IDA

INSTITUTE FOR DEFENSE ANALYSES

A GH-Based Ontology to Support Applications for Automating Decision Support

Francisco L. Loaiza, Task Leader

Steven P. Wartik

March 2005

Approved for public release;
unlimited distribution.

IDA Paper P-3934

Log: H 04-001665

This work was conducted under contracts DASW01 98 C 0067/ DASW01 02 C 0012, Task BC-1-2245, for the Assistant Secretary of Defense (CIO/G-6). The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

© 2004, 2005 Institute for Defense Analyses, 4850 Mark Center Drive, Alexandria, Virginia 22311-1882 • (703) 845-2000.

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (NOV 95).

INSTITUTE FOR DEFENSE ANALYSES

IDA Paper P-3934

A GH-Based Ontology to Support Applications for Automating Decision Support

Francisco L. Loaiza, Task Leader

Steven P. Wartik

Preface

This document was prepared under the Task Order BC-1-1508. It was performed in response to a task objective to formulate sustaining base data exchange requirements using existing information exchange standards specifications.

We wish to thank Mr. Bruce Haberkamp (Army CIO/G6) for his support of this project.

The following Institute for Defense Analyses (IDA) research staff members were reviewers of this document: Dr. L Roger Mason, Jr., Dr. Reginald N. Meeson, Dr. Eugene Simaitis, and Mr. David Wheeler.

Contents

EXECUTIVE SUMMARY……ES-1

1. INTRODUCTION……1

1.1 Background 1
1.2 Purpose of the Document 5
1.3 Intended Audience 6
1.4 Organization of this Document 6

2. PROJECT OVERVIEW 7

3. A C2 ONTOLOGY 9

3.1 Background on the Generic Hub 9
3.2 A GH Ontology 13
3.2.1 Business Rules 13
3.2.2 The Knowledge Model 14
3.2.3 Representing the GH Using Frames 15
3.2.4 Modeling Type Equivalence 17
3.2.5 Modeling Measurable Things 18
3.2.6 Modeling Order 21
3.3 Creating a C2 Ontology 21
3.4 Implementing the C2 Ontology 24
3.4.1 Abstract Classes 27
3.4.2 Inverse Slots 29
3.4.3 Metaclasses 30
3.5 Status of the Implementation 31

4. USING THE ONTOLOGY FOR DECISION SUPPORT 33

4.1 Operational Architecture 33
4.2 System Architecture 33
4.3 A Prototype Implementation 37
4.3.1 Agent Technologies 38
4.3.2 Database Technologies 39
4.3.3 XML Technologies 40
4.3.4 Prototype Components 40
4.3.5 The Data Set 42
4.3.6 The Decision Support Application 43
4.3.7 Implementing Inferencing 46
4.3.8 Metrics 47

5. CONCLUSIONS AND DIRECTIONS 49

5.1 The GH Ontology 49
5.2 The C2 Ontology..... 50
5.3 Use of Ontologies 51
5.4 The Prototype..... 51

REFERENCES ……....REF-1

ABBREIVATIONS AND ACRONYMS……ACROS-1

ANNEX A: Design of the Ontologies……A-1

ANNEX B: Design of a Net-Centric System Using the GH Ontology……...B-1

ANNEX C: Friendly Organization Implementation……..C-2

ANNEX D: GHS Ontology Implementation……...D-1

List of Figures

Fig. 1 Data Sharing Today……....1
Fig. 2 Net-Centric System Execution Concept…….3
Fig. 3 Entity-Relationship Model of Selected GH5 Elements…….....10
Fig. 4 GH5 Location Entity Hierarchy…….....11
Fig. 5 GH5 Reporting Data Entities and Attributes…….....12
Fig. 6 Translation from ER Model to Frame Model……16
Fig. 7 An Illustration of the Type of Hierarchy……18
Fig. 8 Numeric Expression Class Hierarchy……20
Fig. 9 Unit of Measurement Mapping Hierarchy……..... 20
Fig.10 Scenario for C2 Ontology Use……..... 23
Fig. 11 Protégé-2000……28
Fig. 12 Location Hierarchy in Protégé-2000……...29
Fig. 13 Metaclasses in the GH Ontology…….31
Fig. 14 High-Level System Architecture…….34
Fig. 15 Agent System Architecture……..36
Fig. 16 Agent System Sequnce Diagram…….38
Fig. 17 Prototype System Components…….. 41
Fig. 18 Hostile Organization User Interface…….. 42
Fig. 19 Migration of ORGANISATION-TYPE from GH4 to GH5……. 43
Fig. 20 The Decision Support Application User Interface……. 45
Fig. 21 Example Courses of Action……... 46
Fig. 22 Example Justification for Course of Action…….. 46

Executive Summary

Background

The United States Department of Defense is pursuing new approaches to manage information that will accelerate decision making and improve joint war fighting. The DoD's Chief Information Officer (CIO) advocates building a foundation for net-centric operations. This foundation will ultimately entail broad transformation. High-bandwidth network hardware and powerful data fusion tools will be required to take advantage of net-centric benefits. No less important will be new operational processes used by decision makers to react to the myriad sources that will make data available to them.

In a memorandum issued in May 2003, the DoD CIO established an aggressive plan for achieving net-centricity. This memorandum called for plans to be in place by the end of 2003, and for implementation guides to be published during 2004. Architecture development is to be ongoing. The memorandum envisioned active development on net-centric efforts starting at least as early as 2005. Given the scope of information extant in the DoD, this schedule is extraordinarily bold.

The memorandum that lays out the net-centric data strategy establishes seven goals (Table 1), and defines approaches to achieve each goal. It provides an assessment of the challenges DoD faces in providing net-centricity. These challenges are accompanied by mitigation techniques. It is not yet known how effective the techniques will prove, but they seem likely to, at a minimum, establish an environment in which net-centricity can exist.

DoD recognizes that net-centricity has hazards. One is information overload. The total information awareness available to any application on the Global Information Grid (GIG), the network foundation of net-centricity, comes by definition from the huge amount of information the GIG can supply – perhaps orders of magnitude more than some decision makers currently process. Every user of the Google search engine has been confounded by several hundred thousand web pages in response to a simple query. The chance of overlooking information may be of little consequence for the average web

Table 1. Net-Centric Data Goals

Goal	Description
Visible	Applications and users can discover the existence of data assets.
Accessible	Applications and users post data to a shared space.
Institutionalize	DoD processes and practices incorporate net-centric approaches.
Understandable	Applications and users can comprehend data syntax, structure, and semantics.
Trusted	Applications and users can access the pedigree, security level, and access control level of each data asset.
Interoperable	System interfaces, sometimes predefined and sometimes unanticipated, provide many-to-many exchanges of data.
Responsive to User Needs	User perspectives are incorporated into data approaches via continual feedback.

search. Within a combat situation, it can threaten war fighters' lives. DoD applications must be prepared to deal with information overload.

The Data Understandability Goal

One goal of the net-centric data strategy is enabling data to be understandable. For data to be shared, applications and users must be able to obtain and infer data syntax and semantics. Relying on existing data standards is inadequate because these standards are subject to change. Instead, *metadata* that describes data must be available. The DoD expects each Community of Interest (CoI) to create and publish metadata as part of the CoI's commitment to net-centricity.

Data syntax and semantics are to be explicated through ontologies. In this study, we define an ontology as a formalized specification of the semantics applicable to the constituents of our universe of discourse. An ontology formally describes the meaning, properties, and interrelationships of said constituents. In addition, an ontology can specify pertinent inference rules. Because of the latter, an ontology can help applications reason about data. Furthermore, an ontology can provide an intermediate layer that insulates applications from the specific encoding of the data.

The DoD intends to provide metrics and incentives that will encourage CoIs to develop ontologies. These ontologies will be published in a central registry and available to all CoIs through the GIG.

Achieving the goal of enabling data to be understandable may be key to handling information overload. If data can be understood, unnecessary data can be filtered out, or at least flagged as of low interest. Google prioritizes data, but since web page developers do not publish a detailed ontology, Google relies on strategies that can be considered primitive relative to what could be accomplished within DoD's vision. That Google's first few pages almost invariably contain all interesting information hints at what a sophisticated ontology might provide. Of course, the Google user still has to examine several hits to determine if a link is truly interesting. DoD's vision is that an agent will be able to undertake this role.

The above arguments are conceptual. Success of net-centricity will depend on whether CoIs can develop sufficiently expressive ontologies, whether real-world applications can infer enough from them to prevent information overload, and whether an ontology's value exceeds the cost of defining, maintaining, and using it.

An Example for the C2 CoI

IDA has undertaken a study on the creation of a Command and Control (C2) ontology for a net-centric environment. The study is, therefore, concerned with expressing C2 concepts. IDA envisions the C2 ontology to be a fully documented specification that will become a managed resource similar to the Extensible Markup Language (XML) metadata currently published in the DoD Metadata Registry. This C2 ontology will help applications and users understand the syntax and semantics of C2 data.

The ontology is an exploration of the kinds of metadata that could be published in the C2 domain. It is also intended as a test bed to study the degree to which applications can fil-

ter out information of no use to a decision maker and prioritize the rest. Testing this fundamental issue of net-centricity will help DoD plot the course of net centricity.

IDA is testing data filtering by building a prototype system that performs decision support in a simulated battlefield environment. The system simulates a set of hostile organizations and a set of mutually friendly organizations. The friendly organizations collect and share intelligence. Each friendly organization has a decision support application that uses the C2 ontology to analyze the data it receives, and to present the following to a war fighter:

- Threats detected: Inferences regarding threats to any friendly organization.
- Prioritized courses of action: The application's analysis of reasonable decisions in response to threats, measured on a scale of zero to one.
- Rationale: A justification of the priority the application has assigned to each course of action.

In this context, data filtering occurs when a course of action has a priority of zero.

The C2 Ontology

The first component of IDA's study is an ontology (referred to as the GH (Generic Hub) ontology from now on) that expresses two selected C2 concepts threats and operational readiness. Both concepts are central to decision support in C2. Their formalization in the GH ontology will support demonstrations of how applications can put metadata to use.

The GH ontology is based on the Generic Hub data model, version 5 (GH5). The GH5 is a data model developed for land C2 operations and adopted by NATO as a standard.[1] GH5 models instances and types of battlefield objects, namely, FACILITY, FEATURE, MATERIEL, ORGANIZATION, and PERSON, as well as actions, capabilities, holdings, locations, and other typical C2 data. Originally intended to model data exchanges of brigade-and-above echelons, the model is capable of representing information down to the billet level.

Basing the GH ontology on the GH5 is a significant advantage, even aside from the reuse of a well-accepted standard. Many programs already use the GH5 (or some other version of the GH), proving it to be a practical data model. A GH ontology is therefore of immediate interest to the C2 community.

The GH5 is specified in IDEF1X notation. It models the C2 concepts as entities, attributes, and relationships. The GH5 specifies data elements, their valid content, and relationships among entities.

IDEF1X notation, however, goes only so far towards making data understandable. The GH5 may contain an entity named FACILITY and what it means is explicated in terms of the attributes and relationships of FACILITY in the model, but this is not necessarily the totality of what at the semantic level is encompassed by the concept of a facility. This study is an attempt to explore the ability of a GH ontology to express those other semantic facets that traditional data modeling techniques do not cover.

---

[1] Originally STANAG 5523, but that standard has been withdrawn and will be replaced by a new STANAG whose number is indeterminate as of this time.

For example, many GH5 data elements model physical properties such as length, height, width and weight. The GH ontology developed in this study formally expresses what (if any) physical property each data element models, and what semantic interrelationships exist among properties. For example, the EQUIPMENT-TYPE entity has attributes such as equipment-type-height, equipment-type-width, and equipment-type-length. The ontology explicitly defines their semantics to let applications know that these attributes are linear dimensions, that they model height, width, and length, respectively, and that by virtue of being linear dimensions other operations can be performed on them. Thus, the GH ontology expresses formally the fact that length and width can be multiplied to yield an area and that length, width and height can be combined to give a volume. The GH ontology also captures the fact that they are measured in meters (as per the GH5).

The GH ontology also adds information on which elements are conceptually equivalent and which are not. In the GH5, bridge-span-quantity and hangar-area-quantity are both integers. A human can easily infer that they model different concepts; an application cannot, unless – as in the GH ontology – available metadata explicitly states the separate concept each data attribute denotes.

To be of use to an application, i.e., to be machine-process able, the formalized specifications of an ontology must be written in an appropriate language. The most common approach, which comes from research in knowledge representation, is to represent an ontology as a set of frames. The emerging standard for frame-based ontology specification appears to be the Open Knowledge Base Connectivity (OKBC) protocol. In OKBC, a frame is an n-ary relation, of which there are several forms:

- Classes, which describe concepts. Classes are roughly equivalent to IDEF1X entities.
- Slots which describe properties of classes. Slots are roughly equivalent to IDEF1X attributes.
- Facets, which describe properties of slots. Some facets are roughly equivalent to IDEF1X attribute constraints, e.g., enumerations, value ranges.
- Instances of classes. An ontology containing instances is referred to as a *knowledge base*. An ontology shows how to reason; a knowledge base contains information about which one might want to reason.

The equivalences noted above between OKBC and IDEF1X concepts were leveraged to produce the GH ontology. The following mappings are used:

- Each GH5 entity is represented as a class in the GH ontology. OKBC supports class hierarchies, and offers more freedom and flexibility than IDEF1X subtypes (e.g., multiple inheritances, no need for a discriminator column); the GH ontology capitalizes on these features.
- Each GH5 attribute is represented as a slot in the GH ontology.

Furthermore, the GH ontology has:

- A class for each distinct concept modeled by attributes (weight, dimension, radio frequency, etc.).

- A class hierarchy for expressing relationships between physical concepts (e.g., length × width = area).
- A class hierarchy for expressing units of measurement and their interrelationships.
- The ability to express ordering. For example, the values of attribute ammunition-type-calibre-code, which express known calibers of ammunition, are ordered according to size.

These and other features of the GH ontology are an indication of the potential benefits for using ontology-based approaches for future net-centric information systems. For example, the adoption of the GH ontology would allow those CoI's that store their data in relational databases that conform to the physical schema of GH5 to formally capture and centrally manage their business, and inference rules as yet another resource, instead of having to implement, and re-implement them for each GH5-accessing application.

The GH ontology developed in this study is implemented using Protégé-2000, a free, open-source software ontology editor and knowledge base access tool. Protégé-2000 implements the OKBC standard, and provides a sufficiently powerful model to support development of the kind of reasoning needed in a net-centric environment.

In addition to the GH ontology just described, the IDA team has also begun work on a more general C2 ontology. It is written on top of the GH ontology. In other words, its concepts, such as that of a threat, are defined solely in terms of concepts expressed by the GH ontology. Reasoning about threats in the C2 ontology is, therefore, accomplished in terms of GH5 data elements.

The IDA team recognizes that the full definition of concepts such as these is highly complex and would require extensive resources. For this reason, whereas the GH ontology fully represents the GH5 and every GH5 element has an analog in the GH ontology, the C2 ontology is still in an experimental stage.

A Prototype Net-Centric System

The GH and C2 ontologies IDA has developed are intended in part to demonstrate that an ontology can be used to manage information overload in a net-centric environment. How well an ontology performs remains a theoretical question until an application is developed that uses the ontology.

IDA is answering this question by implementing an agent-based system that uses the ontology according to net-centric precepts. The system tests the viability of making data understandable within the C2 domain. It is also intended as a paradigm from which future developers of net-centric software may gain guidance in operational, system, and technical architectures.

The prototype comprises:

- A set of mutually friendly organization subsystems. Each subsystem consists of a decision support application plus a set of supporting agents that access the C2 knowledge base to draw inferences relevant to decision support. These agents provide the decision support application either periodically or on demand, as appropriate to the kind of data on which the agent operates. The decision support

application accesses a DBMS that stores a GH5 data set, populated with continually updated intelligence on situational awareness.

- A set of hostile organization subsystems that, under certain agent-defined conditions, may be considered to pose a threat to one or more friendly organizations.
- A simulation state subsystem responsible for maintaining ground truth (e.g., each hostile organization subsystem is responsible for reporting its current location to the simulation state subsystem).

Each decision support application is in communication with the agents in its subsystem. These agents continually monitor the knowledge base for changes that should be brought to the attention of the decision support application. One agent is devoted to threat detection. When conditions in the knowledge base indicate the existence of a threat, it notifies the decision support application. It also notifies a threat response agent, which formulates courses of action that seem reasonable in light of the threat. The threat response agent sends these courses of action to the decision support application, which displays them to the decision maker. See Figure ES-1.

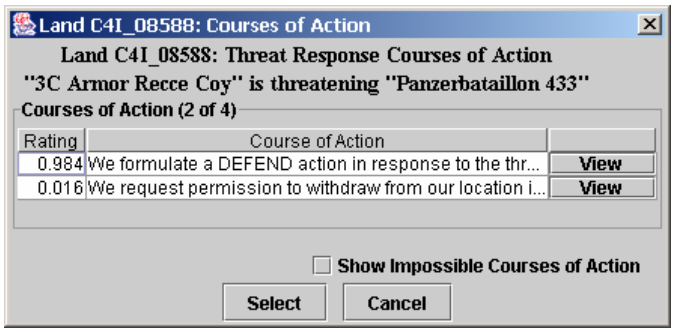

Figure ES-1. Example Courses of Action

The system is implemented using freely available software components and standards. These include the FIPA-OS implementation of the Foundation for Intelligent Physical Agents (FIPA) standards, the open source software MySQL relational database management system, and the Zeus XML DTD parser generator. Aside from these components, the system contains approximately 18,000 lines of Java and 2,000 lines of other languages. The system uses C2 data from a NATO exercise conducted at Ft. Worth, Texas.

Conclusions and Directions

Implementing DoD's net-centric data strategy poses some interesting operational and technical challenges. The ontology IDA has developed addresses the degree to which C2 data models can be made syntactically and semantically understandable, a crucial goal of net-centricity.

In addition, the study shows that an ontology can be employed to add useful semantics and rules to underlying data models in a reproducible and reusable manner. It goes without saying that the IDA C2 ontology does not incorporate all the reasoning capability that it could. This is an obvious area for extension. Further research is necessary to assess all the C2 ontology extensions needed to be useful in a net-centric environment. Nevertheless, the prototype system already documents preliminary aspects of the C2 ontology utility.

Many of the tools IDA used to create the ontologies and implement the prototype are of recent origin. They are of reasonable quality but not validated for use in mission-critical applications; they crash occasionally, and their application and user interfaces are evolving. DoD, in collaboration with industry and academia, may need to devote additional resources to the development of a software infrastructure for net-centricity.

1. Introduction

1.1 Background

Throughout history, the common complaint of decision-making war fighters has been lack of information. The absence of a common and complete operational picture has resulted in mishaps ranging from minor discomforts to the Light Brigade's infamous blunder. Making decisions that look reasonable in hindsight generally requires full situational awareness, something war fighters traditionally find sorely lacking.

The DoD's Global Information Grid (GIG) [GIG 2001] is a significant step towards the ability to provide full situational awareness. The GIG, which aims to interconnect all war fighting data, is a major paradigm shift for the DoD. It is in part a recognition that the old strategy of centralized data administration across the entire department is impractical. There exist too many deeply entrenched cultures within DoD to create fully seamless data exchanges among all domains. Data sharing cannot be mandated in such an environment. To put it another way, data cannot be "pushed" against someone's will. Figure 1 illustrates this environment. Some systems are developed with well-defined interfaces and can exchange data, but these systems tend to be within individual domains. Other systems outside the domain must rely on data content in a globally accessible database. With no control over this content, other systems incur risk and their developers often find that duplicating the content is less costly, or as least better justifiable, than using the existing data. The result is a plethora of stovepipe systems.

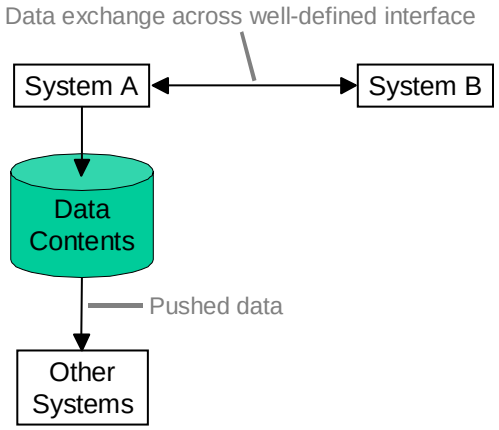

Figure 1. Data Sharing Today

A 2003 memorandum from the office of the DoD Chief Information Officer presents a new vision for handling data [Stenbit 2003] based on the Semantic Web [Berners-Lee 2001]. In this vision, the DoD will nurture individual Communities of Interest (CoIs). Each CoI will be responsible for identifying the set of data it manages. Each CoI will be encouraged, though not strictly required, to make its data visible on the GIG. The DoD

believes this increased data visibility will better enable effective decision-making than the current environment.

This paradigm is a double-edged sword. On the one hand, CoIs are under no obligation to make data visible, nor are they required to avoid duplicating existing data sources; this gives each CoI considerable freedom and flexibility. On the other hand, the existence of data from one CoI should discourage others from undertaking the effort of regenerating the same data over and over.

The DoD intends CoIs to encourage the growth of *net-centricity*. Net-centricity is the realization of network-centric operations, wherein people, processes, and systems can freely share information. Net-centricity represents DoD's approach to abandoning centralized data information and consequent stovepipe systems. The growth of net-centricity can lead to more data, better managed.

[Stenbit 2003] establishes seven goals, the achievements of which are crucial to achieving net-centricity. The following is a summary of these goals:

1. Make data visible: Users and applications must be able to discover the existence of data.
2. Make data accessible: There must exist shared spaces to which users and applications can post data.
3. Institutionalize data management: Approaches to managing data are incorporated into DoD processes and practices. Note that this goal covers only data management, not the data itself.
4. Enable data to be understandable: Users and applications must be able to comprehend data structure and semantics.
5. Enable data to be trusted: Users and applications must be able to determine and assess the degree to which they must trust each datum. In other words, attributes such as security and pedigree must be known.
6. Support data interoperability: Metadata must be available to allow mediation or translation of data between interfaces where necessary.
7. Be responsive to user needs: Approaches will evolve in response to user feedback.

Figure 2 shows how realizing these goals will help achieve net-centricity. The area inside the dashed box denotes a single CoI. Within this area, all systems communicate using well-defined interfaces, just as in Figure 1. This is possible because the CoI can exert control over the systems it defines, mandating standards as it sees fit. However, instead of simply making its data content available, a system within a CoI publishes its data to a shared space. The shared space contains not just the data (Data Contents) but also information that can help other systems discover if the data meets their needs (Discovery Metadata Catalog) and information on how to interpret the data (Structural Metadata). Rather than creating fixed interfaces, other systems may dynamically search for data – i.e., browse through the discovery metadata catalogs in a known set of shared spaces – and choose the set that best fits their current situation. Systems A and B are still free to

communicate through their well-defined interfaces, if that is most efficient, but they are being much more systematic about the availability and nature of their data.

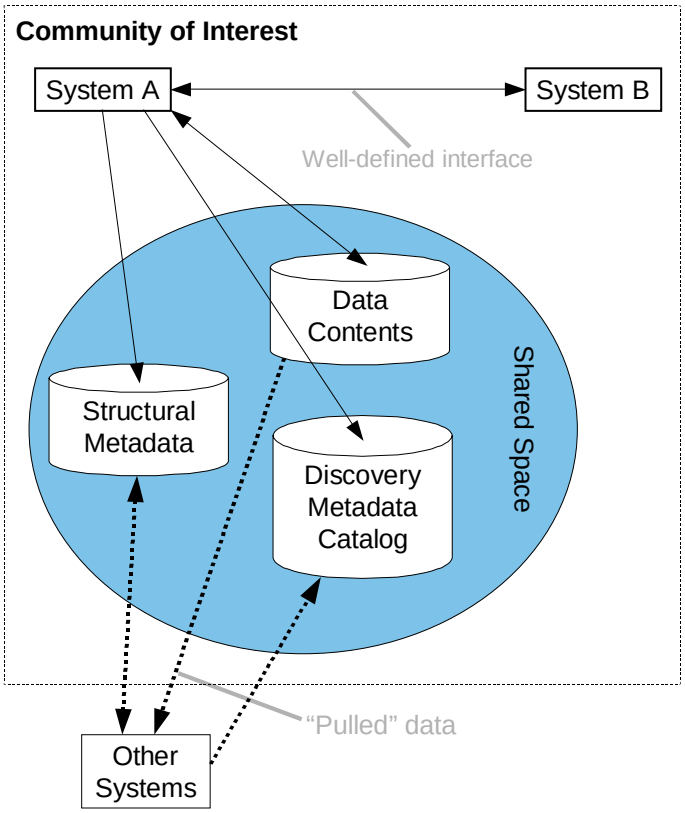

Figure 2. Net-Centric System Execution Concept

Net-centricity provides an excellent foundation for improving decision support applications. A decision support system can continually search the GIG for whatever data is most appropriate, from information on the next town to prospects for resupply from halfway across the globe.

By the same token, net-centricity carries the hazard of information overload for the war fighter. Increasing the amount of information available to a decision support system – as the GIG is intended to do – increase the amount of information that system must process. A decision-making war fighter may then see a corresponding increase in the amount of information presented to him. Any extra information he must process increases the time he needs to reach a decision. Given that the GIG is globally connected, and that many decision support systems rely on local information, it is reasonable to expect that decision support systems of the future will have orders of magnitude more information available. Clearly, these systems must be able to decide what information is relevant and discard the rest. Otherwise, future decision makers will spend so much time analyzing data that they will be unable to make timely decisions.

The goal of making data understandable addresses information overload. Understandability means more than structural and semantic descriptions. It implies an ability to determine the value of data. An application that is pulling data must be able to filter the data based on perceived need.

The net-centric strategy identifies CoI-specific ontologies as one facet of making data understandable. An ontology is an expression of the nature and relations of some entity or set of entities. Examples of what an ontology encompasses include data categorization schemes, thesauri, vocabularies, and taxonomies. The major purpose of an ontology in a net-centric environment is to describe the data of a CoI in ways that promote use of that data by systems outside the CoI. An ontology contributes to the discovery metadata catalog by helping other systems locate data content describing concepts of interest. For example, a system searching for data describing a tank might find a discovery metadata catalog listing a tank as a known concept in a shared space. Of course, the system needs to know if the tank in question is a tracked fighting vehicle or a container of fuel; ontologies in the form of thesauri and vocabularies help disambiguate.

An ontology also contributes to structural metadata by describing entities and their relationships. A (tracked fighting vehicle) tank has a (fuel) tank. Taxonomies of a well-defined ontology make such relationships clear, and allow systems to draw inferences such as the capacity of the (fuel) tank of a tank.

To be useful in a net-centric environment, an ontology must be formal and executable. System developers have often employed ontologies informally as part of system documentation (e.g., drawings of data models), which may suffice for a human reading that document but of no use to a decision support software application. The premise of an ontology in the net-centric data strategy is that other systems can access it to determine whether the data it describes is useful or not.

The DoD CIO envisions that each CoI will develop and maintain ontologies of its data and register those ontologies in a central DoD Metadata Registry. Other systems search this registry (or registries; several may exist) when they need information. The registry both describes the type of information (the details are in the structural metadata catalog) and the shared space in which to find it. Other systems use registries to identify likely data sources, then use a shared space to retrieve data and determine its meaning.

The power of ontologies, then, will stem from their ability to promote dynamic access to data. Applications will no longer be constrained by a fixed set of interfaces. Each time they desire information, they can search as broadly – or as narrowly – as is appropriate to their context.

The underlying assumption is that ontologies can be sufficiently expressive to let applications determine the relevance of data. An ontology is a specification of data structure and semantics. To support net-centricity, an ontology will have to be written using a formal language that is not only understandable to humans but also machine-process able. When that is the case, an application looking for some type of data ought to be able to use an ontology to determine:

1. Whether a CoI offers that data.
2. Whether or not the data is somehow better (e.g., more trustworthy) than seemingly equivalent data from another CoI.
3. Whether the data is in a form that will be useful.

4. How the data can be retrieved.

No general-purpose solutions exist to all these problems. Much research has been devoted to aspects of them, and many researchers have proposed solutions that have at least been demonstrated to work in constrained environments.[2] Much of the recent interest in ontologies has arisen out of the World Wide Web and the belief that ontologies will be useful in electronic commerce. How the results from that community translate to military applications remains to be seen. A model builder searching eBay for tanks may find items for sale in categories like Fuel Tanks and Women's Shirts (tank tops) as well as Toys and Hobbies. The cost to society of these improper hits is low. The war fighter whose decision support system mistakes a fuel container for a tracked fighting vehicle might not be so lucky.

Tim Berners-Lee, the creator of the World Wide Web, has said that his original vision of the Web isn't at all what exists today. He wanted the development of a Semantic Web in which applications could relate and differentiate concepts [Berners-Lee 2001]. This sort of web could help provide applications with the necessary technology to achieve DoD's vision. This paper explores semantic web concepts in the context of decision support.

1.2 Purpose of the Document

The Institute for Defense Analyses (IDA) has undertaken an exploration of how ontologies might be used in a net-centric environment. The objective of this exploration is to determine the validity of the assumption that ontologies can be sufficiently expressive to determine the relevance of data. The purpose of this document is to describe progress made to date, to discuss steps that remain to be taken to complete this exploration, and to identify issues that need to be resolved to make ontologies live up to their potential in contributing to net-centricity.

A related issue is whether an ontology adds expressive power above and beyond existing data modeling technologies, and whether the expense of adding expressiveness is justified – i.e., can ontologies be built quickly and cheaply? To state the issue in concrete terms: why use ontologies instead of relational database management systems? Any information that can be described in an ontology can also be stored in a DBMS (the ontology editor tool used in this project offers the option of storing an ontology in a DBMS). Ontology technology must offer something relational technology does not.

This paper does not directly discuss the merits of ontologies vs. relational databases. To do so would be unfair: ontology tools are not as mature as relational database tools. Any comparisons would be based mostly on what research says is computationally possible. That gives no satisfactory answer to the question of development cost. In any case, this project was begun after DoD had already decided to develop ontologies for the GIG. The issue of their cost is therefore irrelevant to our objectives.

---

[2] The Control of Agent Based Systems (CoABS) program is one example. See http://coabs.globalinfotek.com/, which provides an overview of, and links to, several relevant projects.

Certain places in the document, especially Section 5, cover the merits of ontologies in a general way. Where possible, we show how and why ontologies increase expressive power. However, we do so qualitatively.

1.3 Intended Audience

The intended audience for this document includes everyone contributing to achieving the DoD net-centric data strategy, especially:

1. Analysts developing ontologies.
2. Designers and developers of decision support systems.
3. Designers and developers of the GIG.
4. Department of Defense officials responsible for implementation of the DoD CIO's net-centric data strategy.

This document assumes familiarity with entity-relationship modeling, and with command and control (C2) concepts. Familiarity with the Generic Hub information exchange data model [NATO 2002] is helpful but not required.

1.4 Organization of this Document

This document is organized as follows:

- Section 1 (this section) presents concepts, potential, and liabilities of a net-centric environment.
- Section 2 gives an overview of the project IDA has undertaken to address selected liabilities.
- Section 3 gives a detailed discussion of C2 and C2-related ontologies IDA has created. These ontologies illustrate what a community of interest might publish on the GIG.
- Section 4 discusses the prototype system IDA has created to demonstrate an operational architecture for use of the ontology.
- Section 5 discusses the implications of the work performed to date, then presents areas for future work.

2. Project Overview

This task is part of work currently being performed at IDA, in support of the United States Army, aimed at building non-trivial work products that implement portions of a net-centric environment. These work products are focused on the net-centric data strategy goal of making data understandable (Goal 4) – in particular Goal 4.1, defining CoI-specific ontologies. The current task explores the feasibility of building an ontology that would be of use to war fighters. The existence of an ontology of this nature is essential to determining the viability of Goal 4 using today's technology, and to understanding any technical innovations that are necessary before it can be realized.

This project documents the viability of Goal 4 and the lessons learned that will guide others in developing ontologies and systems. The project makes recommendations on technologies for developing, maintaining, propagating, and accessing ontologies. It describes a system architecture for accessing the ontologies in a net-centric environment.

This project documents the following work products:

1. An ontology for the domain of command and control (C2). C2 is a domain with a long, important history in warfare. It constitutes an area for which a CoI will almost certainly exist, perhaps at the Service level, perhaps at the Joint level for all of DoD.

   The ontology developed in this project is based on the Generic Hub (GH), an existing Information Exchange Data Model (IEDM), and includes all GH entities, attributes, and relationships. In that sense it is a complete C2 ontology. However, IEDMs typically consist of low-level concepts. They do not define higher-level abstract notions that are of interest to a war fighter. For example, although the GH models the concept of a battlefield object and contains enough information to determine whether one battlefield object presents a threat to another, it does not explicitly model the concept of threat – something a battlefield commander needs to know.

   The results of the task include selected high-level concepts in the C2 ontology. The presence of these concepts demonstrates the ability of a C2 ontology to model complex concepts and to support reasoning about those concepts.

2. A prototype system of decision support applications. Creating a C2 ontology is a useful exercise in its own right – it aids in the study of formal relationships between entities – but does not demonstrate that the ontology is formal or complete enough to be useful. Other systems must be able to use the ontology to identify, search, access, and interpret C2 data. The mechanisms by which these actions occur must not take place using the proverbial well-defined interface of Figure 1; they must happen using the paradigm of Figure 2.

   The IDA team has built a prototype system that uses the C2 ontology in conformance with the architectural principles of net-centricity. The system demonstrates

how to avoid information overload through ontology-based decision support applications.

The prototype system is not a production level application; that would be well beyond the scope of IDA's task. It has been implemented to highlight operational, system, and technical architectures that other developers of net-centric systems might find useful.

3. Documentation of Practices and Principles. In addition to the description and analysis related to the two products mentioned above, the document also contains lessons learned in developing them, which hopefully future developers of ontologies and ontology-based applications will find helpful. The report contains both a methodology for developing an ontology, and a methodology for developing ontology-based applications. The report also provides recommendations on technologies and how they can be employed.

3. A C2 Ontology

In a net-centric environment, where an ontology must be software interpretable, the ontology must be formally defined. Formality implies rigorously defined structures and semantics. Extensive formalization of C2 structures and semantics has been accomplished in the data layer specifications for information exchanges among C2 databases. Rather than to reinvent these C2 concepts, IDA has chosen to base its C2 ontology on an existing formal C2 data model. This C2 data model is the Generic Hub (GH) Information Exchange Data Model (IEDM).

3.1 Background on the Generic Hub

The GH was developed by the Army Tactical Command and Control Information System (ATCCIS) to support land C2 operations in a multinational environment for echelons to include Brigade, Corps and above. It emphasizes data that national armies would share under such conditions and purposely omits details traditionally handled by national C2 systems, such as personnel-related information. The model also reflects the philosophy that planning documents must be boiled down to the specific actions contained therein, and mapped to **who** does the action, against **whom**, with **what**, **where**, and **when**.

For the development of the C2 ontology, the IDA team chose version 5.2 of the Generic Hub (released January 31, 2003), hereafter referred to as the GH5. The GH5 is the NATO standard IEDM for joint and coalition operations. It has also been adopted by the Army's Future Combat Systems (FCS) program, and has been used at the Naval Postgraduate School and the Naval Undersea Warfare Center (NUWC) for various data interoperability projects.

Figure 3 uses an entity-relationship diagram to show selected GH5 entities and relationships, in particular those that play a role in this paper. The GH5 models battlefield objects. Each instance of a battlefield object is modeled as an OBJECT-ITEM. There are five kinds of battlefield objects: FACILITY, FEATURE, MATERIEL, PERSON, and ORGANISATION. An OBJECT-ITEM has one or more types. A type is modeled as an instance of OBJECT-TYPE, which is a super type of five kinds of battlefield object types that correspond to the different types of battlefield objects, i.e., FACILITY-TYPE, FEATURE-TYPE, MATERIEL-TYPE, PERSON-TYPE, and ORGANISATION-TYPE.

Each of the type entities has attributes (not shown) that further categorize the type as one of a set of enumerated elements. For example, a PERSON-TYPE may be DETNEE (a detainee, i.e., a person in custody), GOVEMP (a government employee), JRNLST (a journalist), or any of nineteen other values. Instances are not mutually exclusive. The GH5 would model a journalist in custody as an instance of PERSON associated with two instances of PERSON-TYPE, the first with an attribute value of JRNLST and the second with a value of DETNEE.

Figure 3 shows associative entities, i.e., those that model attributes of many-to-many relationships, in italics. For example, the HOLDINGS entity shows the association between an OBJECT-ITEM and the OBJECT-TYPEs it possesses. One attribute of HOLDINGS is the quantity held. For example, that a battalion holds a certain quantity of ammunition is modeled by an association between the instance of ORGANISATION modeling the battalion and the instance of MATERIEL-TYPE modeling that kind of ammunition; the quantity is captured as an attribute of the instance of the relationship between OBJECT-ITEM and OBJECT-TYPE. (The GH5 can also model associations between instances of OBJECT-ITEM; a battalion can, if it wishes, track every individual bullet. Such detail is seldom worth the effort, however, and Figure 3 does not show the GH5 elements for modeling it.)

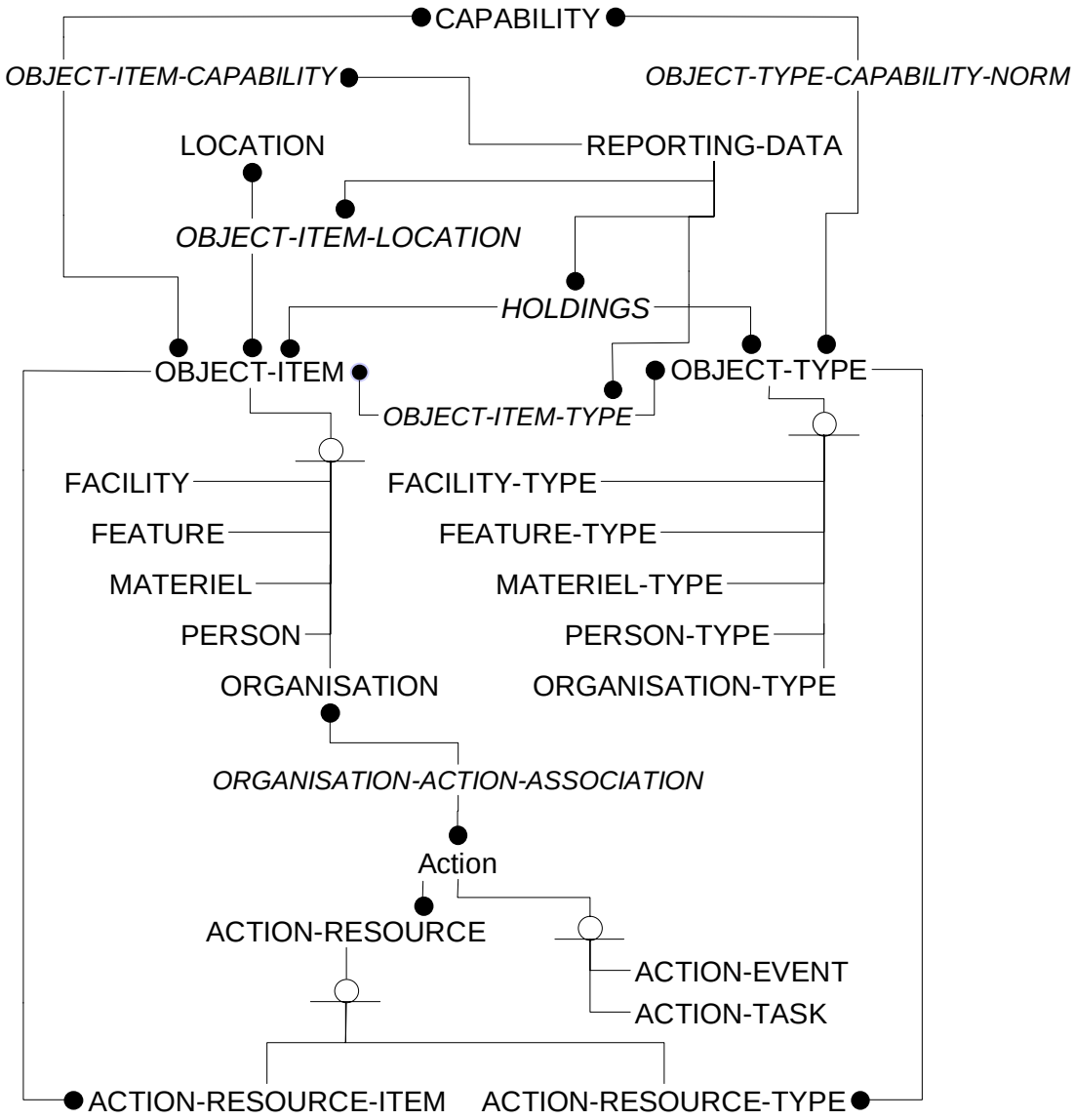

Figure 3. Entity-Relationship Model of Selected GH5 Elements

The potential abilities and effects of each battlefield object, and each battlefield object type, are described as a set of CAPABILITY instances. Each CAPABILITY instance denotes some physical ability or effect; the GH5 enumerates a standard set. There are capabilities

that describe storage, mobility, surveillance, firepower, mission, and engineering abilities. An OBJECT-TYPE has a nominal set of capabilities (OBJECT-TYPE-CAPABILITY-NORM). An OBJECT-ITEM has an actual set of capabilities (OBJECT-ITEM-CAPABILITY).

A battlefield object has a location. The GH5 models location as an association between OBJECT-ITEM and LOCATION. The LOCATION entity is the super type of a set of entities that model geodetic locations either as a point or as a shape. A point can be referenced in either two or three dimensions. A shape can be either two or three dimensional. Figure 4 shows the different types of locations the GH5 models.

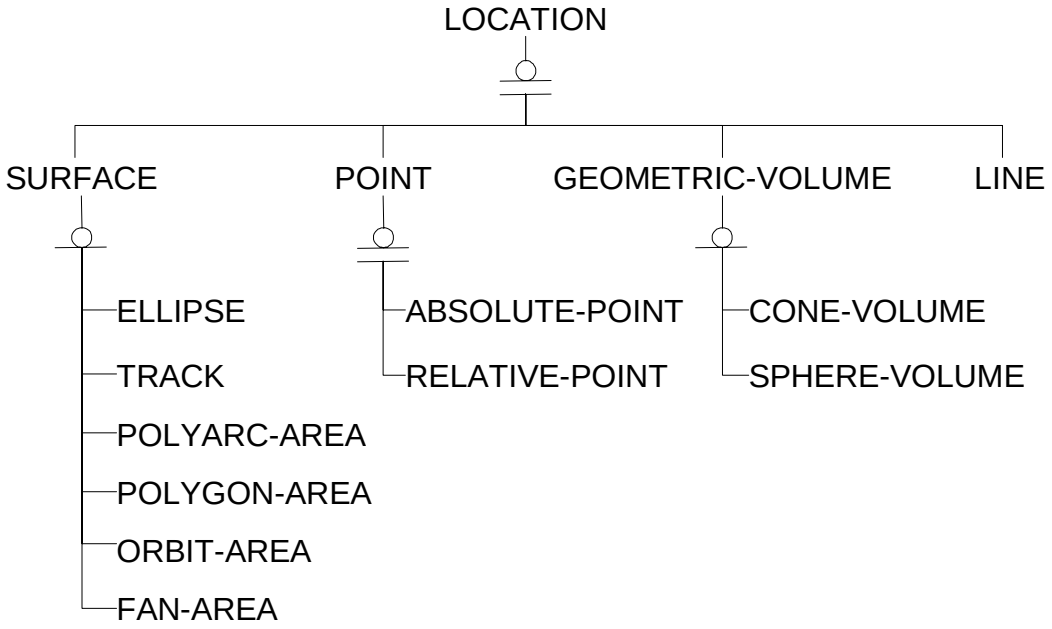

Figure 4. GH5 Location Entity Hierarchy

Except for OBJECT-TYPE-CAPABILITY-NORM, all GH5 associative entities have a relationship to REPORTING-DATA. REPORTING-DATA is a central element. It is shown in Figure 5. Its attributes provide for time-ordered specifications. For example, the location of an instance of OBJECT-ITEM over time is modeled as a set of relationships between that instance of OBJECT-ITEM and a set of instances of LOCATION. Each OBJECT-ITEM-LOCATION has an associated REPORTING-DATA stating the day and time for which the relationship is valid, i.e., the time when the OBJECT-ITEM was reported to be at a Location. Similarly, each OBJECT-ITEM-CAPABILITY relationship has an associated REPORTING-DATA instance stating the time when the OBJECT-ITEM was observed to have the stated CAPABILITY. There is however no REPORTING-DATA associated with an OBJECT-TYPE-CAPABILITY-NORM. OBJECT-TYPE-CAPABILITY-NORM describes nominal, not observed, capability.

The REPORTING-DATA entity has attributes reporting-date and reporting-time that model when an observation was made, but REPORTING-DATA does has model the period during which the observation is valid. That aspect is handled in one of two subtypes: REPORTING-DATA-ABSOLUTE-TIMING and REPORTING-DATA-RELATIVE-TIMING. As their names imply, they model reporting data in either absolute or relative terms. The effective-

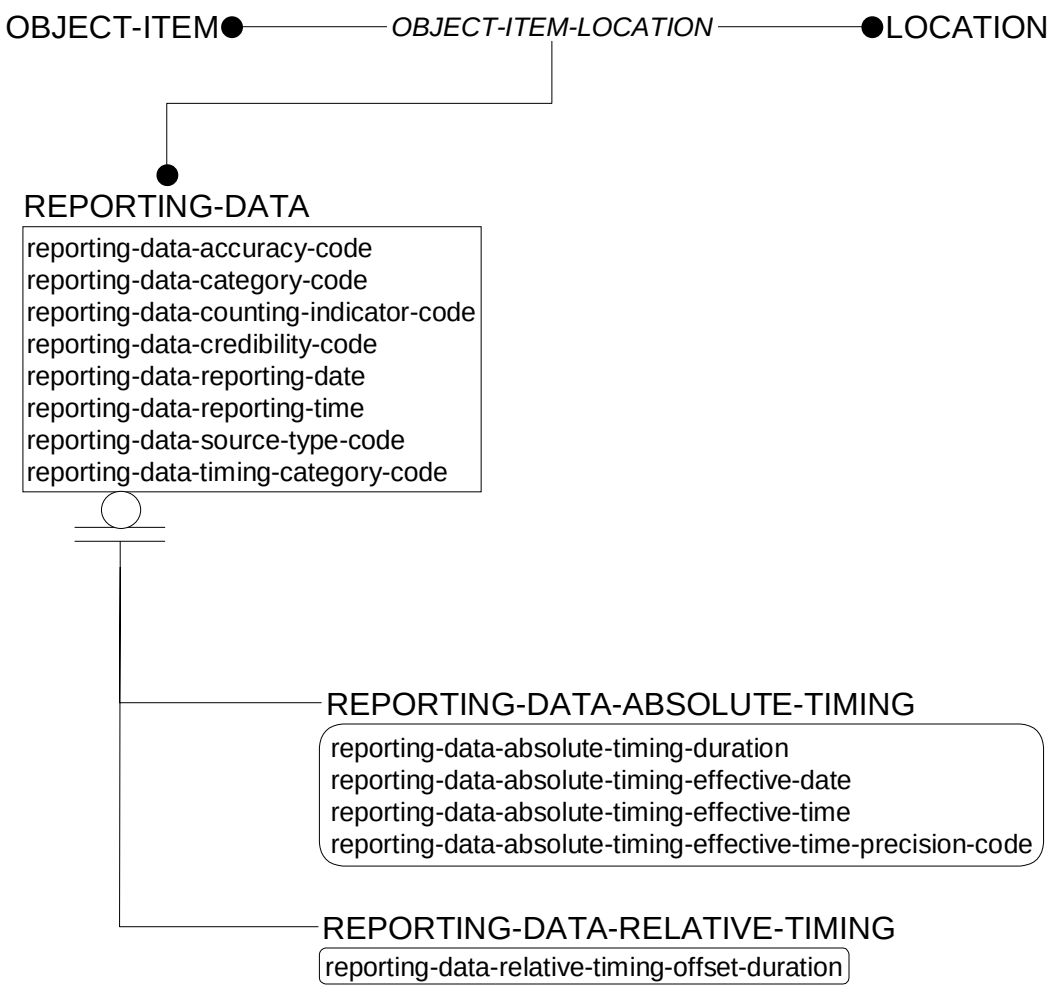

Figure 5. GH5 Reporting Data Entities and Attributes

date and effective-time attributes of REPORTING-DATA-ABSOLUTE-TIMING model the effective starting moment, and the timing-duration attribute lets the effective ending moment be calculated. The effective time of relative recording data is specified using an offset (timing-offset-duration) with respect to an instance of an ACTION-TASK.

The GH5 can model planned and actual occurrences of activities. These are modeled as instances of ACTION. ACTION has two subtypes:

1. ACTION-EVENT, denoting a past or present "incident, phenomenon or occasion of military significance" for which no planning is known.
2. ACTION-TASK, denoting a past, present, or future activity for which planning is known.

An ACTION can be associated with an ORGANISATION. The relationship between the two describes the role of the ORGANISATION with respect to the ACTION (approves, controls, initiates, etc.). An ACTION consumes resources. These may be specified either as resource types or as actual items, depending (usually) on whether or not the ACTION is a template.

Similarly, an ACTION has objectives. These also may be specified either as resource types or as actual items.

3.2 A GH Ontology

The ontologies currently being explored in academia, government research institutions, and industry are expected to support basic reasoning tasks by allowing users and applications to draw inferences about data. The GH is an IEDM. It provides a suitable point of departure for an ontology insofar as its entities, attributes, and relationships categorize concepts. However, like any IEDM, the GH lacks the formal inference capabilities expected of an ontology.

3.2.1 Business Rules

IDA's early investigations therefore focused on identifying GH business rules and figuring out how to formalize them to support inferencing. In practice, it is not difficult to uncover many such rules. Most entities, attributes, and relationships in the distributed GH5 model have explicatory textual definitions. Some of these definitions provide hints intended for application implementers but equally useful for ontology developers. Other sources for business rules come from GH naming conventions. The following are some examples of business rules IDA uncovered and formalized.

Much of the GH models physical phenomena. It follows that applications using a GH-based ontology would want to reason about relationships among the sorts of physical entities the GH models. For instance, the attribute hanger-area-quantity of the AIRFIELD entity measures (obviously) the area in a hanger available for storing aircraft; the textual definition of this attribute further states that the units of this attribute are square meters. The GH also includes an AIRCRAFT-TYPE entity, attributes of which include its dimensions – height, length, and width – also in meters. Useful inferences from this information would include:

1. Can an airfield store a particular type of aircraft?
2. Can a squadron of aircraft of the same type be stored at an airfield?
3. Can several squadrons of aircraft of mixed types be stored at an airfield?

The GH contains data elements to model the first and second inference, but not the third. Thus the ontology must include a framework through which inferences related to physical properties of entities can be made. For the aircraft example, the ontology must support modeling of measurable concepts (dimensions and area) and units of measurement. It must be possible to infer that storing aircraft requires knowing the area each aircraft occupies, that quantity of aircraft stored requires an area equal to at most the sum of area of each aircraft, and that the units of measurement of area are equivalent – or if they are not, how to convert between them.

Equally important is knowing what is not relatable. In the GH, an attribute whose name ends with -quantity denotes some measurable quantity. However, quantities are not unit less, and unless the units are equal or convertible, the attributes they measure are not comparable. The attribute bridge-span-quantity is defined as follows:

> The numeric value that represents the number of sections that a specific Bridge may have.

Obviously hangar-area-quantity has no relationship to bridge-span-quantity, despite the similarity of their names. It follows that the ontology must be precise as to conceptual interrelatedness at the attribute level, not just at entity level as is true for the GH. The GH specifies a domain for each attribute, and shares domains across attributes where appropriate. For example, it specifies separate domains for the latitude and longitude coordinate attributes of an ABSOLUTE-POINT. Latitude is a 9-digit number with 6 digits following the decimal point, and longitude is a 10-digit number with 6 digits following the decimal point, a formulation that is quite logical when one recalls that longitudes have a range twice as large as latitudes.

However, the GH is not always precise enough. It uses the same domain for EQUIPMENT-TYPE height, width, and length, implying interchangeability among the three aspects of size. It also uses some domains named for their representation. The loaded-weight-quantity and unloaded-weight-quantity attributes of EQUIPMENT-TYPE have domain qty-non-negative-real-12-3-optional. As it happens, these two attributes model equivalent concepts (weight) but the GH also uses that domain for the capability-norm-quantity attribute of OBJECT-TYPE, which can model weight but can model other properties too. Most string-valued domains are named for their representation; both action-name and physical-address-street-name use the domain name-50 although they are clearly not interchangeable. Such genericity is acceptable in an IEDM, but diminishes the power of an ontology.

IDA formulated business rules that segregated domains insofar as was possible. The result is that the ontology can only be used to draw inferences among properly related concepts.

3.2.2 The Knowledge Model

After deciding to formalize certain GH concepts, IDA needed to choose a knowledge model in which to represent the ontology. The IDEF1X model [NIST 1993] used to represent the GH5 was inadequate. GH5's designers did not formally state many of the business rules needed in an ontology because they could not.

The emerging standard for representing ontologies appears to be the Open Knowledge Base Connectivity (OKBC) protocol [Chaudhri 1998]. In this model, derived from earlier work on the Ontolingua server [Farquhar 1997], knowledge is represented as a set of *frames*. A frame is an n-ary relation, of which there are several standard forms:

- Classes, which describe concepts. Classes are organized hierarchically. There is a special class called: THING that is the super class of all other classes.
- Slots, which describe properties of classes. A slot is said to be *attached to* a class. A slot has a value.
- Facets, which describe properties of slots.
- Instances of classes.

These forms are not dissimilar to the elements of an entity-relationship model. As will become clear, they are far more powerful.

Slots are independent of classes. If an ontology contains a slot *S*, *S* may be shared among several classes. The implication is that all the classes have a common property. This is unlike the IDEF1X model, where an attribute's name space is that of its entity. Two entities may have a name attribute; but, though a reader might assume from the common names that the attributes model the same kind of information, nothing in IDEF1X enforces or even alludes to that assumption.[3] In OKBC, then, changing a slot implies that the change applies to all classes containing that slot.

There are two kinds of slots: *template* slots and *own* slots. A template slot is attached to a class. It is inherited by all subclasses of that class. An own slot is attached to a frame. An own slot describes a property of a frame, and is not inherited if attached to a class. An own slot is useful for specifying something about a specific frame: some property of a class or slot (e.g., a human-readable name). Template slots have the interesting characteristic of becoming an own slot for an instance of a class. For example, if an ontology has a slot *S*, it would be useful to provide *S* with an own slot to denote a name for *S*. If the ontology has a class *C*, and *C* has template slot *S*, then an instance of *C* has an own slot *S*. In other words, each instance of *C* has a unique value of the property denoted by *S*.

Facets specify constraints on slots. OKBC defines a standard set of constraints, including type (e.g., string, number, date, instance, class) and cardinality. OKBC also allows ontologies to extend the set of facets attached to a slot.

An OKBC ontology is a set of slots, classes, and facets.[4] An ontology defines concepts but does not identify concrete instances. An ontology plus a set of instances of the classes in the ontology is termed a *knowledge base*. An ontology provides a framework for reasoning; a knowledge base allows reasoning about specific data. The distinction is analogous to the difference between an IDEF1X specification of a physical database and a data set structured according to that specification.

3.2.3 Representing the GH Using Frames

The following is a description of the general approach IDA used to translate the GH into a frame-based representation of an ontology.[5] Figure 6 illustrates the concepts for the translation of the three GH entities OBJECT-ITEM, OBJECT-ITEM-ASSOCIATION, and ORGANISATION.

1. Each GH entity was modeled as a class in the GH ontology. Each class had an own slot to denote the class' name, taken from that of the entity. Because of considerations of implementing the ontology in programming languages, IDA

[3] The GH naming convention of prefixing entity names to attributes deliberately avoids misleading readers into such thinking.

[4] The OKBC standard gives implementers considerable freedom. What actually constitutes an ontology is more than this sentence implies.

[5] This ontology is referred to in this report as "the GH ontology." The phrase "the GH" refers to the GH IEDM.

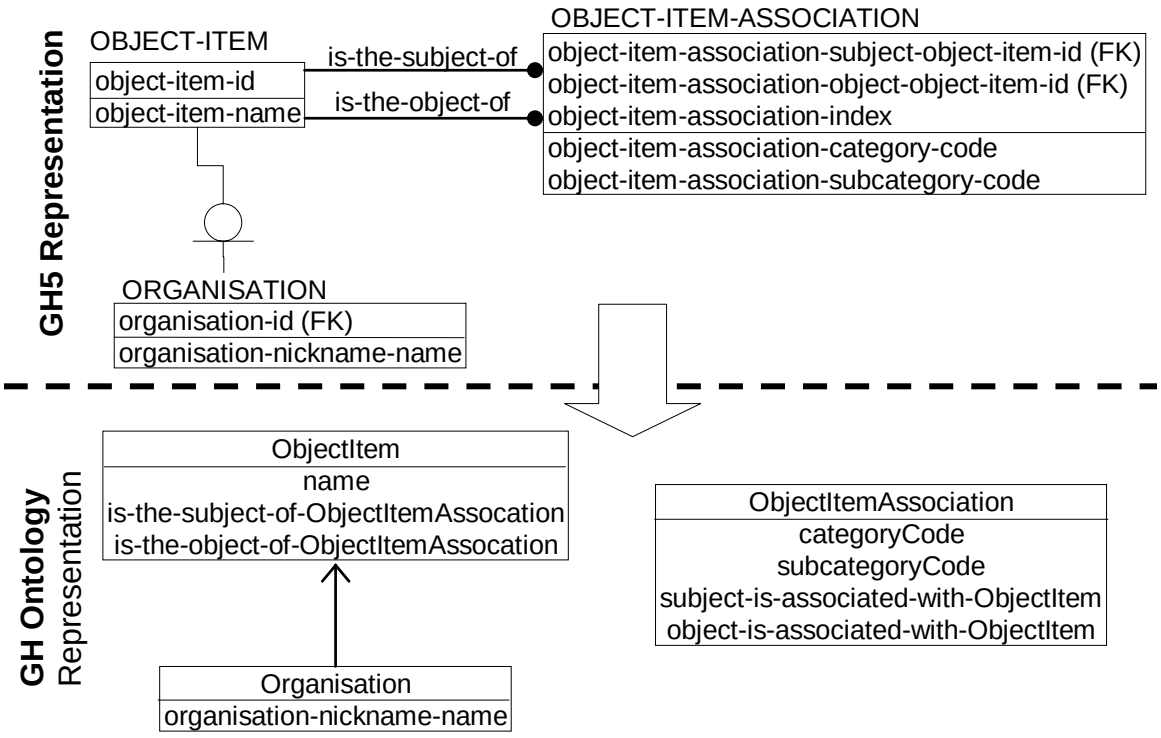

Figure 6. Translation from ER Model to Frame Model

adapted the GH's naming conventions. The hyphens were eliminated, and the uppercase names were changed to initial capitals. Thus entity OBJECT-ITEM became ObjectItem. An own slot was also used to provide documentation on each class. The documentation was copied verbatim from the GH description of the entity.

Entity subtype/super type relationships in the GH were converted to class subtype/super type relationships in the GH ontology. Class ObjectItem has subclasses Facility, Feature, etc. IDA did not encounter any situations in which multiple inheritances were desirable.

Occasionally, IDA identified sets of classes that were related but whose corresponding entities in the GH did not share a super type. IDA grouped these classes under a common super class. Classes MaterielTypeEstablishment and OrganisationType-Establishment, for example, have the super class ObjectTypeEstablishment in the GH ontology.

2. Each organic GH attribute was modeled as a template slot in the GH ontology. Each slot had a facet to denote the slot's name, taken from the name of the attribute. As with classes, naming conventions were adapted: the prefixed entity name was dropped, hyphens were dropped, and words other than the first were capitalized. So airfield-hanger-area-quantity became hangerAreaQuantity. (Many naming conflicts resulted, especially with attributes whose names translated to "name", but the tool IDA used to implement the ontology could handle those conflicts.)

   The slot has a facet specifying cardinality of 1. The GH specifies which attributes are mandatory and which are optional in the GH ontology; a Boolean required

facet was used to express this information. A facet for recording documentation was also used; the documentation was taken directly from the attribute.

3. Each GH attribute that is a foreign key, and therefore implements a relationship, was modeled in the GH ontology as a template slot in the class derived from the entity of which the attribute was a foreign key. The cardinality facet of the slot specifies that the slot can have multiple values, with no inherent upper limit. The facet denoting the slot's name is formed from the name of the relationship in the GH. The value type of the slot is "Instance"; in other words, the value of the slot is an instance of a class. When the value type is "Instance", one also specifies the set of classes of which an instance must be a member. For these slots, the set is the class derived from the entity containing the foreign key attribute. For example, the attribute object-item-id of GH entity HOLDING was represented in the GH ontology as slot has-Holding, which was attached to class ObjectItem. The value of slot has-Holding is a set of instances of class Holding.

3.2.4 Modeling Type Equivalence

One business rule IDA desired to implement was a specification of which slots contained conceptually equivalent values. Mixing aircraft-parking-area-quantity and lower-frequency-quantity would not do, even though the GH specifies that both happen to be 9-digit integers. The ontology should clearly identify the domains as different.

The problem was solved as follows. All classes derived from GH entities were made subclasses of class Land C4I Entity, itself a subclass of: THING. IDA then created another subclass of: THING, called Type. Class Type groups the domains of GH attributes into representational hierarchies; e.g., Enumerated-Type, Numeric-Type, and String-Type.

Every slot in an OKBC ontology has a facet specifying its value type. In the GH ontology, the value type of each template slot derived from an organic attribute is "Instance"; the instance must be an instance of Type. That two slots that model the same concept can be specified by having their value type be restricted to the same subclass of Type. For example, class EquipmentType has template slots loadedWeightQuantity and unloadedWeightQuantity, both of which model weight in kilograms. Consequently, both these slots are restricted to instances of class Weight-Quantity as their value type. Figure 7 depicts these relationships. It shows part of the class hierarchy of the GH ontology; thin lines show subclass relationships. Two templates slots of class EquipmentType are shown, and, as indicated by the arrows, these slots' values are both instances of class Weight-Quantity. Class Weight-Quantity has own slots that specify significant digits, precision, and minimum value for any instance.

Class Type has a template slot type-instance-value. This slot is used to store the actual value of an instance of the type. In other words, when an instance of EquipmentType is created in a knowledge base, the instance has own slots loadedWeightQuantity and unloadedWeightQuantity. The value of these slots is an instance of Weight-Quantity, a subclass of Type. Classes inherit template slots from super classes, and template slots of classes become own slots of instances, so each instance of Weight-Quantity has an own slot

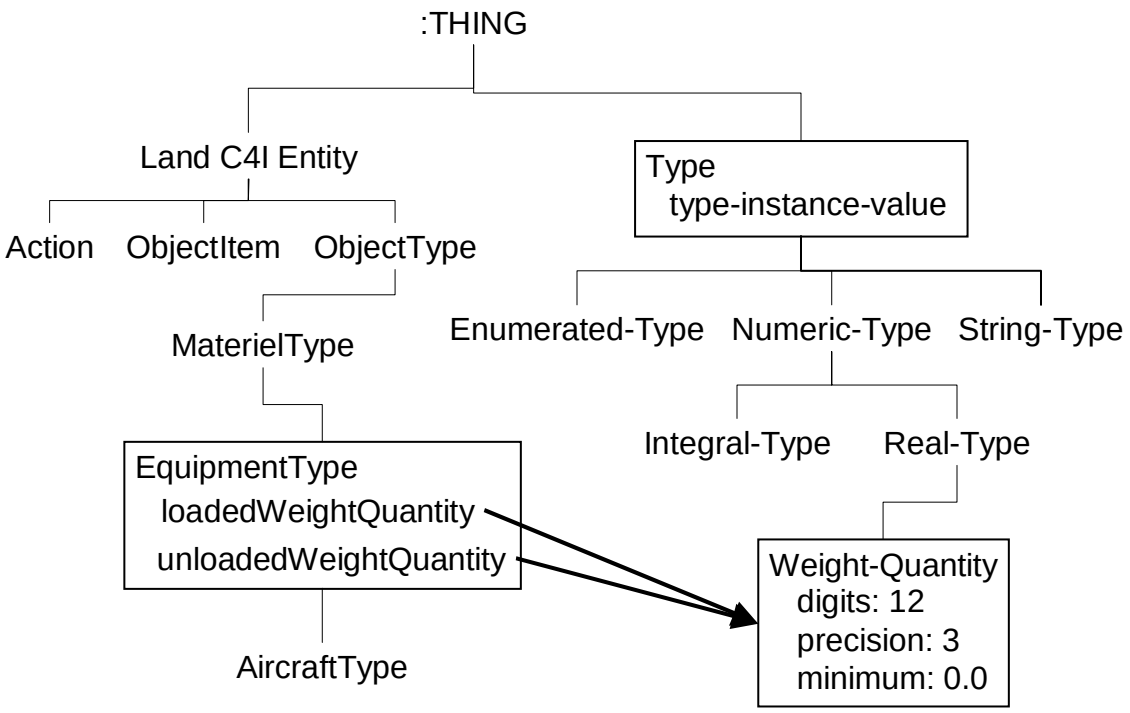

Figure 7. An Illustration of the Type Hierarchy

type-instance-value. This own slot receives the actual value, i.e., a number denoting a weight.

3.2.5 Modeling Measurable Things

The properties specified so far for Weight-Quantity only constrain its representation. Aside from its name, nothing about Weight-Quantity suggests that it models weight, nor is there anything in the ontology to indicate what weight is. Inferencing requires that this information be present.

GH attributes that model weight are modeling a measurable property of a physical entity. A study of the GH reveals many attributes that model measurable properties. Not all attributes that denote a measurement model physical properties. For example, attribute action-task-status-completion-fraction models the degree to which an action is completed as a floating-point value between 0 and 1. Also, some attributes that denote a measurement are represented non-numerically in the GH with a fixed set of codes. An example is action-task-priority-code, which denotes "the rank of a specific ACTION-TASK in the view of the planning organization." Its values are the codes 1, 2, 3, 4, and 5 – which, of course, could have been represented equally well as integers. A better example is reporting-data-credibility-code, whose valid codes are A, B, C, D, E, and F. These values measure level of credibility of reporting data.

An ontology based on the GH should exploit this concept of measurability. IDA therefore added to the GH ontology a class named Measurable-Thing. An instance of Measurable-Thing denotes some measurable characteristic. An application can search the ontology to find things the GH measures. Any two instances of Measurable-Thing are distinct and

should be considered different. The GH ontology contains instances that denote the concepts of length, width, and area.

Of course, measurable properties have relationships. Length times width equals area. Any instance of EquipmentType has length and width slots; therefore, it should be possible to determine whether that EquipmentType can fit in an area specified by some other GH slot. To support this inference requires knowledge of the relationship between length, width, and area. IDA therefore added to the ontology a Measurable-Thing-Mapping class. An instance of this class specifies a relationship:

(Measurable-Thing-Mapping {Measurable-Thing ...} expression Measurable-Thing)

In other words, there exists a mapping, defined by some expression, from a set of measurable things to a measurable thing.

The expression must be a numerical formula that can accommodate expressions such as length × width = area. The GH ontology supports this through a class hierarchy shown in Figure 8. Class Numeric-Expression, the root of expressions, is used throughout the ontology to model numeric values; specifically, the value of slot type-instance-value is an instance of Numeric-Expression for any subclass of Numeric-Type. An instance of Numeric-Expression is an algebraic formula, a degenerate case of which is a simple Integer-Literal or Real-Literal. A more interesting case is the formula to multiply length times width, which is expressed as an instance of class Product. This class has a template slot operands, the value type of which is a set of Numeric-Expression instances. The operands of the expression are length and width, which are expressed as instances of Measurable-Thing. The expression that multiplies length and width is therefore something like the following:

(Product {(Variable (Measurable-Thing length)) (Variable (Measurable-Thing width))})

Drawing inferences about related physical properties requires knowledge of their units of measurement. The GH ontology has a Unit-of-Measurement class with template slots that model the unit's name (e.g., meter), common symbols ("m"), and concept modeled (distance), plus special template slots for such sometimes-necessary properties as minimum and maximum values. The GH ontology also has a class Unit-of-Measurement-Mapping that, analogous to Measurable-Thing-Mapping, can be used to specify interrelationships among units. The class is the root of a hierarchy that reflects the many different kinds of mappings that exist between units (see Figure 9). The three classes of mappings currently recognized are multiplicative (e.g., kilogram × 1000 = gram), additive (ºKelvin + 273 = ºCelsius), and complex ($(°\text{Fahrenheit} - 32) \times 5/9 = °\text{Celsius}$).

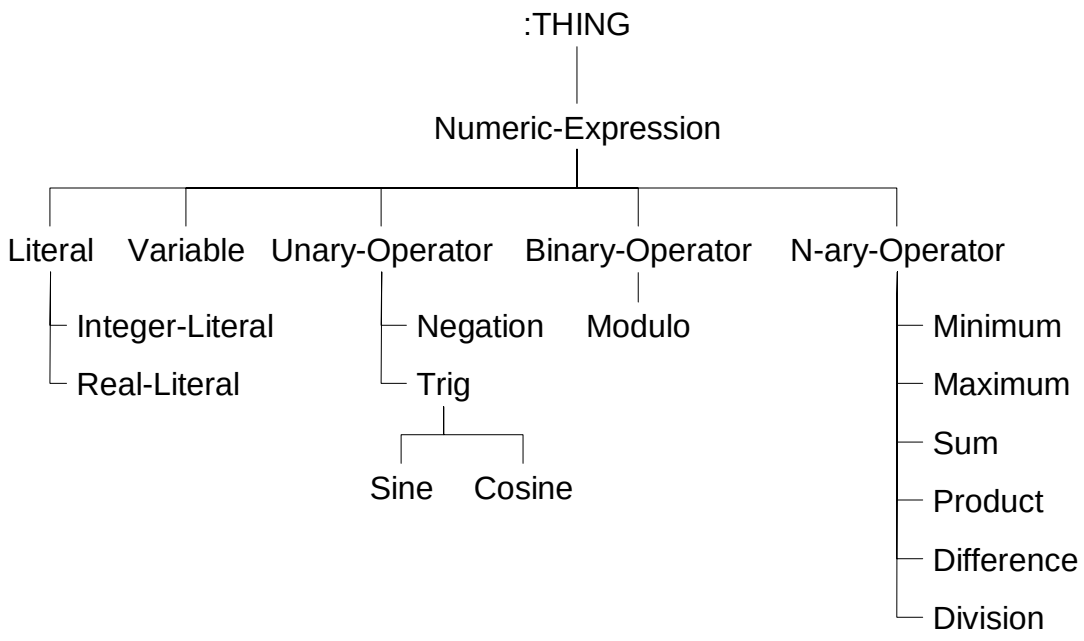

Figure 8. Numeric Expression Class Hierarchy

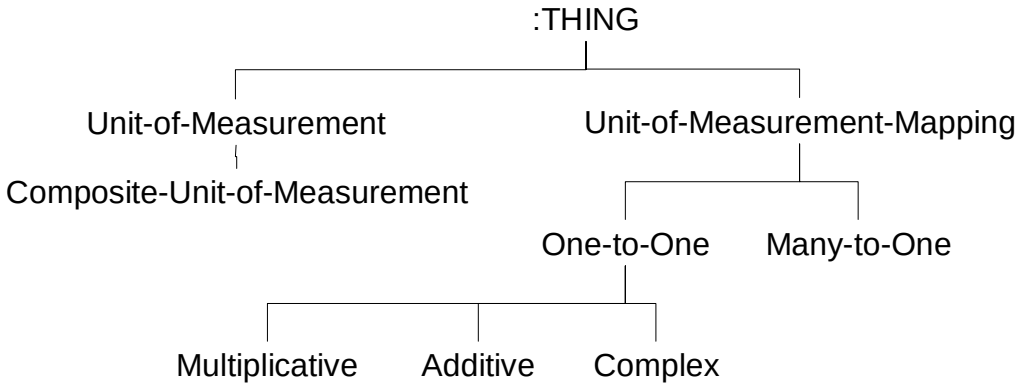

Figure 9. Unit of Measurement Mapping Hierarchy

Some units of measurement are composites of other units. Kilometers per hour, for instance, is a composite of kilometers and hours. Such units are modeled as instances of class Composite-Unit-of-Measurement. A template slot of Composite-Unit-of-Measurement specifies the composite units. An instance of the Many-to-One class specifies a mapping between a set of units of measurement and a single composite unit of measurement. The ontology has an instance of Composite-Unit-of-Measurement that denotes speed in kilometers per hour. An instance of Many-to-One specifies the knowledge needed to combine a distance in kilometers and duration in hours to obtain speed in kilometers per hour. Other instances of Unit-of-Measurement-Mapping specify how to convert distances to kilometers and durations to hours. The GH allows capabilities to be expressed in many different metric and English units, but the GH ontology contains the information necessary to convert between them.

3.2.6 Modeling Order

Some of the GH data elements expressed as codes have an implied ordering. Examples of such elements are:

- The action-task-priority-code attribute, whose legal values are codes 1, 2, 3, 4, and 5. A value of 1 denotes an ACTION-TASK of higher priority than 2.
- The ammunition-type-calibre-code attribute, whose legal values include 0338, 045, and 50.
- The network-security-classification-code attribute, whose legal values are CTS (COSMIC top secret) NC (NATO confidential), NR (NATO restricted), NS (NATO secret), and NU (NATO unclassified).
- The unit-type-size-code attribute, whose legal values include ARMY, FLEET, FLIGHT, and SQUAD.

Each of these domains introduces an ontological challenge. The ordering in action-task-priority-code is intuitively understood by humans, but opaque to an application not specifically programmed for it. The GH represents code values as strings, not numbers, and there is no good reason to believe an arbitrary set of strings has an inherent order, even though an ASCII sorting the values for action-task-priority-code happens to yield the correct ordering. The values for ammunition-type-calibre-code also sort in size order from smallest to largest, a result of careful naming on the part of GH's designers. Note, however, that the smallest value of ammunition-type-calibre-code represents the minimum ammunition size, whereas the smallest value of action-task-priority-code represents the maximum priority. The first element of the natural ordering of a GH coded domain is not necessarily the least.

A lexicographic sort of a GH coded domain does not always yield a meaningful ordering. The values for network-security-classification-code do not adhere to the usual spectrum of security classifications.

Furthermore, some coded domains have a partial, not a total, order. An ARMY is larger than a SQUAD but has no relationship to a FLEET or a FLIGHT.

The C2 ontology developed by IDA addresses these concerns by introducing an OrderSpecification class into the ontology. OrderSpecification has three template slots. Two model instances of a class. The third describes an ordering relationship between them. An instance may be denoted as less than, less than or equal to, or equal to another instance.

3.3 Creating a C2 Ontology

The GH models individual entities, such as actions and types of materiel, relevant to C2. The GH ontology IDA has created is an adaptation of the GH5. It is still not a complete C2 ontology, because it does not explicitly model all command and control concepts. These concepts are based on the entities in the GH5, but are at a higher level of abstraction. An example would be a threat. The concept of threat is central to C2; a war fighter

must understand threats he faces to determine his course of action. A C2 ontology should explicitly include the concept of threat.

The GH does not directly model threats. It models the entities that can pose threats, i.e., battlefield objects. It also models information about battlefield objects that can help determine if a given object poses a threat. For instance, the threat posed by a unit could be based on the unit's last known location, direction of motion, and types of weapons held. This is only one definition of threat, of course, and may not be proper for a given situation, but the point is that the GH possesses this information and can help war fighters in threat recognition, evaluation, and response. (More precisely, our analyses support the conclusion that the GH currently possesses enough information to help war fighters recognize, evaluate, and respond to a wide range of threats. If experience with the GH ontology reveals shortcomings, IDA will propose extensions to the GH.) The IDA team, therefore, decided to define C2 ontology in terms of the GH ontology. However, C2 ontology likely will contain concepts that are not necessarily modeled in the GH.

One of the advantages of basing the C2 ontology on the concepts of the GH is that C2 concepts can be defined in terms of existing GH classes, slots, and facets. There is no need to define an *ad hoc* set of concepts specific to the C2 ontology. Consider modeling of the concept of "threat" in the C2 ontology. The C2 ontology contains a Threat class. The template slots of this class specify the battlefield object that is threatened, the battlefield object that poses a threat, the time at which the threat exists, and the nature of the threat. (These properties are illustrative and have not been subject to expert review; they are not claimed to be complete or fully realistic.) A threat is defined in terms of such GH ontology classes as:

- ObjectItem, to specify the battlefield objects that pose and are subject to threats.
- ObjectItemStatus, to determine if a battlefield object is hostile.
- ReportingData, to assess intelligence on hostile battlefield objects.
- ObjectItemLocation and Location, to determine if a battlefield object is moving in a direction that would pose a threat to another battlefield object.

Although decision support applications will likely use the more abstract classes of the C2 ontology rather than the classes of the GH ontology directly, the fact that most, if not all, of the higher-level concepts will be built using GH ontology classes will ensure data interoperability with the underlying GH-conformant data stores.

In other words, the C2 ontology will be a formalization of doctrine couched in terms of GH concepts and aimed at directly supporting decision makers. Moreover, it will centralize the implementation of doctrine, eliminating the need for individual applications to reimplement portions of doctrine over and over.

Figure 10 illustrates applications making use of the C2 ontology, or rather a knowledge base whose instances are instances of classes from the C2 ontology. That the GH knowledge base is within the C2 knowledge base reflects how C2 concepts are defined in terms of GH concepts; it also indicates that designers are intended to think in terms of C2 rather

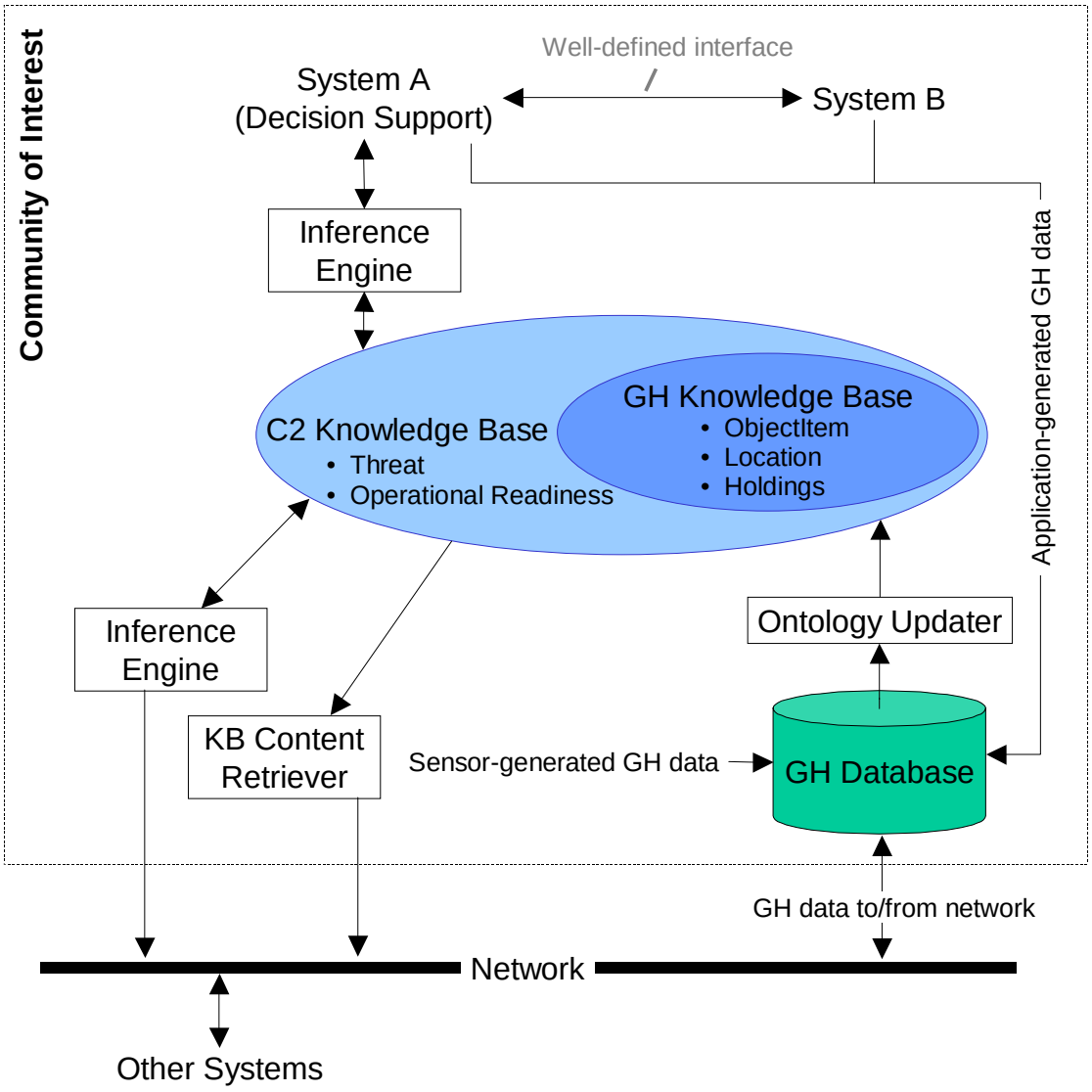

Figure 10. Scenario for C2 Ontology Use

than GH KB classes. The knowledge base is populated from two sources. The primary source is a GH database. (Arguably, in an environment where a knowledge base is being used, the knowledge base replaces the database. This may prove true, but considerable research has gone into optimizing database storage efficiency and query performance; no such body of work exists for knowledge bases. IDA's working assumption is that databases will continue to be necessary.) A tool in a community of interest, here named KB Updater, is responsible for populating the knowledge base and keeping it up to date with respect to database updates, which themselves can come from three sources: sensors, local applications, or a network.

Applications do not access the knowledge base directly, but rather through a fixed set of tools that provide an abstraction of the information. One of these tools is an inference engine. The inference engine attempts, at the request of other tools and applications, to draw conclusions about the data in the knowledge base. These conclusions are stated in terms

of ontological concepts; when the inference engine is able to draw new conclusions, it places them in the knowledge base. Consequently the inference engine is the second way in which the knowledge base may be modified. For example, the decision support application in Figure 10 might periodically request the inference engine to conduct a threat analysis. The inference engine could reach conclusions such as "Unit A threatens unit B", which it would state as an instance of class Threat. It would place this instance in the knowledge base, reflecting that the threat was perceived at that time.

Systems A and B do not directly update the knowledge base either, but rather update the database. The KB Updater tool has the responsibility to update the knowledge base. This is one possible scenario chosen to show the constituents of a CoI based on the GH, and should not be construed as the only possible system architecture.

The other systems outside the CoI have access to the knowledge base in much the same way as systems within it. Their primary access should be through an inference engine. Because a CoI cannot hope to predict all the ways in which other organizations might want to use its data, a separate KB Content Retriever tool exists that allows direct access to the knowledge base. A vigilant CoI would monitor uses of this tool and attempt to infer how other CoIs are using the knowledge base; the CoI would evolve the ontology based on this information.

3.4 Implementing the C2 Ontology

An ontology used in a net-centric environment must be executable, i.e., its contents must be interpretable by and accessible to computer applications. Fortunately, several programs exist that implement the OKBC protocol.

Owing to its project's goals, IDA was able to establish the following requirements an implementation of OKBC had to satisfy:

- *The implementation should be freeware.* This is an exploratory project, and IDA hopes to distribute its results freely. Software licenses complicate distribution.
- *The implementation should run on multiple, heterogeneous platforms.* A demonstration of a net-centric environment is not very convincing if it's run on a single computer. Even a limitation to Microsoft Windows platforms seems constraining given certain movements in the DoD towards Linux (for example, see [Erwin 2003]). The more diverse the environment, the better.
- *The implementation should be useful on knowledge bases of realistic sizes.* One purpose of this project is to determine whether today's technology is sufficiently powerful to make use of ontologies in a net-centric environment. Some inference algorithms are notoriously slow, especially on large data sets. IDA explored what inference capabilities are practical and (ideally) to predict when the DoD can expect to field more powerful capabilities.
- *The implementation needs a well defined application program interface (API).* Actually, any implementation of OKBC will provide an API. The use of OKBC was not a given when the project began, however, and in any case every ontology

tool IDA explored used frames in some way, even if not in complete conformance with OKBC. Some of these tools are simply ontology editors. Though they are useful for ontology design and presentation, they do not support inferencing or even the construction of inference engines.

- *Semi-automated translation of the GH into an ontology should be possible.* Ontology tools usually offer GUI ontology and knowledge base editors, but the GH5 is large enough to make manual entry of an ontology a laborious task. IDA wanted to translate the GH5 into the language in which a tool represents an ontology using as little effort as possible.

The World Wide Web has spurred considerable interest in tools and technologies for implementing ontologies. Many of these tools are web-oriented. Their inputs and outputs are received and sent using the Hypertext Transfer Protocol (HTTP) (e.g., SESAME [Karyakof 2002]) or at least using an explicit client/server model (e.g., Ontolingua [Farquhar 1997], Comtec [Suguri]). The deployment of ontologies in a net-centric environment ultimately relies on client/server connections. It is however irrelevant in the early stages of a project such as this one, where the emphasis is on the design and development of the ontology and associated inference capabilities.

IDA settled on Protégé-2000 [Noy 2000] as the tool to model the ontology.[6] Protégé-2000 is an open-source ontology and knowledge base editor that implements OKBC. It is under development by Stanford Medical Informatics at the Stanford University of Medicine, dating back (in very early form) to 1987. Its funding organizations include the Defense Advanced Research Projects Agency, the National Cancer Institute, the National Institute of Standards and Technology, the National Library of Medicine, and the National Science Foundation. A page on Protégé-2000's web site[7] lists (as of this writing) sixty ongoing projects around the world making use of Protégé-2000, in areas ranging from finance to medicine.

Protégé-2000 is implemented in the Java programming language. Java's availability, portability, ease of deployment in a net-centric environment, and wide support base enhance Protégé-2000's value for this project. (Many ontology and inference tools are written in Lisp, reflecting their origins in artificial intelligence research departments.) Java programs are not known for speedy execution, but the flexibility of Java makes it advantageous.

Protégé-2000's creators have studied its performance degradation as the number of frames increases. They aver that its performance does not degrade for knowledge bases with up to 1,000,000 frames. IDA's studies with a sample GH data set indicate that a single, isolated knowledge base requires on the order of 20,000 frames (see Section 3.5). It is possible that a knowledge base with access to the GIG might contain orders of magnitude more data, but predicting GIG size is a risky venture. It is an issue that merits both further study and close watch. In any case, Protégé-2000's creators believe they can improve its performance on large knowledge bases.

---

6 The Protégé-2000 web site is http:// protege.stanford.edu/.

7 http://protege.stanford.edu/projects.html.

One problem with Protégé-2000 is that it is primarily an ontology and knowledge base editor and only secondarily an inferencing tool. It can be used to create ontologies; more precisely, it can be used to define concepts and interrelationships to the extent permitted by a frame-based model. It can also be used to create instances of classes in an ontology, that is, to create a knowledge base. However, it is deliberately limited in certain important respects. Most significantly, its constraint checking capabilities are minimal and not enforced. A Protégé-2000 ontology can specify simple constraints on slot values: type, range (for numeric types), and cardinality. It cannot specify such useful constraints as the maximum length or format of a string. Protégé-2000 lets ontology designers add constraints like these, but the constraints are not implicitly enforced by Protégé-2000. Instead, constraints must be enforced by applications, or through user actions performed as part of the methodology used to develop and maintain a knowledge base (this is why IDA developed classes for types and units of measurement, rather than making them slot facets).

Furthermore, Protégé-2000 flags the entry of erroneous slot values but does not prohibit them. A slot may be marked as required, but Protégé-2000 will save a knowledge base in which that slot has no value. The API does not even indicate whether a value is invalid for a slot when the slot's value is set, though methods exist to test that condition. In other words, Protégé-2000 makes the user (or application) responsible for the consistency and correctness of a knowledge base.

Protégé-2000 offers a more general constraint facility in the Protégé Axiom Language (PAL). PAL is intended for model checking purposes. It is a variant of the Knowledge Interchange Format (KIF) [Genesereth 1992], a first-order predicate logic language. PAL may be used both to enforce constraints beyond those specifiable as facets, and to perform simple queries on a knowledge base. It is explicitly *not* intended as a general-purpose predicate logic language (as is KIF). Again, Protégé-2000's designers are requiring others to supply inference capabilities.

Fortunately, Protégé-2000 was designed to be extensible. It supports "plug-ins", which is Java packages that extend Protégé-2000's functionality in some integrated way – that is, the extra functionality is available to the rest of Protégé-2000. One useful plug-in is Algernon, an inferencing capability adapted from an earlier Lisp-based system [Kuipers 1994] to work with Protégé-2000 knowledge bases. Algernon is a very general-purpose tool, implementing forward and backward chaining, both staples of rule-based inferencing. Moreover, its Protégé-2000 plug-in can access native and user-defined Java functions. This provides the often-missing capability of procedural abstraction, allowing Algernon rules to be stated concisely in comparison to PAL.

In short, Protégé-2000 is not ideal but possesses many desirable characteristics and can be extended. One goal of knowledge base design is to encapsulate business rules. Protégé-2000 offers that capability.

Figure 11 shows the Protégé-2000 interface to the GH ontology. The application displays the class hierarchy in a window on the left. This window contains the classes discussed in Section 3.2, which form a tree rooted at: THING. The window on the right side shows the

template slots of the currently selected class (here, Action). The column headings in the lower right table – Name, Type, etc. – are facets associated with a template slot. The values in the table are the values of the facets. In other words, Action has a template slot whose Name facet has the value name. This slot's value is an instance of class Action-Name, which though not shown in Figure 11 happens to be a descendent of class Type.

Protégé-2000 has some interesting features that merit mention with regard to how they influenced the GH ontology's design. The following subsections cover them.

3.4.1 Abstract Classes

A Protégé-2000 class may be either *abstract* or *concrete*. An abstract class can have no instances. Instead, subclass of an abstract class can have instances. This modeling technique, common in object-oriented languages, is used to designate high-level concepts that must be further specified before it makes sense to create instances.

A Protégé-2000 class that is abstract has a green "A" to its right. Figure 11 shows that: THING, SYSTEM-CLASS, and Numeric-Expression (among others) are abstract. Protégé-2000's designers have decided that a: THING is too vague to instantiate. Similarly, IDA has decided that a Numeric-Expression is too vague to instantiate. Some of its subclasses from Figure 8 can be instantiated, however – generally the leaves of the class hierarchy tree. The implication is that one must provide a certain amount of detail before a numeric expression can be stated unambiguously.

Despite the ontology's rearrangement of the GH into a class hierarchy, subclasses of Land C4I Entity use abstract classes sparingly. In fact there is only one: ObjectTypeEstablishment, the super class of MaterielTypeEstablishment and OrganisationTypeEstablishment. The GH does not contain an OBJECT-TYPE-ESTABLISHMENT entity, but the IDA team, noting similarities between entities OBJECT-TYPE-ESTABLISHMENT and MATERIEL-TYPE-ESTABLISHMENT, opted to take advantage of those similarities in the GH ontology. Because no GH data set will ever contain an instance of OBJECT-TYPE-ESTABLISHMENT, class ObjectTypeEstablishment can be abstract.

Arguably, other classes in the GH ontology should be abstract. Consider class Location (see Figure 12). By itself, an instance of Location does not specify a location. It only specifies the subclass type (the categoryCode slot) and a relationship to an ObjectItemLocation instance. Moreover, the GH entity from which it is derived – LOCATION – states that the location-category-code attribute (from whence categoryCode derives) is a complete discriminator for subtypes. In theory, then, all possible subclass of Location are known, so any instance can be assigned appropriately.

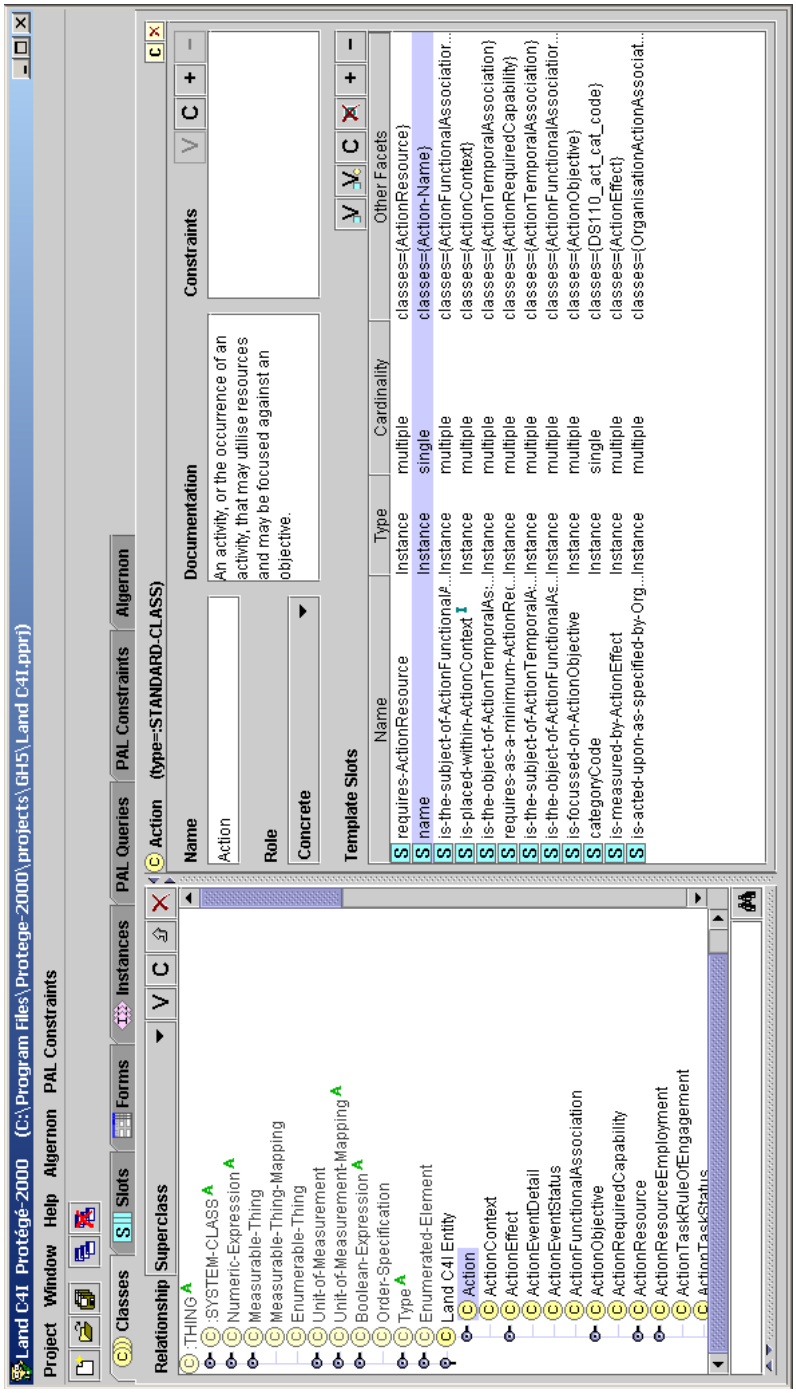

Figure 11. Protégé-2000 User Interface to the GH Ontology

Unfortunately, GH doesn't require that an instance of LOCATION have a corresponding instance of a subtype entity. A GH data set containing an instance of LOCATION but no matching instance of a subtype is still valid. Translating such a data set into the knowledge base is problematic. The instance can be assigned to a subclass of Location – location-category-code is a required attribute, so a subtype can be inferred – but if Location is abstract, then its subclasses Geometric-Volume, Point, and Surface should be abstract too, as

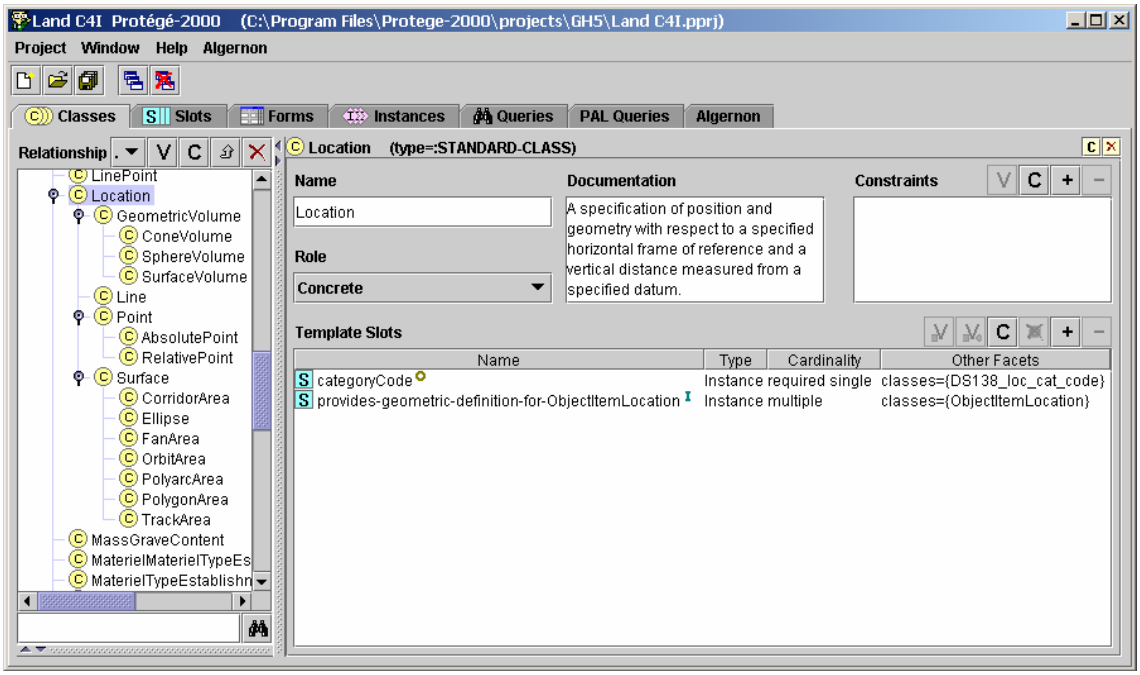

Figure 12. Location Hierarchy in Protégé-2000

none of them convey enough information to specify a location. In other words, an instance of LOCATION doesn't have enough information to identify a concrete subclass of Location.

Given this situation, if GH classes such as Location, ObjectItem, ObjectType, etc., are abstract, there will exist GH data sets that cannot be translated into knowledge bases. IDA has opted for a GH ontology that can model all GH data sets. The trade-off is that performing an inference requires making additional tests in order to ensure that an instance is of a class specific enough to be useful.

3.4.2 Inverse Slots

Section 3.2.3 described how GH relationships are represented in the ontology: as multiple-valued slots whose value types are an instance of a particular class. Figure 12 shows how the relationship between LOCATION and OBJECT-ITEM-LOCATION was translated into template slot provides-geometric-definition-for-ObjectItemLocation.

This technique lets Protégé-2000 represent 1: N relationships. It also lets Protégé-2000 represent M: N relationships – but only if no associative entity is involved. Unfortunately, every M: N relationship in the GH has an associative entity. When one is using a query language such as SQL to access a GH data set, this isn't a problem. An inner join gives a succinct and efficient statement of the set of all locations associated with a battlefield object.

Drawing inferences from a Protégé-2000 knowledge base is another matter, because PAL isn't nearly as compact, expressive, or efficient as SQL. If finding the locations of a battlefield object required finding all associated ObjectItemLocation instances, then searching all Location instances for those whose provides-geometric-definition-for-ObjectItemLocation slot

matches one of the ObjectItemLocation instances, implementations of Protégé-2000 inference rules would be large and unwieldy.

Protégé-2000 provides a technology that addresses this issue. Two slots may be declared each others' *inverse*. The interesting property of inverses is that creating a value for one slot of an inverse set automatically creates an appropriate value for the other slot. In the GH ontology, ObjectItemLocation has a slot obj_item_loc-is-associated-with-Location that is the inverse of provides-geometric-definition-for-ObjectItemLocation. Each association of a Location with an ObjectItemLocation automatically fills in both slot values. One can easily find the Location associated with an ObjectItemLocation. This solves the query problem.

IDA therefore created the GH ontology such that every template slot modeling a 1: N relationship has an inverse slot. The name of the inverse slot comes from the child-to-parent phrase of the GH. The child's name is prefixed to the slot's name to avoid naming conflicts, as many GH entities are the parent of relationships with the same child-to-parent name (e.g., many entities are the child of an is-referenced-to REPORTING-DATA relationship).

3.4.3 Metaclasses

Protégé-2000 fully implements everything required by the OKBC standard. However, OKBC allows features that Protégé-2000's developers chose not to provide. For example:

- OKBC allows a frame to be an instance of multiple classes. Protégé-2000, however, restricts a frame to an instance of a single class.
- In OKBC, an own slot can be attached directly to any frame. In Protégé-2000, an own slot attached to a frame is always derived from some template slot.

These decisions have consequences for ontology design in Protégé-2000. One consequence is that characteristics of a class beyond those allowed by the default facets must be specified using a *metaclass.* A metaclass is a class that happens to be a subtype of class: STANDARD-CLASS. An ontology developer may specify the metaclass of a class. By default, this metaclass is: STANDARD-CLASS, which means that its properties happen to be exactly the template slots of: STANDARD-CLASS. These slots include: NAME, (a string): ROLE (abstract or concrete), and: DIRECT-TEMPLATE-SLOTS (the template slots of the class).

A subclass of: STANDARD-CLASS inherits the slots, and may add its own. When the ontology developer designates a class's metaclass as something other than: STANDARD-CLASS, he allows additional properties to be specified for the class. These properties apply to all instances of the class.

Figure 13 shows some of the metaclasses used in the GH ontology. Metaclasses are used extensively to assign meaning to subclasses of Type. Figure 13 highlights Textual-Class, which is used for types that store text. The Textual-Class metaclass has two additional template slots (in blue) that allow for specification of whether a slot's value is of fixed or varying length, and of the maximum length of a string value for the slot. This extra information allows for capture of the information about text-valued attributes in the GH.

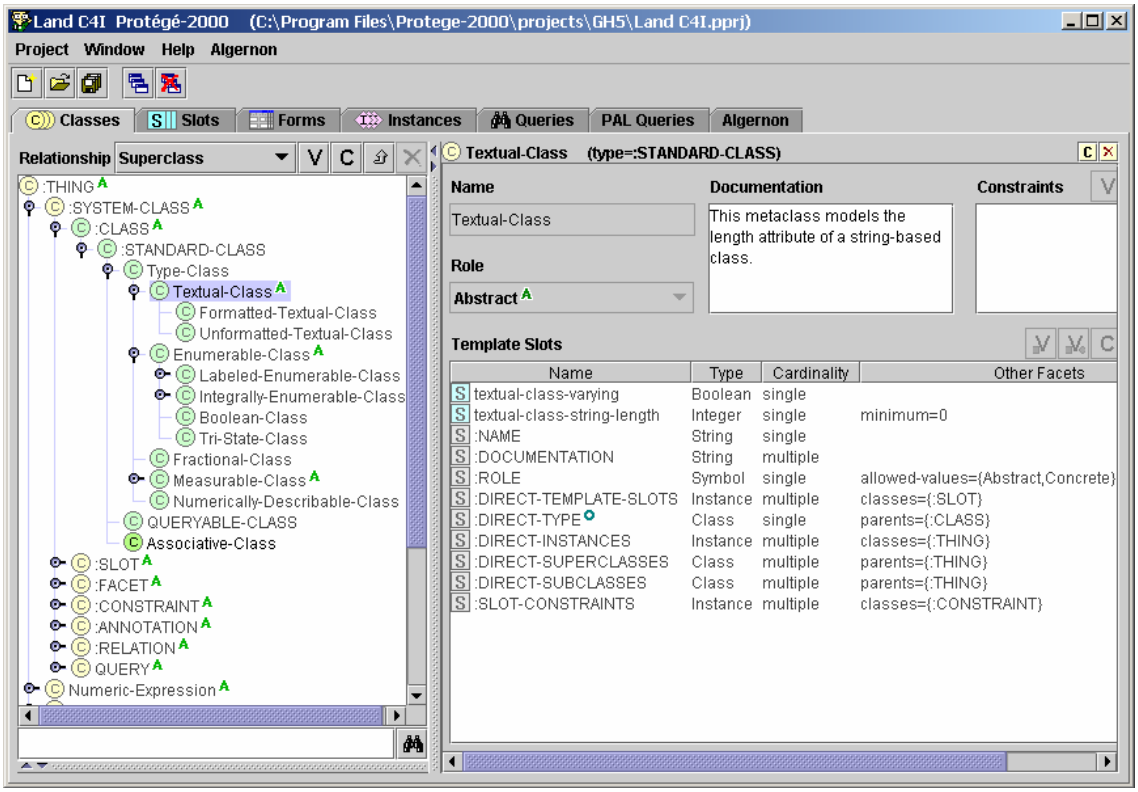

Figure 13. Metaclasses in the GH Ontology

3.5 Status of the Implementation

IDA's implementation of the GH ontology models all elements of version 5.2. Version 6 of the GH was released in November 2003; IDA will study the new version to determine when the changes should be transferred to the ontology.

The C2 ontology is still in a rudimentary form. IDA's goal is to select a few key C2 concepts and implement them as carefully as possible to test the realism that an ontology can offer but creating a complete C2 ontology is outside the scope of the current task. To date the focus has been solely on threats. In other words, the C2 ontology consists of the GH ontology plus a single extra class, Threat, with slots that denote the threatener, threatenee, reporting organization, and perceived degree of threat. Currently only instances of Organisation can be a threat or be threatened.

Some metrics may help give a feel for the size and complexity of the C2 ontology. It has:

- 9,482 frames
- 898 classes
- 706 slots
- 7,878 instances (most of which are values from the GH coded domains)

These numbers do not include the system-defined frames Protégé-2000 includes in every ontology.

In other words, the number of frames is on the order of 10^4, 2 orders of magnitude less than that which would cause degradations in Protégé-2000's performance.

IDA has obtained sample GH data from a NATO exercise. It can be modeled using approximately 20,000 frames. This suggests that real data sets are of a size that Protégé-2000 can manage.

4. Using the Ontology for Decision Support

This report documents how to use the C2 and GH ontologies in a net-centric environment. Net-centric warfare gives decision makers access to the wealth of information available on the GIG. Of course, decision makers are not expected to search the GIG themselves; that would be far too time-consuming. Nor are applications supposed to push arbitrary and overwhelming amounts of data onto decision makers.

Instead, DoD envisions that local applications will serve decision makers by continuously looking for, and analyzing, information the applications deem relevant. These applications are known as agents. The objectives of agents are:

- To reduce the need for routine human operator intervention. Currently, human subordinates must be employed to access and search the GIG. This is an inefficient use of resources.
- To identify the best data to pull from the GIG for analysis and inspection. An agent can be responsible for *filtering* data based on selected criteria. In a C2 scenario, criteria would include factors such as current mission, current location, operational readiness, and perceived threats.

As part of the study, the IDA team investigated the components of a canonical architecture for decision support in net-centric warfare. The rest of this section discusses that architecture, and the prototype system IDA has implemented based on the architecture.

4.1 Operational Architecture

At least for the short term the operational architecture of a decision support system in a net-centric environment will not differ greatly from that of current environments. The significant distinction is that the net-centric environment requires fewer operators. Where once humans performed searches for information, agents will search the GIG.

This difference likely leads to consolidation of information within networks. In the current environment, someone is directed to conduct a search; when he is finished, he may report the results of that search directly – i.e., verbally – to a decision maker. By doing so, he is filtering information; therefore, the exact data presented to the decision maker is not recorded. By contrast, in a net-centric environment an agent performs the filtering, and can leave a record of what information was omitted and what was presented.

4.2 System Architecture

The overall system architecture for decision support systems in a net-centric environment will likely contain the elements shown in Figure 14. There will exist a certain number of

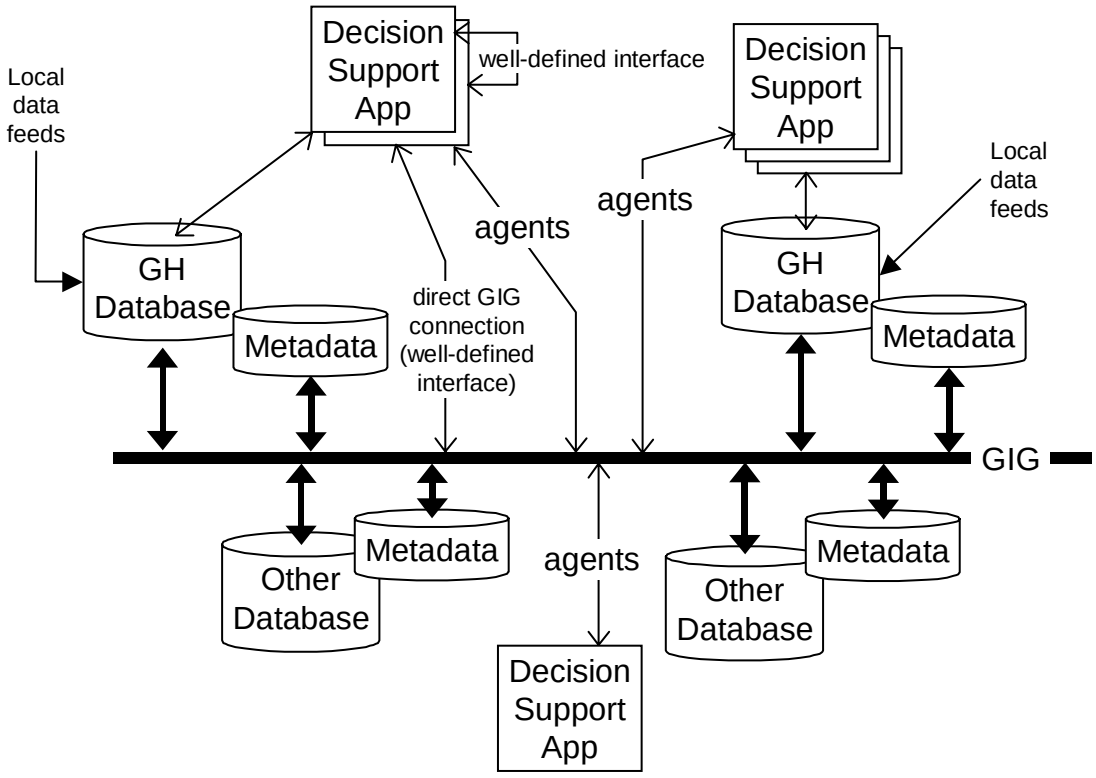

Figure 14. High-Level System Architecture

GH-based databases, each operated independently, but each with metadata. These databases will be fed by local data feeds (sensors, human intelligence), by local applications (decision support systems, combat support systems, etc.), and by updates from the GIG.

Communication involving decision support applications can take several forms. Applications may communicate directly with other applications (decision support or otherwise) over well-defined local interfaces. They may communicate with known applications or database management systems via the GIG; these communications also count as occurring through well-defined interfaces, since the initiating application knows of the queried application in advance, expects it to respond to predefined queries, and anticipates the format of requested data.

In other words, nothing about net-centricity precludes communication via today's technology. Instead, net-centricity provides access to more data – specifically, to any database that contains metadata. This access occurs, as Figure 14 shows, through agents. Each agent is accessing the GIG, looking for information relevant to a decision maker's current situation (which, presumably, is specified through a decision support application). Standard protocols will exist to help an agent locate and retrieve metadata. Standard languages must also exist that allow an agent to interpret metadata and data. OWL, the Web Ontology Language [OWL 2003], appears to be the emerging standard.

The open question raised by Figure 14 is the nature and architecture of the agents. Collectively, agents serve many different, and sometimes mutually exclusive, roles in a net-centric environment:

- Some agents are periodic, continually searching for and filtering information in the belief that it may be useful. Others are aperiodic, initiated in response to commands from a user or from another agent.
- Some agents pull information, on request from a user. Others push information, believing that the information is vitally important and must immediately be brought to a user's attention.
- Some agents search the entire GIG. Others confine themselves to a community of interest.
- Some agents accept metadata to interpret their inputs or produce metadata so other programs can interpret their outputs. Others communicate through well defined interfaces.

In short, "agent" has no single accepted definition. It is understood to be an autonomous application, but beyond that vague description, which includes many applications that are not considered agents, there is not much agreement on the properties an agent possesses.

In the context of decision support systems in a net-centric environment, IDA has found that several types of agents can be useful. The following examples illustrate how each of the above types of agents might be useful in decision support applications:

- Periodic agents can search the GIG for mission-relevant information. For example, decision making is aided by knowledge of supply availability. Given knowledge of an organization and its mission, an agent could periodically search for supply locations near an organization. These searches would be based on organization location, current status, and projected needs.
- Aperiodic agents can be launched in response to specific situations, such as the detection of a threat. A threat response agent would use doctrine-derived rules to formulate acceptable courses of action.
- An agent that gathers information to aid in decision support is pulling data.
- An agent that informs a decision maker of a situation that needs immediate attention is pushing data.

The canonical system architecture IDA is developing recognizes that a community of interest is likely to need different types of agents, all operating concurrently.

The architecture developed to date is in response to the concepts in the C2 ontology: threats and operational readiness. It is shown in Figure 15. This is the system architecture a single unit might deploy. Agent-based decision support in this architecture comprises the following agents:

- A *knowledge base updater agent.* This is a periodic agent whose role is to monitor a GH database for changes, and to incorporate those changes into the GH ontology.

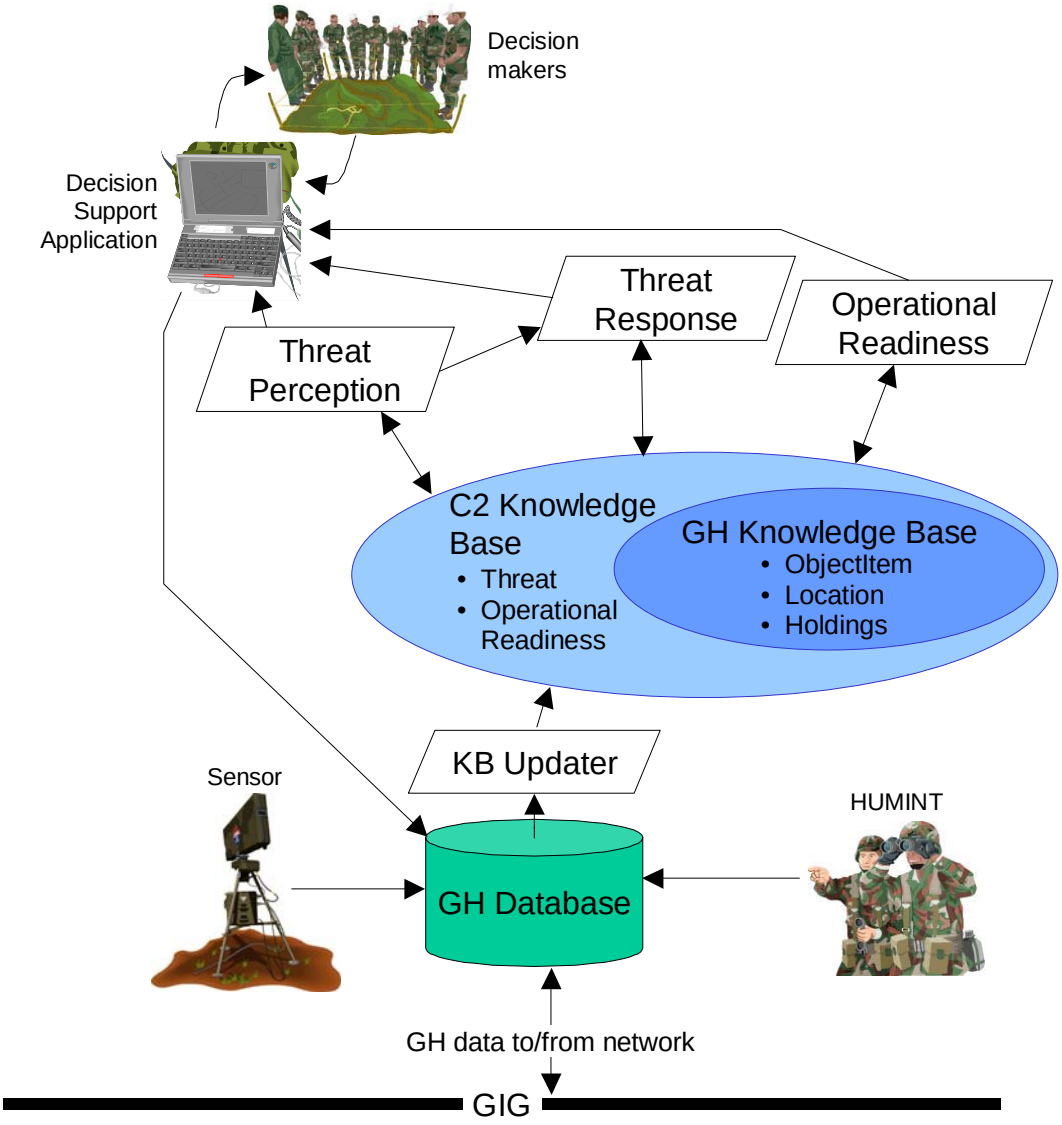

Figure 15. Agent System Architecture

- A *threat perception* agent. This is a periodic agent whose role is to monitor the knowledge base for battlefield objects that, according to doctrinal rules, appear to pose a threat to the organization deploying the agent.
- A *threat response* agent. This is an aperiodic agent, initiated by the threat perception agent in response to a detected threat. Its role is to establish courses of action in response to the threat.
- An *operational readiness* agent. This is a periodic agent. It monitors the knowledge base and formulates statements of operational readiness of the organization that initiates it.

Figure 15 is a realization of Figure 10. The notion of a general purpose inference engine has been replaced by concrete agents that serve specific roles. (These agents could be implemented using inference engines; whether to do so is a technical decision.)

Unlike Figure 10, Figure 15 shows no connection between agents and the GIG. On the one hand, Figure 15 is a simplification of the actual architecture, because the software tools used to implement the agents (see Section 4.3) effects inter-agent communication, such as metadata searches, using a network. On the other hand, Figure 15 shows a deliberate design decision: the prototype's purpose is to demonstrate that the ontology is a useful encapsulation of C2 concepts. The prototype is not intended to demonstrate metadata search capabilities. A convincing demonstration thereof necessarily requires some independence between the ontology developer and the agent developer. Each should be from a different CoI in order to prove that the ontology describes data sufficiently deeply and precisely.

An agent does not make decisions. It assists decision makers by providing them with information. The arrows in Figure 15 that lead from the agents to the decision support application demonstrate this notion. These arrows indicate information provided. The decision maker will evaluate this information, make a decision, and enter it into the decision support application, which will then record it in the GH database. The KB Updater agent will eventually incorporate the decision directly into the knowledge base.

For example, suppose the information in the knowledge base causes the threat perception agent to infer the existence of a threat. The threat perception agent will notify the decision support application. The decision support application's user interface will signal the threat. Simultaneously, the threat perception agent notifies the threat response agent, which examines the knowledge base to determine appropriate courses of action. After doing so, it prioritizes the courses of action and notifies the decision support application. The decision support application presents the courses of action to the decision makers. Each course of action is accompanied by a rationale that helps the decision makers decide if the threat response agent's prioritization is appropriate. The decision makers reach a decision and enter it in the decision support application, which casts it into GH entities – an ACTION-TASK, say – and enters it into the database. Later, the KB Updater incorporates a new ActionTask instance into the knowledge base. Figure 16 presents this information precisely in the form of a Unified Modeling Language (UML) sequence diagram [Booch 1996].

This architecture is by no means a mandate. Agent-based systems are full of design tradeoffs. Consider the decision to make the KB Updater agent periodic. One can argue that updating the knowledge base the instant new data is received yields the most up-to-date decision making capabilities. The ability to do so depends in part on whether the DBMS supports change detection and some mechanism to influence external processes (i.e., something to modify the knowledge base); in other words, periodicity is determined partly by technical factors.

4.3 A Prototype Implementation

IDA has implemented a prototype system that tests the ideas and architecture in this paper. The system comprises the applications and data shown in Figure 15: the decision support system, the agents, and the DBMS, as well as GH data sets managed by a DBMS and knowledge bases managed by Protégé-2000.

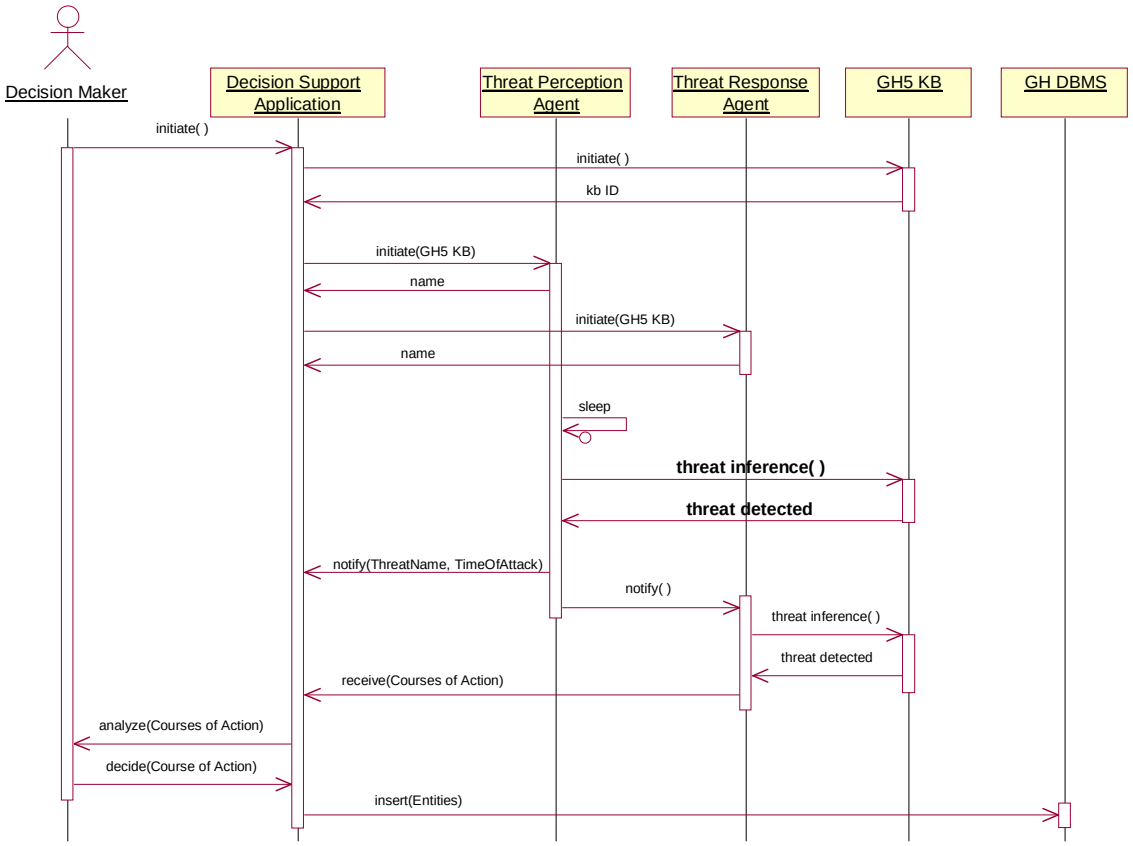

Figure 16. Agent System Sequence Diagram

The two primary objectives in building the prototype were:

- To determine whether an ontology derived from the GH is useful for drawing inferences about command and control-related information. The ontology should encapsulate at least as much data syntax and semantics as the GH does. The prototype must be able to use this metadata to access and manipulate a GH data set.
- To explore the types of metadata that should be created for a C2 CoI. Ontology development is effort intensive. Empirical evidence on what metadata is useful will help ontology developers devote resources to fruitful areas.

4.3.1 Agent Technologies

Agents interoperate. They are expected to exchange information. Exchanging information requires an infrastructure. A precondition to building the prototype was to search for existing agent infrastructures, and to choose one that would minimize the implementation effort for the prototype while providing a robust and realistic platform.

IDA used an implementation of the standards defined by the Foundation for Intelligent Physical Agents (FIPA) [FIPA 2001].[8] FIPA, originally conceived for robotics, has evolved into a comprehensive set of standards for agent-based systems. The standards cover such areas as:

[8] FIPA's web site is http://www.fipa.org/.

- *Abstract architecture:* a general statement of how agents can pass messages to each other.
- *Agent communication*: a standard set of communicative acts agents may perform languages in which they can perform the acts, and protocols for performing the acts. FIPA also defines how to include non-standard acts and languages.
- *Agent management:* How an agent may announce its presence on a network, and how an agent may search for other agents.
- *Agent message transport:* How messages are transported across networks.

FIPA's web site lists ten implementations.[9] Each implements a selected set of FIPA standards. Some are intended as general-purpose agent development platforms. Others have specific goals (e.g., supporting web-based commerce). IDA used the following factors to make a selection:

- *Standards supported.* IDA focused on the implementations that implemented FIPA standards related to ontologies.
- *Support for general-purpose agent development.* Some of the implementations have specific goals (e.g., supporting web-based commerce) that limit their expressiveness.
- *Ability to interface to Protégé-2000's API.* Some of the implementations have their own language in which to create ontologies. By the time IDA had to select an agent infrastructure, much of the C2 ontology was already written. Re-writing the ontology in the native language of an agent infrastructure was a possibility, but none seemed as powerful as Protégé-2000.

  This factor effectively narrowed the search to agent infrastructures implemented in Java.
- *Open source software infrastructure.* Agent technology is relatively new, so documentation is poor and responsive support lacking. In such an environment, open source is the only practical choice.

IDA decided to use FIPA-OS [Poslad 2000].[10] FIPA-OS is probably the most mature implementation of FIPA standards. It was originally developed by the telecommunications company Bell Northern Research, and has gone through ten major releases that have incorporated extensions and bug fixes from individuals and institutions around the world. It has an active user community, an invaluable resource to hasten the learning curve.

4.3.2 Database Technologies

The implementation also required a DBMS to manage a GH data set. IDA opted not to purchase a commercial database server because of the considerable expense. That left three reasonable choices: Microsoft Access, MySQL, and Oracle's Personal DBMS. IDA eliminated Microsoft Access because its limit on the number of integrity constraints for

[9] http://www.fipa.org/resources/livesystems.html.

[10] The home page for FIPA-OS is http://fipa-os.sourceforge.net/.

an individual DB is much lower than the number of integrity constraints in the GH5. With respect to MySQL the IDA team concluded that it was viable even though its suite of database maintenance tools is less mature than Oracle's. IDA therefore implemented the prototype using both version 9i of Personal DBMS and version 4.17 of MySQL.

4.3.3 XML Technologies

Early in the prototype's implementation, it became apparent that the Extensible Markup Language (XML) [W3C 2000], the web's de facto standard for document exchange, would play an important role. XML facilitates the transmission of entities as structured information. To convert an entity – an instance of a Java class, for example – into XML requires a parser. IDA needed an implementation of a parser.

One seldom writes an XML parser, but instead uses an XML parser generator. These tools take as input a specification of the structure of an XML document. These specifications are written in two standard languages. The earlier language, document type definition (DTD) [W3C 2000] is less expressive than its successor, XML Schema (XMLS) [W3C 2001]. However, the XML documents used in the prototype could be adequately expressed as DTDs, and the expressive power of XMLS increases the complexity of the generated parser. IDA decided to express XML document structure as a DTD.

IDA used the Zeus parser generator.[11] Zeus fully supports DTDs defined by the XML 1.0 specification. Parsers produced by Zeus are 100% Java code.

4.3.4 Prototype Components

The prototype simulates a set of mutually friendly organizations wanting to make decisions about courses of action regarding threats from hostile organizations. Each friendly organization has access to a DBMS with a GH data set. Organizations may share the same data set, but no organization has direct access to more than one data set (but inter-agent messages provide indirect access).

Each friendly organization has its own C2 knowledge base. It uses this knowledge base to reason independently of other organizations.

Each friendly organization has a set of sensors. A sensor provides intelligence about the location of hostile organizations within a fixed area.

Each friendly organization runs a decision support application. This application initiates agents and accesses the GH database as shown in Figure 16.

Characteristics of hostile organizations are known to friendly organizations to the extent that they can be described in a GH data set. Currently, the important characteristic of a hostile organization is its location over time. A friendly organization calculates a hostile organization's track from this information and deduces whether it lies in the hostile organization's path. That is the current definition of threat.

---

[11] Zeus is available at http://zeus.enhydra.org/.

These considerations led IDA to design the system as the components shown in Figure 17. (The figure shows three friendly organizations and three hostile organizations, but the number of each is a runtime parameter.) Each friendly organization rectangle represents a unique application process, with the components shown in Figure 15.

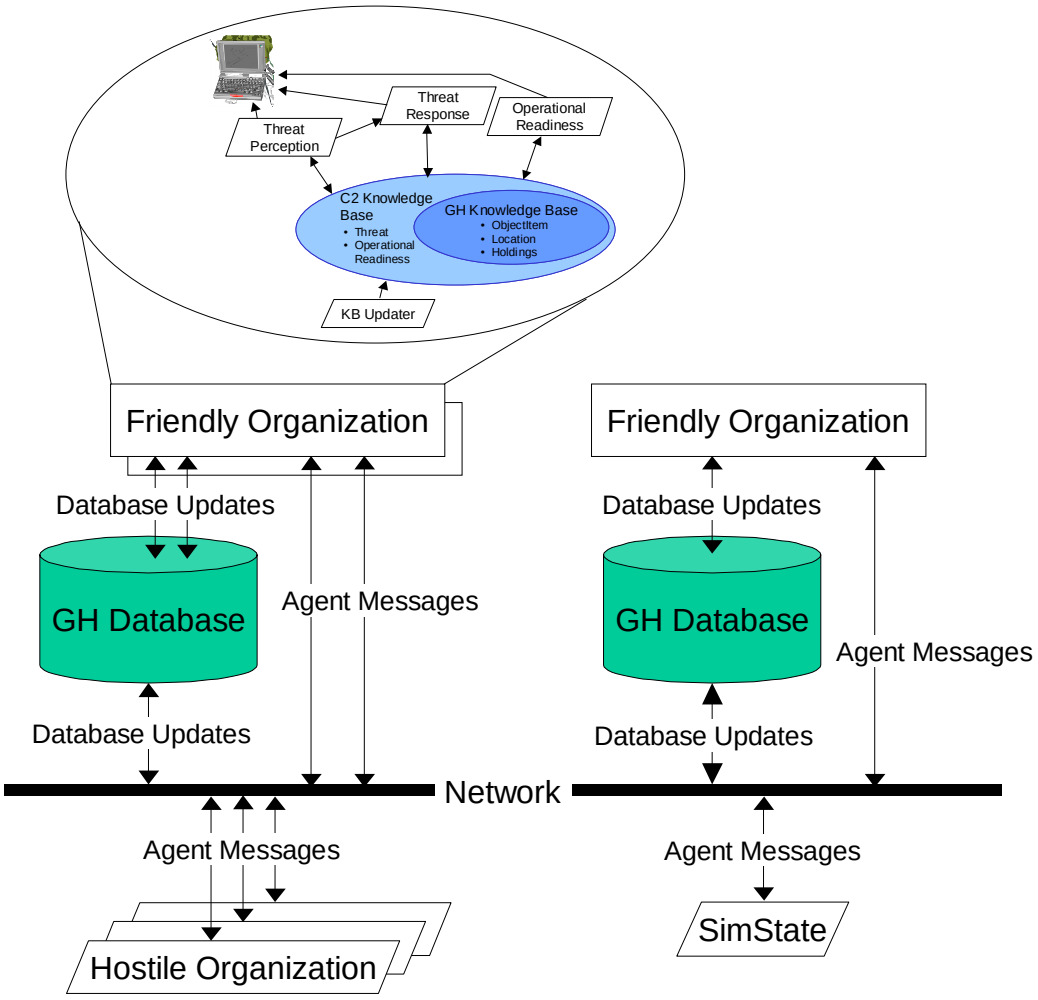

Figure 17. Prototype System Components

With one exception, all inter-process communication occurs through agents. (The exception is a friendly organization's communication with the DBMS. The mechanism for communication to the DBMS is a decision of the DBMS vendor.) Each agent knows the types of agents with which it might profitably share data, and is responsible for searching for all agents of that type. The agents use FIPA-OS agent management tools to carry out this task. The threat perception agent, for instance, searches for all known threat response agents. In consequence, whenever the threat perception agent of one friendly organization detects a threat, the threat response agents of all friendly organizations analyze the threat to see if it affects their mission in any possible way.

The prototype simulates zero or more hostile organizations. A hostile organization is identified by name. Its characteristics come from the GH data set. A hostile organization is required to have a location. As part of executing the prototype, it may be set in motion;

parameters of motion are its speed, bearing angle, and the interval in seconds between the time it recalculates its position. Figure 18 shows the user interface a hostile organization displays.

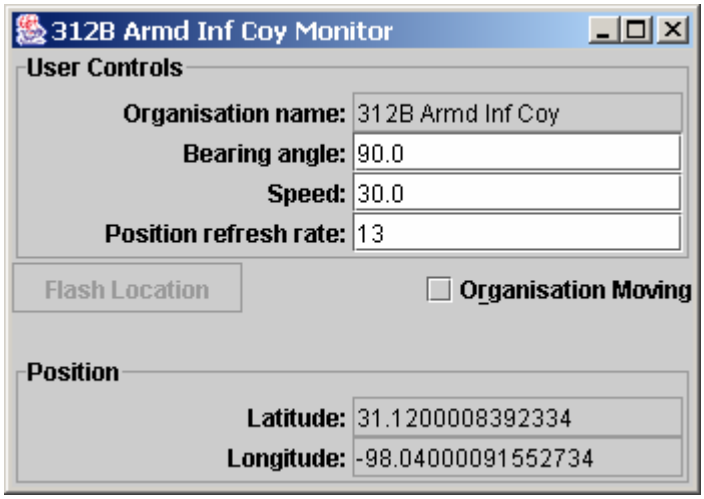

Figure 18. Hostile Organization User Interface

Hostile organizations are implemented as agents. They are not true agents, in that they do not draw inferences or interact with a knowledge base. Using FIPA-OS agent technology simplifies the implementation of their communication infrastructure.

An execution of the prototype creates a single SimState application. This application, which is implemented as an agent (as with hostile organizations, to take advantage of FIPA-OS's communication utilities), maintains ground truth. Each time an organization changes position, it communicates its new location to the SimState agent. (The hostile organization in Figure 18 does so every 13 seconds.) Currently, battlefield object location is the only ground truth the SimState agent maintains. Sensors of friendly organizations get their information by sending a message to the SimState agent, asking for the locations of all battlefield objects within the range of the sensor.

4.3.5 The Data Set

To add realism, IDA used an unclassified data set from a NATO exercise that occurred in and around Fort Hood, Texas.

The data set IDA received used version 4 of the GH. IDA converted the data set to version 5.

Version 5 mainly extends version 4, and where possible, IDA added extra specificity to the data set. For instance, Figure 19 shows that version 5 has a richer set of subtypes for ORGANISATION-TYPE than does version 4. For each UNIT-TYPE instance, IDA added corresponding GOVERNMENT-ORGANISATION-TYPE and MILITARY-ORGANISATION-TYPE instances.

Version 4 models associations between battlefield objects as relationships between subtypes of OBJECT-ITEM including an associative entity such as ORGANISATION-PERSON-

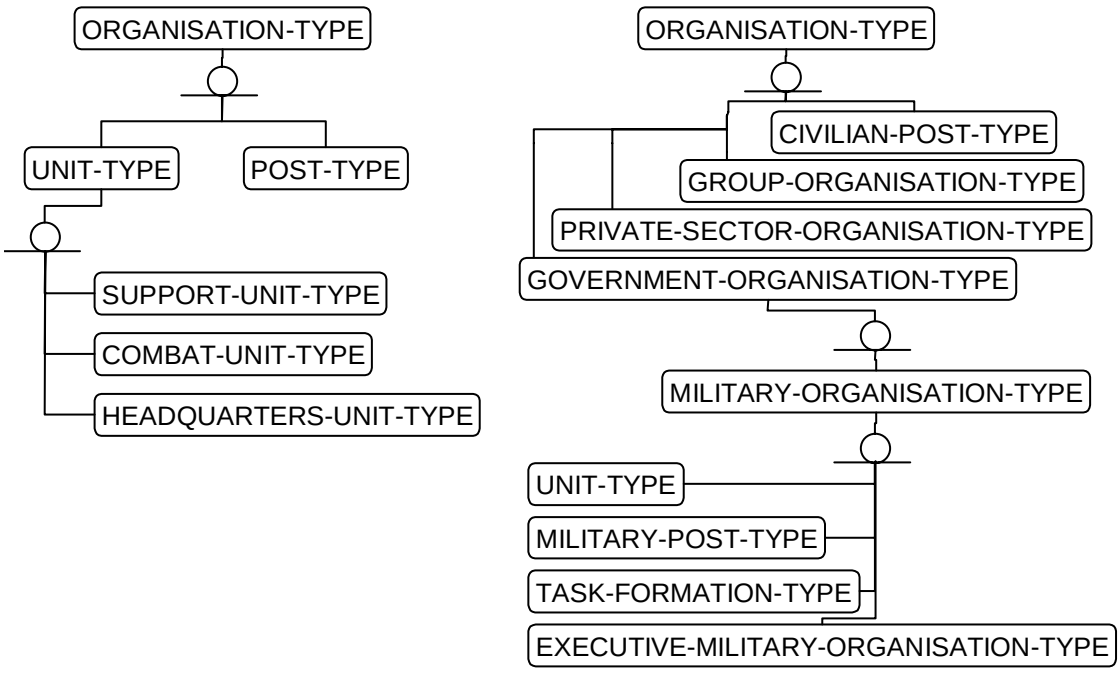

Figure 19. Migration of ORGANISATION-TYPE from GH4 to GH5

ASSOCIATION. Version 5 uses a more general scheme: OBJECT-ITEM has subject and object relationships to associative entity OBJECT-ITEM-ASSOCIATION. IDA converted all battlefield object associations to the version 5 approach.

Occasionally, differences between the versions could not be resolved by an exact mapping. In such cases IDA made an arbitrary decision that seemed justifiable in the context of the prototype. For example, version 4 includes the attribute point-horizontal-precision-code, which records the precision to which a POINT is measured. Legal values for this attribute are codes such as "100M", meaning precision of 100 meters. Version 5 has no exact matching attribute. The attribute absolute-point-angular-precision-code denotes the precision to which an ABSOLUTE-POINT is measured, but its values state precision in degrees, not meters. IDA mapped point-horizontal-precision-code to the closest value in absolute-point-angular-precision-code.

4.3.6 The Decision Support Application

The most significant component of the prototype is the decision support application. It is responsible for launching the agents that draw inferences, and for presenting a graphical user interface demonstrating that the types of information inferred from the knowledge base can be useful in decision support.

IDA's objective for the prototype was to demonstrate how the GH ontology can be used to reason about C2-related issues. Figure 20 shows the interface. The screen is divided into the following areas:

- *Map*: The area in the upper left displays unit locations atop a topographic map. A unit is designated by a "U". Friendly units are shown raised, in blue. Hostile units are shown lowered, in red. The organization running the decision support application (here, Panzerbataillon 433) has a cyan-colored "U".

  The topographic map comes from TopoZone.com, a commercial web site that distributes United States Geographical Survey maps. IDA eschewed military maps to ensure that the prototype contained no classified information.

- *Organization Holdings*: The table in the lower left displays personnel and materiel possessed by the organization running the decision support application. More precisely, the table displays the name of each OBJECT-TYPE instance associated with an organization through an instance of HOLDING, along with the value of the holding-operational-quantity attribute. The values are refreshed as the organization consumes and obtains materiel (though the prototype currently does not include a consumption model). The values come from the knowledge base, not the database, and therefore reflect ongoing reasoning.

- *C4I Display*: The box in the lower right corner shows two items relevant to C4I. The top item is the organization's current task. When the application starts, this task is obtained from the database. It is refreshed as the organization makes decisions or receives orders.

  The lower item shows updates to the knowledge base. These are either generated by the decision support application, or received from other applications through the underlying GH database.

- *Threats*: The box on the right side shows the threats currently known to the organization. In Figure 20, there is one threat, named 3C Armor Recce Coy, which happens to be the red organization (in the prototype, if the user moves the cursor over a "U"; a pop-up displays the unit's name). The threat perception agent has detected that 3C Armor Recce Coy is moving and that Panzerbataillon 433 is in its path. Decision makers must act.

When the threat perception agent detects a threat, it notifies all threat response agents. Each threat response agent examines the knowledge base to determine possible courses of action. The threat response agent for Panzerbataillon 433 formulates courses of action and gives them to the decision support application, which then displays them to the user as shown in Figure 21.

In any given situation, doctrine dictates the courses of action. The threat response agent has identified four. Each is assigned a rating, which is a value from 0.0 to 1.0 that prioritizes the courses of action. If the rating is 0.0, it is not shown unless the user selects the Show Impossible Courses of Action checkbox. The two viable courses of action are that the organization defend its position or withdraw. The threat response agent uses knowledge of the threat's holdings and its own holdings to compute the rating.

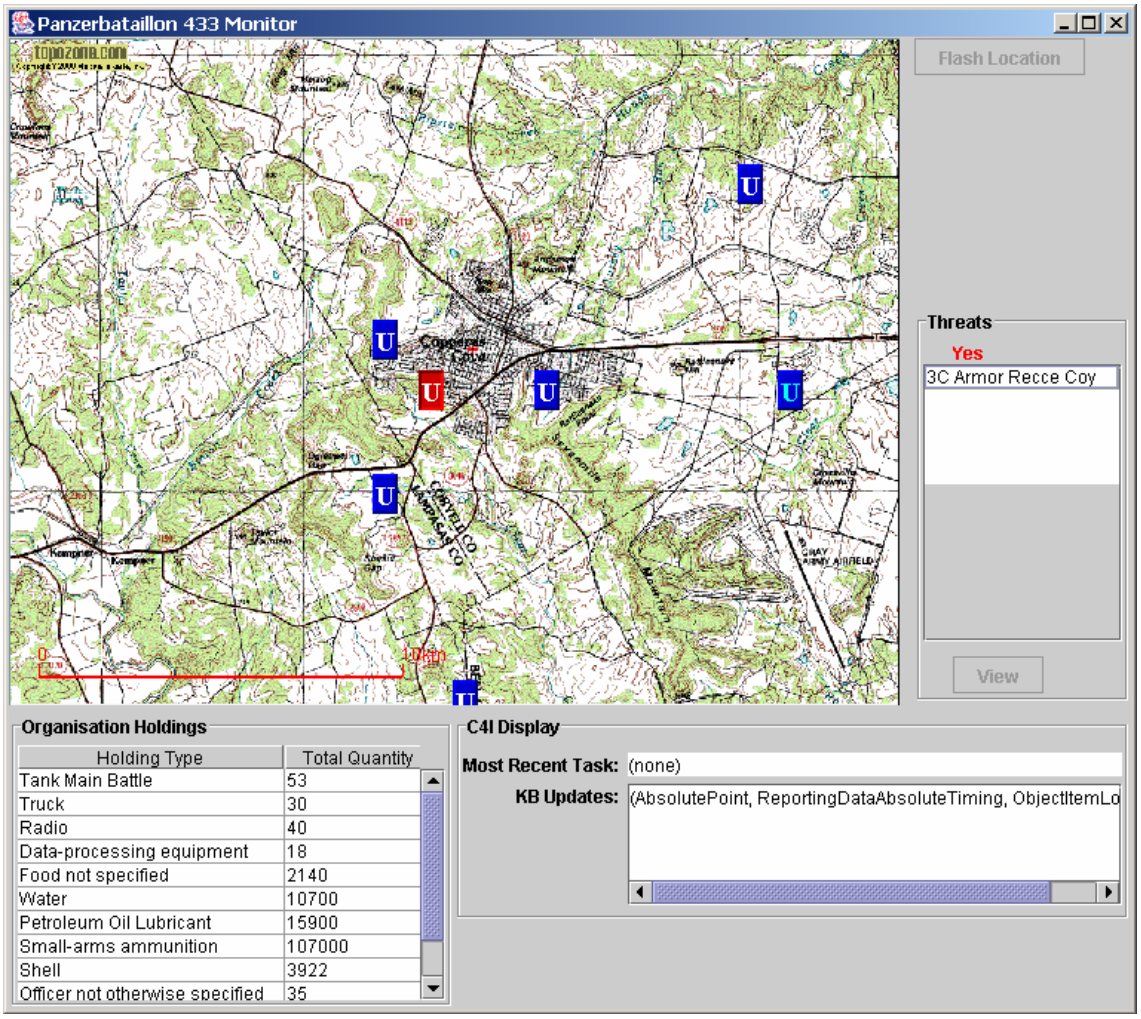

Figure 20. The Decision Support Application User Interface

The decision maker is unlikely to accept the threat response agent's prioritization without obtaining more information. Clicking the View button for a course of action supplies details on how the agent reached its conclusion. Figure 22 shows the details for the top-rated course of action in Figure 21. It provides a simple but logical set of reasons for the rating. In a real system, the decision maker would want more details on each step – e.g., how "adequate" are the supplies? – but for the prototype the information in Figure 22 makes the point.

The decision maker may select a course of action from the given set. When he does, the decision support application phrases this course of action as a set of GH entities: an ACTION-TASK, an ORGANISATION-ACTION-ASSOCIATION, ACTION-RESOURCE instances, etc. These entities are added to the GH database, from whence they are propagated to other GH databases and eventually added to the knowledge base of each active decision support application.

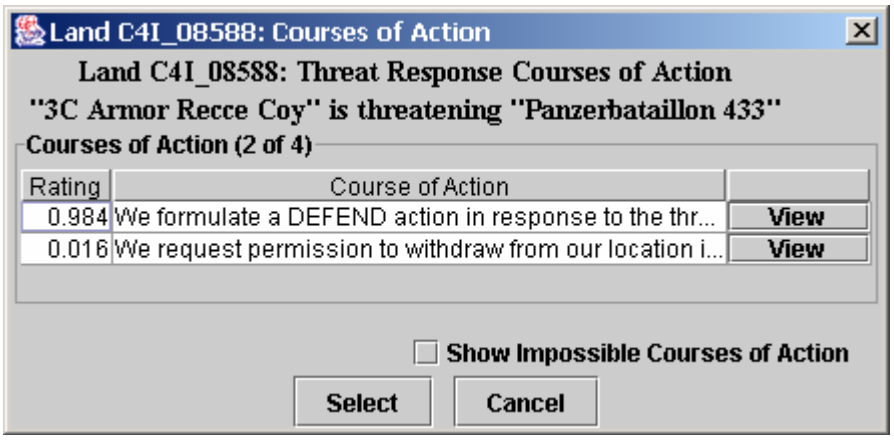

Figure 21. Example Courses of Action

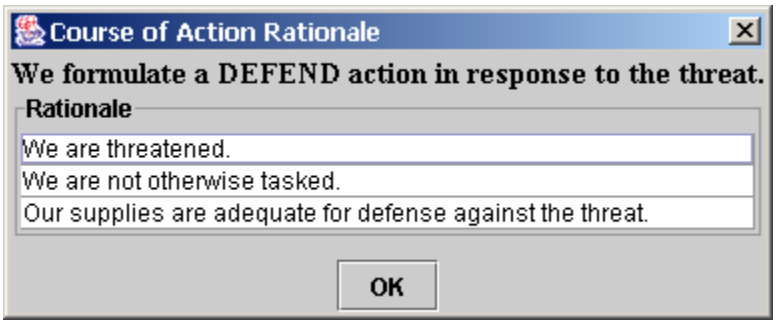

Figure 22. Example Justification for Course of Action

4.3.7 Implementing Inferencing

Reasoning about C2 using the knowledge base is a central function of the prototype. Examples mentioned so far include:

- How the threat perception agent detects that one organization threatens another. Detection includes the important sub function of filtering out false positives to prevent information overload.
- How the threat response agent determines the courses of actions applicable in a given situation.

The ability to express these examples succinctly will influence the implementation strategies.

IDA considered several technologies for drawing inferences on Protégé-2000 knowledge bases. Two were studied carefully. The first was PAL, the language developed as an integral part of Protégé-2000. The second was Algernon, available as a Protégé-2000 plug-in.

IDA found PAL inadequate. It is verbose, it lacks procedural abstraction capabilities, and its support for mathematical analysis is limited to simple arithmetic. This last problem means it cannot be used to model threat detection, which requires computing distance from geodetic coördinates. IDA did create a PAL frame that models whether an organization is hostile – a part of threat detection – but decided that PAL's minimal abstraction

facilities resulted in inordinately and unjustifiably complicated expressions IDA ultimately did not use the frame in the prototype.

This criticism of PAL is unduly harsh, for PAL is not intended for general-purpose reasoning. It is better suited to constraint enforcement. There was little need for constraint enforcement in the prototype, however, so IDA did not end up using PAL.

IDA found Algernon to be better. Algernon *is* suited to general-purpose reasoning. Furthermore, it can interface to system- and user-defined functions written in either Java or Lisp, a feature that greatly enhances its practical application. The version available at the time the prototype was being developed had numerous bugs, and though Algernon's author was responsive to bug reports, IDA eventually decided that using Algernon would introduce delays. Algernon was not used for this implementation, but subsequent work at IDA is actively pursuing the use of Algernon.

IDA ultimately implemented inferencing in Java. Agent methods directly invoke methods in the Protégé-2000 API. This is acceptable for the specialized inferencing performed in the prototype. It is less desirable in a real decision support application because it decentralizes inferencing rules.

4.3.8 Metrics

The prototype is implemented mainly in Java, or in languages translated by parser generators into Java. The current system consists of:

- Approximately 18,000 lines of non-generated Java, including documentation.
- Approximately 2,000 lines of other languages. These languages are:
  - Document Type Definition (DTD): specifications of XML documents.
  - XML documents, in particular a specification of the mapping between a GH data set and a GH knowledge base.
  - Javacc, a language for specifying parsers written in Java.

5. Conclusions and Directions

This paper has discussed the design and implementation of artifacts intended to evaluate the feasibility of using an ontology to make C2 data understandable. Data understandability is a necessary component of net-centricity. Creating an ontology is not the only task needed to make data understandable. One must also provide descriptors for the ontology that will allow agents to find the ontology on the GIG. However, an ontology is a necessary step.

The creation of the ontology and implementation of the prototype has given the IDA team the opportunity to work with many state of the art tools in the area. This section presents the IDA team's experiences, reflects on what has been accomplished, and discusses what remains.

5.1 The GH Ontology

The GH ontology IDA has created is structurally similar to the GH IEDM. Each class in the GH IEDM maps to a class in the GH ontology (though not vice versa). Each organic attribute and relationship in the GH IEDM maps to a slot in a GH ontology class.

This similarity partly reflects the capabilities offered by Protégé-2000, which are similar to those offered by IDEF1X. OKBC permits forms of knowledge modeling that Protégé-2000's designers chose not to implement. For instance, OKBC permits N-ary relationships, whereas Protégé-2000 allows only binary relationships (although facets can be used to model restricted ternary relationships). An ontology is supposed to offer information on relationships, and the GH ontology does not describe the M: N relationships that exist between certain classes as explicitly as information modelers might desire.

The reasoning capabilities that IDA has included in the ontology have to do with physical and enumerable properties. The property a slot measures is described, and inferences can be made as to whether two slots measure the same property, or whether a set of properties can be combined to yield another property (e.g., $\text{height} \times \text{length} \times \text{width} = \text{volume}$). This kind of capability should prove useful in certain C2 situations, such as deciding the feasibility of shipping a given materiel type in a given container.

IDA has not yet attempted to implement reasoning involving actions – or rather, no frames in the ontology implement action-based reasoning. A realistic demonstration of such reasoning is likely to require a significant investment of time and subject matter expert advice.

In hindsight, the choice of Protégé-2000 as the ontology representation tool seems correct. The deficiencies in the core implementation of Protégé-2000 have been noted in this paper, but Protégé-2000 has an active user community that is busy implementing plug-ins. These plug-ins support (and Protégé-2000 is moving towards) the best standards on

knowledge representation devised by the World Wide Web Consortium (W3C). Use of tools supporting web standards will make the ontology portable and reusable.

The GH ontology was implemented using a frame-based ontology, as opposed to description-logic ontology. Description-logic ontologies are becoming increasingly popular. They are the basis for OWL, the Web Ontology Language [OWL 2003], and the ongoing Semantic Web efforts centered around OWL [Berners-Lee 2001]. At the time this project began, automated support for OWL was minimal, and reasoning tools nonexistent. The past few years have seen considerable growth of support for the Semantic Web, however, and construction of prototype tools that use OWL-based ontologies is now practical. It is therefore reasonable to ask how this project, were done over again, should make use of OWL. The answer appears to be that metadata – the information used for ontology discovery – should be expressed in OWL. Description logics are suited to determining similarity between concepts. However, description logics are not well suited to expressing the kinds of queries that the prototype makes to determine threats; for that part of the application, the frame-based ontology seems preferable. An OWL extension called Semantic Web Rule Language (SWRL) [SWRL 2004] has been proposed to support rules in OWL knowledge bases. Eventually SWRL may prove comparable in power to tools that analyze frame-based ontologies, but currently almost no tool support exists for it.

Time constraints, and a tight focus on project objectives, prevented IDA from exploring other technologies and standards that are potentially useful and certainly relevant. Technologies and standards of which we are aware, but did not investigate, include:

- The DoD Discovery Metadata Standard (DDMS) that facilitates making data visible [Stenbit 2003]. Creating DDMS descriptors from the C2 ontology will be necessary to ensuring that the ontology is seen and used.
- The Resource Description Framework (RDF) [RDF 1999], a W3C standard for promoting interoperability. The future of message interchange among agents is likely to involve RDF.
- The Java Expert System Shell (JESS), a popular reasoning engine [Friedman-Hill 1997]. JESS should be investigated as an alternative to Algernon and PAL.

5.2 The C2 Ontology

The IDA team worked on the C2 ontology but did so more to explore how C2 concepts might be represented in terms of the GH ontology than to create a product that an application could use, even in a laboratory setting.

The team did gain enough experience to appreciate what the creation of a realistic C2 ontology would entail. Formalizing the concept of a threat would require consultation with, and consensus of, subject matter experts. These experts would be responsible for stating conditions under which a battlefield object might reasonably be termed a threat. The key to formalizing their knowledge would be to ensure that their statements can be expressed in terms of GH concepts. Any nontrivial statement of a threat is likely to be highly complex.

5.3 Use of Ontologies

A few words on the merits of ontologies are in order. IDA invested nontrivial effort in developing the GH ontology. In theory, an ontology provides no benefits over a DBMS, i.e., anything that can be expressed in an ontology can also be expressed in a database. In practice, IDA observes the following:

- The OKBC frame-based model is more abstract than the relational data model. The GH ontology has fewer implementation details than the corresponding relational database. The ontology needs no keys, and it substitutes relation names for foreign keys. The GH ontology is closer to the ER version of the GH5 than the database is. The ER version being easier to understand, IDA claims the GH ontology is easier for application developers to understand as well.
- A significant portion of the effort in developing the ontology was necessary because of the ontology's experimental nature. As Section 3 indicates, IDA studied many design tradeoffs in the course of the GH ontology's development. IDA expects that subsequent ontologies would use lessons learned from this project, and that their development time would be less.
- Conventional wisdom states that business rules should be kept outside of applications. Both ontologies and databases are appropriate places to store rules. SQL standardizes constraint-based rules (e.g., a column may not be empty; a column's valid values are drawn from a prespecified set). An ontology allows for constraint-based rules. It also allows for knowledge acquisition rules (e.g., the existence of certain conditions means a threat object should be created). SQL does not offer knowledge acquisition rules. Many DBMSs extend SQL to allow for knowledge acquisition (Oracle's triggers and its PL/SQL language can achieve the effect), but their use would restrict database portability. An ontology, then, offers the advantage of freedom from using nonstandard database features (though current technologies for knowledge acquisition are still immature; see Section 5.4).

5.4 The Prototype

IDA wanted the prototype to have realistic system architecture for a decision support system. For this reason, it obtains data from, and updates, a database.

IDA also wanted the prototype's architecture to be viable in a net-centric environment. Work in this area is incomplete. The prototype *is* viable insofar as creating an ontology, but it does not publish the ontology as it should, nor does it perform searches for arbitrary ontologies and attempt to intuit their content's relevance. It searches specifically for C2 ontologies and assumes that the agents for those ontologies understand its protocols. This, however, may not be too different from how a real agent would behave. As Section 1.1 pointed out, a decision support application has to be picky about conceptual equivalence. In any case, the agents are programmed to deal with requests that aren't understood or even simply ignored.

That the inferencing was implemented in Java is unfortunate in the sense that IDA has not been able to evaluate the feasibility of formalizing C2 knowledge in a very high level

language (VHLL). Whether switching to a VHLL is technically advisable is controversial. Artificial intelligence researchers have been touting the benefits of expert systems for two decades but have yet to produce evidence that writing in a VHLL is cheaper or yields more reliable systems than does writing in a procedural language such as Java. However, rules written in a VHLL can be incorporated into an ontology (Protégé-2000 offers this feature), centralizing them and thereby saving developers from potentially having to re-implement them in multiple agents.

References

[Berners-Lee 2001] Tim Berners-Lee, James Hendler, and Ora Lassila, "The Semantic Web." *Scientific American,* May 2001.

[Booch 1996] Grady Booch, *Best of Booch: Designing Strategies for Object Technology.* Cambridge University Press, New York, NY, 1996.

[Chaudhri 1998] V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, and J.P. Rice. OKBC: A programmatic foundation for knowledge base interoperability. In: Fifteenth National Conference on Artificial Intelligence (AAAI-98). Madison, Wisconsin: AAAI Press/The MIT Press, 1998.

[Erwin 2003] S. Erwin, "Land Warrior Follows Simpler Path." National Defense Magazine, November 2003. http://www.nationaldefensemagazine.org/article.cfm?Id=1238.

[Farquhar 1997] A. Farquhar, A., R. Fikes, and J.P. Rice, The Ontolingua server: a tool for collaborative ontology construction. International Journal of Human-Computer Studies. 46: pp. 707–727, 1997.

[FIPA 2001] B. Burg, J. Dale, and S. Willmot, "Open Standards and Open Source for Agent-Based Systems." AgentLink News, Issue 6, January 2001. http://www.fipa.org/docs/input/f-in-00024/f-in-00024.html.

[Friedman-Hill 1997] E. Friedman-Hill, *Jess, the Rule Engine for the Java Platform.* SAND98-8206, Sandia National Laboratories, Livermore, CA. http://herzberg.ca.sandia.gov/jess/docs/index.shtml.

[Genesereth 1992] M. Genesereth and R. Fikes, *Knowledge Interchange Format Version 3.0 Reference Manual.* Computer Science Department, Stanford University, Stanford, CA, June 1992. http://meta2.stanford.edu/kif/Hypertext/kif-manual.html.

[GIG 2001] A reference to the GIG. Possibilities include the CRD (approved Aug '01) and the original memorandum ('99).

[Kiryakov 2002] A. Kiryakov, D. Ognyanov, and B. Popov, *Ontology Middleware: Analysis and Design.* Deliverable 38, On-To-Knowledge project, March 2002. http://www.ontoknowledge.org/downl/del38.pdf.

[Kuipers 1994] B. Kuipers, *Algernon for Expert Systems.* Computer Science Department, University of Texas, Austin, TX, January 1994. http://www.cs.utexas.edu/users/qr/algy/algy-expsys/algy-expsys.html.

[NATO 2002] Working Paper 5-5, Edition 5.0. “The Land C2 Information Exchange Data Model.” ATCCIS Baseline 2.0. Army Tactical Command And Control Information System Working Group, SHAPE, Belgium. March 2002.

[NIST 1993] *Information definition for information modeling (IDEF1X)*, National Institute of Standards and Technology, Gaithersburg, MD, 1993 (FIPS Publication 184). http://www.itl.nist.gov/fipspubs/idef02.doc.

[Noy 2000] N. Noy, R. Fergeson, and M. Musen, *The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility.* Stanford Medical Informatics, August 2000. http://smi-web.stanford.edu/pubs/SMI-Abstracts/SMI-2000-0830.html.

[OWL 2003] *OWL Web Ontology Language Overview.* W3C Candidate Recommendation, August 2003. http://www.w3.org/TR/owl-features/.

[Poslad 2000] S. Poslad, P. Buckle, and R. Hadingham, “The FIPA-OS Agent Platform: Open Source for Open Standards.” In: Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, UK, pp. 355–368, 2000.

[RDF 1999] *Resource Description Framework (RDF) Model and Syntax Specification.* W3C Recommendation, February 1999. http://www.w3.org/TR/REC-rdf-syntax.

[Stenbit 2003] John Stenbit, *Department of Defense Net-Centric Data Strategy,* Department of Defense, 6000 Defense Pentagon, Washington, DC 20301-6000, May 9, 2003. http:www.defenselink.mil/nii/org/cio/doc/ Net-Centric-Data-Strategy-2003-05-092.pdf.

[Suguri] H. Suguri, E. Kodama, M. Miyazaki, H. Nunokawa, and S. Noguchi, *Implementation of FIPA Ontology Service.* http://ias.comtec.co.jp/ap/comtec-ontology-service.pdf.

[SWRL 2004] *SWRL: A Semantic Web Rule Language Combining OWL and RuleM,* W3C Member Submission, 21 May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[W3C 2000] *Extensible Markup Language (XML) 1.0 (Second Edition).* October 2000. http://www.w3.org/TR/REC-xml.

[W3C 2001] *XML Schema Part 0: Primer.* W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-0.

Abbreviations and Acronyms

ATCCIS	Army Tactical Command and Control Information System
C2	Command and Control
C4I	Command, Control, Communications, Computers, and Intelligence
CIO/G6	Army Office of the Chief Information Officer
CoABS	Control of Agent Based Systems
DBMS	Database Management System
DoD	Department of Defense
DTD	Document Type Definition
ER	Entity-Relationship
FCS	Future Combat System
GH	Generic Hub
GH5	Generic Hub, Version 5
GIG	Global Information Grid
ICAM	Integrated Computer-Aided Manufacturing
IDA	Institute for Defense Analyses
IDEF1X	ICAM Definition Method 1 Extended
IEDM	Information Exchange Data Model
NATO	North Atlantic Treaty Organization
OKBC	Open Knowledge Base Connectivity
SHAPE	Supreme Headquarters Allied Powers, Europe
SQL	Structured Query Language
UML	Unified Modeling Language
VHLL	Very High Level Language
XML	Extensible Markup Language

Annex A: Design of the Ontologies

This annex describes in depth the ontologies used in the *Ontologies for Decision Support* project. The reader will understand the design decisions and rationale for the ontologies. The discussion should prove useful to someone who wants to extend IDA's efforts, or to someone tasked with developing an ontology for another domain.

1. Structure of the Ontologies

To aid in making C2 data understandable, the IDA team created two distinct ontologies:

1. The GH5 ontology, which directly models knowledge from a GH5-conformant data set.
2. The C2 ontology, which models higher-level C2 concepts in terms of GH data.

The IDA team separated these ontologies into 9 distinct Protégé-2000 projects. Each Protégé-2000 project is a distinct ontology. Protégé-2000 lets one project include a set of projects (circular references are not permitted). The nine projects the IDA team created have the inclusion relationships shown in Figure 1.

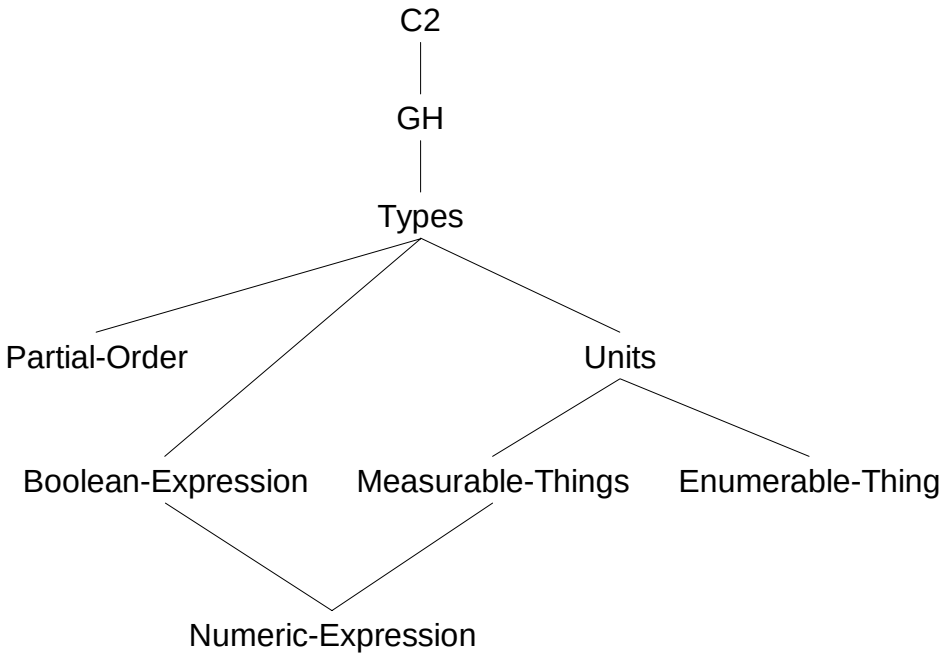

Figure 1. Protégé-2000 Project Inclusion Hierarchy

The ontology was split into separate projects to encourage the encapsulation of design decisions within individual projects. Thus a change in how units of measurement are represented is made within a single project.

A frame-based ontology does not encapsulate design decisions to the same degree as does an object-oriented programming language. Many classes of changes to a project, such as renaming or adding slots, require updates to projects that include the modified project. Nevertheless, splitting components of the entire C2 ontology across projects is useful for some practical reasons:

1. It permits several individuals to work on portions of an ontology simultaneously.
2. It shortens load time for individual ontologies. Protégé-2000 requires a non-trivial interval to load the entire GH ontology. When one is concentrating on unit-of-measurement issues, loading just the Units (of Measurement) ontology is much quicker than loading the entire package.
3. It forces a developer to concentrate on relevant concerns. There is less tendency to introduce arbitrary and ad hoc frames into an ontology if one's mind is focused on a narrow concept.

The next several sections discuss each ontology in Figure 1. The low-level, building block ontologies are discussed first.

2. The Numeric Expression Ontology

The Numeric-Expression ontology supports the statement of numeric information. Numeric information may be expressed as:

- Literal values. Protégé-2000 supports literal values but none of the other types of numeric information.
- Variables. A variable is a reference to a frame. The frame is assumed to contain sufficient information to obtain a numeric value. It is the responsibility of any ontology that includes Numeric-Expression to define how this information may be obtained.
- Unary operators. A unary operator is anything that, when applied to an instance of a Numeric-Expression, yields an instance of a Numeric-Expression.
- Binary operators. A binary operator is anything that, when applied to two Numeric-Expression instances, yields an instance of a Numeric-Expression.
- N-ary operators. An n-ary operator, when applied to one or more Numeric-Expression instances, yields an instance of a Numeric-Expression.

Figure 2 shows the classes that form the Numeric-Expression ontology. Classes for inner nodes are abstract, because only for the leaf nodes is enough information available to describe an instance.

2.1. Slots

The Numeric-Expression class has a single string-valued template slot named description. When a user creates an instance of the Numeric-Expression class, he should enter a description of an expression in this slot. The forms of the Numeric-Expression ontology use description as the form browser key; therefore, this description will identify the instance.

2.1.1. Slots of the Literal Class

The Literal class has two template slots: value and base. The value slot denotes the literal value of an instance. Its type facet is string, though that facet must be overridden in concrete subclasses.[1] The base slot denotes the expected numeric base of a value. It is intended as additional information to be used when the value is converted to or from a string representation. Its default value is 10.

[1] Protégé-2000 has an Any type that is arguably more appropriate than string. However, the version of Protégé-2000 used to develop the Numeric-Expression ontology did not properly implement overriding Any slots in subclasses. Protégé-2000's developers claim to have fixed this in subsequent versions.

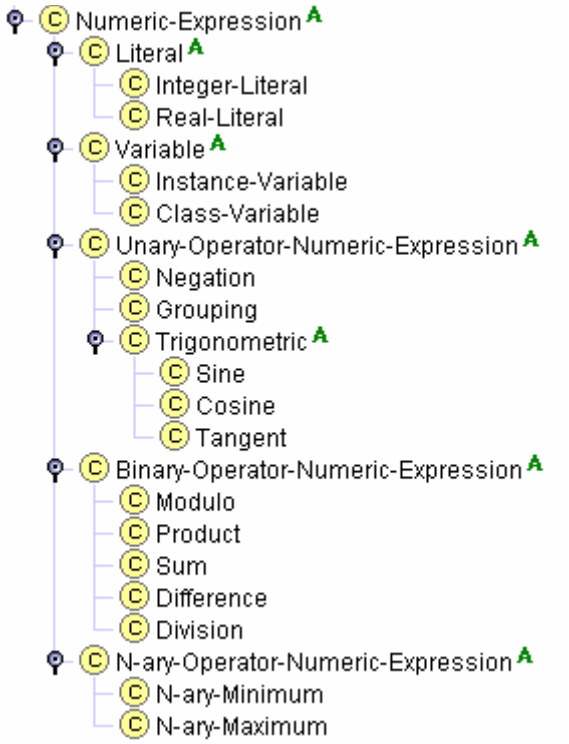

Figure 2. Classes of the Numeric-Expression Ontology

The Literal class currently has two subclasses, Integer-Literal and Real-Literal. These model, respectively, integer and real numbers by overriding the type facet of the value slot to Integer and Float.

In the form for the Literal class, the value slot is used as the browser key.

Protégé-2000 represents numbers using 32-bit representations. This suffices to represent GH5 integral and real numbers that represent physical and conceptual properties. It is not enough for GH5 attributes that represent keys, which are as much as 18 decimal digits. The GH ontology does not represent keys, so the 32-bit representation caused no problems. If a GH5 attribute had required more than 32 bits, the simplest approach to modeling it in Protégé-2000 would have been to create a subclass of Literal that represented literal values as a string; that, along with the base slot, would have allowed representation of larger or more precise literals.

2.1.2. Slots of the Variable Class

The Variable class has a template slot variable-ref. Its subclasses specialize this slot's type. In class Instance-Variable, the variable-ref slot is an instance of :THING, i.e., to any frame. In class Class-Variable, the variable-ref slot is a subclass of :THING (or :THING itself).

Class Instance-Variable supports the creation of numeric expressions that are functions of a set of variables. Class Class-Variable was originally intended to support the creation of numeric expressions involving metadata. It is not used in the current version of the GH ontology.

2.1.3. Slots of the Unary-Operator-Numeric-Expression Class

The Unary-Operator-Numeric-Expression class has a single template slot named unary-op-value. This slot, which is of type Numeric-Expression instance, denotes the expression to which a unary operator is to be applied. The operator itself is denoted by the subclass of Unary-

Operator-Numeric-Expression: Negation, Grouping (i.e., parenthesizing), and assorted trigonometric operators.

2.1.4. Slots of the Binary-Operator-Numeric-Expression Class

The Binary-Operator-Numeric-Expression class has two template slots: binary-op-left and binary-op-right. These slots are of type Numeric-Expression instance. As with Unary-Operator-Numeric-Expression, the actual operator is denoted by subclass. The binary operators currently included in the Numeric-Expression ontology are the arithmetic operators and the modulo operator.

2.1.5. Slots of the N-ary Operator-Numeric-Expression Class

The N-ary Operator-Numeric-Expression class has a single template slot named operands. This slot has multiple cardinality but requires at least one element. Order among the values is not significant. This approach is a consequence of Protégé-2000's decision to return the values of a multiple-cardinality slot as an unordered collection. Maxima and minima operators are implemented as subclasses of this class. At one time addition and multiplication were subclasses of this class, but they were ultimately made binary operators to keep them together with subtraction and multiplication.

2.2. Instances in the Ontology

The Numeric-Expression ontology has instances for common literal values such as 0 and 1. (Instances for these values exist as both integers and reals.) Consolidating common values in the Numeric-Expression ontology makes other ontologies less cluttered.

2.3. Use of the Ontology

Other ontologies use the Numeric-Expression ontology to form knowledge of numeric expressions. Numeric expressions are useful in the following circumstances:

- Expressing rational and irrational values. Although a quantity such as $1/3$ or $\sin(45)$ can be expressed as a literal to the limit of Protégé-2000's precision, expressing it as a formula aids in understanding the origins of the expression.
- Expressing functions. The GH data model lets capabilities be stated in many different units of measurement. An ontology should be able to convert hours to minutes, i.e., it should be able to express the formula $m = h \times 60$.

Protégé-2000 has no built-in capacity to evaluate instances of a class. Thus a Numeric-Expression instance, even one involving only constant values, has no inherent meaning to Protégé-2000. The IDA team regretted this missing capacity, as some useful constraints could have been stated using Numeric-Expression instances. Furthermore, an application that uses the Numeric-Expression ontology may have to implement an interpreter. An alternate approach that the IDA team did not explore would be to implement a Protégé-2000 plugin that recognizes the Numeric-Expression class. Queries in Protégé-2000's PAL are implemented as a plugin, so the approach seems consistent with Protégé-2000's design philosophy.

3. The Boolean Expression Ontology

The Boolean-Expression ontology serves an analogous purpose to the Numeric-Expression ontology. It allows the statement of Boolean expressions. These expressions can be used to describe conditions that are true at some moment in time, for example. Describing them as a

complex formula with bound variables may be more meaningful than using simple true and false values. Boolean expressions may be:

- Negation (the not operator)
- Implication (the ⇒ operator)
- Numerical comparisons (<, ≤, etc.)
- Multiple-operand expressions, (AND, OR, etc.)

Assuming all variables are bound, an instance of class Boolean-Expression denotes either truth or falsehood.

Figure 3 shows the classes in the Boolean-Expression ontology. Classes for non-leaf nodes are abstract, because only leaf nodes have enough information to make creating direct instances logical.

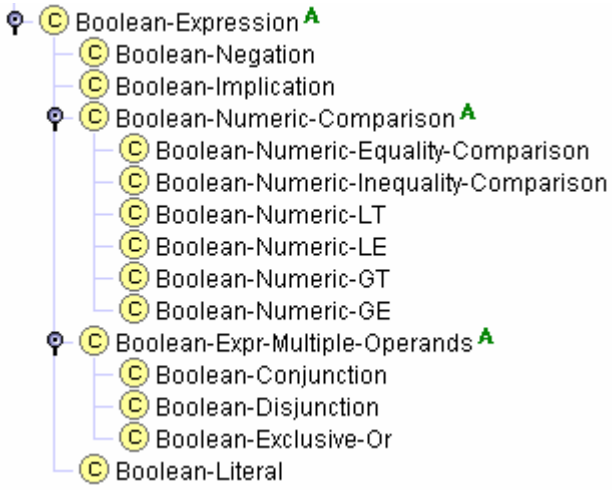

Figure 3. Classes of the Boolean-Expression Ontology

3.1. Slots

The Boolean-Expression class has a single string-valued template slot named boolean-expr-description.[2] When a user creates an instance of the Boolean-Expression class, he should enter a description of an expression in this slot. The forms of the Boolean-Expression ontology use boolean-expr-description as the form browser key; therefore, this description will identify the instance.

3.1.1. Slots of Boolean-Negation

The Boolean-Negation class has a single template slot named boolean-expr-negation-operand. This slot, which is of type Boolean-Expression instance, denotes the expression to which negation is to be applied.

3.1.2. Slots of Boolean-Implication

The Boolean-Implication class has two template slots: implication-premise and implication-conclusion. Both are of type Boolean-Expression instance. They denote the premise and conclusion, respectively, of an implication operation.

---

[2] Protégé-2000 does not permit two slots to have the same name, and description is already used in the Numeric-Expression ontology. Facets of description could have been overridden for Boolean-Expression, but that approach courts disaster should description be modified in Numeric-Expression.

3.1.3. Slots of Boolean-Numeric-Comparison

The abstract class Boolean-Numeric-Comparison has two template slots: boolean-numeric-left-operand and boolean-numeric-right-operand. They denote the left and right operand to apply to a numeric comparison operator. Each is an instance of class Numeric-Expression. The actual numeric comparison operator is specified as a subclass of Boolean-Numeric-Comparison. The Boolean-Expression ontology provides the usual set.

3.1.4. Slots of Boolean-Expr-Multiple-Operands

The abstract class Boolean-Expr-Multiple-Operands has a template slot named boolean-expression-operands. This slot is of multiple cardinality. Each value is an instance of Boolean-Expression. At least one instance is required. The values of the slot represent the operands to be applied to a Boolean operator, which is specified as a subclass. The order in which the operands are applied to the operator is unspecified. This approach is a consequence of Protégé-2000's decision to return the values of a multiple-cardinality slot as an unordered collection.

The Boolean-Expression ontology currently defines three multi-expression operators: conjunction, disjunction, and exclusive-or.

3.1.5. Slots of Boolean-Literal

The Boolean-Literal class has a template slot named boolean-expr-value. This Boolean-valued slot denotes whether an instance denotes truth or falsehood.

3.2. Instances in the Ontology

The Boolean-Expression ontology contains two instances. One denotes truth, the other falsehood. Their labels are True and False, respectively.

3.3. Uses of the Ontology

The GH ontology makes extensive use of the Boolean ontology. However, it currently does so only in the simplest way: it uses instances of Boolean-Literal to represent the many true/false attributes in the GH5. These instances seldom have labels of True or False. The GH prefers to use YES and NO.

Protégé-2000 has no built-in capacity to evaluate instances of a class. Thus, as with Numeric-Expression, a Boolean-Expression instance has no inherent meaning to Protégé-2000. An application that uses the Boolean-Expression ontology may have to implement an interpreter, or alternately implement a Protégé-2000 plugin that recognizes the Boolean-Expression class.

4. The Enumerable Thing Ontology

The Enumerate-Thing ontology models the concepts of things that can be enumerated. Enumeration is distinguished from measurement. Measurement represents a property of an individual thing. Enumeration identifies a group of things that are homogenous with respect to some criterion.

The Enumerable-Thing ontology contains a single class named Enumerable-Thing that is a subclass of :THING.

4.1. Slots

Class Enumerable-Thing has three template slots:

1. Slot enumerable-thing-reference is a reference to the thing of which instances are being enumerated. Its value is any class in the ontology.
2. Slot enumerable-thing-name denotes a name for an instance of Enumerable-Thing. It is used as the browser key for the class's form.
3. Slot enumerable-thing-symbol is a multiple-cardinality slot of strings. It may be used to record commonly used symbols and abbreviations for an enumerable thing. (In practice, this slot has not been used.)

4.2. Uses of the Ontology

The Enumerable-Thing ontology is used in the GH ontology to describe entities such as persons and ammunition.

The enumerable-thing-reference slot refers to a class, not to an instance. In the GH ontology, class Person – not an instance of Person – is an enumerable thing. The intent is to help an inferencing engine reason about a knowledge base through properties of things that accrue from innumerability.

5. The Measurable Things Ontology

The Measurable-Things ontology models the physically and conceptually measurable properties. It models only the properties, not the entities for which properties can exist. Other ontologies are able to characterize entities in terms of their measurable properties.

The ontology distinguishes between atomic and composite measurable things. As the names imply, an atomic measurable thing cannot be decomposed, whereas a composite measurable thing is composed of a set of other measurable things.

The Measurable-Things ontology also models mappings between measurable things. For example, it models measurement of length, width, and area, and also models that length times width equals area.

Figure 4 shows the classes of the Measurable-Thing ontology. Both Measurable-Thing and Measurable-Thing-Mapping are subclasses of :THING. Composite measurable things are modeled as a subtype of measurable things.

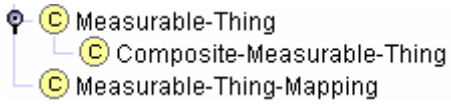

Figure 4. Classes of the Measurable Thing Ontology

Measurable things are conceptual and are stated without units of measurement. Units of measurement are expressed in another ontology.

5.1. Slots of Measurable-Thing

The Measurable-Thing class has three template slots:

1. The measurable-thing-name slot denotes a common name for an instance of a measurable thing. This name is presumed to be something that might be used as metadata.
2. The measurable-thing-description slot denotes a textual description for an instance. It is not intended to convey meaning.

3. The measurable-thing-subtype-of slot is an instance of a Measurable-Thing. It is not required, but if given, indicates a hierarchical relationship that can be useful as additional metadata. For example, time of day is a subtype of time.

The subclass Composite-Measurable-Thing contains an additional template slot named measure-expr. This slot, which is an instance of Numeric-Expression, expresses a functional relationship that specifies how a set of Measurable-Things composes some instance of a Measurable-Thing. An example is given below.

5.2. Slots of Measurable-Thing-Mapping

Each instance of the Measurable-Thing-Mapping class specifies how an instance of Measurable-Thing can be described in terms of a set of Measurable-Thing instances. It uses the following four slots:

1. The measurable-thing-label slot gives a textual description of the mapping. It is the form browser key.
2. The from-measurable-things slot is a multiple-cardinality slot of Measurable-Thing instances. At least one instance is required. It denotes the list of things that are to be combined according to the measurable-thing-expr slot.
3. The measurable-thing-expr slot is an instance of a Numeric-Expression. This expression must include instances of class Instance-Variable (see Section 2.1.2), each of which must be refer to a value in the from-measurable-things slot.
4. The to-measurable-thing slot is an instance of Measurable-Thing. It denotes the thing that results when the from-measurable-things are combined according to the measurable-thing-expr.

The Measurable-Thing ontology also adds class Measurable-Thing-Instance-Variable as a subclass of Instance-Variable. Its purpose is to separate variables used to reference measurable things from other variables used in other ontologies. The Numeric-Expression instance for the measurable-thing-expr slot should reference instances from this class.

5.3. Instances in the Ontology

The Measurable-Thing ontology includes instances of measurable things, and mappings between them, that are relevant to the GH ontology. Some examples of measurable things are:

- Distance: an amount of separation between two locations.
- Height, width, and length: Each of these is a subtype of distance.
- Area and volume: These are instance of Composite-Measurable-Thing. There exist instances of class Measurable-Thing-Instance-Variable that reference the Height, Width, and Length Measurable-Thing instances. Thus the measure-expr slot of Area is a Product instance where the left and right operands are the Width and Length instances of Measurable-Thing-Instance-Variable.
- Time, age, and time of day: The second and third are subtypes of the first.

An example of a measurable thing mapping is the relationship between speed, time, and acceleration. Speed and time are instances of Measurable-Thing. Acceleration is an instance of Composite-Measurable-Thing. There exists an instance of Measurable-Thing-Mapping in which:

- The from-measurable-things slot has as its value the two instances Speed and Time.
- The to-measurable-thing slot has as its value the instance Acceleration.

- The measurable-thing-expr slot has as its value a Division instance in which the left operand is the instance of Measurable-Thing-Instance-Variable that refers to Speed, and the right operand is the instance of Measurable-Thing-Instance-Variable that refers to Time.

5.4. Uses of the Ontology

The Units-of-Measurement ontology extends the Measurable-Thing ontology with units of measurement for each measurable thing. In other words, Measurable-Thing describes a class of concepts and their interrelationships, whereas Units describes different ways to express each concept.

There are some constraints that should be associated with this ontology. For example, in an instance of Measurable-Thing-Mapping:

- Every variable in the measurable-thing-expr slot must refer to an instance of a Measurable-Thing.
- The cardinality of the from-measurable-things slot should equal the number of instances of Measurable-Thing-Instance-Variable in the measurable-thing-expr slot.

Unfortunately, these constraints cannot be stated in Protégé-2000 (a recursive application function is needed). The IDA team opted to include constraints in the ontology, but stated them in natural language.

6. The Partial Order Ontology

The Partial-Order ontology facilitates the specification of a partial (or total, as a degenerate case) ordering among a set of instances. It consists of a single class, Order-Specification, that is a subclass of :THING.

6.1. Slots of Order-Specification

The Order-Specification class has the following slots:

1. The order-spec-description slot provides a textual label for each instance. It is used as the form browser key.
2. The order-spec-a and order-spec-b slots both have instances of :THING as values.
3. The order-spec-relationship slot is one of the symbols <, <=, or =.

The interpretation is that the instance denoted by order-spec-a has the relationship specified by order-spec-relationship to order-spec-b.

6.2. Uses of the Ontology

The GH ontology uses Partial-Order to specify relationships among values in coded domains. These domains are string-valued codes. Most of the codes are written such as to be in a lexicographic order that also corresponds to a logical ordering between elements, where such an ordering is possible. However, the ordering is not uniform. Some domains have smallest elements first, some have largest elements first. Moreover, some domains are partial and not total orders (for instance, unit size code), and almost all include the element NOS (not otherwise specified) which should not be considered in any order.

The Partial-Order ontology lets the order among elements be explicitly specified. One element can be stated as being less than, less than or equal to, or equal to another element. Transitive properties can be applied to determine if a relationship exists between a given pair of elements. The model of unit size, which in the GH data model encompasses units from air, land,

and sea branches of the services, is the most complete application of an ordering specification in the GH ontology.

For the ordering to make sense, the values of order-spec-a and order-spec-b must be from the same domain. This is enforced using the following PAL constraint:

```
(defrange ?os :FRAME Order-Specification)
(forall ?os
    (and (instance-of (order-spec-b ?os) (get-class (order-spec-a ?os)))
        (instance-of (order-spec-a ?os) (get-class (order-spec-b ?os)))))
```

Protégé-2000 requires an application or user to apply this constraint explicitly; it is not automatically enforced.

Currently an application must examine the ontology to determine ordering constraints. A plugin would be preferable, but has not been implemented because of time constraints.

7. The Units of Measurement Ontology

The Units ontology models different units of measurement. It builds on the Measurable-Things ontology by describing different standards by which a property can be measured. Like Measurable-Things, it includes a capability to specify interrelationships between instances. Figure 5 shows the classes in the ontology. Unit-of-Measurement and Unit-of-Measurement-Mapping are both subclasses of :THING.

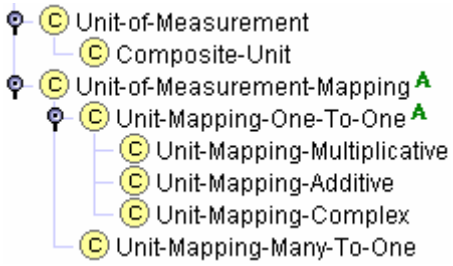

Figure 5. Classes of the Units Ontology

7.1. Slots of Unit-of-Measurement

Each instance of the Unit-of-Measurement class denotes a unit of measurement. The class has the following slots:

- The unit-name slot is used to supply a textual name for an instance. The value should be a full name, not an abbreviation. The value could be used as metadata. The slot's value must be unique with respect to a knowledge base.
- The unit-symbol slot is a multiple-cardinality slot whose values are strings. It is used to list common abbreviations and symbols for a unit of measurement. (Protégé-2000 is not restricted to ASCII; it supports the UTF-8 coding system, so ontology developers can use non-textual glyphs as symbols.) If the unit-name slot has the value kilometer, the unit-symbol-slot might have values km and k. These values can also be used as metadata, but they are not required, nor even expected, to be unique across a knowledge base.
- The unit-base-type slot specifies the most natural numeric representation for the instance. Its possible values are integer, real, complex, and string. The GH ontology only

uses integer and real. The slot gives an indication as to the nature of a unit: whether or not it records a quantity (usually an integer), for example.

- The unit-concept slot describes the concept that the unit of measurement models. It is a multiple cardinality slot whose values are instances of either Measurable-Thing or Enumerable-Thing.
- The unit-range-min and unit-range-max slots specify minimum and maximum permitted values for the unit. These slots are optional. However, many units of measurement cannot be less than zero (especially quantities) or some arbitrary value (e.g., degrees Celsius). A maximum value is less common but still useful for angles, color hues, and the like. It may also be useful in some domains to prevent nonsensical values.

Here is an instance of class Unit-of-Measurement:

unit-name	kilometer
unit-symbols	{ k, km }
unit-base-type	real
unit-concept	Distance (instance of Measurable-Thing)
unit-range-min	0.0
unit-range-max	38000

The value for unit-range-max is the approximate circumference of the Earth in kilometers. This would be a reasonable maximum value for land-based warfare.

Class Unit-of-Measurement has a subclass Composite-Unit-of-Measurement. This subclass is used to model units of measurement that are composed of other units of measurement, as Composite-Measurable-Thing models non-atomic measurable things. Composite-Unit-of-Measurement has a template slot unit-constituent-units. This slot is of multiple cardinality; its values are instances of Unit-of-Measurement that make up this instance. The following is an example:

unit-name	kilometers per hour
unit-symbols	{ km/hr, kph }
unit-base-type	real
unit-concept	Speed (instance of Measurable-Thing)
unit-range-min	0.0
unit-range-max	[not specified]
unit-constituent-units	{ kilometer, hour }

The values for unit-constituent-units must include one value for each constituent. Thus its value for square meters is { meter, meter }. The order of the values is not significant. Note that the Composite-Unit-Measurement class does not contain enough information to infer how constituent units form a composite unit. Future versions of the Units ontology may need to add this information, depending on inference needs.

The values for the unit-range-min and unit-range-max slots are instances of all subclasses of Numeric-Expression except Variable. The intent is that the minimum and maximum values should be constants, either literals or expressions that evaluate to constants. Of course, an

instance of an expression can contain a Variable instance as an operand. This violation is not detected.

7.2. Slots of Unit-of-Measurement-Mapping

The Unit-of-Measurement-Mapping class specifies how one unit can be converted to another unit. The conversion is specified as a restricted Numeric-Expression; the restriction depends on the type of mapping. The class has two template slots:

1. The unit-mapping-label slot specifies a textual label. It is used as the form browser key.
2. The to-unit slot specifies the Unit-of-Measurement instance that is the target of this mapping. That is, to specify a mapping from some unit to kilometers, one specifies that the kilometer instance of Unit-of-Measurement is the target of this mapping.

Class Unit-of-Measurement-Mapping is abstract. The nature of a mapping is specified in a subclass. Unit-of-Measurement-Mapping has two subclasses. One handles one-to-one mappings, the other many-to-one mappings.

7.2.1. Slots of Unit-Mapping-One-to-One

The Unit-Mapping-One-to-One class specifies a mapping from one unit of measurement to another. The target unit is specified by the to-unit slot of the parent class. This class has a template slot from-unit that specifies the source unit of measurement; its value is an instance of Unit-of-Measurement.

Usually the unit-concept slot of the from-unit and to-unit instances will refer to the same Measurable-Thing (or Enumerable-Thing) instance. This is not required, because the unit-concept slot has multiple cardinality to account for units of measurement that express multiple concepts. An example is degrees in the sense of location; degrees can refer to both latitude and longitude. Arguably, this flexibility leads to imprecision, and requiring equivalence of concepts between the from-unit and to-unit slots would be better. (In that case a knowledge base would have to express degrees of latitude and degrees of longitude as distinct units of measurement.)

The Unit-Mapping-One-to-One class is abstract, but has three subclasses, each of which expresses a specific kind of one-to-one mapping between units of measurement:

1. The Unit-Mapping-Multiplicative class denotes a mapping between units expressed as a constant multiplier. It has a template slot multiplier, whose value is an instance of class Literal, to denote the multiplier. The relationship between centimeters and millimeters can be expressed using this class:

unit-mapping-label	cm => mm
from-unit	Centimeter (instance of Unit-of-Measurement)
to-unit	Millimeter (instance of Unit-of-Measurement)
multiplier	10 (instance of Integer-Literal)

2. The Unit-Mapping-Additive class denotes a mapping between units expressed as a difference in magnitude. It has a template slot additive-factor, whose value is an instance of class Literal, to denote the difference. The relationship between degrees Kelvin and degrees Celsius is expressed as follows as an instance of Unit-Mapping-Additive:

unit-mapping-label	ºK => ºC
from-unit	degrees Kelvin (instance of Unit-of-Measurement)
to-unit	degrees Celcius (instance of Unit-of-Measurement)
multiplier	273 (instance of Integer-Literal)

3. The Unit-Mapping-Complex class denotes a mapping between units that must be expressed as a function of the source unit. It has a template slot relating-expression. The value of this slot is an instance of a Numeric-Expression. The expression must involve an instance of class Instance-Variable that refers to a Unit-of-Measurement instance. Analogous to the Measurable-Things ontology, the Units ontology extends class Instance-Variable with a subclass Unit-Instance-Variable, and the instance variable contained in the Numeric-Expression should be instances of this class. As with the Measurable-Things ontology, this constraint is not enforced.

   The Unit-Mapping-Complex class is used to specify a mapping from degrees Celsius to degrees Fahrenheit. The formula used is:

```
(Sum
    (Product
        (Division
            (Integer-Literal 9)
            (Integer-Literal 5))
        (Unit-Instance-Variable :variable-ref [degrees Celsius]))
    (Integer-Literal 32))
```

7.2.2. Slots of Unit-Mapping-Many-to-One

The Unit-Mapping-Many-to-One class models mappings from a set of units to a composite unit of measurement. The class has the following template slots:

1. The to-unit slot is overridden such that its value must be an instance of Composite-Unit rather than Unit-of-Measurement.
2. The from-units slot models the units of measurement that form the source of this mapping instance. It is a multi-cardinality slot whose values must be instances of Unit-of-Measurement. At least one value is required.
3. The many-to-many-mapping-expr slot is an instance of Numeric-Expression specifying a formula that describes the mapping. The formula should contain an instance of the Unit-Instance-Variable class for each value in the from-units slot, and the Unit-Instance-Variable instances should refer to these values in the variable-ref slots. The ontology does not enforce this requirement.

For example, the Units ontology specifies a mapping from kilometers and hours to kilometers per hour as follows:

unit-mapping-label	km/hr => kph
from-units	{ kilometer, hour }
to-unit	kilometers per hour (Instance of Composite-Unit)
many-to-one-mapping-expr	(Division (Unit-Instance-Variable :variable-ref [kilometer]) (Unit-Instance-Variable :variable-ref [hour]))

7.3. Instances of the Ontology

The Units ontology contains instances for those units of measurement commonly used in the GH data model to reflect physical properties. The GH data model uses the metric system, so the Units ontology contains meter, liter, kilometer, etc. The Units ontology contains English units as well.

The Units ontology contains mappings between common units. Examples include converting between metric length units (centimeter to millimeter, meter to centimeter) and conversions between English and metric units (inch to centimeter, square meter to square foot).

The GH data model attribute capability-unit-of-measure-code specifies a large set of units. The Units ontology should ultimately contain all those units.

7.4. Uses of the Ontology

The Types ontology makes extensive use of the Unit-of-Measurement class to allow a type to be attached to a unit. In the GH ontology, each slot that models a physical property is of a type that has an attached Unit-of-Measurement instance. The means by which this is accomplished are discussed in the next section.

The Unit-of-Measurement-Mapping class is not currently used. It was developed to support reasoning capabilities that so far have not been needed in the sample applications that use the ontology.

The Units ontology contains a PAL constraint that enforces uniqueness of type names. It is up to applications, or to users, to ensure that this constraint is satisfied; Protégé-2000 will no enforce it automatically.

The Units ontology is a building block, and therefore it is the responsibility of any ontology that uses it to establish a proper framework for interpreting units of measurement. One aspect of this framework is adherence to the criteria established by the unit-range-min and unit-range-max slots so that they are satisfied for any instance declared to be of a particular unit. This particular case is discussed for the Types ontology in Section 8. Protégé-2000 will not enforce adherence automatically, but an ontology can specify a contract between itself and an application or user, adherence to which will guarantee constraint satisfaction.

8. The Types Ontology

The Types ontology models data types – domains of values. These values may share elements, but they are distinguishable. The Types ontology is a building block that allows another ontology to distinguish domains of slots beyond the few primitive types offered by Protégé-2000 and other knowledge base development tools. This distinction is important in the GH data model, where (for example) string-valued attributes have a specified maximum length. Protégé-2000 does not allow maximum string length to be specified.

Many GH data model attributes are represented as a six-character string denoting a code. Protégé-2000 allows the values for a slot to come from a prespecified set of strings, which could constrain possible values to the desired set. However, it doesn't describe the different meaning a code can have. The value 'AT' can mean, in different contexts, the Ashmore and Cartier Islands, an anti-tank weapon, and specified time.

In an ontology, it is important to have as much information as possible about the possible value for a slot. The Types ontology works towards that end.

The Types ontology also assigns meaning to a slot. At the most primitive level, meaning is assigned in terms of a type's name and its relationship to other types in the type hierarchy (see Figure 6). More complex meaning comes from use of the ontologies discussed in the previous sections. How this is achieved will be discussed shortly.

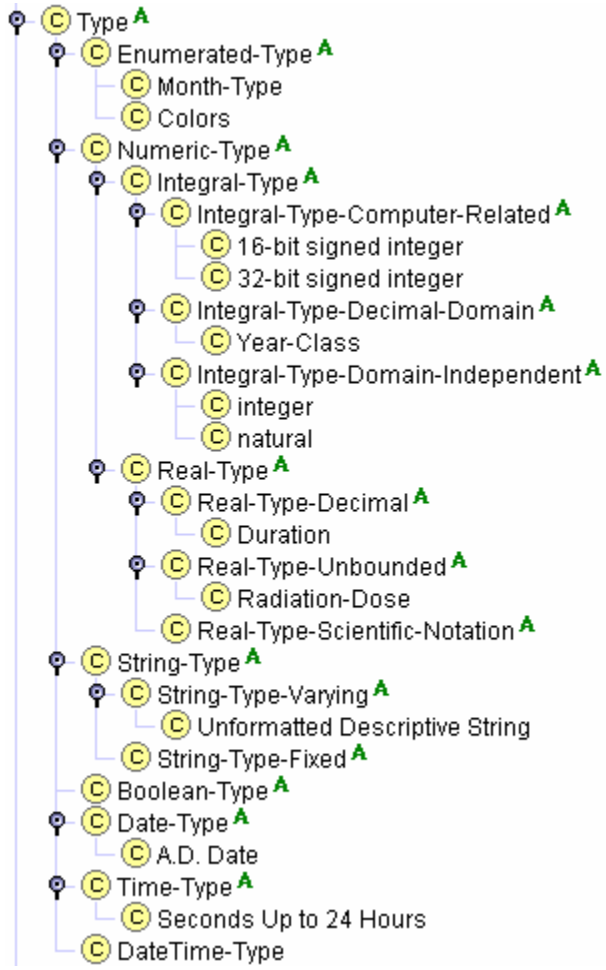

Figure 6. Classes in the Types Ontology: Type Hierarchy

The Types ontology includes a basic hierarchy of types categorized according to representation. The intent is to assist users in placing and locating types. Thus Figure 6 shows, as subclasses of Type, classes for representing enumerations, numeric types, string types, Boolean types, date times, time types, and date/time types. These classes are further subdivided into according to representation: integers vs. real numbers, fixed vs. varying length strings, etc. Only leaves of the hierarchy are concrete; the interior classes are all abstract.

An ontology defines a type not by creating an instance but by creating a subclass somewhere in the Type hierarchy. Figure 6 contains some illustrative classes, such as 16-bit signed integer, Duration, and Seconds Up to 24 Hours. The meaning of these classes should be clear enough to a reader based on their name, but their meaning to an application is less clear. Section 8.2 will discuss how additional properties are associated with each of these classes.

8.1. Slots of the Type Class

The Type class contains a single template slot named type-instance-value. Its meaning and use require some explanation. An ontology uses the Types ontology to model the types of organic

slots. Each organic slot's value should be declared to be an instance of some subclass of Type. Organic slots therefore do not use Protégé-2000's built-in domains directly. Instead, they refer indirectly to these domains. The value of slot type-instance-value for an instance is an appropriate domain, such as a Protégé-2000 string.

Figure 7 shows an example. The GH ontology has a class Object-Item with two template slots: name and alternate-identification-text. A value for the name slot must be an instance of Object-Item-Name, which the GH ontology declares as a subclass of String-Type-Varying. In class String-Type-Varying the value type of slot type-instance-value is overridden to be a string. Creating a value for the name slot entails creating an instance *o* of Object-Item-Name, assigning the string denoting the object item's name to the type-instance-value slot of *o*, then assigning *o* to be the value of the name slot.

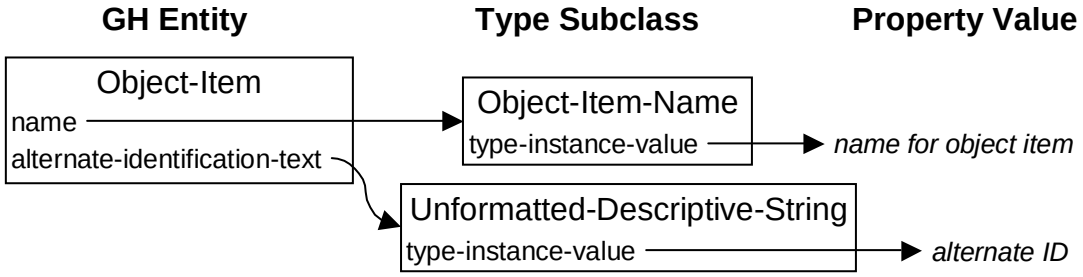

Figure 7. Representing Organic Slot Values

Creating a value for the alternate-identification-text slot is similar, except:

- The string denoting the alternate identifier is assigned to the type-instance-value slot of an instance of class Unformatted-Descriptive-String rather than Object-Item-Name.
- The GH ontology specifies that the value type of the alternate-identification-text slot must be an instance of Unformatted-Descriptive-String rather than Object-Item-Name.

Thus it is not possible to assign an alternate identifier to a name. In this way the Types ontology establishes the means by which another ontology can specify strict typing. (However, Protégé-2000 doesn't enforce strict typing; the knowledge base user has that responsibility. The Types ontology simply provides enough information to make strict typing possible.)

Sometimes the domain of type-instance-value is another class defined as part of the GH ontology hierarchy, such as Numeric-Expression. In that case the value of the type-instance-value slot will be an instance of Numeric-Expression. The intent of allowing numeric expressions is to facilitate recording exact values, and to show their derivation: If one slot has the value $a+b$ and another has the value $1/(a+b)$, the fact that a knowledge base uses the same instance of Numeric-Expression to represent $a+b$ may explain some similarity. Of course, a Literal is a Numeric-Expression, and in practice most slot values will be instances of Literal.

Subclasses of Type have no slots other than type-instance-value. Additional properties are described using a metaclass.

8.2. Metaclasses in the Types Ontology

A subclass of Type contains no information about a type other than what can be gleaned from its name and relative location in the class hierarchy. This information is useful to humans but provides no real information that applications or other ontologies might use for reasoning.

To let this information be added, the Types ontology declares metaclasses. In Protégé-2000, every class has a metaclass. The slots associated with a metaclass are exactly those slots an ontology developer uses to characterize a class. The default metaclass is class :STANDARD-CLASS, a subclass of :CLASS, and its slots are the default properties for a class: name, template slots, concrete or abstract, etc. Any subclass of class :CLASS is a valid meta-class. The usual approach to adding metaclasses to an ontology is to extend :STANDARD-CLASS, as Protégé-2000 is programmed to understand the slots associated with :STANDARD-CLASS. To declare meta-classes that are descendants of :CLASS but not of :STANDARD-CLASS would necessitate the time-consuming task of writing a plugin.

Figure 8 shows the metaclasses declared by the Types ontology in support of expressing type-related information. The metaclasses are rooted by Type-Class, a subtype of :STANDARD-CLASS. Descended from Type-Class are classes that characterize different kinds of data types. The set of classes is based on needs of the GH ontology. Other ontologies may need to add other classes.

The descendents of Type-Class are hierarchically organized based on characteristics of a domain. Four characteristics are recognized:

1. Textual domains, in which the elements are strings. Values of these domains tend to be descriptive rather than analytic.
2. Enumerable domains, in which there exist a finite set of elements. This does not exclude domains that may be enumerated using ordinal numbers.
3. Measurable domains, in which a value directly describes a physical or conceptual property.
4. Numerically describable domains, in which a value describes a numerical amount of some entity.

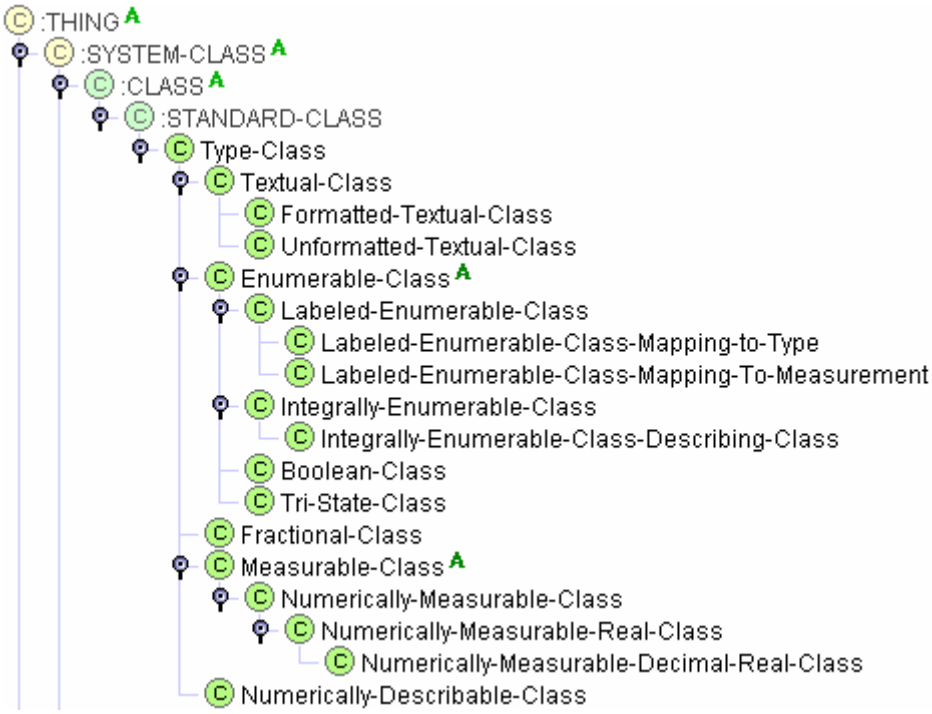

Figure 8. Metaclasses for Specifying Type Information

These characteristics are not mutually exclusive. Some enumerable domains are also measurable, and vice versa. The slots of a metaclass help an ontology developer decide whether it is most appropriate.

The metaclass Type-Class has no template slots. It exists to distinguish type-related metaclasses from other metaclasses an ontology may declare. Moreover, it is not an abstract class. It is a useful placeholder for domains that an ontology does not otherwise characterize.

8.2.1. Metaclasses for Textual Domains

An ontology developer declares a type (i.e., a subclass of Type) to have a metaclass that is a subclass of Textual-Class when the type is meant to model strings. Textual-Class has two template slots:

1. The textual-class-varying slot is a Boolean-valued slot that specifies if the domain is to consist of varying-length or fixed-length strings.
2. The textual-class-string-length slot declares the maximum permitted length for any string in the domain. If the textual-class-varying slot is false, then the maximum length is also the minimum length.

Class Textual-Class has two subclasses: Formatted-Textual-Class and Unformatted-Textual-Class. The former is used to model domains in which the values have a specified format. This format is described by the template slot formatted-textual-class-format-description. The Types ontology does not assign any meaning to the content of this slot; applications, users, and other ontologies are free to interpret it. Possibilities include regular expressions and hyperlinks to external standard definitions (the latter is more logical in a net-centric environment, if more expensive to implement).

Class Unformatted-Textual-Class is used for string-valued domains that have no format constraints.

The metaclass does not specify a character set. This extension would be useful in net-centric environments and should be added to a future version of the Types ontology.

8.2.2. Metaclasses for Enumerable Domains

An ontology developer declares a type to have a metaclass that is a subclass of Enumerable-Class when the type models a fixed set of values. These values need not have a string representation. Any enumerable set – integers, logical values, or even real numbers – will do in theory. The Types ontology does not currently support enumerations of real numbers.

The Enumerable-Class metaclass has no template slots. It is the root of a hierarchy of enumerable class kinds.

8.2.2.1. Labeled Enumerable Classes

The first kind of enumeration supported by the Types ontology is a labeled enumerable class. In this class, each element of the enumeration has a label that is not presumed to have any meaning when compared to other elements. This is in contrast to integrally enumerable classes, where the elements have an inherent order.

Labeled-Enumerable-Class has two template slots. The first, which is required to have at least one value, is named labeled-enumerable-class-elements. Its values are instances of a subclass of Enumerated-Element, which is a subclass of :THING. Section 8.4 covers the Enumerated-Element class in detail. For now, it is sufficient to know that the class has two template slots. The first denotes the six-character label of the element. The second models the descriptive text that

accompanies each GH code. All values of labeled-enumerable-class-elements must be instances of the same class.

The second template slot of Labeled-Enumerable-Class is named labeled-enumerable-class-element-ordering. This slot allows information on ordering among elements to be given, should such information make sense for an enumeration. An ordering is stated as a set of Order-Specification instances (see Section 6). The own slots of an Order-Specification instance specify two elements; these elements must be instances of the elements given by the labeled-enumerable-class-elements slot.

The GH ontology has many labeled enumerable classes (that is, subclasses of Enumerated-Type whose metaclass is Labeled-Enumerable-Class). These enumerations tend to be those about which little can be inferred, except perhaps ordering information. More interesting are the two subclasses of Labeled-Enumerable-Class:

1. Class Labeled-Enumerable-Class-Mapping-To-Type serves as a metaclass for those enumerations in which the instances describe some subclass of Type. It has a template slot labeled-mapping-to-type that records this class. For example, the GH attribute capability-day-night-code has values that describe a period of a 24 hour day: DAY (daytime), N (nighttime), and DN (daytime or nighttime). Because daytime varies with each day of the year, these values do not correspond to any precise time range until placed in context. However, one can still note that the code corresponds to some time range. Thus the corresponding class in the GH ontology, DS181_capab_day_night_code, uses Labeled-Enumerable-Class-Mapping-to-Type as its metaclass, and has class DateTime-Type (see Section 8.3.7) as the value of its labeled-mapping-to-type own slot.
2. Class Labeled-Enumerable-Class-Mapping-To-Measurement is the metaclass for enumerations in which instances are codes denoting some measurement. The class has two template slots: labeled-enumerable-class-mtm-measurable-thing and labeled-enumerable-class-mtm-units-of-measurement. These describe, respectively, the thing that each instance measures, and the units in which it is measured. (The details of the measurement are a property of an enumerated element instance, and are specified in the instance of Enumerated-Element.) For example, the GH attribute organisation-status-radiation-dose-code has values 1, 2, and 3, each of which gives a range of radiation exposure measured in centigrays. Thus the DS394_rad_dose_code class in the GH ontology uses Labeled-Enumerable-Class-Mapping-To-Measurement as its metaclass, with Radiation Dose as the measurable thing, and centigray as the unit of measurement.

   As another example, the GH data model attribute action-task-start-precision-code has values DAY (day), HR (hour), and MINUTE (minute) that denote the precision of measurement for determining when a task will commence. These values correspond to the measurable thing "time", of which an instance is defined in the Measurable-Things ontology. Thus the DS284_timing_precision_code class in the GH ontology uses Labeled-Enumerable-Class-Mapping-To-Measurement as its metaclass, with Time as the measurable thing. The slot describing units of measurement is left undefined, because for this enumeration each value is a distinct unit of measurement.

8.2.2.2. Integrally Enumerable Classes

An integrally enumerable class is one in which the enumerated values are integers rather than character strings. A class that uses Integrally-Enumerable-Class as its metaclass has values that are naturally countable, as opposed to using integers as arbitrary labels. For this reason, DS394_rad_dose_code is *not* an integrally enumerable class, but values such as quantity of beds, candidate target priority, and line point sequence are. Note too that these values are not what one would normally think of as measurements, though this is arguable.

Integrally-Enumerable-Class has two template slots: minimum and maximum. These slots prescribe minimum and maximum values for the domain that an instance of Integrally-Enumerable-Class models.

The Types ontology declares five primitive types as integrally enumerable:

1. Class integer, which denotes an integer. The class has no predefined minimum or maximum.
2. Class natural, which denotes the natural integers. The class has a minimum of 0, and no predefined maximum.
3. Class 16-bit signed integer, which has a minimum of -32,768 and a maximum of 32,767.
4. Class 32-bit signed integer, which has a minimum of -2,147,483,648 and a maximum of 2,147,483,647.
5. Class A.D. Date, which is intended to model dates between 1 January 1 and 31 December 9999. The class has no predefined minimum or maximum, however. (In other words, its definition is incomplete.)

Despite sharing the same metaclass, these classes are distributed throughout the class hierarchy rooted at Type (see Figure 6).

Integrally-Enumerable-Class has a subclass named Integrally-Enumerable-Class-Describing-Class. It serves as the metaclass of classes that enumerate some class in the ontology. It has a single template slot named integrally-enumerable-class-referenced-class whose value type is a class. For example, the GH data model attribute holding-quantity is modeled in the GH ontology as a slot whose domain is the class Holding-Quantity. Holding-Quantity, a subtype of Type, uses Integrally-Enumerable-Class-Describing-Class as its metaclass. The own slot value of integrally-enumerable-class-referenced-class for the Holding-Quantity class is the Object-Type class. For an associative class such as Holding, the Integrally-Enumerable-Class-Describing-Class metaclass identifies the class to which a slot refers.

8.2.2.3. Boolean Classes

A Boolean class is one with two values, one of which can be mapped to truth and the other to falsehood. Boolean-Class serves as the metaclass for these classes. It has two template slots: bool-type-true-literal and bool-type-false-literal, which denote the literal values for truth and falsehood, respectively, for a domain. The metaclass is motivated by the many GH coded domains that use YES and NO instead of true and false.

8.2.2.4. Tri-State Classes

Tri-State logic has three values: truth, falsehood, and indeterminacy. Tri-State-Class serves as the metaclass for type classes that model tri-state logic domains. It has three template slots:

bool-type-true-literal, bool-type-false-literal, and tri-state-type-unknown-literal. The metaclass is motivated by the GH data model coded domains that add an "unknown" code to a domain expressing truth and falsehood.

8.2.3. Metaclasses for Fractional Domains

Fractional-Class serves as a metaclass for types that express a unitless fraction. The GH has attributes that express percent completion; Fractional-Class is the metaclass for the types of the slots modeling those attributes.

Fractional-Class has three template slots. The precision and digits slots, which are required, express the significant digits in the fraction as (decimal) digits following the decimal point and total digits, respectively. At least one digit is required, and the number of digits must equal or exceed the precision. Fractions need not be between 0 and 1.

The third template slot, fractional-class-referenced-class, expresses the class of which a fraction is being stated. This slot is optional.

Fractional-Class is used to express concepts that are not measurable things. Sometimes this merely means no obvious way has been found to measure something. For instance, the GH data model attribute action-task-status-completion-fraction is modeled as slot completionFraction, the type of which is Task-Completion-Fraction; the metaclass of Task-Completion-Fraction is Fractional-Class, and the own slot value of fractional-class-referenced-class is Action. The intent is to state a completion fraction of an Action. An Action is not a measurable thing, so completionFraction is not modeled as a measurement. Arguably, "percent completion" is a measurable thing, and should be expressed as such. But just what it measures is complex. Fractional-Class provides a convenient fallback.

8.2.4. Metaclasses for Measurable Domains

Measurable-Class is the root of a hierarchy for describing types of measurable things (see Figure 9). Measurable-Class has two template slots. Slot measurable-class-measurable-thing denotes the thing being measured by a type whose metaclass is Measurable-Class; its value type is an instance of Measurable-Thing. Slot measurable-class-unit-of-measurement denotes the unit of measurement for the measurable thing. In other words, a type that uses Measurable-Class as its metaclass explicitly binds itself to a measurable thing and a unit of measurement.

Measurable-Class is abstract; its subclasses are concrete. It has one direct subclass of Measurable-Class, named Numerically-Measurable-Class. It is intended for modeling types where measurements are expressed as numbers. This encompasses all measurements in the GH. Numerically-Measurable-Class has two template slots: minimum and maximum. These slots, whose value types are instances of Numeric-Expression, state the minimum and maximum values allowed for a measurement.

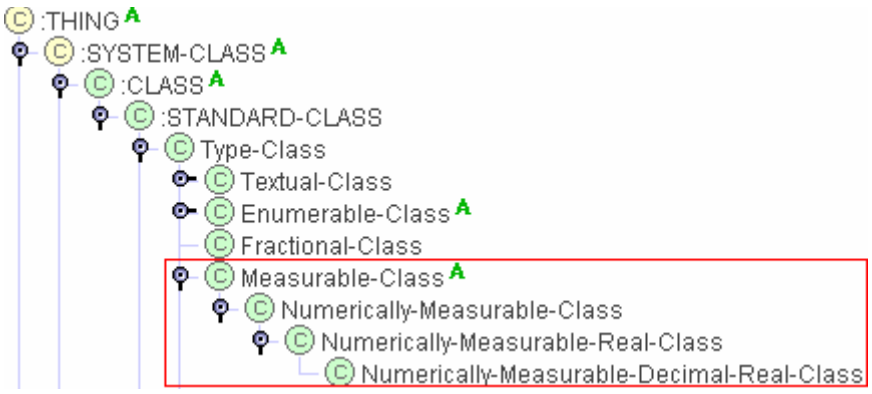

Figure 9. Metaclass Hierarchy for Measurable Type Classes

(Measurable-Class has no other direct subclasses. The Types ontology was designed to allow for non-numeric measurements, but no need for them has arisen.)

Numerically-Measurable-Class has a subclass named Numerically-Measurable-Real-Class. Its purpose is to model types that state measurements as real numbers. It has two template slots: min-inclusive and max-inclusive. These slots are Boolean valued, and describe whether the values of the minimum and maximum slots, respectively, are to be included in the allowed values for a measurement – that is, whether the range is closed or open.

The Numerically-Measurable-Decimal-Real-Class extends Numerically-Measurable-Class with template slots digits and precision that, as for fractional classes (see Section 8.2.3), add information on significant digits. This metaclass is the one most commonly used in the GH ontology to model GH attributes that represent physical or conceptual properties.

Numerically-Measurable-Real-Class is sometimes used for types that model coded domains mapping to measurements. These coded domains do not specify required numerical accuracy and so cannot benefit from the additional slots of Numerically-Measurable-Decimal-Real-Class. An example is the type DS362_nbc_event_spl_siz_code, which models the size of a spill of a nuclear, biological, or chemical (NBC) weapon. The codes of this value express some range of liters (e.g., LARGE corresponds to a spill of between 208 to 1,500 liters, not inclusive). The measurable thing (liquid volume) is known, and its units (liters) are known, but the precision isn't.

8.2.5. Metaclasses for Numerically Describable Domains

The Numerically-Describable-Class metaclass is used to characterize types for which instances describe a numerical amount of some class in the ontology. The numerical amount refers to a quantity rather than to a measurable thing. This metaclass is useful for an amount that is not enumerable and therefore cannot be characterized as an Enumerable-Class. An example from the GH data model is the attribute object-item-capability-quantity.

Numerically-Describable-Class has three template slots. The digits and precision slots specify the significant digits used to state a quantity (see Section 8.2.3). The numerically-describable-class-referenced-class slot models the class of which a type is expressing some amount. For the GH ontology class Capability-Amount, the own slot value of numerically-describable-class-referenced-class is class Capability, that is, an instance of Capability-Amount describes a Capability.

8.3. Subclasses of the Type Class

The class hierarchy of types shown in Figure 6 provides a basic categorization of types an ontology might use. This categorization is based on representation of domain literals. Aside from a few standard types that are expected to be widely applicable and globally useful, the subclasses of Type are abstract. An ontology extends these subclasses to add additional types in support of relevant domains.

The following subsections discuss subclasses in the Type hierarchy. As mentioned on page 15, these classes have no template slots except for the type-instance-value slot defined in class Type. Subclasses generally override the value type of the type-instance-value slot; this is discussed in each subsection.

8.3.1. Enumerated-Type

Class Enumerated-Type is the root of classes whose members are a finite enumeration of string literals. In the GH ontology, Enumerated-Type is the root of all types that represent GH's coded domains.

The value type of slot type-instance-value is overridden to be a set of instances of Enumerated-Element. A subclass of Enumerated-Type should further override the value type to the subclass of Enumerated-Element that is relevant for the enumeration.

The Types ontology provides one expository concrete enumerated type, defined by class Month-Type. This type is an enumeration of months of the year. The value type of its type-instance-value slot is Enumerated-Element-Mapped-Onto-Numeric-Range (see Section 8.4.2).

The metaclass of Month-Type is Labeled-Enumerable-Class-Mapping-To-Type (see Section 8.2.2.1). The class is mapped to class Date-Type, implying that any instance of the class must be interpreted as a date. The class has twelve valid elements. Instances of Order-Specification establish the expected total order among them.

8.3.2. Numeric-Type

The Numeric-Type class is the root of all classes that represent a numeric type (see Figure 10). The type-instance-value slot of the class has as its value type a set of instances of Numeric-Expression. In other words, evaluating the slot yields a numeric value. This value may involve variables. The knowledge base provides a context in which to evaluate a numeric expression, and that context must supply values for all variables in an expression.

The metaclass of Numeric-Type is Type-Class. This conveys no information except that Numeric-Type denotes a type.

Numeric types are further categorized as being integer or real. It is expected that a Numeric-Expression instance associated with the type-instance-value slot of an Integral-Type will evaluate to an integer, and that a Numeric-Expression instance associated with the type-instance-value slot of a Real-Type will evaluate to a real. Protégé-2000 does not provide for determining the type of a Numeric-Expression instance, however.

The Integral-Type and Real-Type classes are further specialized by abstract classes that categorize types according to the origins of a numeric domain. Integral types are divided into:

1. Computer-related, i.e., those that model integers as represented on computers. The Types ontology includes two concrete subclasses of Integral-Type-Computer-Related: 16-bit signed integer and 32-bit signed integer. Both of these classes have Integrally-Enumerable-Class as their metaclass. This allows them to establish minima and

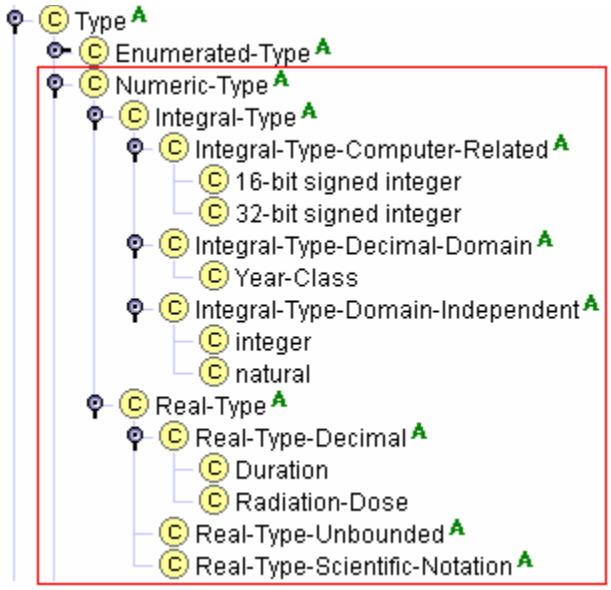

Figure 10. Classes for Numeric Types

maxima. The values, instances of Integer-Literal, are the expected bounds for 16- and 32-bit signed values.

2. Decimal domain, in which constraints on minima and maxima are expressed as powers of ten. (This property is characteristic of many GH data model domains, though on reflection the need for such grouping is not strong.) The Types ontology offers the concrete class Year-Class, which models A.D. years. The metaclass of Year-Class is Numerically-Measurable-Class. Its measurable thing is Time, its unit of measurement is year, and its minimum value is 1 (an instance of Integer-Literal). Because Protégé-2000 provides no built-in capability to evaluate numeric expressions, the constraint that the minimum or maximum value is a power of ten is not enforced.
3. Domain-independent, which is for integral domains drawn from mathematics. The Types ontology provides concrete classes integer and natural. Both use Integrally-Enumerable-Class as their metaclass. The integer class has neither a minimum nor a maximum. The natural class has a minimum of 0 (instance of Integer-Literal) and no maximum.

Real types are divided into the following categories:

1. Decimal types, in which properties of the type are expressed as a (digits, precision) pair.

   The metaclass of Real-Type-Decimal is Type-Class rather than one of its subclass with digits and precision template slots, because metaclasses with these slots are scattered throughout the Type-Class hierarchy. A subclass of Real-Type-Decimal must have a metaclass with digits and precision template slots, and the number of digits must equal or exceed the precision.

   The Types ontology specifies one concrete subclass of Real-Type-Decimal: Duration. Duration is a Numerically-Measurable-Decimal-Real-Class. It measures Time, though its unit of measurement is not specified. It is given 13 digits, three of which follow the decimal point. This happens to be convenient for modeling GH data model attributes that describe duration.

2. Unbounded types, in which no constraints are imposed on the digits or precision. The Types ontology provides one expository class, Radition-Dose, that is a subclass of Real-Type-Unbounded. It is a Numerically-Measurable-Class. It measures Radiation Dose in centigrays. Its minimum is 0.0 (a Real-Literal).

   Section 8.2.2.2 explained that the GH data model describes a radiation dose as a labeled enumeration, where each label maps to a range. Ratiation-Dose is the type underlying that range.
3. Numbers expressed in scientific notation, where digits and exponent magnitude are specified. Neither the Types ontology nor the GH ontology contains any types that require scientific notation. It is included for expository completeness. If a subclass of Real-Type-Scientific-Notation were to be created, a corresponding metaclass would need to be added that captures the parameters of the type.

8.3.3. String-Type

The String-Type class is the root of all classes that describe types whose contents are textual. Figure 11 shows the hierarchy of string types in the Types ontology.

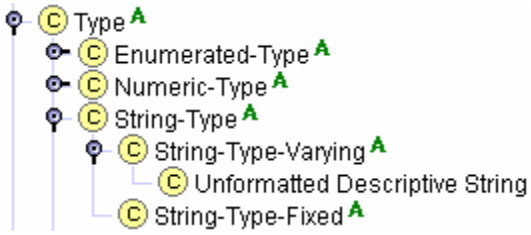

Figure 11. Classes for String Types

All types have a textual representation, but subclasses of String-Type are lexicographically ordered. The Types ontology does not allow specification of the details of this ordering but probably should (e.g., do spaces count?). It also does not, and probably should, allow specification of the character set. The GH uses ASCII, and the Types ontology, being developed to support the GH, assumes that degenerate case.

The metaclass of String-Type is Textual-Class. Any subclass of String-Type must have a metaclass that is a subclass of Textual-Class.

The value type of the type-instance-value slot of String-Type is overridden to specify that it must contain a string. Protégé-2000 does not place any constraints on strings other than those imposed by the underlying Java implementation. The textual-class-string-length slot of the metaclass imposes a length constraint.

There are two categories of string types: varying length and fixed length. Any subclass of String-Type-Varying must set the textual-class-varying slot to true. Any subclass of String-Type-Fixed must set the textual-class-varying slot to false.

The Types ontology includes the expository concrete class Unformatted Descriptive String. This class, whose metaclass is Unformatted-Textual-Class, specifies a type that represents varying length strings of up to 255 characters. Instances of the type are not presumed to have any inherent semantics.

8.3.4. Boolean-Type

Class Boolean-Type is the superclass of all Boolean types that do not express any recognizable meaning beyond truth and falsehood. That is, for such a domain, truth or falsehood has no

implications for other instances in a knowledge base. The type-instance-value slot's value type is overridden to be an instance of class Boolean-Expression.

8.3.5. Date-Type

Class Date-Type is the superclass of all classes that represent dates. A date represents only month, day, and year (or a subset thereof) information – no time information is present.

The type-instance-value slot's value type is overridden to be a string. This representation is used because Protégé-2000 does not support date/time types. An application may need to parse the string to use it for inferencing operations.

The Types ontology contains the expository class A.D. Date, which represents a type that models a month/day/year combination. See Section 8.2.2.2 for details on Integrally-Enumerable-Class, its metaclass.

8.3.6. Time-Type

Class Time-type is the superclass of all classes that represent times. As with class Date-Type, the type-instance-value slot's value type is overridden to be a string. Subclasses are free to further override its value type to a numeric value, or instance of Numeric-Expression, but strings permit forms such as "00:10:20", which may prove useful in certain circumstances. The Types ontology opts for maximum flexibility.

The Types ontology contains the expository class Seconds Up to 24 Hours, which represents a time of day. Its metaclass is Numerically-Measurble-Class; it measures Time, using unit of measurement second. It has a minimum of 0 and a maximum of 86,399; these values are instances of Integer-Literal. The implication is that instances of this class measure time precise to one second, not fractions thereof. However, the class does not explicitly constrain instances to integral values.

8.3.7. DateTime-Type

Class DateTime-Type is the superclass of all classes that represent moments in time requiring more precision than a single day. As with class Date-Type, the type-instance-value slot's value type is overridden to be a string. As with class Time-Type, subclasses are free to override this value to any type that makes more sense in a given context.

Class DateTime-Type is concrete. Its metaclass is Numerically-Measurable-Class. Its measurable thing is Time.

8.4. Enumerated Elements

The Enumerated-Element class is the root of a class hierarchy wherein each class models the elements of an enumerated domain. Except for two expository classes, Enumerated-Element and all its subclasses in the Types ontology are abstract. An ontology that uses Types is expected to add, in the appropriate spot in the hierarchy, a subclass of Enumerated-Element that denotes an individual domain.

Once an ontology designer has created a subclass and its instances, the instances are by convention regarded as complete. No other ontology, knowledge base, application, or user should create duplicate instances of the class. Instead, references to the existing instances should be made. This convention is necessitated by Protégé-2000's approach to implementing frames. In particular, comparisons for equality based on object references will be false for two distinct instances even if those instances share the same label. It is therefore necessary for a single set of instances to exist for each Enumerated-Element subclass.

The Enumerated-Element class hierarchy is organized according to how a domain of enumerated elements can be mapped to elements of another domain. If the elements cannot be mapped, the class representing the domain should be a direct subclass of Enumerated-Element.

The Enumerated-Element class has two template slots:

1. The enumerated-element-label slot models the label associated with an enumerated element. It is a string, and is required.
2. The enumerated-element-description slot models a textual description associated with an enumerated element. It is a string, and is optional.

These two slots model essential information of the GH data model. Other ontologies may want to add template slots to model such items as the source of an element or a label to be used when displaying the element to a user.

The following are the steps an ontology designer takes when modeling an enumeration:

1. Create a subclass *s* of Enumerated-Type. It may be worth noting that, in the GH ontology, all subclasses of Enumerated-Type are direct subclasses. However, an ontology designer is free to further categorize enumerations.
2. Choose a subclass of Enumerable-Class as the metaclass of *s*. If the only concepts associated with the enumeration are the elements and perhaps and ordering, the subclass will be Labeled-Enumerable-Class.
3. Create a subclass *e* (direct or indirect) of Enumerated-Element. By convention, the name of *e* ends with "-Elements". This distinguishes it from the name of the type to which it corresponds.
4. Create an instance of *e* for each element of the enumeration, supplying the label and, optionally, a description. If *e* is an indirect subclass of Enumerated-Element, add appropriate mapping information to slots of the instance.
5. Override the type-instance-value slot of *s* such that the value type is "instance of *e*".
6. If the metaclass of *s* is Labeled-Enumerable-Class or a subclass thereof:
    1. Add to the labeled-enumerable-class-elements slot of *s* all instances of *e* created in step 4.
    2. If the elements have a partial or total order, create instances of Order-Specification that describe orderings and add them to the labeled-enumerable-class-elements-ordering slot.
    3. Add any mappings that might be specifiable via template slots of subclasses of Labeled-Enumerable-Class.

Figure 12 shows the organization of classes that represent elements of enumerations. The following subsections cover each subclass.

8.4.1. Class Enumerated-Element-Mapped-Onto-Enumerated-Element

This class models an enumeration in which each element denotes an element of another enumeration. It has a template slot enumerated-element-mapping-to-element. The value type of this slot is an instance of Enumerated-Element.

Class Enumerated-element-Mapped-Onto-Enumerated-Element was introduced into the Types ontology in the expectation that some GH data model coded domains might translate into others. It was, ultimately, not used, i.e., no subclasses were created during development of the GH ontology.

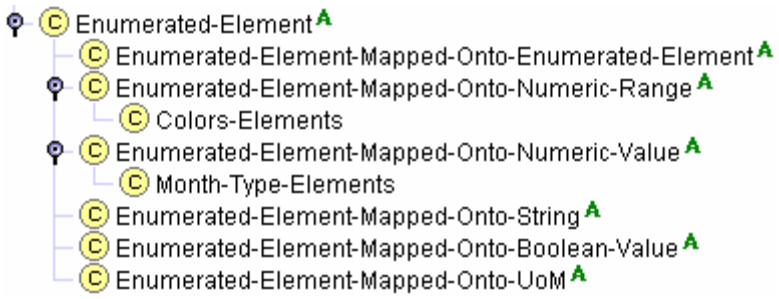

Figure 12. The Enumerated-Element Class Hierarchy

8.4.2. Class Enumerated-Element-Mapped-Onto-Numeric-Range

This class models an enumeration in which each element maps onto a continuous range of numeric values. It has four template slots. Slot enumerated-element-mapped-onto-numeric-range-lb, whose value type is an instance of Numeric-Expression, models the lower bound of the range. Slot enumerated-element-mapped-onto-numeric-range-lb-inclusive, a Boolean-valued slot, models whether the lower bound of the range is closed (true) or open (false). Slots enumerated-element-mapped-onto-numeric-range-ub and enumerated-element-mapped-onto-numeric-range-ub-inclusive play analogous roles for the upper bound.

Class Enumerated-Element-Mapped-Onto-Numeric-Range has one expository subclass in the Types ontology. Class Colors-Elements models an enumeration of colors (red, blue, green, etc.) with each color mapped to a frequency range in the electromagnetic spectrum. For example, the instance labeled "Red Spectral Range" has a lower bound of 450.0 and an upper bound of 495.0, roughly corresponding to the frequencies the human eye perceives as red.[3]

The Colors class declares the following ordering among its elements:

red < orange < yellow < green < blue < violet

This common view of colors is the opposite of the ordering imposed by wave-length range, as the red spectrum has a longer wave-length than the orange spectrum. Although some ordering can be inferred from mapped enumerated elements, it is not necessarily correct to assume that this ordering reflects a usual worldview. To avoid ambiguity, an ontology designer should explicitly specify an ordering.

8.4.3. Class Enumerated-Element-Mapped-Onto-Numeric-Value

This class models an enumeration in which each element maps to a single numeric value. It has a template slot named enumerated-element-numeric-value-mapping, the value type of which is an instance of class Numeric-Expression. This slot specifies the numeric value to which an instance maps. The interpretation of that value (the measurable thing it denotes, for instance) must come from the enumerated type of which the element is a component. The slot is not required to have a value: some enumerations have an "undefined" value.

Class Enumerated-Element-Mapped-Onto-Numeric-Value has one expository subclass in the Types ontology. Class Month-Type-Elements models an enumeration of the months of the year. It provides the elements of class Month-Type (see Section 8.3.5). Each element maps to an instance of Integer-Literal: the instance labeled January maps to 1, the instance labeled February maps to 2, etc.

---

[3] These values are approximations. Concrete classes in the Types ontology are expository and make no claims of accuracy.

8.4.4. Class Enumerated-Element-Mapped-Onto-String

This class models an enumeration in which each element maps to a single string value. It has a template slot enumerated-element-mapped-onto-string-value, the value type of which is a string. The slot specifies the string to which an instance maps.

This class was originally intended to model enumerations in which a label represents some other value. The GH ontology does not use it.

8.4.5. Class Enumerated-Element-Mapped-Onto-Boolean-Value

This class models an enumeration with two elements, one of which corresponds to truth and the other to falsehood. It has no template slots; it is used for categorization. The class of which the elements are instances should have Boolean-Class as its metaclass, and must specify which literal denotes truth and which falsehood.

8.4.6. Class Enumerated-Element-Mapped-Onto-UoM

This class models an enumeration in which each element maps to a unit of measurement. It is used to model GH data model attributes such as capability-unit-of-measure-code. It has a template slot enumerated-element-uom, the value type of which is an instance of class Unit-of-Measurement. The implication is that the instance denotes the specified unit.

8.5. Analysis of the Types Ontology

The Types ontology is designed to support strong typing, in the belief that ontology-based inferencing should take care to ensure that facts are compatible. Even if two GH data model attributes have the same representation, their values are not necessarily interchangeable or comparable. Strong typing lets an ontology distinguish between the domains of slots in cases where a distinction is desirable. But there is no hard and fast rule preventing comparison of domains.

Strong typing comes at a cost. The Types ontology adds a layer of instances in order to define the value of a slot. The approach has proven workable, but is arguably overkill.

This section presents some issues in the design of the Types ontology. It gives alternate design approaches that could have been used, and compares them to the decisions made in the Types ontology. It also points out weaknesses where no clear solution exists.

8.5.1. Symbols for Enumerations

If “Symbol” is specified as a possible value type of a Protégé-2000 slot, then the application designer can specify a set of symbols (each given as a string), and the slot is restricted to these strings.

“Symbol” could be used as the value type for an enumerated type (i.e., subclass of Enumerated-Type). Symbols for class Month-Type would include January, February, etc. This is simpler than defining a corresponding subclass of Enumerated-Element.

Using symbols has the following drawbacks:

1. No description is associated with a symbol. Adding a description would require creating a separate class hierarchy in which instances of labels can be related to instances of descriptions. That approach would necessitate something akin to the Enumerated-Element hierarchy anyway.
2. Symbols are not ordered. Although the Protégé-2000 user interface always lists symbols in the order in which the ontology designer entered them, the Java API method to

retrieve symbols returns an unordered collection. Protégé-2000's designers do not want users to depend on the order.

These are not fatal flaws that preclude symbols from being used. Rather, it is believed that ordering of labels is useful for a GH ontology, and that the current structure is clearer than mapping symbols to descriptions.

8.5.2. Inherent Semantics for Dates and Times

The Types ontology contains classes for dates and times. These classes draw upon measurable things and units of measurement defined in other ontologies.

For all that, the classes in the Types ontology still lack semantic meaning. The ontology cannot be used to make inferences about dates and times ("3 weeks from today is …").

This problem arises from Protégé-2000's lack of built-in support for dates and times. If dates and times were primitive types in the same way that integers and strings are, one could expect that Protégé-2000 knowledge bases would provide better support for reasoning about dates and times.

8.5.3. Specifying Type Using a Facet

The Types ontology is structured such that the value type of a slot is an instance of some subclass of class Type. Another approach would be to add to each slot a facet that specifies the type. To achieve this, the class :STANDARD-SLOT would be extended with a class that allows for specification of a metaclass as a standard facet of a slot (class Type-Specifier-Slot in Figure 13). Each slot whose metaclass is set to Type-Specifier-Slot can then specify a type to be associated with a slot.

Consider the GH data model entity OBJECT-ITEM-CAPABILITY, which has organic attributes capability-quantity (a number of twelve decimal digits with three digits after the decimal point) and object-item-capability-mission-primacy-code (a 6-character string modeling a coded domain with elements PRIME, SCNDRY, and THIRD). The GH ontology currently models the attributes as slots quantity and missionPrimacyCode. The value type of the quantity slot is an instance of class Capability-Amount; the value type of object-item-capability-mission-primacy-code is an instance of class DS327_msn_primacy_code. If the GH ontology used the approach in Figure 13, then the quantity slot could be specified as shown in Figure 14. Note that in this approach:

1. Slot value types are specified directly, rather than indirectly as instances of Type. In Figure 14, the value type is Float. Constraints on that type come from the Slot-Type facet.
2. Other standard facets, such as Minimum, can be used. (Note, however, that some metaclasses already include information on minima and maxima. Inclusion of this information for a slot would constitute an additional constraint.)

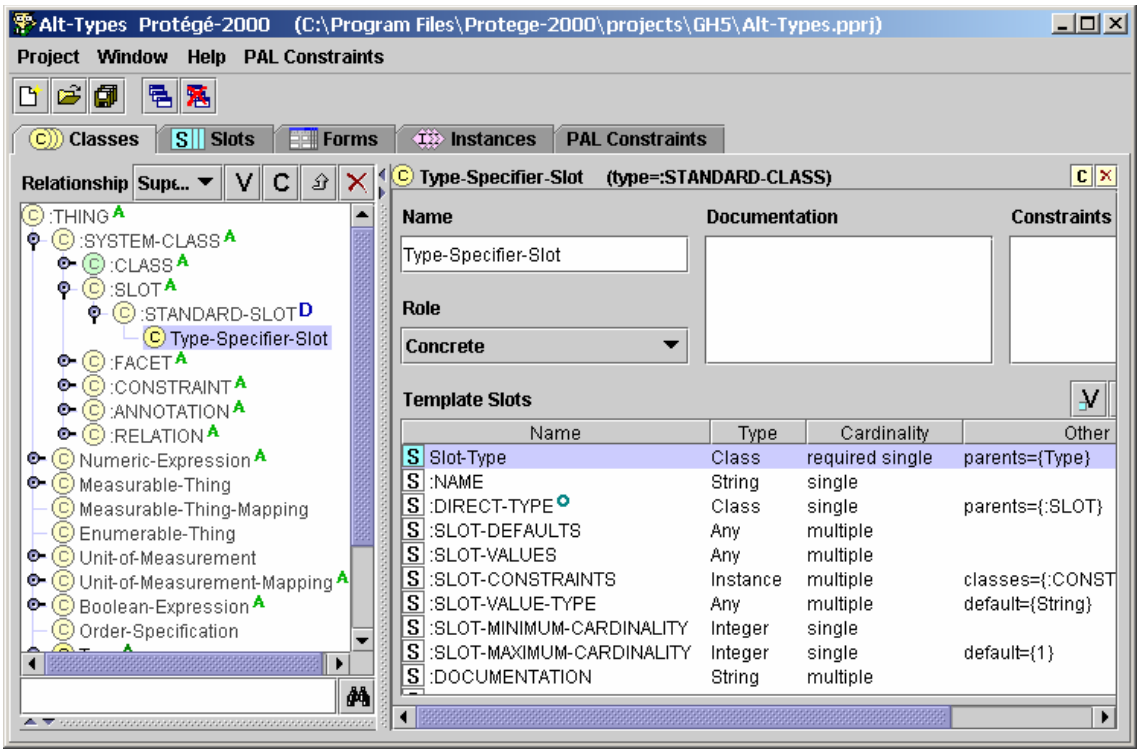

Figure 13. Specifying Types Using a Facet

This approach has the disadvantage of requiring more complex naming conventions than are currently used in the GH ontology. The reason derives from two Protégé-2000 restrictions:

1. Slot names are global, i.e., two slots in the same ontology cannot have the same name.
2. A slot can have one and only one metaclass.

If a class needed a slot to record quantity but wanted to record a quantity of something other than capability amount, it could not use the quantity slot. The naming conventions used by the GH data model, in which attribute names are prefixed with entity names, would suffice.

8.5.4. Use of the Numeric-Expression Class

The Types ontology encourages use of class Numeric-Expression to model instances of numeric values. Minima and maxima, for example, are modeled as numeric expressions rather than Protégé-2000 numeric literals.

This decision was made in the belief that many useful constants are best expressed as formulas rather than as literals because the use of a literal presumes, especially for a real number, a particular representation. Representing sin(45°) as 0.7070107 states in so many words that one is either assuming 8 significant digits or an unspecified number of digits with '7' as the first digit after the decimal point. The Numeric-Expression class frees an ontology designer from this sort of restriction.

The Numeric-Expression class has some drawbacks. Most notably, Protégé-2000 cannot evaluate an instance of a Numeric-Expression. This is of course an advantage too, as the point of Numeric-Expression is requiring the ontology designer to specify a proper context in which an instance should be evaluated. An application designer must interpret that context and write an evaluator. This is, fortunately, not a difficult chore.

Use of numeric expressions interferes with constraint writing. Numeric expressions that deal with mappings necessarily involve variables. Other numeric expressions, such as those specifying minima and maxima, should be prevented from having free variables. Writing a constraint to preclude free variables requires a recursive evaluation capability, which PAL lacks.

Use of numeric expressions requires some care. Two instances of Integer-Literal whose value slots are both 0 will not, when compared using Java's equals() method, be perceived as equal. The equals() method tests object equivalence, not conceptual equivalence – unless it is overridden, and Protégé-2000's implementation of instances doesn't override equality testing in this way. Thus evaluation of numeric expressions is necessary to test equality. Evaluation carries risks too, as round-off errors may arise: to a computer, $f(1)+\cdots+f(n)$ and $f(n)+\cdots+f(1)$ are not necessarily equal, after all.

It is likely that a production ontology would want to avoid use of formulas other than those with bound variables. Literals should be preferred.

9. The Generic Hub (Version 5) Ontology

The GH ontology models the entities defined in the Generic Hub, version 5. Every GH data model entity and attribute has a counterpart in the GH ontology.

Figure 15 gives an overview. It shows some of the classes in the ontology. These classes are subclasses of class GH-Entity. GH-Entity has both direct and indirect subclasses. A reader familiar with the Generic Hub will recognize variants of many Generic Hub elements.

The GH ontology strives to add meaning to GH data elements. Wherever possible, it defines conceptual equivalence among elements; lack of definition should be taken to imply lack of conceptual equivalence. This equivalence is based on concepts of the building block ontologies:

- Type equivalence.
- Identification of thing being modeled (measurable or enumerable).

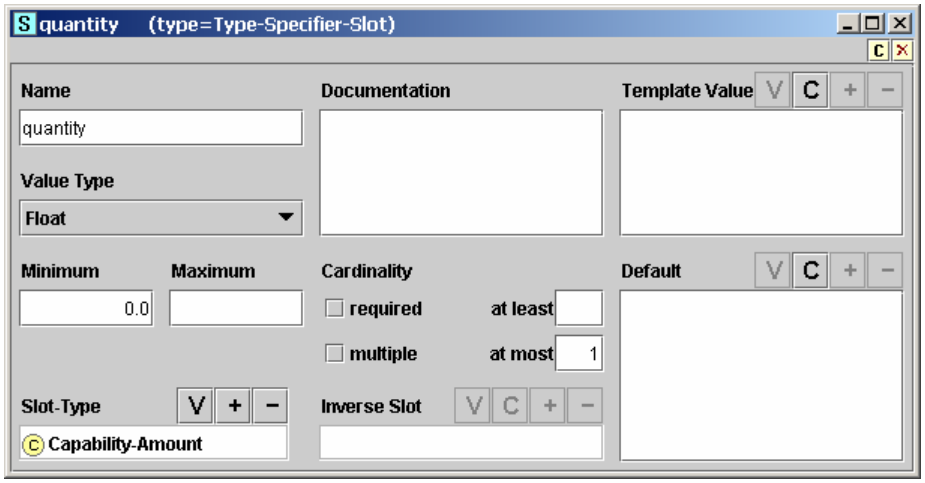

Figure 14. Specification of Quantity Slot Using Slot-Type Facet

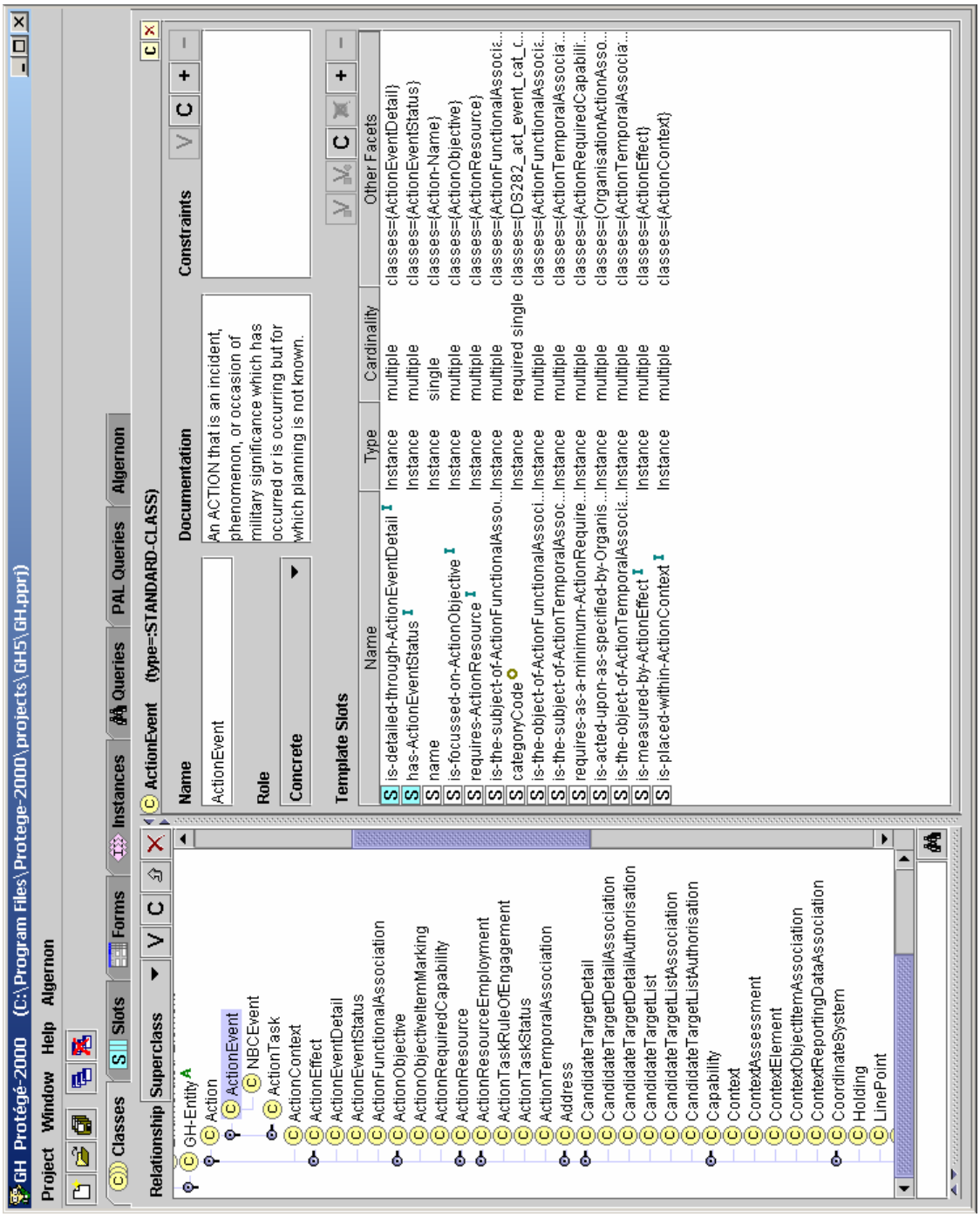

Figure 15. GH Ontology Classes (Partial)

- Identification of units of measurement.
- Mapping formulas.

These concepts provide the basis for building an inference engine that could be used to reason about data in a GH knowledge base.

The Generic Hub is defined using the IDEF1X model. Three basic rules were used to translate the Generic Hub to an OKBC-based ontology:

1. Each Generic Hub entity was translated to a class in the GH ontology.
2. Each organic attribute in the Generic Hub was translated to a slot in the GH ontology.
3. Each relationship in the Generic Hub was translated to a multiple cardinality slot in the GH ontology.

That the resulting ontology is derived from the Generic Hub is easily recognizable.

The design of an IDEF1X model uses many of the same considerations as the design of an object-oriented model. In particular, both consider how to encapsulate data elements – IDEF1X by entity, object-oriented by class. Some similarity between the Generic Hub and the GH ontology is to be expected.

Nevertheless, IDEF1X is not equivalent to a frame-based object-oriented model. The following discusses the details of translation. It presents strategies for adding expressiveness to the ontology. It also presents conflicts that arise due to the differences between the models.

9.1. Inheritance

An object-oriented model has inheritance. In Protégé-2000, a class inherits all template slots of its superclasses.

IDEF1X has subtypes. Subtypes are conceptually similar to inheritance. However:

- A subtype does not inherit attributes. Attributes of a supertype must be retrieved through a JOIN operation.
- Integrity constraints are needed in IDEF1X to ensure that deleting a supertype deletes any corresponding subtypes.
- A supertype must have a discriminator attribute. The discriminator is implicit in an object-oriented model, because a "type of" operator is available. If the discriminator is incomplete – that is, if there does not exist an entity for every possible value of the discriminator – then the "type of" operator is not sufficiently powerful.
- The Generic Hub does not specify whether it is legal to have an instance of a supertype that has no corresponding subtype.

The first two points do not pose a problem to the design of an ontology. The third and fourth require some care. Rules must be established for how the discriminator will be treated.

Suppose e_s is a subtype of *e* in the Generic Hub. The following rules were used in the design of the GH ontology:

1. The class denoting *e* is a superclass of the class denoting e_s.
2. The class denoting *e* has a slot modeling the discriminator of *e*, even if the discriminator is complete. This rule results in redundant slots (the information can be inferred from the "type of" operator), but gives the GH ontology consistency.

The GH ontology does not use multiple inheritance. No data elements in the Generic Hub seem to benefit from multiple inheritance.

The GH ontology introduces one class for which there is no corresponding element in the Generic Hub. This class is named ObjectTypeEstablishment. It is the superclass of the two kinds of establishments that exist in the Generic Hub (materiel type and organization type). These establishments share attributes (name and effective date) and their commonality is effectively modeled using an abstract superclass.

9.2. Organic Slots

Each slot in the GH ontology that models an organic Generic Hub attribute has properties derived from those of the attribute:

1. If the attribute is required, the slot is required. (Note that Protégé-2000 flags violations of the "required" facet in its user interface but is willing to save an invalid model. The user or application is ultimately responsible for ensuring model consistency.)
2. The slot's value type is an instance of some concrete subclass of Type.
3. The slot has single cardinality.

There are some special rules for specific type categories. The following subsections cover them.

9.2.1. Organic Slots Modeling Coded Domains

If the slot models a coded domain, its value type is an instance of some concrete subclass of Enumerated-Type. The categoryCode slot of class Action has instance of DS110_act_cat_code as its value type. The subcateogoryCode slot of class Capability has instance of DS182_capab_subcat_code as its value type.

The enumerated types representing coded domains are all direct subclasses of EnumeratedType. The names for these types are taken from the Generic Hub validation rules for the attribute they represent. The alphanumeric prefix of each names helps a reader locate a type.

The metaclass of a type class modeling a coded domain is a subclass of Enumerable-Class. Most of these classes use Labeled-Enumerable-Class as their metaclass. A Labeled-Enumerable-Class notes only the elements of the class, optionally, it may specify a partial or total ordering among elements.

Slots denoting category codes use a type whose metaclass is Labeled-Enumerable-Class. There is no ordering among category code elements.

Orderings have been added wherever possible. The type DS106_pers_stat_physical_stat (the physical status of a person) ranks codes based on a person's mobility; greater mobility is highest. Thus IN (incapacitated, moveable by stretcher) is less than IW (incapacitated, cannot perform tasks but can walk), which is less than SI (incapacitated, can perform tasks), which is less than FT (fit).

Some types specify a partial rather than a total order. The elements of DS166_unit_type_size_code cover Army, Navy, and Air Force designations for units. The ordering for this type includes the usual Army terms (squad < platoon < company < …); however, neither a fleet nor a flight is directly comparable.[4] Moreover, the type includes elements such as TSKEL ("a unit organized for a specific task") whose size depends on context. These elements are omitted from the ordering. Many types have elements NKN (not known) and NOS (not otherwise specified) that likewise cannot participate in an ordering.

Where possible, types modeling coded domains use a metaclass more descriptive than Labeled-Enumerable-Class. Examples include:

[4] The ordering specifications for DS166_unit_type_size_code type are known to be incomplete in the current version of the GH ontology.

- Action resource criticality indicator (DS225_act_res_crticality_ind), whose two values are NO and YES. Its metaclass is Boolean-Class.
- Presence of mines (DS313_mine_prsnc_code), whose three values are NO, YES, and NKN. Its metaclass is Tri-State-Class.
- Timing precision code (DS284_timing_precision_code), whose values are SECOND, MINUTE, HR, DAY, etc. Its metaclass is Labeled-Enumerable-Class-Mapping-To-Measurement; the measurable thing is Time. No unit of measurement is specified, because the value *is* the unit of measurement.

There are cases where different metaclasses might have been used. Consider materiel-status-usage-status-code (DS259_mat_stat_usage_stat_code), which denotes the usage of a specific materiel. Its possible values are ACTIVE (performing the function for which it is designed), DEACTV (not performing the function for which it is designed), and NKN (not known). This class's metaclass could be Tri-State-Class, where ACTIVE corresponds to truth, DEACTV to falsehood, and NKN to an undefined value. The conceptual difficulty of this approach is that the ontology does not link truth to anything, as it has no implications concerning the meaning of active materiel.

Each enumerated type is linked via its elements to a subclass of Enumerated-Element (see Figure 12). Each of classes is instantiated with all instances of the coded domain it models. The slots of the class let both the code and a description be stated. The GH ontology contains all the codes in the Generic Hub. This, as discussed in Section 8.4, facilitates comparison.

The classes denoting the enumerated elements are divided among the Enumerated-Element class hierarchy. Classes in this tree add slots according to meaning. Instances of classes modeling Generic Hub coded domains must give values for as many slots as possible.

Most enumerated element classes in the GH ontology are direct subclasses of Enumerated-Element. All these classes have :STANDARD-CLASS as their metaclass. In other words, any semantics related to an enumeration are expressed as part of the type or as part of an instance, not as part of an element's class.

9.2.2. Organic Slots Modeling Dates and Times

The Generic Hub contains several attributes that model dates and times. These elements are stored as strings. A date is an eight character string of the form YYYYMMDD. If DD is "00", the date is precise only to within one month. If MM and DD are both "00", the date is precise only to within one year.

A time is stored as a six character string of the form HHMMSS. Times are always precise to within one second.

The GH ontology models dates and times using the types A.D. Date and Seconds Up To 24 Hours, respectively. The implication is that any two slots of value type A.D. Date may be compared, as may any two slots of value type Seconds Up To 24 Hours.

The type-instance-value slot of A.D. Date stores a string. The format of this string is identical to that used in the Generic Hub.

The type-instance-value slot of Seconds Up To 24 Hours stores an integer. Metaclass constraints restrict this integer to between 0 and 86,399, inclusive. Note that translating between a Generic Hub data set and a GH knowledge base requires conversion of times.

9.3. Relationships

The Generic Hub uses two kinds of relationships: one-to-many and many-to-many. All many-to-many relationships have an associative entity.

9.3.1. One-to-Many Relationships

Modeling one-to-many relationships in Protégé-2000 is straightforward. For each Generic Hub one-to-many relationship *r* from entity *A* to entity *B*, the GH ontology associates with the class modeling *A* (C_a) a slot that models *r*. This slot has multiple cardinality. Its value type is instance of the class modeling *B* (C_b).

Given an instance of C_a, it's easy to use Protégé-2000 to find all associated instances of C_b, through both the user interface and the Java API. Given an instance of C_b, it's harder to find the associated instance of C_a. In Java, one can find the instance using a simple, if time-consuming, algorithm. A Protégé-2000 user must perform a tedious search using internal instance names.

The GH ontology always defines an inverse slot for each one-to-many relationship. The slot in C_a that models *r* has an inverse slot associated with C_b. This slot is required only if the corresponding foreign key of *B* is required. Its value type is a single instance of C_a. Because many-to-many relationships are modeled as two one-to-many relationships (see below), inverse slots also exist for many-to-many relationships.

9.3.2. Many-to-Many Relationships

Protégé-2000 supports many-to-many relationships only if they don't have an associative entity. Unfortunately, this excludes all many-to-many relationships in the Generic Hub.

A few associative entities have no attributes other than their keys (e.g., ACTION-TASK-RULE-OF-ENGAGEMENT). These entities could be omitted from the GH ontology, but a solution is needed for modeling the rest of the associative entities.

There are two possible approaches to modeling many-to-many relationships in Protégé-2000. The first is to mimic the structure of the Generic Hub. Suppose the Generic hub defines a many-to-many relationship between entities E_1 and E_2 with associative entity E_a. Let C_1, C_2, and C_a be the classes that model E_1, E_2, and E_a, respectively. Let C_1 and C_2, have slots r_1 and r_2 that define a one-to-many relationship to C_a (plus inverse slots for C_a, as noted above). The set of all instances of C_2 associated with an instance of C_1 is the instance of the inverse of r_2 of the set of all instances of C_a associated with C_1 through r_1.

This approach is reasonable if one does not mind the indirection necessitated by the imposition of C_a between C_1 and C_2. A second approach to modeling many-to-many relationships is to treat them as first-class objects. The GH ontology could contain a class MNRelationships with the following template slots:

- instA, which defines the instance of C_1 that participates in the relationship. Its value type would be an instance of a GH class.
- instB, which defines the instance of C_2 that participates in the relationship. Its value type would be an instance of a GH class.
- associativeInst, which defines the instance of C_a that participates in the relationship.

The value type of all of these slots is an instance of a GH class. All slots are required.

This second approach simplifies the task of finding the set of all instances of C_2 associated with instance c_1 of C_1 and vice versa. One issues a query of the form:

Find all instances of MNRelationships where the class of instA is C_1, the class of instB is C_2, the class of associativeInst is C_a, and the value of instA is c_1.

This approach has the following drawbacks:

1. The MNRelationships class will have a large number of instances. Searching it may become time-consuming.
2. Certain information on many-to-many relationships vanishes from the ontology. The name of a relationship, which can be captured in a slot, is lost. This may hinder collection and dissemination of metadata.
3. An associative entity must participate in exactly one instance of a relationship. This is automatically enforced using the first approach, but it's possible to add an instance to MNRelationships in violation of this constraint.

These drawbacks have workarounds. For instance, rather than defining a single class MNRelationships, the GH ontology could define a separate class for each many-to-many relationship. Own slot of such classes could specify metadata.

The GH ontology uses the first approach. It relies on a model that is known to be workable and can be regarded as the cautious option. It is also more explicit about showing relationships between classes, which is probably an advantage when defining metadata.

9.3.3. Associative Classes

Each class that models a Generic Hub associative entity has metaclass Associative-Class. An Associative-Class explicitly specifies the two entities involved in the association. The class has two template slots, associative-class-class-1 and associative-class-class-2. Both are required, and both have a class that is a subclass of GH-Entity as their value type. This extra information may prove useful in disambiguating associations. Figure 16 shows the specification of class ObjectItemType, the class associated with the many-to-many relationship between ObjectItem and ObjectType.

Associative classes used to play an important role when an application loaded a GH knowledge base. Subsequent changes in implementation techniques have obviated the need for them. They are currently retained in the belief that specific knowledge of the role a class plays may be useful.

9.4. Absence of Keys

The GH ontology does not have any slots that model values of Generic Hub key attributes, either primary or foreign. A knowledge base does not need such slots; each instance in an object model has a unique name that is assigned at the time the instance is created. This name obviates the need for primary keys.

Translating a Generic Hub data set into a GH knowledge base requires translating foreign keys into slot value instances. Updating a Generic Hub data set with changes from a GH knowledge base requires knowledge of the keys associated with the data used to create an instance. Annex B presents one approach to implementing this functionality.

9.5. Naming Conventions

Elements in the Generic Hub follow certain naming conventions that are suited to a DBMS but not necessarily elsewhere. The GH ontology was created with the expectation that the contents of a knowledge base might need to be shared among applications. Specifically:

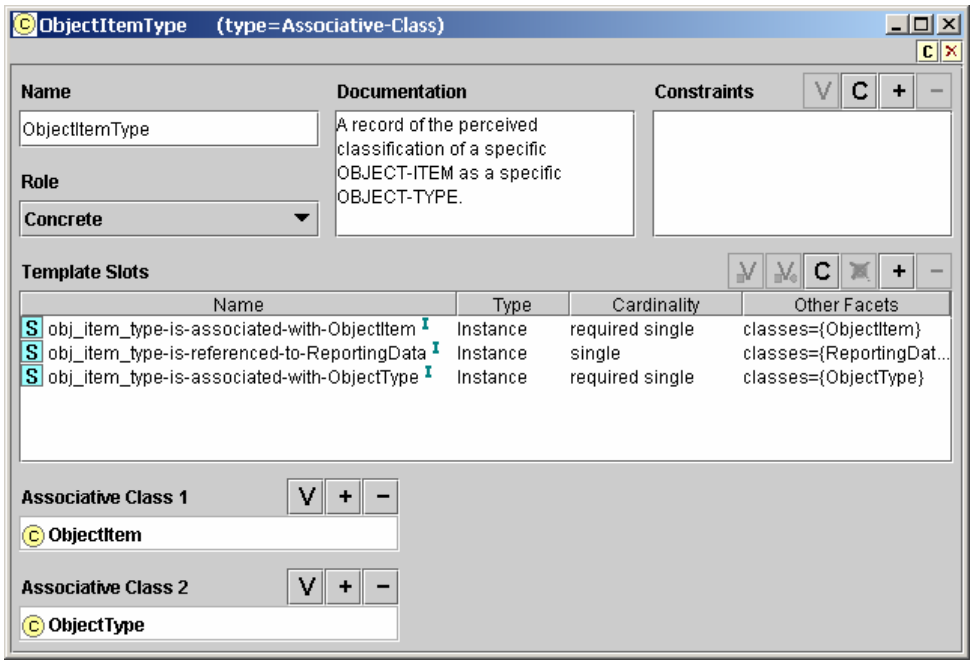

Figure 16. The Specification of the ObjectItemType Class

1. Names should be meaningful. The physical model of the Generic Hub uses valid SQL names such as ACT and GEO_FEAT. If class names in an ontology are to serve as meta-data, they should be composed of actual words, not abbreviations.
2. Names should adhere to conventions of programming languages. One way to share knowledge among application is through XML, using a data model such as DTD or XSD. These models have rules for names and their transformation into Java identifiers.

9.5.1. Naming Conventions for GH-Entity Subclass Elements

In the GH ontology, names are closer to those in the Generic Hub logical model than the physical model:

1. Class names are taken from entity names. The upper case names are converted to initial upper case, and hyphens are removed.
2. Organic slot names are derived from attribute names. In the Generic Hub, an attribute's name is prefixed with the name of the entity to which it belongs. In the GH ontology, a slot's name is formed by removing the entity's name, then concatenating the separate words of the attribute's name (removing the hyphens) with each word other than the first converted to initial upper case. For example, the attribute reporting-data-timing-category-code within the GH entity REPORTING-DATA becomes timingCategoryCode.
3. Slot names for one-to-many relationships are derived from the relationship phrase and the name of the associated class. Words in the phrase are in lower case and are separated by hyphens; also, a hyphen separates the phrase from the associated class name. For example, class ObjectItem has slots named has-ObjectItemStatus and is-specified-with-ObjectItemCapability.
4. Inverse slot names are constructed from three parts: the name of the (physical) table implementing the entity at the "many" end of the relationship, the inverse phrase, and

the name of the class at the "one" end of the relationship. For example, the inverse of has-ObjectItemStatus is obj_item_status-is-ascribed-to-ObjectItem.

The table name must be prefixed to the slot name to avoid conflicts. In particular, the Generic Hub has fourteen relationships of the form "REPORTING-DATA provides application information for *E*" each of which has an inverse relationship "*E* is referenced to REPORTING-DATA". A slot can have at most one inverse; thus the inverse slot implementing the inverse relationship can't be shared by the fourteen relationships of REPORTING-DATA. Prefixing the slot name with the table name makes the slot name unique. (On reflection, the ontology would have been more readable if the class name had been used instead of the table name.)

These naming conventions result in shorter (and therefore simpler) names than in the Generic Hub. They do, however, yield conflicts. Attributes action-name, object-item-name, and object-type-name are all modeled in the GH ontology as a slot named name. None of these slots shares the same value type; all are instances of a different subclass of String-Type. Protégé-2000 permits slot facets to be overridden on a per-class basis, which allows the different value types to be specified. Some care is needed when viewing and modifying slots, however, as it is easy to accidentally view a slot's global rather than local definition.

Slots named categoryCode have a more interesting conflict. Entities OBJECT-TYPE, MATERIEL-TYPE, CONSUMABLE-MATERIEL-TYPE, and AMMUNITION-TYPE all have a (distinctly named) category code attribute; these attributes all translate into slots named categoryCode. The issue is not the common name, but rather that the classes representing these entities are part of an inheritance hierarchy. If class C_s is a subclass of *C*, Protégé-2000 only displays the union of the template slots of C_s as determined by name. If C_s overrides a property of some template slot of *C*, Protégé-2000 lets the user see the slot only with the overridden properties. The slot as it appears in *C* is invisible. In other words, for an instance of AmmunitionType a user may set the categoryCode slot to a valid ammunition type but cannot set the categoryCode slot of any other class in the hierarchy.

One fix would be to use the Generic Hub's naming conventions. Instead, the GH ontology relies on the fact that the category code of a superclass can always be inferred. Given an instance of AmmunitionType, the category code for class ConsumableMaterielType must be AMMO, the category code for class MaterielType must be CM, and so on. No information is lost. In fact, one can argue that including all the category codes under different names merely permits errors in the ontology. Deriving category codes from the class hierarchy is safer.

Generic Hub discriminator attributes that are incomplete must also be considered. For example, the facility-category-code attribute has seven values, but for two – NOS and ROADWY – there is no corresponding subtype entity. For such values, an instance of Facility must be a direct instance. This rule can be enforced with a PAL constraint:

```
(defrange ?f :FRAME Facility)
(forall ?f
    (=> (or (= (enumerated-element-label (type-instance-value (categoryCode ?f) "NOS")
            (= (enumerated-element-label (type-instance-value (categoryCode ?f) "ROADWY"))
        (direct-instance-of ?f Facility)
```

In fact, rules related to the value of the categoryCode slot for the Facility class can be stated using PAL:

```
(defrange ?f :FRAME Facility)
(forall ?f
   (and (=> (direct-instance-of ?f Airfield)
            (= (enumerated-element-label (type-instance-value (categoryCode ?f))) "AIRFLD"))
     (=> (direct-instance-of ?f Bridge)
       (= (enumerated-element-label (type-instance-value (categoryCode ?f))) "BRIDGE"))
     (=> (direct-instance-of ?f MassGrave)
       (= (enumerated-element-label (type-instance-value (categoryCode ?f))) "MSSGRV"))
     (=> (direct-instance-of ?f Minefield)
        (= (enumerated-element-label (type-instance-value (categoryCode ?f))) "MINE"))
     (=> (or (= (enumerated-element-label (type-instance-value (categoryCode ?f))) "NOS")
          (= (enumerated-element-label (type-instance-value (categoryCode ?f))) "ROADWY"))
       (direct-instance-of ?f Facility))))
```

This constraint does not include a test for the NETWRK value because, of all subclasses of Facility, only class Network has its own definition of slot categoryCode.

9.5.2. Naming Conventions for Types

The types used in the GH ontology – that is, the subclasses of Type – also follow naming conventions:

1. Names of types derived from coded domains are taken from validation rules in the Generic Hub. For instance, the name of the type for the categoryCode attribute of the Action class is DS110_act_cat_code.
2. Other type names are derived from the attribute they model. Examples include Bridge-Spans, Holding-Quantity, and Capability-Amount. Capability-Amount is the value type of the quantity slots of both ObjectItemCapability and ObjectTypeCapabilityNorm. The generic name reflects its shared use.

These two conventions are not consistent, and should be resolved. During the design of the ontology, the use of the validation rule name facilitated cross-referencing. Now that the ontology is complete, it would be better to use a more meaningful name, perhaps retaining the validation rule name through the use of a metaclass.

The convention for other type names is inconsistent with the conventions used for class names. Hyphens should be used consistently.

10. The Command and Control (C2) Ontology

The C2 ontology differs from the ontologies upon which it is built. The difference stems from the intent of how the ontology will be populated:

- The ontologies underlying the GH ontology are intended to be pre-populated with concepts relevant to modeling physical and conceptual properties of C2 data. This information, then, is fixed.
- The GH ontology is intended to be populated in real time with instances obtained from sensor and network data – in other words, from information drawn from GH-conformant data sets. A GH ontology must be continually updated with the latest information on battlefield object location, status, supply, plans, etc. This information ultimately derives from observations.

- By contrast, the C2 ontology is intended to be populated by information drawn from reasoning. This information is likely to be generated by an inferencing engine (see below) though there is no inherent reason it cannot be entered manually.

Whereas the GH ontology contains atomic C2 data, the C2 ontology contains high-level abstract notions of command and control. The C2 ontology intends to capture the abstract concepts that warfighters use as part of reasoning during the decision-making process.

Whereas the GH and underlying ontologies are useable and close to production quality, the C2 ontology is preliminary. Identification, formalization, and incorporation of a useful set of abstract warfighting concepts is a huge, multi-year task (one way to grasp its magnitude is to think of formalizing doctrine). The purpose of the C2 ontology to date has been to study how concepts can be formalized, thereby providing a framework for an eventual effort.

The size and complexity of the overall task dictated that the "framework" ontology would need to concentrate on some subset of C2. Two areas were selected: threats and organization operational readiness. Both are regarded as complex but vital in C2. Their successful formalization in an ontology would provide a convincing demonstration of certain vital net-centric precepts.

Even the formalization of these two areas – indeed, of either one – is a task of considerable complexity. The C2 ontology in its current state covers only threats, and does not claim to have an adequate formulation of that concept. Whether the subjective notion of "threat" can ever be formalized is still an open question. And whether a GH-conformant data set (or, correspondingly, a GH ontology) provides enough information to identify all conceivable threats is still unknown.

10.1. Classes of the C2 Ontology

The C2 ontology adds a single class hierarchy to the underlying ontologies. This hierarchy is rooted by a class named Threat, which is a subclass of :THING. Threat is an abstract class comprising subclasses that, as one descends deeper into the hierarchy, specialize the kind of threat in terms of the OBJECT-ITEM posing the threat and the OBJECT-ITEM being threatened. Only leaf notes of the hierarchy are concrete classes. The C2 ontology currently recognizes threats posed by organizations and persons, and threats to facilities, organizations, and persons. See Figure 17.

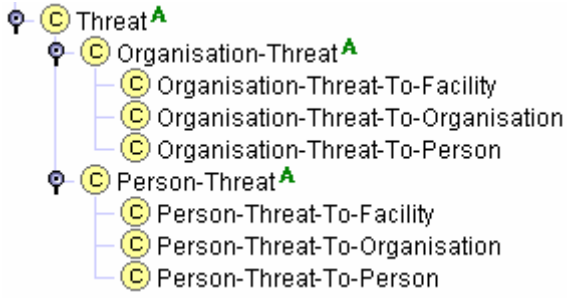

Figure 17. C2 Ontology Classes

10.2. Slots of the C2 Ontology

The C2 ontology defines two slots. Both are template slots of class Threat:

1. The threatening-object-item slot denotes the object item that poses a threat.
2. The threatened-object-item slot denotes the object item that is threatened.

The value type of both of these slots is an instance of class ObjectItem. Slots in the C2 ontology build on properties defined in the underlying ontologies. The C2 ontology states that threats are posed by battlefield objects, that battlefield objects may be threatened, and that a threat is a binary relationship between two battlefield objects.[5]

The C2 ontology defines PAL constraints that specialize these slots according to their placement in the hierarchy. Class Organisation-Threat has a constraint that requires the value of the threatening-object-item to be an instance of class Organisation, whereas Class Person-Threat has a constraint that requires the value of the threatening-object-item to be an instance of class Person. Classes Organisation-Threat-To-Facility and Person-Threat-To-Facility both require the threatened-object-item slot's value to be an instance of Facility. These constraints are straightforward to express; for example,

```
(defrange ?ot :FRAME Organisation-Threat)
(forall ?ot
    (instance-of (threatening-object-item ?ot) Organisation))
```

states that in every instance of Organisation-Threat the OBJECT-ITEM posing the threat must be an instance of an ORGANISATION.

The Threat class has a more complex PAL constraint that requires the threatening OBJECT-ITEM to be hostile according to its most recent OBJECT-ITEM-STATUS. This constraint is worth presenting as an illustration of the complexity that constraint writers are likely to encounter:

```
(defrange ?status :FRAME ObjectItemStatus has-ObjectItemStatus)
(defrange ?hc :FRAME DS145_obj_item_stat_hstly_code)
(defrange ?hce :FRAME DS145_obj_item_stat_hstly_code-Elements)
(defrange ?statp :FRAME ObjectItemStatus has-ObjectItemStatus)
(defrange ?rd :FRAME ReportingData)
(defrange ?rdp :FRAME ReportingData)
(defrange ?dt :FRAME Date-Type)
(defrange ?dtp :FRAME Date-Type)
(defrange ?tm :FRAME Time-Type)
(defrange ?tmp :FRAME Time-Type)
(defrange ?t :FRAME Threat)
(forall ?t (exists ?status (and (has-ObjectItemStatus (threatening-object-item ?t) ?status)
  (exists ?hc (and (hostilityCode ?status ?hc)
    (exists ?hce (and (type-instance-value ?hc ?hce)
      (enumerated-element-label ?hce "HO")
        (exists ?rd (and (provides-applicable-information-for-ObjectItemStatus ?rd ?status)
          (not (exists ?statp (and (has-ObjectItemStatus (threatening-object-item ?t) ?statp)
            (exists ?rdp (and (provides-applicable-information-for-ObjectItemStatus ?rdp ?statp)
              (exists ?dt (and (reportingDate ?rd ?dt)
                (exists ?tm (and (reportingTime ?rd ?tm)
                  (exists ?dtp (and (reportingDate ?rdp ?dtp)
                    (exists ?tmp (and (reportingTime ?rdp ?tmp)
```

[5] A more realistic definition of a threat is a binary relationship between two sets of battlefield objects, but the complexity of that relationship introduces details that are beyond the experimental nature of the C2 ontology. Here and elsewhere, however, this section notes such simplifications.

```
(=> (= ?dt ?dtp) (> ?tmp ?tm))
  (> ?dtp ?dt))))))))))))))))))))))
```

The complexity of this constraint – which, as discussed below, is incomplete – is attributable to at least three factors:

1. *The lack of user-defined functions in PAL*. Procedural abstractions would eliminate common subexpressions (note the repeated occurrences of (has-ObjectItemStatus ...).
2. *The difficulty of expressing ordering relationships in a set-theoretic language*. The expression that constrains an instance of ObjectItemStatus to the most recent with respect to all other instances associated with an ObjectItem is easy enough to state: there does not exist another instance whose reporting date is greater, or whose reporting date is equal and whose time is greater. But expressing it in PAL requires introducing an extra set of variables bound to ReportingData, Date-Type, and Time-Type. If instead one could write an expression denoting the last instance of the sorted ObjectItemStatus instances, the above expression would be considerably simpler. Unfortunately, sets aren't ordered, so most set-theoretic languages don't include operators that take advantage of order.
3. *The representation decisions in the Type ontology*. Section 8.5 discussed the advantages and disadvantages of the type-instance-value slot and the Enumerated-Element class. The above expression shows how they add to the complexity of a PAL constraint.

Study of the constraint statement shows that it is deciding which ObjectItemStatus is most recent based on the reportingDate and reportingTime slots of the associated ReportingData instance. This raises three issues:

1. The constraint assumes a GH knowledge base is valid. The GH ontology requires an ObjectItemStatus instance to be associated with a ReportingData instance. The constraint doesn't check to ensure if the association exists. PAL permits such a test (using its own-slot-not-null predicate); it has been omitted for simplicity.
2. Slot reportingDate is required; reportingTime is not. The constraint needs to test whether reportingTime is null. Moreover, the ontology designers need an accepted definition of the meaning of order for comparing two ReportingData instances where one's reportingTime slot is null and the other's is not.
3. Arguably, the comparison should be based on effective rather than reported timing. An effective timing specification derives from either the effectiveDate and effectiveTime template slots of subclass ReportingDataAbsoluteTiming or from the offsetDuration template slot of subclass ReportingDataRelativeTiming added to the plannedStartDate and plannedStartTime slots. PAL can differentiate between subclasses. It lacks date arithmetic operators, however.

In other words, the constraint is incomplete, and completing it would greatly increase its complexity.

10.3. Uses of the C2 Ontology

The C2 ontology is intended to be used in a rule-based inferencing environment. Applications will query a C2 knowledge base, either periodically or on demand (or both). The queries will be predefined, and will be a specification of how command and control information

may be derived from the information in a GH knowledge base (or, more likely, a C2 knowledge base, as rules may rely on previously derived C2 information).

For example, an organization *O* might *ask*:[6]

(Threat :threatened-object-item *O* :threatening-object-item ?x)

The result of this query is a set consisting of all Organisation and Person instances in the knowledge base that pose a threat to *O* according to rules in the C2 ontology.

Alternately, an application may opt to add a new fact to a knowledge base directly. This would happen if the application had certain knowledge of the existence of a threat. It would *tell* the knowledge base:

(Organisation-Threat-To-Organisation :threatened-object-item *O* :threatening-object-item *T*)

thereby creating a new instance of class Organisation-Threat-To-Organisation.

Ask and *tell* are inferencing terminology. When an application "tells" a knowledge base of a fact, rules stored in the knowledge base may be automatically triggered that add additional facts; this is known as *forward chaining*. When an application "asks" a knowledge base a query, rules may also be automatically triggered that add additional facts; this is known as *backward chaining*.

Chaining helps populate knowledge bases. It adds information based on rules that ontology designers formulate. Note that arbitrary information is not preserved, i.e., intermediate computations of a query are not necessarily saved. Instead, rule designers define the concepts that a query reveals.

10.3.1. Backward Chaining

Backward chaining ensures that information generated as a result of a query is preserved. A query is stated as an application of a template slot to an instance. For instance,

((:instance Unit ?u) (has-ObjectItemStatus ?u ?status))

poses the query "find all instances of ObjectItemStatus that are related to an instance of Unit." It assigns the results of the query to the variable ?status.

Backward chaining provides rules for answering queries. In the C2 ontology, an obvious query is:

Find all threats to the unit whose name is *N*.

The query is posed as:

((:instance Unit ?u (name ?u N)
(threatened-object-item ?threat-inst ?u)
(threatening-object-item ?threat-inst ?threatener))

The issue is how one determines the existence of a threat – that is, how instances of class Threat are created. Unless those instances exist, the query will fail.

A C2 knowledge base might be populated with instances of Threat in three ways:

1. A user can manually add instances.

[6] The syntax is taken from the Semantic Language (SL) of the Federation for Intelligent Physical Agents (FIPA). SL is intended for stating queries and their results.

2. An application can scan the knowledge base looking for facts that justify creating an instance of Threat. This scanning might occur periodically or each time new facts are added (or both).
3. An application- or user-initiated query that detects a threat can automatically create an instance of Threat.

Backward chaining rules implement the third strategy. Backward chaining is expressed as a rule of the form:

(consequent ← antecedent)

Whenever a query or one of its clauses matches the consequent of a backward chaining rule, the antecedent is automatically applied to see if it generates instances; if it does, those instances are asserted as new facts in the knowledge base. The canonical example is:

((aunt ?person ?y) ← (parent ?person ?x) (sister ?x ?y))

that is, when one poses a query to find the aunts of some person, the approach to answering that query is to search for the person's parents, then search for the parents' sisters.

For the C2 ontology, a logical backward chaining rule would be:

(threatened-object-item ?threat-inst ?obj-item) ← (*rules defining threat*))

That is, any time a query is posed to determine whether an ObjectItem instance is threatened, the query is answered by invoking the rules defining the concept of a threat. For each successful application of these rules – that is, for each threatened ObjectItem instance found –the threatened-object-item slot of a Threat subclass instance is set to the threatened instance.

This simple explanation omits certain details, in particular the rules defining a threat. Annex B presents more detail on how this would be achieved. Even so, it is not the intent of either annex to present a realistic definition. That would result from an intense study of how C2 could be formalized using Generic Hub concepts. Such a study is beyond the scope of this document.

10.3.2. Forward Chaining

Forward chaining is used to maintain invariant relationships and knowledge base consistency. A traditional use of forward chaining is to maintain inverse relationships. Given the rule:

((parent ?a ?b) → (child ?b ?a))

then whenever an application asserts that *a* is the parent of *b*, the knowledge base also notes that *b* is the child of *a*. The GH ontology does not need this sort of rule; its use of Protégé-2000's inverse slot feature ensures Protégé-2000 will automatically add an inverse relationship.

There are, however, more sophisticated uses of forward chaining, and the C2 ontology could benefit from them. For example, an application cannot just tell a knowledge base of the existence of a threat. The knowledge base needs to know who believes the threat exists, and why. Forward chaining rules can be used to gather, generate, and add this additional information, in the form of GH ontology instances, to a C2 knowledge base. A rule of the form:

((threatened-object-item ?threat ?o1) (threatening-object-item ?o2) →
(*Add relevant GH instances to knowledge base*))

ensures this will happen whenever someone sets the slots of a Threat.

Annex B: Design of a Net-Centric System Using the GH Ontology

This annex presents the design of a simulation prototype decision support system that uses the GH ontology described in Annex A. The prototype explores how decision support functionality can be implemented in a net-centric environment. This annex also is intended to provide a canonical architecture for implementing decision support in that environment.

The purpose of this annex is to discuss design and implementation decisions and details that arose during the prototype's construction. This annex should help organizations intending to implement applications, particularly decision support applications, in a net-centric environment, as they will face many of the issues encountered building this prototype. The information in this annex is meant to provide guidance.

A prototype system is not a production system. Some design decisions in a prototype are made to reduce effort in order to get the system running quickly. Generally, therefore, prototypes:

- Incorporate technology-specific functionality at the expense of flexibility and easy maintainability.
- Use algorithms whose execution time is acceptable only for small data sets.
- Make simplifying assumptions on error handling.

The prototype developed in this study has these features. They are noted, along with discussion of realistic alternatives.

The IDA prototype uses several freely available technologies and implementations. They are discussed briefly in Section 3.1, but this annex is not intended to explicate them. The reader may need to refer to supplementary documentation to fully understand some of the technical discussion. Also, some illustrations assume familiarity with the Unified Modeling Language.

1. Operational View

The prototype system's operational view strives to provide a simple but realistic (if abstract) simulation of roles humans play in a warfighting environment. The prototype currently focuses on a single human role, namely, that of the decision maker. It is assumed that a decision maker will act on information available within the net-centric environment. Within the IDA prototype the characteristics of decision makers is predefined. Once the prototype starts executing, roles do not change.

The IDA team also modeled decision making as hierarchical, recognizing command chains. A higher level decision maker has authorities not granted to a lower level decision maker.

The prototype is organized around military units. A unit is either friendly or hostile. Friendly units are able to communicate with each other.

A friendly unit has a set of associated decision makers. Decision support functionality presents decision makers with threats. (The definition of a threat is complicated, and is taken up repeatedly in this annex; informally, it means the movement of a hostile organization toward a friendly organization.) The objective of decision makers is to determine and enter appropriate responses to threats. The prototype currently only allows decision makers to enter these

responses – i.e., the simulation does not evolve beyond the first stage. Specifically, this means that their associated friendly units do not fire, move, or otherwise take physical action.

Each decision maker sees a specific portion of the battle space. His information on the battle space comes from intelligence sources controlled by his unit and communications from other units. He is responsible for communicating to other friendly units all information newly obtained from his intelligence sources.

The operational architecture deliberately omits a role for information analysts. It is assumed that the delivery of information traditionally supplied by human analysts is the responsibility of the network centric system.

The prototype also includes support roles. Individuals playing these roles act as controllers:

- An individual may control a hostile (enemy) organization. The role of an enemy organization in the prototype is to move; as it moves, it can be detected by an intelligence source connected to a friendly organization. The controller of an enemy organization specifies whether it is currently moving, and if so, its speed and bearing angle. Simulations work with discrete events, so the controller also specifies the position refresh rate, which is how often the component implementing an enemy organization recalculates its position.
- An individual may control the initiation and termination of the prototype's execution. This is an administrative role. The administrator specifies the number of friendly and enemy organizations that will participate in a simulation, as well as the inclusion or exclusion of some supporting components. Once the prototype is executing, the administrator has the option of instructing all enemy organizations to start or stop. In other words, the administrator can assume the role of enemy organization controller.

Figure 1 depicts the operational view. The emphasis is on the roles active during execution; the administrator is omitted. Intelligence sources (shown as humans, but possibly mechanical or electronic sensors) operate within a battle space to obtain information of use to decision makers. This information is transmitted to intercommunicating decision support applications, which in turn present information to decision makers. The role of the decision support applications is to filter and select the information presented to decision makers. This ensures that decision makers have the best pertinent information available. They make decisions and feed them back into the decision support applications, which in turn update each other with the latest information on friendly organization state and intentions.

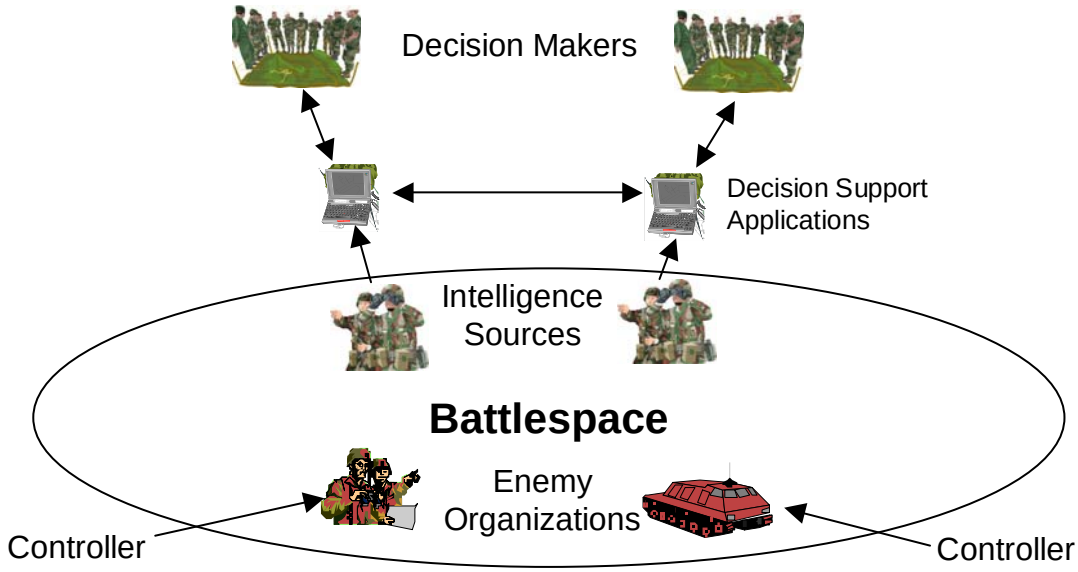

Figure 1. Operational View

2. System View

The system view shows the system decomposed into its component subsystems. It presents the interconnections between these subsystems. It also shows the interactions of external components (humans and external computer systems).

Figure 2 shows the high-level system view after initialization. There exists:

- A set of subsystems implementing friendly organizations. Each friendly organization has an associated set of decision makers, an associated set of intelligence sources, and a location.
- A set of subsystems implementing enemy (hostile) organizations. Each enemy or-

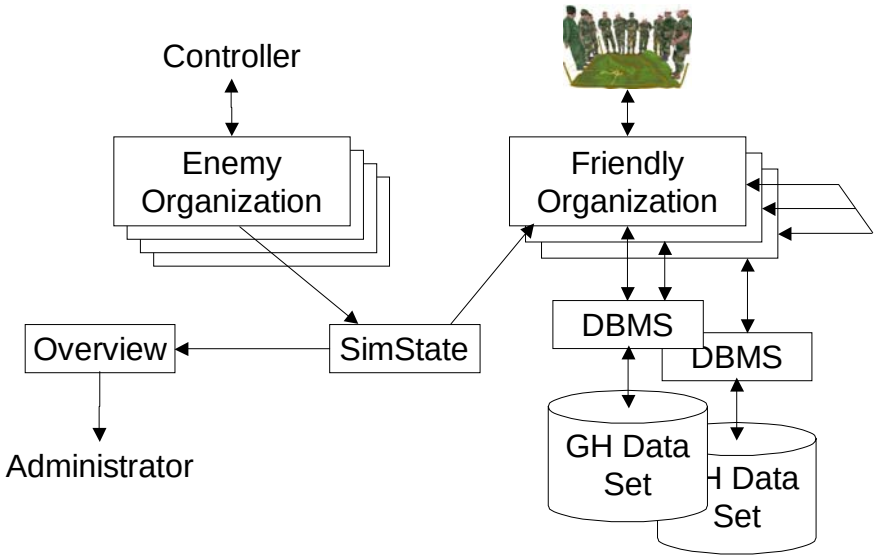

Figure 2. System View (Omitting Start-Up)

ganization has a current location, speed, and bearing angle.

- An overview subsystem. This component presents a two-dimensional display of the location of every known organization, both friendly and hostile. It is used as an aid by administrators.
- A simulation state subsystem. This subsystem maintains ground truth. Each subsystem that may change a component of its ground truth is responsible for communicating the change to the simulation state subsystem. Arrows in Figure 2 connected to SimState show how ground truth is obtained and communicated:
    - Each time an enemy organization changes its location, it sends that new location to the SimState component.
    - A friendly organization periodically requests the SimState component to send the latest information on properties (such as location) of battlefield objects. This simulates humans making observations.
    - The SimState component sends ground truth updates to the overview component. (Note that a friendly organization pulls data, whereas the overview component receives pushed data.)
    - A set of database management subsystems. Each DBMS maintains a GH-conformant data set accessed by some set of friendly organizations. Each friendly organization accesses exactly one DBMS.

An execution of an interacting set of these subsystems is termed a *simulation*. Each simulation must have at least one Friendly Organization, at least one DBMS, and exactly one SimState. Other components are optional. Their inclusion follows from the objectives of a simulation.

Friendly organizations communicate with each other. Enemy organizations currently do not. This reflects the prototype's focus on decision support between friendly organizations.

The system view does not prescribe any hardware configurations beyond those needed to implement the system software. The simulation may be run on one or many different computers. If multiple computers are used, they must be connected by a network that supports TCP/IP.

The GH-conformant data sets need not be identical at the time the prototype components start executing. Differing data sets will cause individual components to have differing views of the battle space, which of course typifies warfighting, and is also what warfighters seek to avoid. One goal of the prototype is to ensure that views converge. However, data is pulled rather than pushed in a net-centric environment, so convergence cannot be mandated as it might in current operational environments. It must instead be incorporated in more subtle ways.

2.1. Enemy Organizations

An enemy organization (also known as a hostile organization) is a semi-autonomous entity. Its controller may set it in motion. Once moving it may be halted.

An enemy organization is identified by its name. An enemy organization must be defined in the GH-conformant data set at the time the simulation begins. Specifically, the following information must exist:[1]

[1] For readability, names are taken from GH's logical rather than physical model.

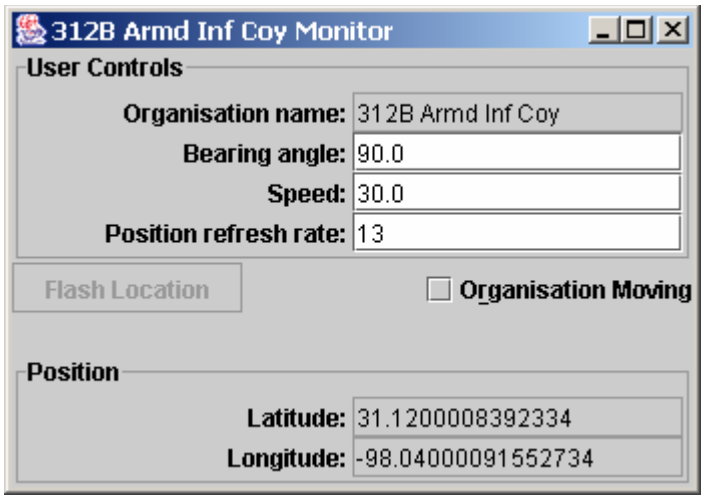

Figure 3. Enemy Organization User Interface

- The OBJECT-ITEM table must have exactly one row whose object-item-name column is that of the enemy organization.
- The ORGANISATION table must have a row whose key is that of the aforementioned row in the OBJECT-ITEM table.
- The OBJECT-ITEM-LOCATION table must have at least one row linked to the OBJECT-ITEM table via the object-item-id portion of the key. If more than one row in OBJECT-ITEM-LOCATION maps to the OBJECT-ITEM, then the relevant row is the one that is most recent according to the reporting date and time of its associated REPORTING-DATA.
- The row in the LOCATION table that maps to the aforementioned row in OBJECT-ITEM-LOCATION must be an instance of a POINT. That is, its location-category-code column must have the value 'PT', and there must exist a corresponding row in the POINT table.
- The row in the POINT table must be fully specified as either an ABSOLUTE-POINT or a RELATIVE-POINT. If it is a RELATIVE-POINT, it must be resolvable into an ABSOLUTE-POINT.

An enemy organization has five attributes: current location, bearing angle, speed, whether it is moving, and refresh rate. The first three attributes are initially taken from information in the Generic Hub data set. Bearing angle, speed, and refresh rate may also be provided at the time the enemy organization is initialized; initialization-time values will override any in the data set. These values are all shown in the user interface of an enemy organization (Figure 3). Bearing angle, speed, and position refresh rate must all be specified before an enemy organization can move.

The movement state of an enemy organization is dynamically controlled, either by a human or by the Loader subsystem. When an organization starts moving, it begins recalculating its position. Periodically, as specified by the refresh rate (in seconds), it computes the position it would have if it were to move at the specified bearing angle and speed. It updates its display of position, and sends this new position to the SimState system.

Any time an enemy organization is not moving, its controller may change its bearing angle, speed, and position refresh rate. Organization name is immutable. An enemy organization is not permitted to morph into another.

On start-up, an enemy organization has the following parameters:

1. The URL of a GH ontology.

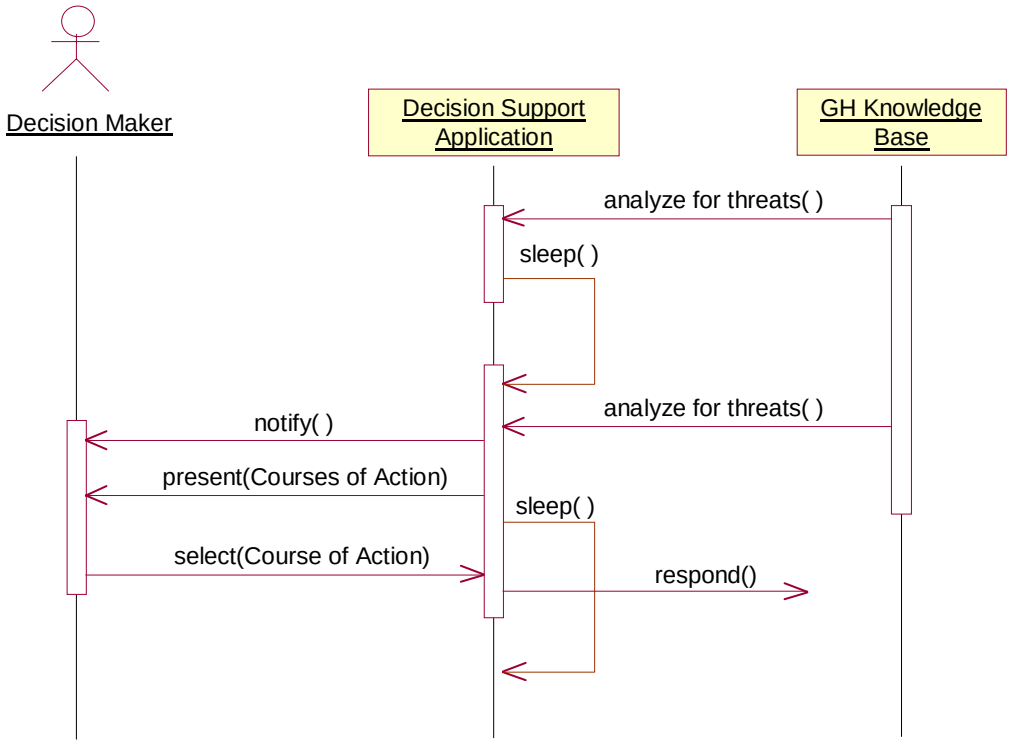

Figure 4. Overview of Friendly Organization Behavior

2. The URL of a Generic Hub data set. This parameter is optional; if not given, the enemy organization uses a canned set of data.
3. The name of the organization. This name must exist in the Generic Hub data set.
4. Initial values for bearing angle and speed. These parameters are optional. If not given, the values are undefined.
5. An initial value, in seconds, for the position refresh rate, the interval between times when the enemy organization is to recalculate its position based on speed and bearing angle. This parameter is optional. If not given, the position refresh rate is undefined.
6. Whether the enemy organization is to have a visible user interface on the computer where it is initiated. This parameter is optional. If not given, the enemy organization will have a visible user interface.
7. A name that is unique with respect to names of all other system components that will execute on the computer running the enemy organization.

2.2. Friendly Organizations

Friendly organizations are the emphasis of the prototype, as its purpose is to explore how ontologies can be used to automate decision support functions. Each friendly organization, then, is provided with decision support functionality. Figure 4 delineates a friendly organization's behavior using a sequence diagram. A decision maker has a decision support application. This application has access to a GH knowledge base. Periodically, the decision support application analyzes the knowledge base using rules that detect the presence of threats. If it finds none, it repeats the analysis after a prespecified interval.

When the decision support application determines that a threat exists, it notifies the decision maker, then presents to the decision maker a set of courses of action that seem reasonable in light of the threat. The decision maker chooses one (or none), and the decision support application responds appropriately (Figure 4 does not elaborate this aspect). Meanwhile, the decision support application schedules another threat analysis.

Each friendly organization must exist in the Generic Hub data set being used at the time its subsystem is initialized. Specifically, the following information must exist:

- The OBJECT-ITEM table must have exactly one row whose object-item-name column is that of the friendly organization.
- The ORGANISATION table must have a row whose key is that of the aforementioned row in the OBJECT-ITEM table.
- The OBJECT-ITEM-LOCATION table must have at least one row linked to the OBJECT-ITEM table via the object-item-id portion of the key. If more than one row in OBJECT-ITEM-LOCATION maps to the OBJECT-ITEM, then the relevant row is the one that is most recent according to the reporting date and time of its associated REPORTING-DATA.
- The row in the LOCATION table that maps to the aforementioned row in OBJECT-ITEM-LOCATION must be an instance of a POINT. That is, its location-category-code column must have the value 'PT', and there must exist a corresponding row in the POINT table.
- The row in the POINT table must be fully specified as either an ABSOLUTE-POINT or a RELATIVE-POINT. If it is a RELATIVE-POINT, it must be resolvable into an ABSOLUTE-POINT.

A decision support system must present its user with the best possible information for making decisions. The emphasis of this implementation is currently on detecting hostile organizations and making decisions on how to respond to their presence. The user interface of a friendly organization therefore shows its users a map of that portion of the battle space occupied by the organization, plus subsidiary information on supplies and threats (see Figure 5). Organizations on the map are shown as “red” and “blue” units. Red units are hostile. Blue units are friendly. The friendly organization controlling the map is the one whose “U” is darker than the others (in Figure 5, the rightmost unit).

The map provides a visual clue on imminence of threats. The decision makers see the movement of a hostile organization in real time. Its vector helps determine whether it presents a direct threat to any friendly organization.

The interface should not be taken as a recommendation on information to include interfaces of automated decision support applications; it is simply information that is useful in interpreting the results of the prototype simulation as it executes.

Information other than the map displayed by the decision support system is as follows:

- The window’s title bar shows the organization’s name.
- The lower left table shows the organization’s holdings of battlefield object types (materiel, persons, and other organizations). Holdings are derived from rows of the GH table HOLDING associated with the organization. The quantity column is derived from the holding-total-quantity column of the Generic Hub. (The prototype does not currently distinguish between total and operational holdings.)
- Threats to the organization appear on the right side as a list of names of hostile organizations. Figure 5 shows one such organization, named “3C Armor Recce Coy”. Note that the list is headed by the word “Yes”, in red, which aids in recognition of a

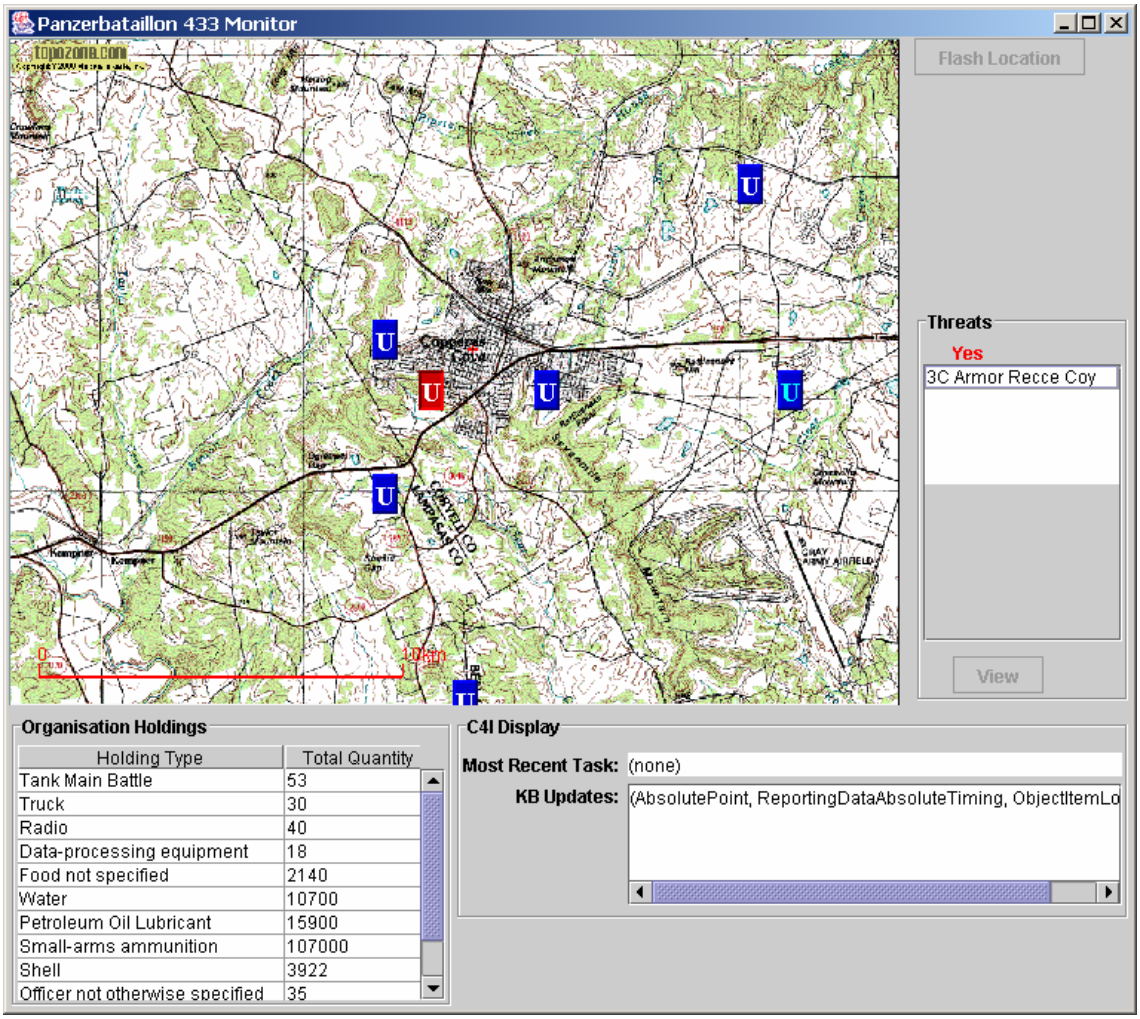

Figure 5. Friendly Organization Main User Interface

situation that needs attention. If there are no detected threats, the list is headed by "No", in black.

The user may select a threat in the list and click the View button below to see details on a threat that may aid decision makers (Figure 6). Currently the information is derived from the estimated time until the hostile organization reaches the friendly organization's location, assuming the hostile organization continues at its current bearing angle and speed. A more sophisticated decision support system would incorporate terrain and weaponry into the analysis.

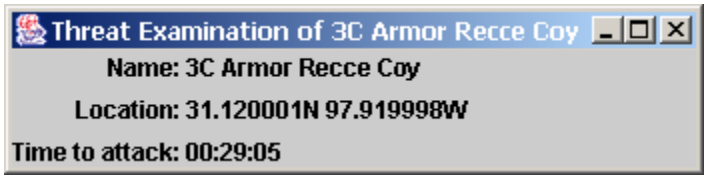

Figure 6. Example Threat Details

- The area in the lower right shows C4I information generated and received from other friendly organizations. As part of decision making, a friendly organization may formulate tasks (expressed as rows in the ACTION-TASK table plus rows in associated ta-

bles). Its current (i.e., most recent) task is displayed in the Most Recent Task field. Furthermore, a friendly organization sends information it formulates to other friendly organizations and, conversely, receives updates from these organizations. These updates eventually translate into updates to a friendly organization's knowledge base. They are displayed in the KB Updates area.

- One possible update is a new task received from a commanding organization. Such an update is displayed in both the Most Recent Task field and the KB Updates area. In the Most Recent Task area, it summarizes the friendly organization's current objective. In the KB Updates area, it shows the friendly organization's communications with other friendly organizations.
- The Most Recent Task field provides information that may be useful in formulating a decision; for example, whether the current task can be overridden. The KB Updates area is mainly gives a sense that the system is functioning correctly in that it is generating and receiving information. It would probably be of little value in a real decision support application.

Friendly organizations send updates to the knowledge base using Generic Hub elements. That is, when one friendly organization wants to notify others of updates, it sends a message containing a set of Generic Hub table rows. This is one of three possible formats. The other two are:

1. Use a custom message format.
2. Use elements in the GH ontology.

Using Generic Hub elements seems most logical. The Generic Hub is an accepted standard. Devising any custom format would entail work that would probably duplicate effort already put into the Generic Hub. The approach using elements of the GH ontology has some merit, but it uniquely identifies an instance by frame name, which is application specific, as opposed to Generic Hub keys, which can be considered more readily interoperable, by, for example, implementing an RDBMS key management that ensures global uniqueness within the entire operational environment. The only difficulty with using Generic Hub elements is the difficulty of translating between the knowledge base and the data set. Section 3.6.4 takes up this point.

A friendly organization also displays threat notification and threat response information. Threat notification information appears immediately upon detection of a threat. It is presented in a pop-up window (Figure 7). The threat list on the right side of the map window is initially redundant with respect to this pop-up, but the threat lists persists for as long as the threat exists.

Threat response information gives decision makers a set of possible responses to a threat. The decision support application generates it automatically upon detection of a threat and presents it as soon as it is ready; this follows shortly after the threat notification. As Figure 7 shows, it contains a list of possible courses of action in response to a threat. The intent is that these courses of action be generated by rules derived from doctrine, and thus useful to decision makers as justifiable. Each course of action has a rating, which is a number between 0 and 1, inclusive. A rating of 1 means the course of action is always justifiable according to doctrine. A rating of 0 means the course of action is never justifiable. A rating in between indicates the degree of justifiability. By default, courses of action with a rating of 0 are not shown.

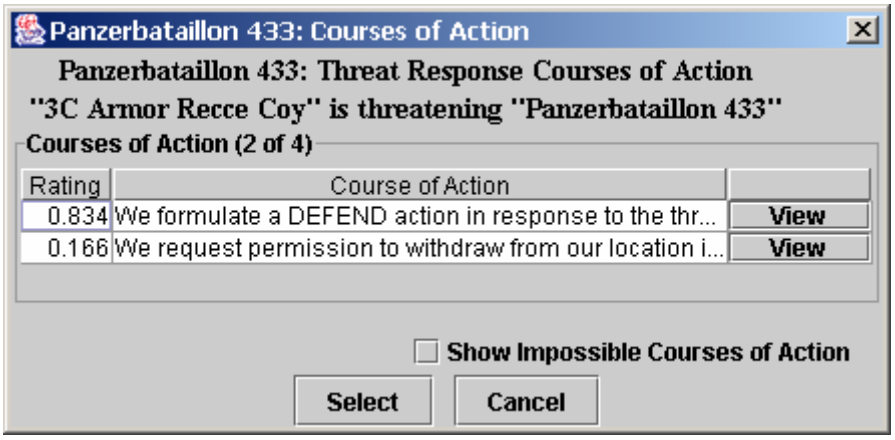

Figure 7. Threat Response: Courses of Action

If a user clicks the View button to the right of a course of action, the decision support application pops up a window containing the rationale for the rating (Figure 8). Figure 8 is notional; it should provide enough information for a decision maker to understand how the decision support application calculated the threat's rating. The decision maker may agree or disagree with the logic but must be able to understand it.

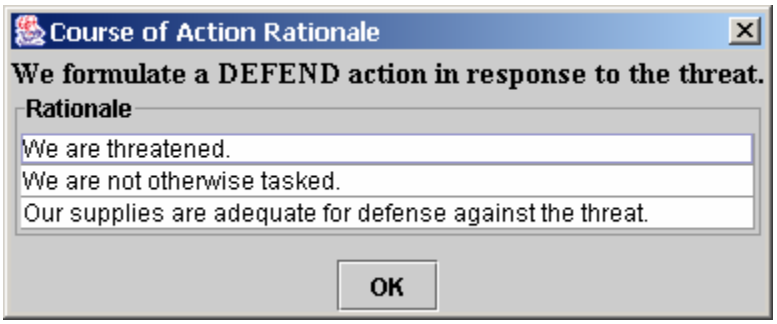

Figure 8. Example Rationale

A friendly organization does not have to be associated with a set of decision makers. If it is not, it has no user interface, and operates autonomously.

A friendly organization that is associated with a Generic Hub data set periodically updates its knowledge base from that data set. It searches the REPORTING-DATA table for rows added more recently than its last update. If it finds any, it creates instances of ReportingData in its knowledge base. It also determines what observation the reporting data refers to and creates instances in the knowledge base to model them.

A friendly organization also updates its Generic Hub data set. Whenever a friendly organization generates or infers information that can be expressed in terms of the Generic Hub data model, it converts the frames modeling that information into Generic Hub format, then invokes its associated RDBMS to save the data.

On start-up, a friendly organization has the following parameters:

1. The URL of a GH ontology.
2. The URL of a Generic Hub data set. This parameter is optional; if not given, a canned set of data is used instead.
3. The name of the organization. This name must exist in the Generic Hub data set.

4. The interval, in seconds, between updates of the knowledge base from the underlying Generic Hub data set. This parameter is optional. If not given, the interval defaults to 60 seconds.
5. The interval, in seconds, between analyses of the knowledge base for threats. This parameter is optional. If not given, the interval defaults to 50 seconds.
6. The interval, in seconds, between searches for system components that formulate courses of action in response to detected threats. This parameter is explained in Section 3.2.2. It is optional. If not given, the interval defaults to 50 seconds.
7. The interval, in seconds, between searches for system components that synchronize changes to Generic Hub data sets. This parameter is explained in Section 3.2.2. It is optional. If not given, the interval defaults to 50 seconds.
8. The URL of a map specification document (see below).
9. A name that is unique with respect to names of all other system components that will execute on the computer running the friendly organization.

A friendly organization requires a map specification document that describes the location and content of a graphic representing a map image to display in a decision support application. A map specification document is an XML document. A map specification document must conform to the XML Document Type Definition (DTD) shown in Figure 9.

```
<!ELEMENT map-spec (map-URL, center-point, north-west-point)>
<!ELEMENT map-URL (#PCDATA)>
<!ELEMENT center-point EMPTY>
<!ATTLIST center-point
        latitude CDATA #REQUIRED
        longitude CDATA #REQUIRED>
<!ELEMENT north-west-point EMPTY>
<!ATTLIST north-west-point
        latitude CDATA #REQUIRED
        longitude CDATA #REQUIRED>
```

Figure 9. DTD for Map Specification

The map-URL element specifies the URL of the graphic image to use for the map. The image must be rectangular; the center-point and north-west-point elements specify its properties. Attributes of these elements must be given in decimal degrees. Figure 10 gives an example.

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE map-spec SYSTEM 'file:///C:/map-spec.dtd'>
<map-spec>
    <map-URL>file://images/KilleenNW.png</map-URL>
    <center-point longitude="-97.7426" latitude="31.1419"/>
    <north-west-point longitude="-97.8490" latitude="31.2104"/>
</map-spec>
```

Figure 10. Example Map Specification XML Document

2.3. SimState

A Generic Hub data set mainly represents observations, especially with respect to instance of battlefield objects. It may specify an organization's position, but that and the organization's

actual position may differ. In any case, it qualifies observations, noting that position is as of some specified time, and given by an organization of specified credibility.

Friendly organizations have intelligence sources that provide observations of the battle space. It is the responsibility of a friendly organization to update a Generic Hub data set based on intelligence source observations.

The prototype simulation, then, must simulate intelligence sources surveying the battle space for battlefield objects during each execution. This could be accomplished by querying the objects directly, but having a friendly organization query an enemy organization is illogical. Instead, the prototype has a SimState (Simulation State) component. When a battlefield object changes (or initializes) a component of its physical state that can be described in terms of the Generic Hub, it is responsible for communicating that change to the SimState component. Intelligence sources may query the SimState component; as part of the query they provide certain parameters (e.g., for an electronic sensor, center point and radius of the area to survey) and the SimState component responds with the set of battlefield objects the source could detect using those parameters, based on the most recently provided state information.

Currently the SimState component only recognizes organizations as battlefield objects, i.e., it will not record changes to the state of any battlefield object other than an organization. Its model of intelligence sources is limited to providing information on organizations within a rectangular area specified by two points: a center and a northwest. This primitive implementation could easily be extended with different sensor types.

On start-up, the SimState component may be provided with a name that is unique with respect to all other components executing on the computer running the SimState component. This name is optional. If omitted, it defaults to ssa.

2.4. The DBMS

The Generic Hub data model provides a physical schema for implementation as a relational database. Any realistic application must expect to access a Generic Hub data set through an RDBMS. The Generic Hub's structure is too complex for a flat file implementation. Non-trivial data sets require indexing operations for acceptable access times. Simultaneous access by multiple applications will require some concurrency mechanism. These features are all characteristic of functionality offered by an RDBMS.

The prototype, therefore, uses an RDBMS to store GH-conformant data sets. Aside from the above features, an RDBMS offers the following:

- *Network access.* The location of the Generic Hub data and the computer on which the DBMS runs need not be tied to the computer running any other system component.
- *Back-up and restore.* Snapshots of database state can be captured and reloaded; this helps establish separate simulation configurations.
- *Transaction management.* This simplifies error recovery.
- *Data replication.* Some RDBMSs can automatically replicate data across multiple data sets. This eliminates the need for friendly organizations to exchange Generic Hub-formatted messages.

Start-up parameters for the RDBMS depend on its implementation. Vendor-supplied documentation should be consulted to determine them. The prototype requires that the RDBMS be accessible via a network protocol such as JDBC.

2.5. Overview

The Overview component complements the SimState component by giving a picture of ground truth. It shows the location of all organizations. See Figure 11.

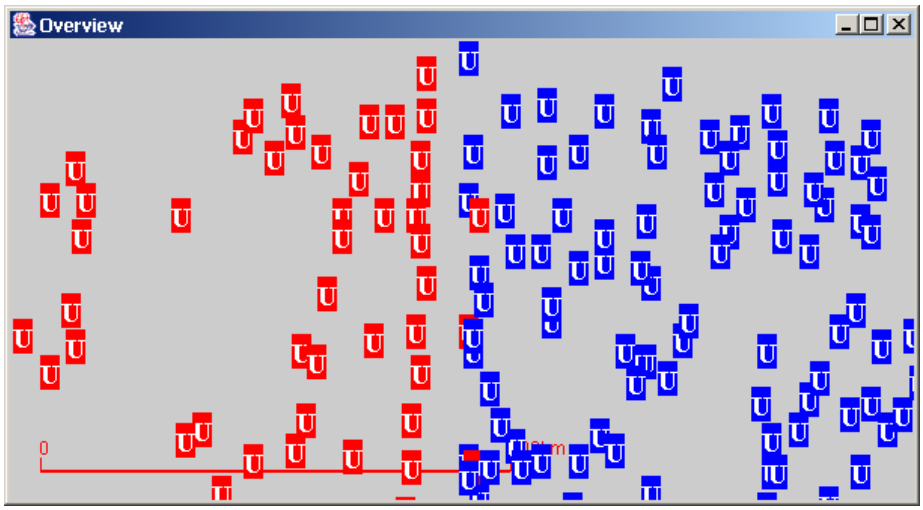

Figure 11. The Overview Component GUI

The Overview component is used by an administrator to monitor the progress of a simulation. It is optional; that is, a simulation can take place without an Overview component.

On start-up, the Overview component requires as a parameter a name that is unique with respect to names of all other system components that will execute on the computer running the Overview component.

2.6. Loader

The Loader component initiates a simulation. It parses a configuration document to determine information on the above components: which ones to include in the simulation. It then launches the components.

Use of the loader is optional. However, it greatly simplifies the initialization process and promotes repeatability. Furthermore, its user interface (shown in Figure 12) allows some visibility into and control over a simulation. A controller can start all enemy organizations moving (assuming their speed, bearing angles, and position refresh rates have been set), stop them, and shut down a simulation with a single button click.

The configuration document is an XML document conforming to the DTD shown in Figure 13. The document must be of type loader-config. This element consists of:

- A set of elements specifying the friendly and enemy organizations to participate in the simulation.

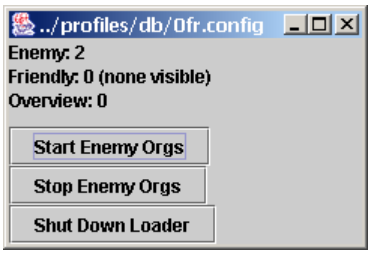

Figure 12. Loader GUI

```
<!ELEMENT c4i-config ((friendly-org-spec | enemy-org-spec)*, DB-spec?)>
<!ATTLIST c4i-config
        DB (false|true) "false"
        overview-agent (off|on) "off"
        ontology-URL CDATA #IMPLIED>
<!ELEMENT enemy-org-spec (org-name)>
<!ATTLIST enemy-org-spec
        speed CDATA #IMPLIED
        bearing-angle CDATA #IMPLIED
        visible (false|true) "true"
        position-refresh-rate CDATA #IMPLIED>
<!ELEMENT friendly-org-spec (org-name, map-spec)>
<!ATTLIST friendly-org-spec
        threat-response-agent-search-interval CDATA "45"
        threat-search-interval CDATA "60"
        visible (false|true) "true"
        DBUpdate-agent-search-interval CDATA "100"
        ontology-refresh-interval CDATA "75">
<!ELEMENT org-name (#PCDATA)>
<!ELEMENT map-spec (#PCDATA)>
<!ELEMENT DB-spec (URL, (user-name, password?)?, schema-map, driver?)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT user-name (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT schema-map (#PCDATA)>
<!ELEMENT driver (#PCDATA)>
```

Figure 13. DTD for Loader Configuration

- An optional element specifying the DBMS that components will use.
- Attributes that specify:
    - Whether to use a RDBMS. If true, the RDBMS element must be given. If false, a canned data set will be used, and RDBMS-related functionality will be lost.
    - Whether to display an Overview component.
    - The URL of a GH ontology. (This attribute is optional in the XML document, but if missing, it must be given as a property to the Java runtime environment. See Section 3.6.8.)

The element content covers the start-up parameters for each component.

The loader allows only one RDBMS to be specified. All components the loader initiates share this RDBMS. A simulation can access multiple RDBMSs by invoking the loader repeatedly, each time with a different configuration document. Individual components may also be invoked independently of the loader, and their start-up parameters may include different RDBMS URLs.

The RDBMS element contains URL and schema map elements, both of which are required. It may also include a user name and password. Some DBMSs require a user name and pass-

word; moreover, the user name and password sometimes determine the user's table space, and therefore specify the particular Generic Hub data set accessed. Such an RDBMS would permit access to multiple Generic Hub data sets by creating different user accounts and associating each account with a unique table space.

The RDBMS element also contains an optional driver element. This element specifies a Java class implementing a JDBC driver manager.

The design of the loader does not attempt to consider all variabilities of RDBMS access. Other elements may be necessary to access other RDBMSs.

3. Technical View

This section presents design and implementation decisions for each component in the System View. It discusses the technologies used to implement the simulation prototype, showing how they contribute to an agile, easily extendable design. It covers the simulation prototype components in terms of their task structure and their packaging (i.e., breakdown into modules). It gives a detailed discussion of selected components that are especially interesting in the implementation of a net-centric system.

In compliance with principles of net-centricity, the system is implemented as a set of interacting agents. These agents base their analyses on ontologies and knowledge bases insofar as is practical. This is as opposed to implementing business rules in a programming language.

3.1. Underlying and Supporting Technologies

Many technical decisions on the design of the prototype simulation stem from the technologies used in its development:

- The prototype simulation is implemented in Java [Java]. All source code is written in Java, is written in a language for which a Java parser generator exists, or is written in a language that is interpreted by components written in Java.
- The primary system for manipulating ontologies and knowledge bases is Protégé-2000 [Protégé 2004]. Extensions to knowledge base functionality derive from plug-ins available for Protégé-2000. (Some design decisions are based on hypothesized plug-ins.)
- The system for implementing agent-based capabilities is FIPA-OS [FIPA-OS]. Because FIPA-OS has support for interprocess communication, it is also used to implement functionality for certain components that are not true agents but are most easily or realistically implemented as distinct processes.
- The system for translating between XML documents and Java objects is Zeus [Enhydra]. Zeus uses Document Type Definitions (DTDs) rather than the newer XML Schema Definitions (XSDs). DTDs are not as powerful as XSDs, but they are much easier to create and use, and they are sufficiently expressive for this prototype simulation.

The following discussion of these technologies will aid in understanding the subsequent detailed discussion of system design.

3.1.1. Java

Many features of Java make it an excellent language in which to implement this prototype simulation. Java is by design platform-independent, freeing implementers from platform-

specific considerations. Its standard libraries provide useful capabilities for letting an application interface with its environment. Many freely available packages provide cutting-edge functionality; perhaps more significant, many freely available packages (including the standard libraries) provide key basic functionality such as sorting and searching.

Java's ability to load classes at run-time adds considerable flexibility. Dynamically declared classes can be used to decouple components of a program; Java's SQL package uses dynamic classes to manage the JDBC driver used to connect, for example, a program to an RDBMS. Dynamic classes are also a useful implementation technique for converting data exchanged between applications (such as agent messages) into Java objects. FIPA-OS and Zeus use dynamic classes for this purpose. So does the prototype simulation.

Java's standard libraries formalize the concept of a URL and make access to, and reading of, a URL simple and straightforward. This greatly simplifies system configuration by decoupling an application from its data.

3.1.2. Protégé-2000

Protégé-2000 is often invoked with a user interface that lets it serve as an ontology definition tool and a knowledge base editor, but it also provides an API that an application may use to access that same editing functionality. The Protégé-2000 data model is based on OKBC. The API provides Java classes and interfaces that provide direct access to the data model. These classes and interfaces are contained in the package edu.stanford.smi.protege.model.

The interfaces specify an OKBC view of data. They define the hierarchy shown in Figure 14. The API of the top-level interface, Frame, specifies methods needed for all interfaces:

- Get/set methods to manipulate a frame's name, own slot values, and facet values.
- Get methods to obtain a frame's own slots, certain properties associated with own slots (e.g., the number of values an own slot possesses), and facets of own slots.
- Add/remove methods to affect facets, and also to change the own slot value for a frame of multiple cardinality.
- A method to obtain the knowledge base in which the frame exists. (Protégé-2000 does not permit multiple knowledge bases to share a frame.)
- Boolean-valued methods to test properties of a frame, including validity (currently validity means only that Protégé-2000 allocated the frame correctly) and editability. Corresponding set methods are defined to set these properties.

Applications usually deal with subinterfaces of Frame rather than directly with Frame itself.[2] Instance, the direct descendent of Frame, adds get/set methods that make a frame an instance of a class. Most applications want frames that are instances of classes, so Instance is more useful than Frame. (A comment in the API documentation for Instance indicates that Protégé-

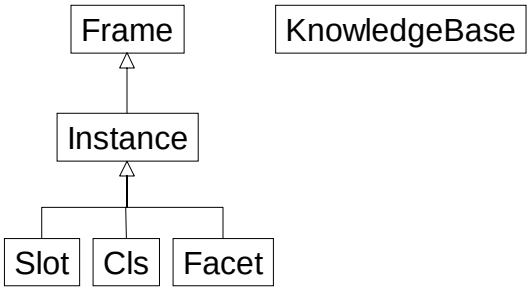

Figure 14. Protégé-2000 Components of OKBC Model

2000's designers considered, but decided against, merging the Frame and Instance interfaces.[3])

The Cls interface (so named because Java defines Class as a class in its language standard) extends Instance with:

- Get/set methods to manipulate template slots, direct subclasses, and direct superclasses. Certain properties of template slots, such as cardinality, can also be manipulated.
- Add/remove methods to affect the template slots, template slot values, and template facet values of a class.
- Get methods that obtain class properties, including the number of direct and total instances, the superclasses, and the subclasses.
- A method to create a direct instance (i.e., an object of type Instance) of a class. The own slots of the instance are undefined unless the corresponding template slots have a default value.
- Boolean methods to test class properties such as:
    - Whether the class is abstract or concrete.
    - Whether the class possesses a specified template slot.
    - Whether the class overrides the definition of a template slot or facet inherited from a superclass.

The Slot interface specifies methods that get or set properties of a slot (but not its value; a Frame specifies value properties of an own slot, and a Cls specifies value properties of a template slot). These properties include cardinality, value type, allowed classes (if the value type is "Instance") allowed parents (if the value type is "Class"), and inverse. In the Protégé-2000 model, a slot is represented as an instance of class :SLOT. Therefore the Slot interface extends Instance.

The Facet interface extends Instance with get/set methods for placing constraints on slots. In Protégé-2000, facets add little power, because their capabilities are all implemented as part of knowledge base functionality. The prototype simulation does not use the Facet interface.

Frames are part of a knowledge base. The KnowledgeBase interface specifies methods for manipulating frames. These include:

- Get methods to obtain all extent frames, instances, classes, slots, and facets.
- Get methods that obtain properties (e.g., count of all instances).
- A set method that specifies whether Protégé-2000 should validate the value of an own slot with respect to its facets whenever its value changes.

Protégé-2000 supplies implementations for all these interfaces. An application accesses them by creating an instance of the Project class, usually via the static method Project.loadProjectFromFile(). The parameters to this method include the name of a file (it may also be a URL) containing a Protégé-2000 "project". A project specifies an ontology, a knowledge base (which may be empty), and certain details useful to Protégé-2000's knowledge base editor; it is the means by which both users and applications access a knowledge base. Applications using Protégé-2000's API usually contain code similar to the following:

---

[3] http://protege.stanford.edu/doc/pdk/api/index.html; under package edu.stanford.smi.protege.model, select interface Instance.

```
Collection errors = new LinkedList();
Project project = Project.loadProjectFromFile("project.pprj", errors);
if ( errors.size() > 0 )
        throw new Exception("Cannot load project");
KnowledgeBase kb = project.getKnowledgeBase();
```

The upshot of this code is that Protégé-2000 has supplied an instance of a knowledge base that can subsequently serve to access all other interfaces of the model. For example, an application can create an instance of a Unit and set its slots as follows:

```
Cls unitCls = kb.getCls("Unit");
Instance unit = unitCls.createDirectInstance(null);    // null makes Protégé-2000 generate a
                                                       // unique frame name automatically.
Slot tiv = kb.getSlot("type-instance-value");          // We'll use this each time we set a slot's value.
// Set the unit's name to "1 (DA) AA Company".
Instance unitName = kb.getCls("Object-Item-Name").createDirectInstance();
unitName.setOwnSlotValue(tiv, "1 (DA) AA Company");
unit.setOwnSlotValue(kb.getSlot("name"), unitName);
// Set the unit's formal abbreviated name to "1C(AA)-1BD".
Instance formalAbbreviatedname = kb.getCls("Unit-Formal-Abbrev-Name").createDirectInstance();
formalAbbreviatedName.setOwnSlotValue(tiv, "1C(AA)-1BD");
unit.setOwnSlotValue(kb.getSlot("formalAbbreviatedName"), formalAbbreviatedName);
// Set the unit's category code to "UN". Note that we don't create a value;
// we use values in the DS149_org_cat_code enumerated type.
Slot eelSlot = kb.getSlot("enumerated-element-label");
Collection orgCatCodes = kb.getCls("DS149_org_cat_code").getInstances();
Instance unInst;
for ( Iterator ccIter = orgCatCodes.iterator(); ccIter.hasNext(); ) {
        unInst = (Instance)ccIter.next();
        Instance element = (Instance)unInst.getOwnSlotValue(tiv);
        if ( element.getOwnSlotValue(eelSlot).equals("UN") )
            break;
}
unit.setOwnSlotValue(kb.getSlot("categoryCode"), unInst);
```

Later, an application can associate a location with this unit:

```
Instance location = …;      // We assume these instances are
Instance speed = …;         // obtained from somewhere.
Instance objItemLoc = kb.getCls("ObjectItemLocation").createDirectInstance();
unit.addOwnSlotValue(kb.getSlot("is-geometrically-defined-through-ObjectItemLocation"),
                     objItemLoc);
location.addOwnSlotValue(kb.getSlot("provides-geometric-definition-for-ObjectItemLocation"),
                         objItemLoc);
objItemLoc.setOwnSlotValue(kb.getSlot("speedRate"), speed);
```

It's worth noting again that Protégé-2000 only verifies a set operation if the method kb.setValueChecking() has been invoked with a true value for its parameter. These checks can be time-consuming, so by default Protégé-2000 places the onus of verification on the application.

3.1.3. FIPA and FIPA-OS

The Foundation for Intelligent Physical Agents (FIPA) [FIPA] is a standards organization that has written and published numerous standards for the design and implementation of agent-based systems. The core of FIPA is an abstract architecture that defines elements of an environment of communicating agents. This architecture has, at its most abstract, four elements:

1. The message transport element, which defines how messages exchanged by agents will be communicated across a network.
2. The agent communication language (ACL) element, which standardizes the structure of messages in terms of a set of parameters with predefined meanings.
3. The agent directory element, which specifies how agents are to announce their presence and how other agents may detect them.
4. The service directory element, which specifies how agents can characterizes the services they offer, and how other agents may search for services.

FIPA states that one (optional) parameter of an ACL is the ontology. In other words, an agent's message may include a reference to an ontology that defines the content of the message and, depending on the sophistication of the ontology, lets other agents interpret message content. The degree to which message interpretation is possible varies from recognizing the name of an ontology (and assuming that name refers to the same ontology one has in mind) to ontology translation services based on first-order predicate calculus.

FIPA-OS is an implementation of many of the FIPA standards. In particular, it fully implements the abstract architecture, and thereby provides a foundation upon which all other FIPA standards can, or have been, implemented.

3.1.3.1. Agents

Figure 15 shows the life cycle of a FIPA-OS agent. An agent is an instance of a subclass of class FIPAOSAgent (in Figure 15, assume class MyAgent is such a class). For an agent to operate, an Agent Management System (AMS) and a Directory Facilitator (DF) must be running in the environment hosting the agent (usually the host computer). Agent *a*'s first order of

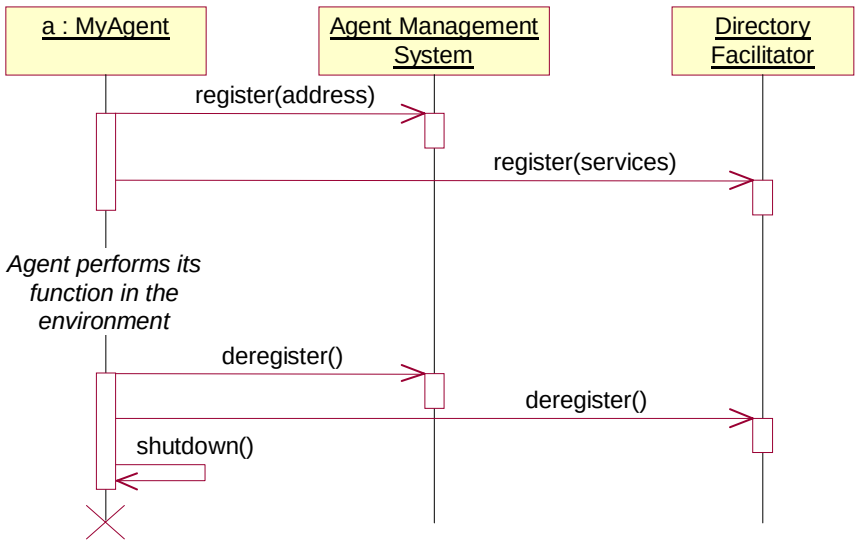

Figure 15. FIPA-OS Agent Lifecycle

business is to register itself with the AMS, supplying its name and address. The AMS, which is FIPA-OS's name for the agent directory element, notes these items; *a* has effectively declared itself as part of the environment, and can now be contacted by other agents. The address *a* gives suffices to let other agents in the environment identify it uniquely, and to contact it. Depending on how the AMS is configured, this environment may be as narrow as the host computer or as wide as the entire Internet.

The agent next registers its services with the DF, which is FIPA-OS's name for the service directory element. The purpose of the DF is to help agents search for services they require. FIPA-OS lets a service be specified in several ways. The agent may simply give its name, implying that its services are standard enough that other agents can be expected to know about them. The agent may also state the names of ontologies it understands, names of languages it supports (natural, artificial, or both), and names of communication protocols it uses. Finally, an agent may provide one or more service descriptions. A service description has a name and a set of agent-specific properties, each of which is a name/value pair. The implication of a property is that its name is something for which an agent might search, and its value will inform an agent of whether the service is useful.

An agent may use any combination of these search terms. Providing a name and a service description is perfectly legal. In fact, a service description may also specify ontologies, languages, and protocols. This might be particularly useful in multilingual environments; an agent could announce that it provides services for the Generic Hub, the Moyeu Générique, and the Generische Nabe, specifying each as a separate service description with property names and values of the GH ontology in respective languages.

In all cases, agents conduct searches by name. The assumption is that names are sufficiently powerful and unambiguous to identify desired services. The validity of this assumption remains to be seen.

After an agent has registered with the AMS and the DF, it begins performing its assigned tasks; when it has finished, it deregisters, then shuts down by invoking the shutdown() method in the FIPAOSAgent superclass. This completes its life cycle.

3.1.3.2. Tasks

Agents, by their nature, engage in synchronous and asynchronous communication with each other. An agent is inherently a multi-tasking entity. Java's Thread class provides support for multiple tasks, but a thread is more general than necessary for agent implementation. FIPA-OS provides an abstract Task class that is tailored to agent needs. It provides methods for:

- Searching for agents.
- Initiating, performing, and terminating conversations with agents.
- Creating a hierarchy of tasks.
- Gracefully terminating a task.

Similar to agents, an application extends the class Task to create a concrete class implementing specific functionality. Because tasks are created in a hierarchy, the agent invokes the root task, then waits for it to terminate. The paradigm for implementing an agent is therefore:

```
public class MyAgent extends FIPAOSAgent {
    public MyAgent(String agentName) {
        super(profile, agentName, "FIPA-OS", true);
        registerWithAMS();                    // Automatically supplies the agent's name.
```

```
            registerWithDF(agentName);          // Can also give a service description.
            setListenerTask( new MyTask() );    // Now MyAgent will wait until MyTask terminates.
        }
        /** Callback method invoked automatically by FIPA-OS when MyTask terminates. */
        public void doneMyTask(Task t) {
            deregisterWithDF();                 // De-register in the opposite order of registration!
            deregisterWithAMS();
            shutdown();
        }
    }
```

Aside from MyTask(), all the methods invoked are defined in FIPAOSAgent.

The task hierarchy an agent uses depends entirely on the functionality the agent provides. A typical agent has both periodic and aperiodic functionality. It may be proactive, reactive, or both. Each functional category will occupy its own place in the hierarchy.

As an example, consider enemy organizations. Each enemy organization is implemented as an agent. An enemy organization periodically:

1. Updates its position.
2. Sends its position to the SimState component.
3. Sends its position to any Overview components.

The SimState and Overview components are implemented as agents that register themselves and their services. The second and third of these actions require locating these agents. It is assumed that one SimState component always exists – the system cannot run otherwise – but Overview components may come and go during execution. An enemy organization must periodically search for Overview components.

Based on these considerations, the agent that implements an enemy organization uses the task structure shown in Figure 16. The root task is named IdleTask; this is a FIPA-OS convention. Its role is to start the other tasks, await their completion, then terminate, at which time FIPA-OS will attempt to invoke the doneIdleTask() method of the agent that started it. If this method exists and is public, it will likely contain code similar to the canonical agent presented above, so the agent will terminate. (If this method does not exist, the agent must have another strategy to terminate.)

The Idle Task starts three tasks. Each of these tasks operates independently:

- The SimState Agent Search Task uses FIPA-OS to search the DF for the SimState

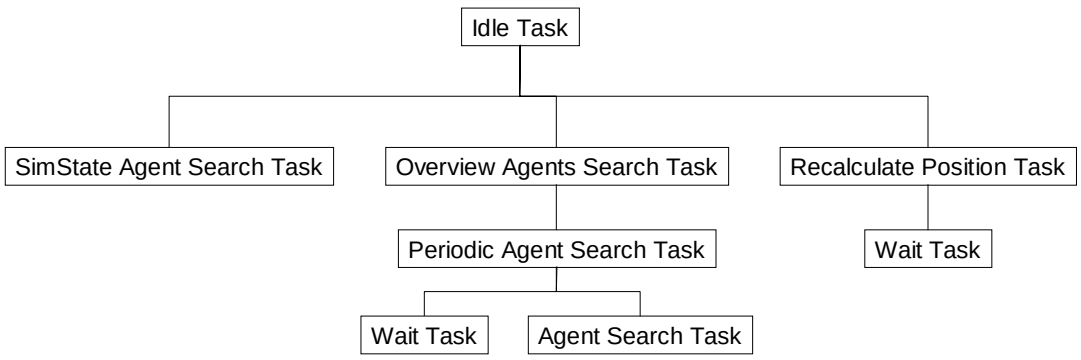

Figure 16. Enemy Organization Agent Task Hierarchy

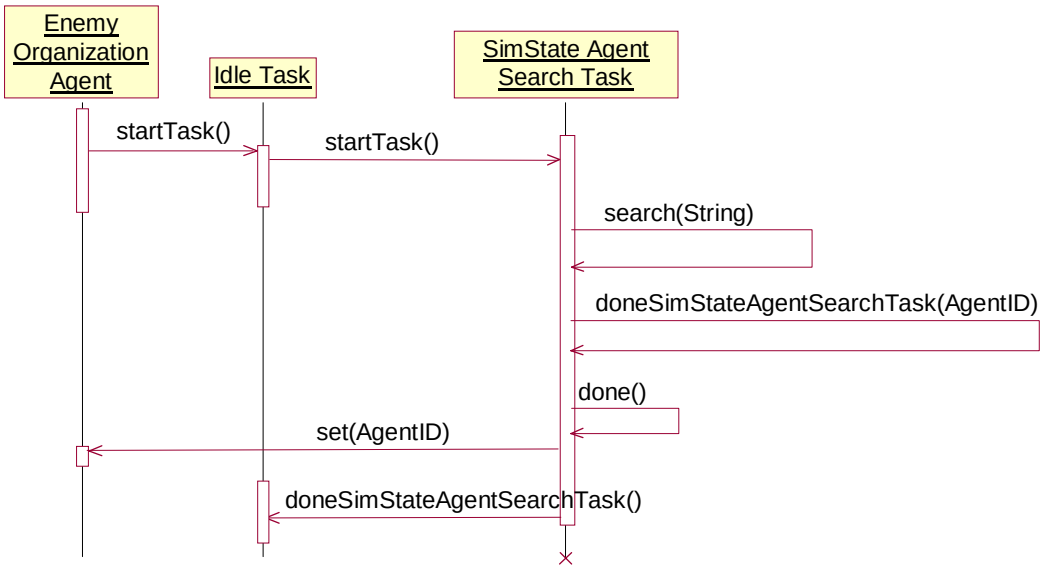

Figure 17. SimState Agent Search: Task Interactions

agent. Upon finding the SimState agent, FIPA-OS passes the SimState Agent Search Task an ID that unambiguously identifies the SimState agent, and that can be used in subsequent communications. The SimState Agent Search Task in turn passes this ID to the Enemy Organization agent, then terminates. See Figure 17.

- The Overview Agents Search Task invokes a Periodic Agent Search Task; as its name implies, this task performs a search, waits for a specified period of time, then repeats this search/wait cycle, continuing until instructed by its parent task to terminate. FIPA-OS provides an implementation of the Wait Task. To use it, one instantiates the WaitTask class, passing the desired wait time in milliseconds as a parameter. When this time has elapsed, FIPA-OS invokes the method doneWaitTask() in the instance that started the Wait Task.

  Figure 18 below shows canonical Java code for implementing a periodic task. The Overview Agent Search Task conforms to this paradigm, substituting an invocation of Agent Search Task for the generic performSomeAction().

- The Recalculate Position Task differs from the other two tasks in that it is launched by the Idle Task not automatically but by external command. This command comes either from an enemy organization controller via the agent's user interface (when the controller checks the Organization Moving box; see Figure 3) or from the loader. Once it starts, it uses the paradigm in Figure 18, performing the necessary mathematics in the doneWaitTask() method to compute the organization's new location.

The Recalculate Position Task may also be stopped via external command. The enemy organization agent is implemented such that the Idle Task receives the command; when it does, it tells the Recalculate Position Task to stop. (In FIPA-OS, a parent task cannot kill its child; it only receives notification of changes to a child's state.)

3.1.3.3. Messages and Conversations

Agents send messages to each other. Every message has an associated *communicative act* that describes the intent of the message. FIPA specifies a predefined set of standard communicative acts, and provides a formal semantic model of each. Examples of communicative acts include:

- *Inform*: The sender informs the receiver that the sender believes a given proposition is true.
- *Not Understood*: The sender informs the receiver that it believes the receiver has performed some action, but it does not understand that action. The most common use for this communicative act is for the sender to notify the receiver that it does not understand the contents of an Inform communicative act the receiver sent to the sender earlier.
- *Request*: The sender requests the receiver to perform some action. Usually the action requests that the receiver respond to the sender with an Inform communicative act.
- *Failure*: The sender informs the receiver that it attempted, but failed, to perform some action.

In FIPA, every message is part of a conversation. A conversation prescribes a message sequence according to an ordering of communicative acts. This ordering is known as the protocol. FIPA-OS supplies several protocols, and lets applications specify their own. Examples of FIPA-OS protocols include:

- **No-Protocol**: The simplest protocol, it defines a one-message conversation that allows any communicative act. In other words, the sender sends the receiver an arbitrary message and expects no reply.
- **FIPA-Request**: In this protocol, agent *A* requests some action of agent *B* (typically that *B* supply information), and agent *B* either agrees or refuses to participate in the

```
public class PeriodicTask extends Task {
    private int delayTime = 10000;

    /** Callback method invoked automatically when the task starts. */
    public void startTask() {
        newTask( new WaitTask(delayTime) );       // Delay for a while.
    }

    /** Callback method invoked automatically when a WaitTask terminates. */
    public void doneWaitTask(Task t) {
        performSomeAction();                      // Do what PeriodicTask is supposed to do.
        newTask( new WaitTask(delayTime) );       // Now repeat the cycle.
    }
}
```

Figure 18. Canonical Code for Periodic Task

conversation. If agent *B* agrees, it attempts to perform the action; it then sends to *A* a message with either an Inform or a Failure communicative act.

Protocols can specify conversations of little or great complexity. A conversation can contain cycles, so agents can communicate as much as they need for as long as they need.

In FIPA-OS, most conversations are initiated by tasks. (Agents may also initiate conversations, but methods in the Task class provide better support.) A task constructs a conversation using the getNewConversation() method, which takes a protocol name as its parameter and returns an ACL message whose communicative act is set to that of the first communicative act for the protocol. The task then fills in the content of the ACL message and invokes the forward() method, which sends the ACL message to the recipient:

```
ACL acl = getNewConversation("fipa-request");        // The parameter is the protocol's name.
acl.setReceiverAID(agent ID of receiver);
acl.setContent(message content);
acl.setOntology(message ontology);
forward(acl);
```

The sender is assuming the receiver knows how to handle a conversation that uses the FIPA-Request protocol.

A task handles conversations by declaring methods with names based on the communicative acts. For the above example, the receiver will handle the conversation if it declares a method as follows:

```
/** Callback method invoked on receiving a message with a Request communicative act. */
public void handleRequest(Conversation c) { ... }
```

From the Conversation parameter, the method can extract the message and determine its content, ontology, sender, etc.

FIPA-OS supports conversations with a more complex protocol by maintaining within an agent the state of the conversation, i.e., the sequence of messages already sent and the valid next messages (including whether the conversation is over). Thus the body of a conversation-handling method looks something like the following:

```
public void handleRequest(Conversation c) {
    ACL acl = c.getACL(c.getLatestMessageIndex());          // Get the message to handle.
    if ( ! acl.getProtocol.equalsIgnoreCase("fipa-request") ) {
        sendNotUnderstood(acl);
        return;
    }

    try {
        // Perform task-specific actions that generate a response.
        Object result = interpret(acl.getOntology(), acl.getContent());
        ACL response = getResponse(c, "inform");
        response.setContent(result.toString());
        forward(response);
    } catch ( InterpretationException e ) {                 // Application-defined execption.
        // Could not complete the action.
        ACL response = getResponse(c, "Failure");
```

```
            response.setContent(acl.getContent());          // Let sender know what failed.
            forward(response);
        }
    }

    /** Gets from a conversation the next message with the specified performative ("performative" is
     * the FIPA-OS name for a communictive act). */
    private ACL getResponse(Conversation c, String performative) {
        for ( Iterator i = c.getNextMessages().iterator(); i.hasNext(); ) {
            ACL acl = (ACL)i.next();
            if ( acl.getPerformative.equalsIgnoreCase(performative) )
                return acl;
        }
        return null;
    }
```

Most methods in a task are devoted to handling communicative acts.

A message has exactly one sender but may have multiple recipients. FIPA-OS creates in the sending task a separate conversation thread for each recipient. Usually the sending task will want to create a child task to handle each thread.

3.1.3.4. Ontologies, Languages, and Message Content

The heart of a message is its content. Every non-trivial message has content that informs the receiver of the sender's intent: a query, a response, unsolicited information, etc. The challenge of an agent-based environment is to ensure that the sender provides enough information to allow the receiver to unambiguously interpret content.

FIPA defines three message parameters that specify semantics. The first is the communicative act. In the context of "Inform", the content "time flies" could be interpreted to mean that things happen quickly, whereas in the context of "Request" it could mean that the recipient is to measure the speed of winged insects, or perhaps batted baseballs.

The second parameter is the ontology name. The elements of the ontology should allow the receiving agent to identify the entities and actions of the content (e.g., insects versus baseballs).

An ontology is identified by name. FIPA does not require the receiver to implement the ontology. It provides for the existence of an ontology server that can interpret content in the context of an ontology. The receiver provides the ontology's name to this server. This technology is still experimental, however, and its viability is unproven for large knowledge bases. The agents in this system do not use an ontology server, instead accessing Protégé-2000 directly.

The third parameter states the language of the content. FIPA does not constrain what this might be. Natural languages (English, French, German) are permitted, though their use usually indicates a message for which precise semantic disambiguation is unnecessary. Some agents use XML, especially for messages in which syntax rather than semantic interpretation is needed. Extensions of XML such as the Resource Description Framework and the Web Ontology Language add some semantics.

Some agents use formal languages, such as Prolog. FIPA defines two preferred languages for agent communication: Choice Constraint Language (CCL) and Semantic Language (SL).

CCL is the preferred language for constraint satisfaction problems (CSPs). A CSP is a problem where, given a finite set of variables (each of a finite domain) and a finite set of constraints between the variables, a solution is an assignment of values to each variable such that no constraints are violated. It is useful for posing questions such as "Given a certain dinner menu, what wine would be appropriate?" and "Under current conditions, what routes are not available to a convoy?"

SL is based on first-order predicate calculus. The syntax is based on s-expressions (parenthesized terms with a prefix operator). Content may be:

- A proposition, i.e., a statement of the truth or falsehood of a fact. A proposition is expressed as a well-formed formula. A proposition is used as the content of an Inform communicative act.
- An action expression, which is a request for an agent to perform some action; the content of the action expression describes the nature of the action. An action expression is used as the content of a Request communicative act.
- An identifying reference expression, which basically establishes one collection of objects as synonymous with another. It is used as the content of the Inform-Ref communicative act, a special type of Inform act. Agent *A* asks *B* for all synonyms of *S*; rather than simply replying with a set $\{s_1, s_2, \ldots\}$, *B* responds with a sentence that states "all synonyms of *S* are $\{s_1, s_2, \ldots\}$". This explicit connection is often necessary for *A* to establish precise relationships.

SL is a very general language, sufficiently so that it can contain well-formed formulas whose solutions may be both undecidable and NP-complete. FIPA recognizes the difficulties such formulas might cause and specifies three subsets of SL. SL0, the minimal subset, has no identifying reference expressions, and allows only well-formed formulas without quantifiers and Boolean operators. SL1 extends SL0 with Boolean operators, facilitating arbitrary Boolean expressions. SL2 adds to SL1 identifying reference expressions, and also restricted forms of existential and universal quantification to propositions: nested quantifiers are not permitted unless the containing proposition is also a quantification. This and other restrictions (in particular, no free variables) keep SL2 expressions decidable.

SL0 is useful for stating simple facts, and much of the inter-agent communication in the IDA prototype simulation uses SL0 as the content language. For example, the enemy organization agent (specifically, the Recalculate Position Task) uses SL0 to inform the SimState agent of its position. It sends a message using the Inform communicative act with content such as:

```
((has-location "1 (DA) Mech Bde" (position :latitude 31.65 :longitude -97.46)))
```

The enemy organization agent establishes "Geo-Position" as the name of the ontology for this message. In this ontology, has-location and position have meaning:

- Has-location is a relationship between the name of an organization and a geodetic coördinate. This relationship can easily be translated to elements of the GH ontology. Note, however, that modeling the same information in the GH ontology requires instances of ObjectItemLocation and ReportingData. The Geo-Position ontology is more concise.
- Position is a two-argument function that generates an instance of class GeoPosition. In other words, it encapsulates the specification of position.

The enemy agent is assuming the SimState agent understands "Geo-Position" as an ontology in which the same semantics apply to these two terms.

3.1.4. Zeus

The prototype uses the Zeus parser-generator to convert between XML documents and Java classes. Zeus, an open-source product of Enhydra, takes as input a DTD and generates as output a set of Java classes. The input-output relationship may be summarized as follows:

- For each element in the DTD, Zeus generates an interface and implementing class whose names derive from that element.
- For each attribute of an element, Zeus generates in the corresponding interface and implementing class a get/set method pair. The result of the get method (or the parameter of a set method) is an instance of java.lang.String.
- For each single-valued or optional element of the element content, Zeus includes a get/set method pair in the element's interface and implementing class. The get method returns an instance of the single-valued element's interface.
- For each multiple-valued element (that is, an element specified with a * or +) in the element content, Zeus generates a get/set/add/remove method quadruple. The get method returns an instance of java.util.List; the parameter to the set method is a List, and the list should be a homogenous list of instances of the Java interface generated from the underlying element. The add and remove methods both take a single parameter that is an instance of the underlying element's interface.
- An interface specifies methods to let an instance *marshal* itself (and the implementing class satisfies the specification). Marshalling means to turn an instance into a valid XML document. Marshalling methods take a parameter that specifies the destination (file, URL, or writer).
- Zeus generates a class containing static methods to *unmarshal* the root element of the DTD, specified when Zeus is executed. Unmarshalling is the opposite of marshalling. The methods take as a parameter a source file, input stream, or reader. Each method yields an instance of the interface generated from the root element.

  Some unmarshalling methods have a Boolean parameter that specifies whether to validate the XML document. If Zeus performs validation, it will throw an exception if the document doesn't conform to its DTD. Conformance is syntactic: All required elements must be present, all elements must be in the correct order, all attributes must have the correct values, etc.

Using the DTD in Figure 13 as input, for example, Zeus generates the following Java code:

- Interfaces LoaderConfig, DBSpec, EnemyOrgSpec, and FriendlyOrgSpec. Zeus would, by default, generate interfaces for all elements, but it has an option to compress elements that only contain #PCDATA into strings, and the IDA prototype simulation uses this option.
- Corresponding classes LoaderConfigImpl, DBSpecImpl, EnemyOrgSpecImpl, and FriendlyOrgSpecImpl.
- The class LoaderConfigUnmarshaller, because Loader-Config is specified as the root element.

Zeus is an excellent tool for prototypes, because like DTDs it is small, concise, and does what it can efficiently and effectively. It should be used with care in production systems be-

cause of the inherent limitations of a DTD in terms of XML validation capabilities. For instance, the DTD in Figure 13 shows that DB is an attribute of the Loader-Config element, and that a Loader-Config may contain a DB-Spec element. A Loader-Config should contain a DB-Spec if and only if DB has the value "true", but expressing this constraint is beyond the power of a DTD.

3.2. Technical View: Component Agent and Task Structure

The components of the IDA prototype simulation are inherently asynchronous. The different controllers operate independently. None are constrained to each other's timing. The major characteristic all components share, then, is the need for independence. A secondary characteristic is the need to communicate with other components, even if their existence is not known until run-time.

These considerations call for implementation technology that supports unconstrained and arbitrary communications. Because the characteristics are essentially those of agents, and because FIPA-OS was being used to implement agents, the prototype implements components as agents, handing asynchronicity and communication using FIPA-OS architecture and methods. Not all components of the prototype are true agents that register, search, and infer. Nevertheless, there are two good reasons to use FIPA-OS whenever possible:

1. Reuse of FIPA-OS architecture and methods obviates the need for extensive design and implementation.
2. FIPA-OS includes an Agent Loader tool (not to be confused with the Loader component of the IDA prototype simulation discussed in Section 2.6) that can be used to control agent start-up and shutdown. The Agent Loader invokes each new agent as a new thread in one Java virtual machine. Executing a Java virtual machine that accesses FIPA-OS, Protégé-2000, and an RDBMS is computationally expensive – it requires approximately 500 MB of RAM – so use of the Agent Loader significantly lessens computer resources needed.

Each component in the System View is implemented as an agent or (in the case of a Friendly Organization) a set of agents (this does not include the RDBMS, which is a COTS component). Each component may be invoked either as a stand-alone application or through the Loader (see Section 2.6). Except for friendly organizations, components may also be invoked through the FIPA-OS Agent Loader tool.

The following sections cover each component's agent and task structure.

3.2.1. Agent/Task Structure of Enemy Organization Component

The Enemy Organization component is implemented as a single agent. The task structure of this component is discussed on p. B-21.

3.2.2. Agent/Task Structure of Friendly Organization Component

The Friendly Organization component is implemented as a set of agents initiated by a single application, shown in Figure 19 as the Decision Support Application (parallelograms represent agents). This application is not itself an agent, for reasons discussed below. It is a Java class that invokes four agents, each of which serves a distinct role:

1. The Threat Perception Agent is responsible for:

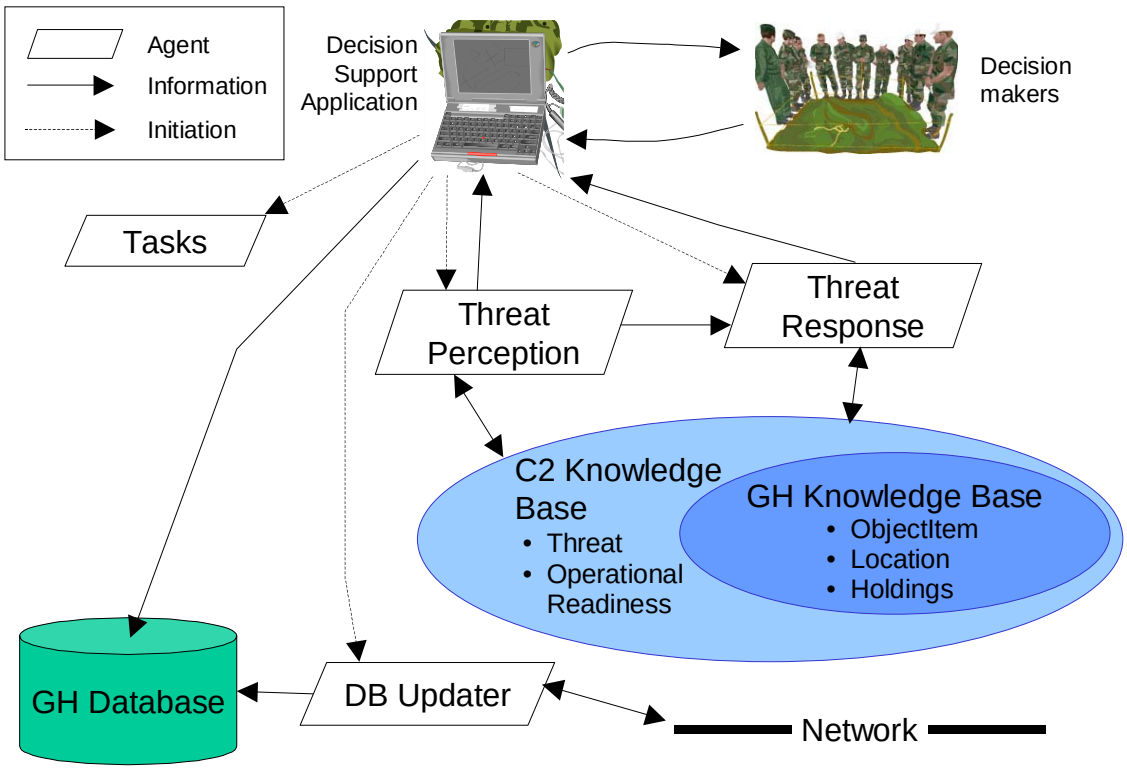

Figure 19. Friendly Organization Agent Structure

a) Identifying threats from a hostile enemy organization to any friendly organization (not just the one associated with the decision support application that invoked the agent).
b) Notifying the Decision Support Application of the threat, thereby giving the application the opportunity to inform decision makers.
c) Notifying all Threat Response Agents (not just the one associated with the decision support application that invoked the agent) of the threat.

2. The Threat Response Agent is responsible for:
   a) Awaiting receipt of threats.
   b) Formulating courses of action that might be used to respond to a threat. These courses of action are expressed as frames in the GH knowledge base, so the agent updates the knowledge base as part of its functionality.
   c) Notifying the Decision Support Application of the courses of action.
3. The DB Updater Agent is responsible for:
   a) Detecting changes in the GH data set associated with the friendly organization, and communicating these changes to all other known DB Updater Agents.
   b) Receiving changes from other DB Updater Agents and incorporating them into the GH data set.
4. The Tasks Agent is responsible for starting and controlling certain tasks performed by each Friendly Organization component. These tasks are discussed below.

   The Tasks Agent is not a true agent. It does not communicate with other agents. Its accesses to the knowledge base are for mundane maintenance chores. It is imple-

mented as an agent because a friendly organization needs to perform certain tasks periodically, and because a FIPA-OS agent can control a task hierarchy.

All agents in a single Friendly Organization component share the same knowledge base. This decision is convenient (the KB Updater Agent only deals with a single knowledge base, and the Threat Perception and Response Agents can be assumed to have identical knowledge) and efficient (each Protégé-2000 knowledge base is memory-intensive).

The Decision Support Application does not access the knowledge base directly. Only agents access the knowledge base (this is a design decision). The Decision Support Application receives information from agents, and updates the GH-conformant data set. Agents, on the other hand, focus on the knowledge base.

3.2.2.1. Threat Perception Agent Structure

The Threat Perception Agent is implemented as a FIPA-OS agent whose task hierarchy is shown in Figure 20. The agent initiates an Idle Task that serves as the root of the task hierarchy. The subtasks are as follows:

- On start-up, the Idle Task initiates the Ontology Search Task and the Threat Response Agent Search Task.
- The Threat Response Agent Search Task is a periodic task that uses the Directory Facilitator to search for all known Threat Response Agents. This list will include the Threat Response Agent of the agent running this search task. The interval between searches (that is, the duration of the child Wait Task) is a parameter of the Friendly Organization component, as described in Section 2.2.
- The Ontology Search Task is the task responsible for searching the knowledge base for threats. In other words, it encapsulates the logic and rules that determine what constitutes a threat. The Ontology Search Task also detects organizations that once were but no longer pose a threat. It is a periodic task; each analysis of the knowledge base and subsequent wait counts as one cycle.
- Each time the Ontology Search Task detects a threat or a canceled threat, it initiates an Inform Threat Response Agents Task (thus several of these latter tasks may be executing simultaneously). This task informs every agent found by the Threat Response Agent

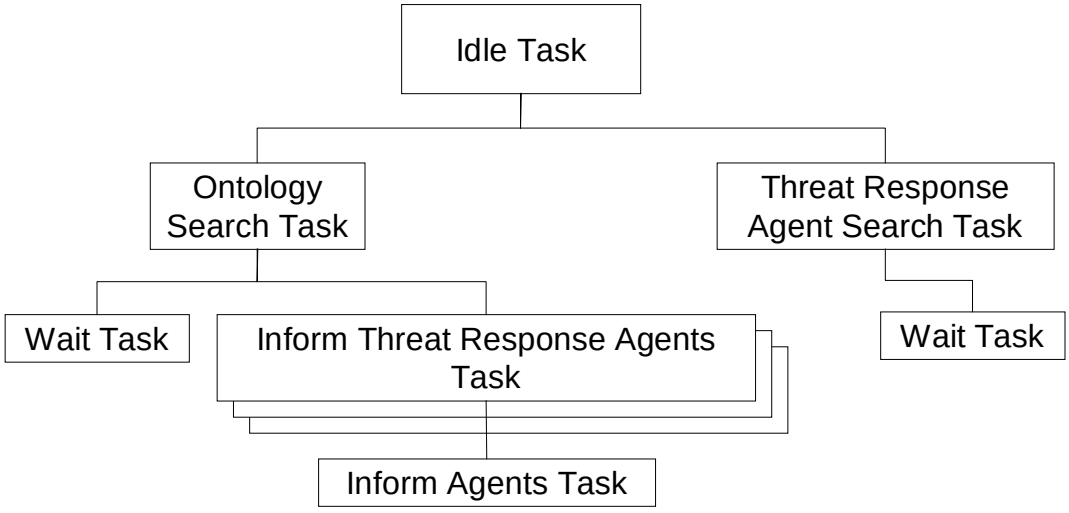

Figure 20. Threat Perception Agent Task Structure

Search Task of the threat. It sends a message expressed in SL0 of the form:

((threat hostile-org-name threatened-friendly-org-name))

if hostile-org-name poses a threat, and

((not-threat hostile-org-name threatened-friendly-org-name))

if it no longer does.

- The Inform Threat Response Agents Task is implemented by having it initiate an Inform Agents Task. The latter task is a procedural abstraction defined by the prototype. It is created with four parameters: a list of agent IDs, information, the name of a language, and the name of an ontology. It creates a FIPA message that contains the information, language, and ontology, sends that message to each agent in the list, then ends.

3.2.2.2. Threat Response Agent Structure

The Threat Response Agent is a FIPA-OS agent whose responsibility is to formulate courses of action in response to information received from a Threat Perception Agent. It is initiated by a Friendly Organization component, which gives it a reference to a Decision Support Application and the knowledge base that component is using.

A Threat Response Agent's task structure is implemented as a single Idle Task that listens for Inform messages. Assuming an Inform message has the correct ontology and language, the Idle Task will interpret the message's contents. Threat content interpretation causes the generation of a list of courses of action. The Threat Response Agent notifies the Decision Support Application of these courses of action. The Decision Support Application is not an agent, so this notification is implemented using library classes from the Java run-time environment, not FIPA-OS classes.

3.2.2.3. Tasks Agent Structure

The Tasks Agent, whose structure is shown in Figure 21, initiates certain tasks that are necessary during the execution of a Friendly Organization component:

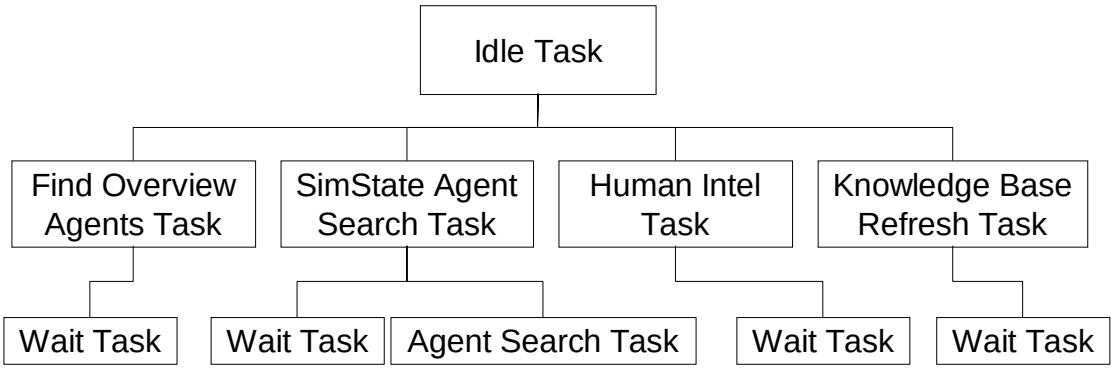

Figure 21. Tasks Agent Task Structure

- It initiates the tasks that simulate the intelligence sources a friendly organization possesses. These sources include human intelligence; in fact, human intelligence is the only intelligence source the IDA prototype simulation currently implements.
- It finds the executing SimState component (this is necessary before intelligence sources can operate).
- It initiates a task that performs a periodic search for Overview components. Overview components are implemented as agents, so the search is implemented using FIPA-OS agent search methods.

- It initiates a task that periodically examines the GH-conformant data set for changes, and incorporates these changes into the Knowledge Base. The changes are then visible to all agents of the Friendly Organization component.

Furthermore:

- An instance of the agent may be instructed to request that all known Overview Agents highlight the location of the Friendly Organization controlling the agent. This is the purpose of the "Flash Location" button in Figure 5. (That the button is deactivated in Figure 5 means no Overview components have been found.) When a decision maker clicks the button, the decision support application signals the Tasks agent to perform the "flash" request. The request is carried out by the agent; it involves no tasks. FIPA-OS agents can send messages too.

All tasks are periodic. The Human Intel Task's delay is based on a Poisson distribution, a recognition that humans do not provide intelligence on a regular basis. (This is not to claim that a Poisson distribution is a comprehensive or even realistic model of human behavior.)

The SimState Agent Search Task only executes until it finds the SimState component. When it does, it terminates. Other tasks run until the Friendly Organization component terminates.

3.2.2.4. DB Updater Agent Structure

The DB Updater Agent is responsible for ensuring database concurrency: a change to one Generic Hub data set must be propagated to all others. Database concurrency is a highly complex area, but it is straightforward in this prototype simulation because the initial Generic Hub data set content is never modified. The decision support application adds rows, only under tight constraints. An update to a GH data set always consists of:

1. Adding a record to the REPORTING-DATA table, and a corresponding record to the REPORTING-DATA-ABSOLUTE-TIMING table.
2. Adding a record to one of several tables that have a column for which a reporting-data-id is a foreign key (OBJECT-ITEM-LOCATION, OBJECT-ITEM-ASSOCIATION, etc.).

An update will sometimes include:

3. Adding rows to other tables (LOCATION, etc.) to which the second table has an integrity constraint.

Two RDBMSs never attempt to lock the same information.

Figure 22 shows the task structure of the DB Updater Agent. The root Idle Task initiates a periodic task, DB Update Agent Search Task, that locates all known DB Updater Agents. This task uses the Agent Search Task procedural abstraction, using the string "db-upd" as the type of agent for which to search. The interval between searches is a parameter of a Friendly Organization component that may be specified on start-up.

The DB Updater Agent also has two important methods:

1. Its handleInform() method listens for messages with the INFORM performative. These messages are expected to come from other DB Updater Agents. They inform an agent of updates to a GH-conformant data set. When such a message is received, handleInform() parses the message and uses its content as a set of updates.
2. Other agents or applications may invoke its propagateChanges() to propagate a set of updates to all other DB Updater Agents found by the DB Update Agent Search Task.

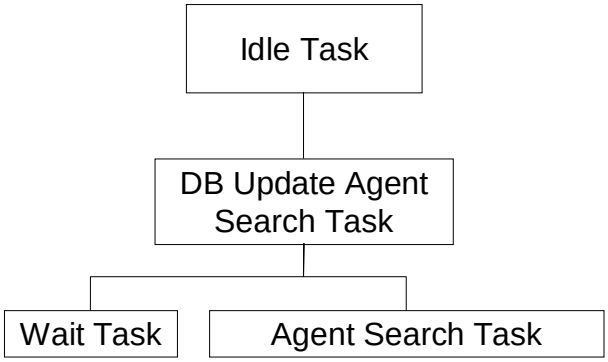

Figure 22. DB Update Agent Task Structure

The content of messages sent by the DB Updater Agents is a list of SQL queries. This is not a good format; for one thing, it ties the prototype to a particular implementation of SQL. Transmitting the updates as an XML document that map to rows in the GH-data model is considered a better approach, but was not used in the IDA prototype simulation because it would have taken more time to implement than was available.

3.2.3. Agent/Task Structure of Overview Component

The Overview component is implemented as a FIPA-OS agent whose role is to listen for messages with INFORM or REQUEST performatives. An INFORM performative is a notification of change in state. Currently, state captures only the location of a battlefield object. The message, which is from the SimState Agent, gives the object's name and location. The Idle Task will request the GUI of the Overview component to change the location of the object. If the object did not exist, it is displayed; if the new location is outside the bounds of the display, it is removed from the display.

The REQUEST message occurs when the controller of a friendly or hostile organization clicks the Flash Location button on that organization's user interface. The message will contain the name of a battlefield object. If the GUI of the Overview component is currently displaying this object, it will change the display to highlight the object's position.

3.2.4. Agent/Task Structure of SimState Component

The SimState component is implemented as a FIPA-OS agent whose primary role is to maintain ground truth regarding the physical state of battlefield objects. The agent invokes an Idle Task that listens for messages with INFORM and REQUEST performatives. An INFORM performative is expected to contain a specification of new state information for a battlefield object. Currently, hostile organization components send these messages regarding their location. The SimState agent makes no reply to these messages.

A REQUEST performative is expected to specify a geographic area. The SimState agent replies with a list of the names and locations of all battlefield objects within the area. The position of these objects is originally determined by the content of the GH-conformant data set specified during initialization of the prototype simulation. Positions are updated by subsequent INFORM messages. The intent is to simulate sensor behavior: a given sensor is able to detect battlefield objects within a prespecified range. Currently the prototype simulation implements only human sensors (i.e., battlefield observers) who are able to detect objects within a rectangular area that has no obstacles. This model is overly simplistic but is easy to implement.

In addition to handling messages, the SimState agent also notifies Overview agents of state changes. The SimState agent performs this function each time it receives a state change. The agent therefore searches for Overview agents and maintains a list of all agents found during the last search. Figure 23 shows the task structure that the SimState agent uses to perform this search. The Find Overview Agents Task is a periodic task.

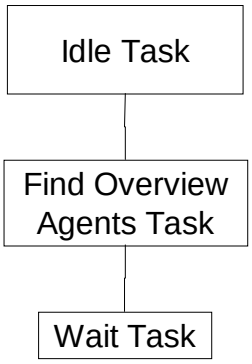

Figure 23. SimState Agent Task Structure

3.2.5. Agent/Task Structure of Loader Component

The Loader component is implemented as a FIPA-OS agent. However, it is not truly an agent. The implementation strategy permits it to be controlled from the FIPA-OS Agent Loader tool, thereby conserving system resources.

The Loader agent invokes no tasks. It only initiates other agents, after which it displays a GUI that a controller may use to start enemy organization movement, or to stop execution of all running components.

3.3. Inter-Agent Messages

Table 1 shows the types of messages agents in this system send to each other.

Table 1. Message Types Sent By Agents

Sending Agent	Receiving Agent	Language	Ontology	Performative	Protocol
Enemy Organization	Overview	identify		Request	No-Protocol
		Syntax: flash *obj-item-name*			
	SimState	SL0	geo-position	Inform	No-Protocol
		Syntax: ((has-location *org-name* (position :latitude *lat* :longitude *lon*)))			
DB Updater	DB Updater	binary	db-update	Inform	No-Protocol
		Syntax: An instance of class java.util.List; each list element is a string that is a syntactically valid SQL INSERT statement.			
Threat Perception	Threat Response	SL0	threat	Inform	No-Protocol
		Syntax: One of: • ((threat obj-item-name-1 obj-item-name-2)) • ((not-threat obj-item-name-1 obj-item-name-2))			
Tasks	SimState	SL0	geo-position	Request	Request-Reply-Protocol

Sending Agent	Receiving Agent	Language	Ontology	Performative	Protocol
		Syntax: ((action *simstate-agent-name* (sensing :center (position :latitude *lat-1* :longitude *lon-1*) :northwest (position :latitude *lat-2* :longitude *lon-2*))))			
	Overview	identify		Request	No-Protocol
		Syntax: flash *obj-item-name*			
SimState	Overview			Inform	No-Protocol
		Syntax: An instance of class ida.sd.gpc.GeoPositionImpl.			
	Tasks	SL0	geo-position	Inform	Request-Reply-Protocol
		Syntax: ((result (action *simstate-agent-name* (sensing :center (position :latitude *lat-1* :longitude *lon-1*) :northwest (position :latitude *lat-2* :longitude *lon-2*))) (set (position :latitude *lat* :longitude *lon* :name *n*) …)))			

The following points should be noted:

- The "identify" language used by the Overview agent was invented for this system. The language contains only one valid sentence: flash *n*, where *n* is the name of a battlefield object. The merits of using an arbitrary language are debatable. On the one hand, the language can be concise. On the other hand, message interpretation becomes system dependent.

  The message could also be expressed using SL0 with a REQUEST performative:

  ((action *overview-agent-id* (flash *obj-item-name*))

  However, parsing the content to extract the action requires much more effort than the customized language. SL0, while preferable for a production system, can be expensive for a prototype, and its use should be considered carefully.
- FIPA-OS permits binary objects – specifically, instances of Java classes that implement interface java.io.Serializable – as message content. A few agents of the prototype (DBUpdater and SimState) send messages with binary content. In a production system, the decision to use binary content should not be made lightly. It presumes the sending and receiving agent have access to the same Java class; in other words, it presumes certain standardization efforts. Standardization is desirable but also costly. Textual message content transmitted using a standard language is preferable.
- The majority of inter-agent communication occurs using the No-Protocol protocol, meaning the sending agent does not expect a response from the receiver. This is probably inappropriate for a production system. Then again, for many of the messages it is not clear that the sender would behave any differently if the receiver failed to receive, or refused to accept, a message.

- Some messages lack an ontology; some lack both a language and an ontology. This is probably a mistake for a production system.
- Table 1 does not include the agent registration/deregistration and agent search messages are sent to FIPA-OS tools. FIPA specifications (see [FIPA-70], p. B-33) govern their content.

3.4. Use of Model-View-Controller Paradigm

Most components of the IDA prototype simulation utilize the Model-View-Controller (MVC) paradigm popularized by the Smalltalk language [Smalltalk 1983]. This paradigm is especially useful for designing and implementing applications with a GUI. Java supports it through two standard library components: the java.util.Observable class and the java.util.Observer interface. An Observable instance may have observers (instances of Observer); the Observable may then invoke a method that automatically signals all observers. A GUI component commonly uses this paradigm to signal some other component that a button has been pressed, a text field has changed, etc.

The MVC paradigm decouples an application's functionality from its user interface. Methods in classes implementing the user interface:

1. Respond to user actions.
2. Are invoked by methods in other classes to display information. Their functionality is concerned with the format of the information, not with its meaning.

Other classes are concerned with a component's essential functionality: the transformation of user inputs into outputs to display or communicate.

The IDA prototype simulation components contain at least two Java classes: one to implement the functionality, and another to implement the user interface. Figure 24 shows classes for the Enemy Organization component that implement the MVC paradigm. Class EnemyOrganisationGUI contains fields that are used to display values to the user. Since most of these values are textual (see Figure 3), they are of type javax.swing.JTextField. Fields _isMoving

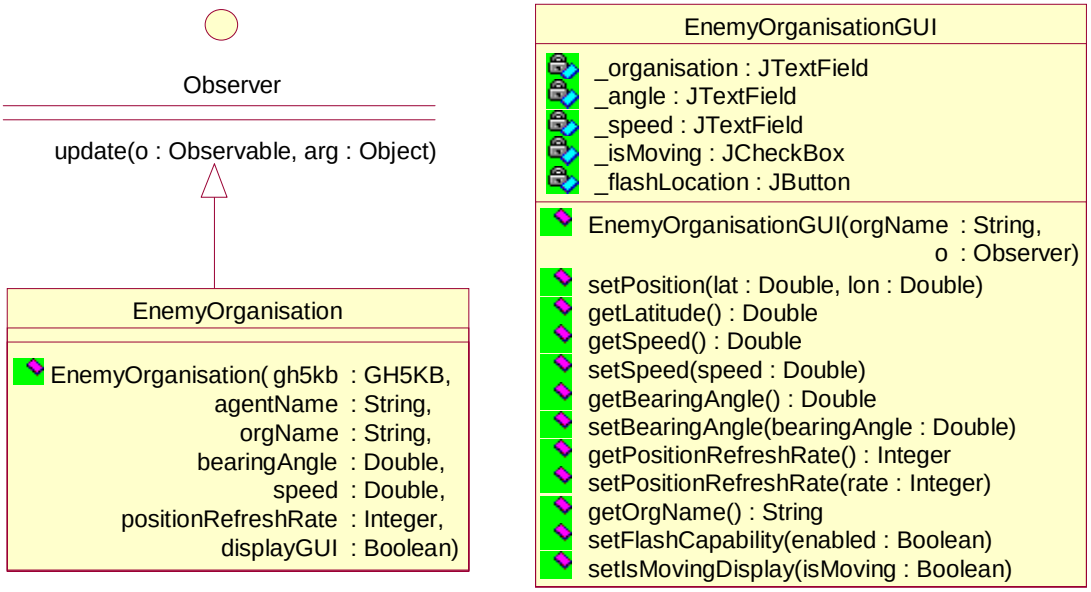

Figure 24. Model-View-Controller Paradigm Example for Enemy Organisation

and _flashLocation illustrate the other two user interface paradigms used.

Most methods of EnemyOrganisationGUI are set/get methods that access the user-visible properties. (Properties that are read-only, such as the organization's name, are fixed by the constructor and cannot be set.) The class constructor is responsible for setting the fields, and for defining the Java events and event handlers that occur in response to user interface events. These event handlers invoke the update() method of the observer given to the EnemyOrganisationGUI() constructor. When EnemyOrganisation is instantiated, it instantiates EnemyOrganisationGUI, passing itself as the value of the new object o. Thus the update() method invoked is that of EnemyOrganisation. The update() method analyzes its arg parameter to determine the nature of the update, responds appropriately, and invokes set methods of the EnemyOrganisationGUI instance. See Figure 25.

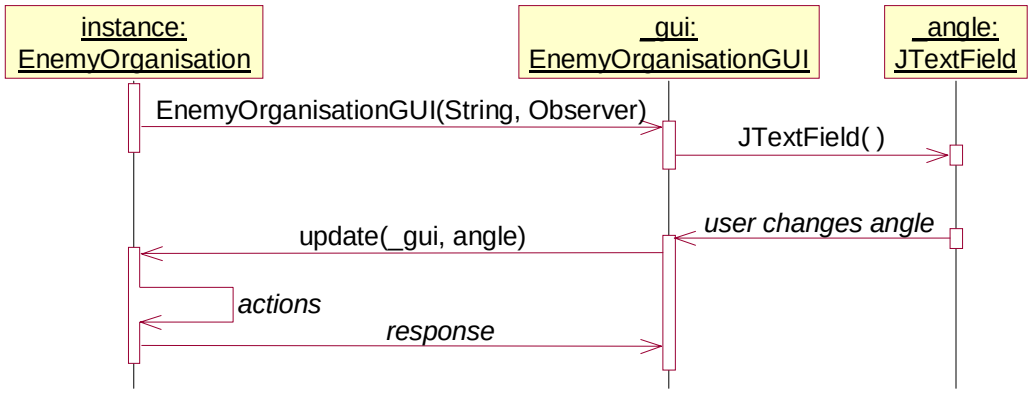

Figure 25. Sample MVC Event Sequence for Enemy Organisation

3.5. Modularization

The IDA prototype simulation is decomposed into modules according to the principle of information hiding: Each module encapsulates a set of secrets not visible to other modules [Parnas 1972]. The modules are represented as four trees:

1. The Environment Hiding module, the primary secrets of which are the interfaces between the other two modules and the environment in which the prototype simulation executes. Changes to the Environment Hiding module are motivated by updates to hardware and software technologies.
2. The Behavior Hiding module, the primary secret of which is the system functionality. Changes to the Behavior Hiding module are motivated by additions or modifications to system features, or to changes in the user interface. (In a production system, changes to the Behavior Hiding module stem from changes to system requirements, but the IDA prototype simulation has no formally defined requirements.)
3. The Software Decision module, which encapsulates physical facts, theories, and data and procedural abstractions. Changes to the Software Decision module are motivated by perceived improvements in execution speed and design clarity.
4. The Software Generation module, which encapsulates packages that assist in code generation. The primary secrets of this module are decisions on the most effective way to specify certain aspects of functionality in non-native languages. The secondary secrets are the algorithms and data structures used in processing those languages.

Changes to the Software Generation module are motivated by new ideas on how to represent behavior and functionality.

Each of these modules is in turn decomposed into sub-modules, each with its own secrets. This decomposition continues until the designer decides that a module's secret cannot profitably be split among sub-modules. At that point, the designer implements the module's secrets in Java. Note that interior nodes of the tree are virtual modules. Only leaf nodes are Java classes.

The prototype's decomposition follows that pioneered in the development of software for the A-7E [Britton 1986]. Java packages and classes replace modules.

Figure 26 shows the decomposition of the prototype system into packages using the information hiding principle. Figure 26 shows only the packages (i.e., the interior nodes); selected classes are discussed in Section 3.6. The following discussion of each package covers the package's secrets and likely motivation for change.

3.5.1. Environment Hiding Package

The Environment Hiding package contains subpackages and classes that encapsulate the environment in which the system operates. The secrets of this package are imposed by the designers of environmental components, and are, therefore, external. Changes in this package occur as a result of updates to the environment, usually to fix bugs, add features, or effect performance improvements. Because the IDA prototype simulation is implemented in Java, environment updates will be updates to software rather than hardware components.

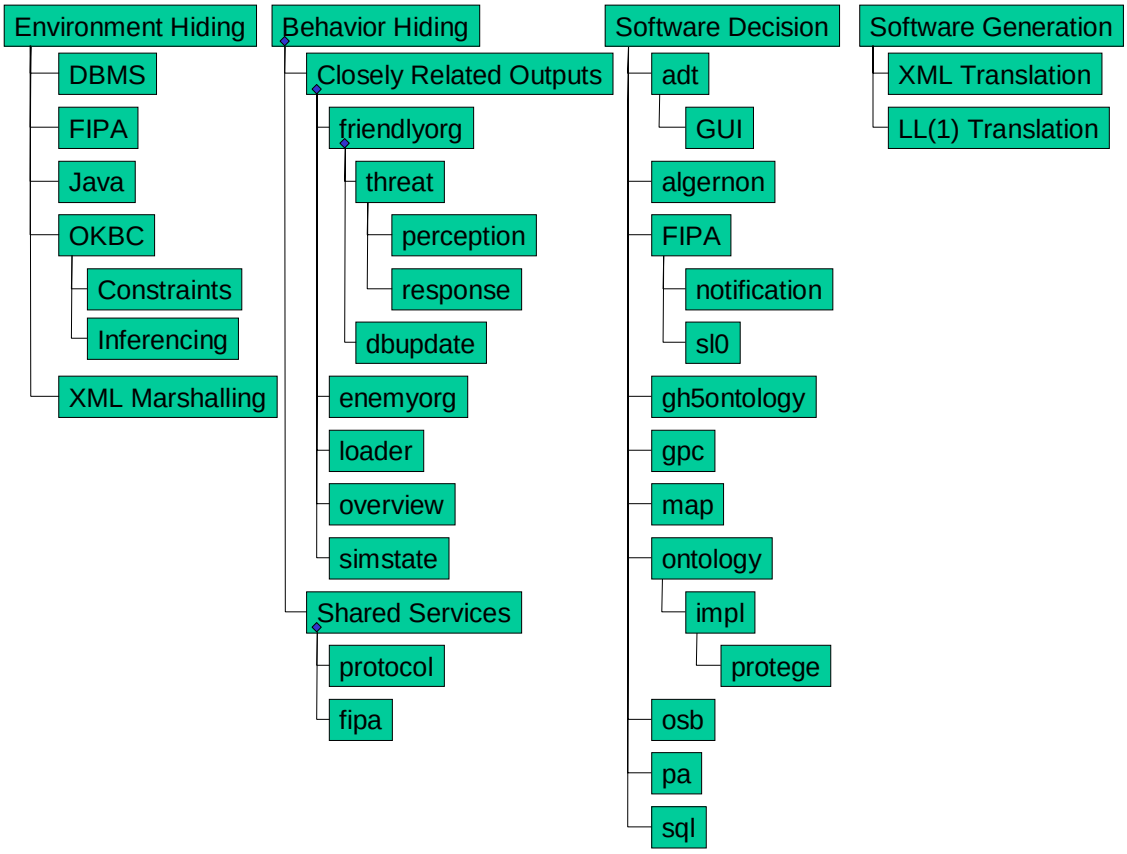

Figure 26. Information Hiding Structure for Prototype System

The secrets of Environment Hiding subpackages are almost invariably the algorithms and data structures used to implement the packages. This reflects a desire to present the subpackage's functionality through an API. That these packages will be reused through extension is not anticipated.

The usual approach to creating subpackages of Environment Hiding is to write code that isolates the rest of the system from change. The IDA prototype simulation is built using software systems that are relatively stable. Major change in them seems unlikely. Therefore, these systems *are* the Environment Hiding subpackages. This approach is untenable in production systems, for reasons discussed in the some of the following subsections.

3.5.1.1. Java

The Java package encapsulates the language in which the prototype is written, and the standard libraries available to interface with the environment in which the prototype runs. The prototype assumes the capabilities present in version 1.4.1. The primary secrets of this package are the implementation of the Java Compiler, the Java Virtual Machine, and the Java runtime libraries. Changes to this package are in response to new releases of Java from Sun Microsystems.

3.5.1.2. FIPA

The FIPA package contains subpackages and classes that encapsulate an implementation of FIPA specifications for agent-based systems. The primary secrets of this package are:

- The algorithms and data structures used to implement FIPA specifications.
- The underlying technologies used to implement inter-agent communication.

The prototype uses FIPA-OS as the implementation of the FIPA package. Agent-based implementation technologies are complex, and even two technologies that implement FIPA are likely to differ greatly based on arbitrary design decisions. A production system is likely to select a single agent-based implementation technology, as opposed to trying to isolate other modules from switches between different technologies. Therefore, changes in this package result from new releases of FIPA-OS.

3.5.1.3. RDBMS

The DBMS package encapsulates an implementation of a database management system. The primary secrets of this package are the algorithms, data structures, and external file system structures used to implement that RDBMS.

The IDA prototype simulation was tested with both version 9i of Oracle (personal edition) and version 4.0.17 of MySQL as the corresponding RDBMS. Use of a particular DBMS is somewhat hidden by Java's `java.sql` library package, which aside from a single vendor-specific driver module is supposed to reduce variations between RDBMSs to differences in their implementations of SQL. In practice, each RDBMS has its own quirks (for example, Oracle and MySQL give different initialization sequences in their documentation) that make RDBMS encapsulation if not problematic then at least time-consuming. A production system will want to devote more effort to RDBMS encapsulation than the IDA prototype simulation has. Then too, each RDBMS implementation offers a distinct subset or superset of the standard SQL-92 specification. A system's designer must consider whether particular non-standard features are important and consider the consequences of tying his system to a specific RDBMS by using them.

3.5.1.4. OKBC

The OKBC package encapsulates an implementation of an OKBC knowledge base. OKBC is a protocol, so this package defines a protocol for communicating with a knowledge base. The primary secrets of this package are the algorithms, data structures, and external file structures used to represent a knowledge base. Changes in this package are in response to changes in the technology used to implement OKBC.

The prototype uses Protégé-2000 to implement the OKBC package. Defining an abstraction of OKBC capabilities is not difficult (the IDA prototype simulation does so – see Section 3.5.3.7) and a production system should consider taking this step to isolate other packages from the effects of switching between OKBC implementations.

3.5.1.4.1. Inferencing Package

The Inferencing package encapsulates rule-based analysis of a knowledge base. The primary secrets of this package are the algorithms and data structures used to implement rule-based analysis.

The prototype uses Algernon as the Inferencing package. A production system should isolate changes in inferencing technology by considering the inferencing capabilities needed and writing a package that presents an interface to exactly those capabilities, with an implementation in terms of Algernon or another suitable inferencing system.

3.5.1.4.2. Constraints Package

The constraints package encapsulates constraint enforcement on a knowledge base. The primary secrets of this package are the algorithms and data structures used to implement constraint enforcement.

The prototype uses Protégé-2000's PAL (Protégé-2000 Axiomatic Language) plug-in to implement constraint enforcement. Constraint enforcement languages differ so widely that isolating change between them may prove impractical. Designers of a production system should consider selecting a specific constraints package that accompanies the OKBC implementation they choose.

3.5.1.5. XML Marshalling

The XML Marshalling package encapsulates how Java objects may be marshaled into XML documents (and conversely, how XML documents may be unmarshalled into Java objects). The primary secrets of this package are the algorithms, data structures, and external file system access techniques used to implement marshalling and unmarshalling operations. Changes to this package are likely to be motivated by a desire to improve marshalling functionality.

The prototype uses outputs of Zeus to implement this package. Zeus generates a set of classes and interfaces according to logical and conventional, but nevertheless arbitrary, rules. These rules are part of the XML marshalling package. Other marshalling technologies may differ. Designers of a production system should consider whether a more robust encapsulation of marshalling operations is desirable.

Zeus determines XML document conformance using DTDs rather than XSDs. XSDs add specification power that might result in some documents being acceptable to Zeus but not to other technologies. Designers should consider whether DTDs meet their needs.

3.5.2. Behavior Hiding Package

The Behavior Hiding package contains subpackages and classes that encapsulate the required behavior of the system. This required behavior is the primary secret of this model. Changes in this package occur as a result of a desire to modify or extend required behavior.

3.5.2.1. Closely Related Outputs Package

The cro package consists of a set of packages that individually control a set of closely related outputs.[4] These outputs are information that is displayed to a user on a GUI, messages sent to other agents, or updates to a Generic Hub data set. The primary secrets of the Individuals package are the rules determining the timing, order, and values of the outputs. Changes in this package occur in response to required changes in these rules.

3.5.2.1.1. Friendly Organization

The friendlyorg package contains subpackages and classes that implement the prescribed functionality of the Friendly Organization component. The primary secret of this package is that functionality. The secondary secrets are the algorithms and data structures used to implement that functionality. Changes in this package occur in response to required changes in friendly organization functionality.

3.5.2.1.2. Enemy Organization

The enemyorg package contains classes that implement the prescribed functionality of the Enemy Organization component. The primary secret of this package is that functionality. The secondary secrets are the algorithms and data structures used to implement that functionality. Changes in this package occur in response to required changes in enemy organization functionality.

3.5.2.1.3. Loader

The loader package contains subpackages and classes that implement the prescribed functionality to interpret a simulation configuration and load a simulation based on that configuration. The primary secret of this package is the content of a configuration. The secondary secrets are the algorithms and data structures used to convert it into a simulation. Changes in this package occur in response to changes in the specification of a configuration.

3.5.2.1.4. Overview

The overview package contains classes that implement the prescribed functionality of the Overview component. The primary secret of this package is the format in which the information is presented. The secondary secrets are the algorithms and data structures used to receive requests and display information. Changes in this package occur in response to changes in the specification of how an overview is presented to a controller.

3.5.2.1.5. SimState

The simstate package encapsulates classes that implement ground truth. The primary secrets of this package are the algorithms and data structure used to collect, maintain, and disseminate ground truth. Changes in this package are likely to occur in response to changes in what constitutes ground truth.

---

[4] The A-7E name for this package, “Function Driver”, seems better suited to real-time systems.

3.5.2.2. Shared Services Package

The ss package encapsulates aspects of behavior common to several Individuals subpackages. If there is a change in an aspect of this behavior, it is likely to affect all subpackages. Each component of ss, then, encapsulates an aspect of behavior shared by at least two subpackages of the cro package.

3.5.2.2.1. FIPA

The fipa package encapsulates shared services required for FIPA-based agents to interact with their environment. The primary secrets of this package are the FIPA rules for performing the services. Changes in this package are likely to be motivated by changes in these rules.

3.5.2.2.2. Protocol

The protocol package encapsulates system-defined protocols used in inter-agent communications. The primary secret of this package is a protocol's implementation. Changes in this package are likely to occur in response to the need to add or modify protocols.

3.5.3. Software Decision Package

The Software Decision package encapsulates subpackages and classes that implement software design decisions based on physical facts (e.g., the Earth's radius), theories, and data and procedural abstractions. The secrets of this package come not from prescribed functionality or environment considerations but from considerations of design, implementation, and testing ease, and run-time efficiency. These secrets are determined by software designers. Changes are usually motivated by a desire to improve either performance, or to simplify or clarify software design.

3.5.3.1. ADT

The adt (abstract data type) package encapsulates subpackages and classes data abstractions deemed useful throughout the system. These abstractions do not implement any concepts directly prescribed by system functionality. They provide lower level support entities. The primary secrets of this package are the algorithms and data structures used to implement the data abstractions. Changes in this package generally result from a realization of how to improve performance.

3.5.3.2. Algernon

The algernon package encapsulates classes that provide procedural abstractions used by other packages to deal with the inference subpackage of Environment Hiding (Section 3.5.1.4.1). The secrets of this package are the algorithms that implement those procedural abstractions. Changes in this package usually result from a realization of how to improve performance. Extensions to this package result from observation of repeated code dealing with Algernon, or from realization of how the introduction of a procedural abstraction could clarify code.

3.5.3.3. FIPA

The FIPA package encapsulates subpackages and classes that provide procedural and data abstractions to deal with concepts in the fipa subpackage of Environment Hiding (Section 3.5.1.2). This package does *not* encapsulate either FIPA or FIPA-OS. It just simplifies use of FIPA-OS through the introduction of these abstractions. The secrets of this package are the algorithms and data structures used to implement the abstractions. Changes in this package are likely to be motivated by a realization of how to improve performance. Extensions to this package result from observation of repeated code dealing with FIPA-OS, or

from realization of how the introduction of a procedural abstraction could clarify code. Changes and extensions may also result from upgrades to FIPA-OS that provide new capabilities.

3.5.3.4. GH5 Ontology

The gh5ontology package tailors the ontology package (Section 3.5.3.7) to support manipulation of a Generic Hub knowledge base. Methods in this package hide design decisions on the implementation of the GH ontology. They provide some procedural abstractions that simplify access to the ontology as a consequence. The primary secrets of this package are:

- The use of Protégé-2000 as the technology to implement the GH ontology.
- Design decisions on how Generic Hub attributes are represented in the ontology; in other words, the classes and slots declared in the GH and C2 ontologies are visible, but classes and slots of underlying ontologies are not.
- The theories underlying methods in the package that implement procedural abstractions.

The secondary secrets are the algorithms and data structures used.

Changes in this package are likely to be motivated by changes to the Generic Hub, or to the representation of the Generic Hub in the GH ontology. They may also result from a realization of how to improve performance. Extensions to this package result from observation of repeated code dealing with the GH ontology, or from realization of how the introduction of a procedural abstraction could clarify code.

3.5.3.5. GPC

The gpc (Geo-Position Context) package contains classes that encapsulate the specification of a location, in geocentric coordinates. The package provides:

- An interface, GeoPosition, and its implementation, GeoPositionImpl, that specifies get/set methods for defining a coordinate, and optionally associating the name of a battlefield object with that coordinate.
- An abstract class, AbstractGeoPositionContext, that partially implements a Context, which is a data abstraction useful in evaluating FIPA messages (see Section 3.6.1).

The primary secrets of the class are:

- The algorithms and data structures used in the implementation of GeoPositionImpl.
- The algorithms of AbstractGeoPositionContext. Note that AbstractGeoPositionContext is abstract. Its data structures are visible to any class that extends it, but not outside that class.

3.5.3.6. Map

The map package contains subpackages and classes that encapsulate a map image displayed in the Friendly Organization component (see Figure 5). A map image is specified in an XML document. The primary secrets of the package are the DTD describing that document, as well as the algorithms and data structures for translating a map specification into an image, and the representation of a map image. Changes in this package occur in response to changes in the information associated with obtaining a map.

3.5.3.7. Ontology

The ontology package contains:

- A set of interfaces that together define the concept of an ontology for the purposes of this system.
- A subpackage impl, each subpackage of which provides a single implementation of the interfaces in the ontology package.

The interfaces derive from Protégé-2000's interpretation of OKBC. They are not as complete as Protégé-2000. They have been defined to suit the needs of the IDA prototype simulation (for example, there is no interface for a facet). They could be expected to expand as the prototype simulation grows in complexity.

The primary secrets of this package are the classes in the subpackages of impl that implement the interfaces. In other words, the view of an ontology in this system is provided solely through the interfaces of ontology.

A subpackage is responsible for providing access to instances of these interfaces. For example, impl has a subpackage protege. The only class of protege visible outside the ontology package is ProtegeProject, which has a method that, given the URL of a Protégé-2000 project specification, returns an instance of interface KnowledgeBase for that project. The KnowledgeBase interface allows access to all classes, slots, and instances in the knowledge base.

Changes to the ontology package are likely to occur in response to new needs for access to an ontology.

3.5.3.8. OSB

The osb (Ontology-SQL Binding) package contains classes that bind a GH ontology to an SQL database. The SQL database is specified via a JDBC (Java Data Base Connectivity) object. The binding permits a knowledge base to be populated from a GH-conformant database, and for changes to a knowledge base to be saved into a GH-conformant database as well. The primary secrets of this package are the algorithms and data structures used to implement the binding. Changes in this package are likely to occur in response to:

- The use of an RDBMS that supports a variant of SQL not currently recognized.
- Need for performance improvements.

3.5.3.9. PA

The pa (Procedural Abstraction) package contains classes that provide procedural abstractions that are independent of any domain defined by the prototype simulation. These include mathematics (e.g., a Poisson distribution) and strings (e.g., concatenate a list of strings). The primary secrets of this package are the algorithms and data structures used to implement the abstractions. Changes in this package are likely to occur in response to:

- Use of more sophisticated modeling constructs for behavior
- Extensions in the design of agents that require additional functionality

3.5.3.10. SQL

The sql package contains classes that encapsulate software design decisions based upon preferred techniques to access a RDBMS via Java's java.sql package. Use of these classes is not mandated but may improve clarity. The primary secrets of this package are the algorithms and data structures used to implement the classes. Changes in this package result from a desire to improve performance.

3.5.4. Software Generation Package

The Software Generation package encapsulates packages that assist in code generation. The primary secrets of this package are decisions on the most effective way to specify certain aspects of functionality in non-native languages. The secondary secrets are the algorithms and data structures used in, and generated by, these code generation packages. Changes to the Software Generation module are motivated by new ideas on how to represent behavior and functionality.

3.5.4.1. XML DTD Translation

The XML DTD Translation package encapsulates the translation between a DTD and a set of Java classes. It hides the details of marshalling and unmarshalling XML documents. Each DTD element is translated into both an interface and a class.

The primary secrets of this package are:

- The algorithms and data structures used to translate a DTD into Java.
- The classes into which an element is translated (i.e., only the interface should be considered visible outside the package), except for the class that unmarshals an XML document based on the root element.

The IDA prototype simulation uses Zeus to implement this package.

3.5.4.2. LL(1) Translation [5]

The LL(1) Translation package encapsulates the specification of a language as an LL(1) grammar, as well as the translation of an LL(1) grammar into Java. The primary secrets of this package are the properties of an LL(1) parser-generator. The secondary secrets are the algorithms used for translating LL(1) grammars into Java. Changes in this package are likely to be motivated by the desire for increased power in manipulating an LL(1) language.

The prototype uses Sun Microsystems's javacc parser-generator to implement this package.

3.6. Detailed Discussion of Selected Packages and Classes

This section gives an in-depth discussion of selected elements of the prototype simulation. It covers important design and implementation decisions. The underlying issues are likely to arise in net-centric systems. This section is intended to help designers understand trade-offs.

3.6.1. The SL0 Package

SL and its subsets are the preferred FIPA languages for communicating most semantics. FIPA-OS provides minimal support for SL, however. FIPA-OS contains an SL parser, but the output of the parser has proven difficult to use [Duncan 2003]. FIPA-OS also contains an SL0 parser, but that parser only verifies that a string conforms to SL0 syntax; it does not generate a parse tree that an application can analyze. Example FIPA-OS applications included in the distribution use ad hoc parsers. Comments in FIPA-OS documentation mention the need for improved SL support.

In anticipation of complex and disparate messages, the IDA prototype simulation implements a subsystem to parse, manipulate, and de-parse SL0 strings. The package is an exploration of

[5] LL(k) parsers process input from **L**eft to right, and construct a **L**eftmost derivation of the sentence. The number of tokens it uses to look ahead is given by the parameter k.

how an agent-based application can effectively utilize messages whose content is expressed using SL0.

The package, sd.FIPA.sl0, has two major classes:

1. Class SL0, which contains nested classes that implement SL0 concepts.
2. Class SL0Parser, a parser-generator that translates strings into instances of classes in SL0.

The following sections describe each class.

3.6.1.1. Class SL0

The SL0 class encapsulates SL0 concepts. A concept is one of the following:

- A non-terminal in the SL0 grammar, which is encapsulated by a nested class. Most of the concepts encapsulated in SL0 are non-terminals. An important purpose of SL0 is to represent parse trees for the SL0 language. These parse trees are built by class SL0Parser and used by agents.
- An approach to manipulating an SL0 expression, which is encapsulated by either an interface or a procedural abstraction.

Terminals are represented as the Java primitive types Integer, Float, and String.

3.6.1.1.1. Encapsulation of Non-Terminals

Class SL0 contains a nested class for most SL0 non-terminals (some exceptions are discussed below). These classes have get/set methods to access the content of a production. For example, the root production of SL0 is:

```
Content = "(" ContentExpression+ ")".
```

Class SL0 contains nested class Content, whose signature is as follows:

```
public static class Content {
    public Content(ContentExpression ce);
    public Content(List contentExpressions);
    public void add(ContentExpression ce);
    public List getContentExpressions();
}
```

The nested classes do not have remove() methods, as these methods are not needed to construct parse trees.

Not all non-terminals appear in class SL0. The parse tree can be simplified by collapsing SL0 productions that have a single expansion. Consider the SL0 productions:

```
Parameter      = ParameterName ParameterValue.
ParameterName  = ":" String.
ParameterValue = Term.
```

Class SL0 has nested classes Parameter, Strng (the abbreviated name avoids confusion with Java's built-in class String), and Term, but no nested class ParameterName or ParameterValue. Instead, it provides methods in class Parameter to access the name and value:

```
public static class Parameter {
    public Parameter (String name, Term value);
    public String getName();
```

```
    public Term getValue();
}
```

Note also the substitution of String for Strng, possible for the same reason. Collapsing productions simplifies the implementations of both SL0 and classes that import it.

If an SL0 production has multiple right-hand side expansions, the non-terminal on the left-hand side is an abstract class, and SL0 contains a concrete class for each expansion; these concrete classes extend the abstract class. This approach expresses the concept a non-terminal encapsulates while accommodating the necessary variations of the expansions. For example, the SL0 production:

```
ContentExpression  = ActionExpression
                   | Proposition.
```

maps to the following classes in SL0:

```
public static abstract class ContentExpression
        extends BaseSL0Symbol { ... }
public static class ActionExpressionContentExpression
        extends ContentExpression { ... }
public static class PropositionContentExpression
        extends ContentExpression { ... }
```

The chain of abstract classes continues for as long as necessary. The SL0 productions:

```
Term               = Constant | ....

Constant           = NumericalConstant
                   | String
                   | DateTime.

NumericalConstant  = Integer
                   | Float.
```

are encapsulated by the nested SL0 classes shown in Figure 27 (italicized names denote abstract classes).

The nested classes that encapsulate non-terminals all extend class BaseSL0Symbol, an abstract class that requires its descendents to include some convenience methods. Most significantly, BaseSL0Symbol ensures that any invocation of toString() will yield an instance of class java.lang.String that is a valid SL0 representation of the instance. This makes use of SL0.Content instances as the content object of FIPA-OS messages easy:

```
SL0.Content c = new Content();
c.addContentExpression(content expression);
ACL message = new ACL();
message.setContentObject(c.toString());
```

3.6.1.1.2. Evaluating Content

When an agent receives an SL0 message, it must parse and evaluate that message. The following is a discussion of what evaluation means and how the IDA prototype simulation implements it.

An SL0 message contains content. Content is composed of one or more content expressions. Each content expression specifies one of the following:

1. A fact to be added to the agent's knowledge base. This fact is stated as a proposition.
2. A request to the agent to state whether its knowledge base contains, or can be used to deduce, a fact. This fact is also stated as a proposition.
3. A request to the agent to perform an action. The action is stated as an action expression, which has the forms:

   ActionExpression = "(" "action" Agent Term ")".

   where the Term specifies the action that the Agent is requested to perform. An SL0 Term can be an arbitrary expression, but as an action expression it is most often a function that the sender believes the receiver understands. For example, the Tasks agent of the Friendly Organization requests the SimState agent to perform the sensing function (Table 1).

A study of these items shows that an agent cannot evaluate content expressions in isolation from its knowledge base. In fact, something more than a knowledge base is necessary, because of the SL0 FunctionalTerm production:

FunctionalTerm = "(" FunctionSymbol Term* ")"
| "(" FunctionSymbol Parameter* ")".

The FunctionSymbol is a string. SL0 establishes no requirement that it be part of the knowledge base; that is, a FunctionSymbol is not necessarily a class or slot. Evaluating a content expression, then, requires that permissible functions be predefined.

Several messages send by agents of the prototype use functional terms. The most pervasive function has the form:

(position :latitude *lat* :longitude *lon*)

The position function, when evaluated, yields a value that is equivalent to an instance of the AbsolutePoint class in the GH ontology.

Evaluating a context expression can require even more information than knowledge base and functions. How an agent handles a context expression that is a proposition depends on the communicative act associated with the message. The interpretation of:

(has-location *obj-item-name* (position :latitude *lat* :longitude *lon*))

depends entirely on whether it is associated with a message whose communicative act is Inform (item 1) or Query-If (item 2).

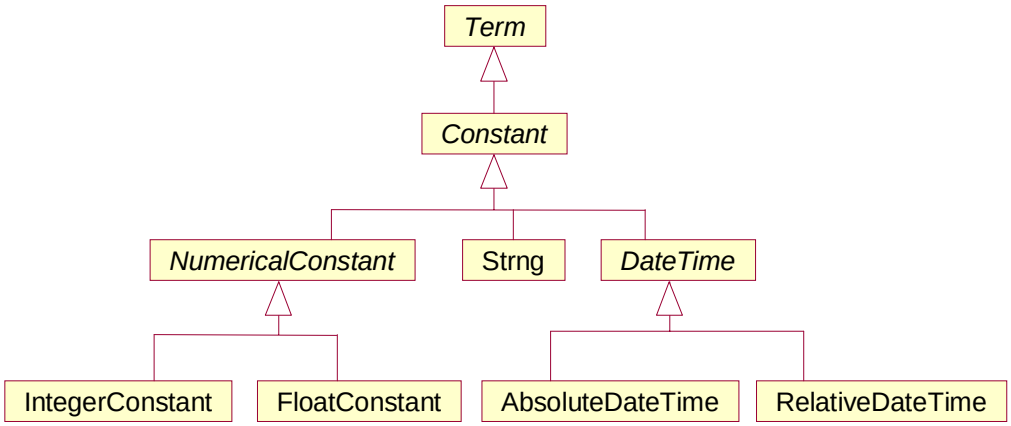

Figure 27. SL0 Class Hierarchy for Constants

These considerations have led to the design for content evaluation that will now be presented. This design is especially suited to SL0, and probably to SL1. It is not so well suited to SL2 or SL.

SL0 classes that encapsulate non-terminals have a method whose signature has the form:

```
public Object evaluate(Context context) throws EvaluationException;
```

(Sometimes the method returns a Boolean or a List.) SL0.Context is an interface. A knowledge base implementation that wishes to allow evaluation of SL0 context should implement the SL0.Context interface.

A context is intended to be more general than a knowledge base. A knowledge base is one type of context, but others are possible. This design approach derives from the need to allow functions, as discussed above. It also accounts for the fact that a full knowledge base can be overkill. The agents in the IDA prototype simulation send messages that do not require the full expressiveness of the GH ontology. Constructing a "pseudo-ontology" that handles just the necessary concepts is often simpler. Section 3.5.3.5 presents more information on pseudo-ontologies.

The SL0.Context interface has the following signature:

```
public interface Context {
    public Boolean evaluateResult(Object term, Object equivalentTerm)
            throws EvaluationException;
    public Boolean truth() throws EvaluationException;
    public String[] getFunctionParams(String functionName);
}
```

The evaluateResult() method must handle evaluation of a Result atomic formula, which has the form:

```
AtomicFormula  = "(" "result" Term Term ")"
```

and means that the sending agent believes the second term to be the result of evaluating the first term. This atomic formula is usually sent in response to a query or an action request, and the receiving agent will want to implement evaluateResult() to assert the fact of equivalentTerm in its knowledge base. For example, the SimState agent sends a result to a Friendly Organization in response to a sensing request. The second term is a set of (object name, position) pairs. The Friendly Organization will want to record these object names as being detected at the specified locations.

The truth() method must implement a proposition whose value is one of the literals "true" or "false". These literals, though valid in SL0, have little use; they are carryovers from more complete SL subsets. FIPA does not specify the effect of asserting "false". Presumably it could wipe a knowledge base clean, whereas "true" would have no effect.

The getFunctionParams() method is part of a larger paradigm for implementing functions, proposition symbols, and predicates. All are handled using dynamic evaluation. Suppose a message contains the position() function given above. The class that implements SL0.Context must:

- Define a function with the following signature:

  ```
  public Object position(Float latitude, Float longitude);
  ```

- Define getFunctionParams() to return:

(String[]) { “latitude”, “longitude” }

when invoked with “position” as its argument.

Now consider what happens when evaluating the SL0 term:

(position :latitude 45.3 :longitude 40.0)

The IDA prototype simulation will parse this string into an instance of SL0.FunctionalTerm. When the evaluate() method is invoked on that instance, it will search its SL0.Context argument for a method named position with two arguments that are both floating-point values. On finding the list of all such methods, evaluate() will invoke the first,[6] passing it the values 45.3 and 40.0, in that order. The evaluate() method does not use the value the position() method returns; bear in mind that this value is part of some larger context defined by the SL0 content expression containing the position function invocation. The evaluate() method only passes on the value as a parameter to the next stage of the contextual evaluation.

This dynamic function evaluation approach is used to handle predicates as well. If an SL0 message contains the atomic formula:

(has-location “3rd Armor Rec Coy” (position :latitude 45.3 :longitude 40.0))

then the evaluate() method that handles the instance parsed from this string will look for a function in the context with the signature:

public Boolean hasLocation(String name, GeoPosition position)

and invoke it with the two terms of the predicate.

Three points deserve mention. First, it must be noted that the predicate name has-location is translated into a Java method name. SL0 allows function names to be arbitrary strings. This won’t do for Java identifiers. Some mapping must be introduced.

Second, the hasLocation() method takes a string and a GeoPosition (see Section 3.6.5) as arguments. That the first argument is a string is obvious from the left term of the SL0 atomic formula. The type of the second argument happens to come from knowledge of the implementation of the position() method.

Third, the hasLocation() method returns a Boolean value because a predicate is a proposition, and FIPA propositions are either true or false. FIPA does not provide much detail on the semantics of evaluation. The effect of a false proposition is unclear.

There are other ways to implement support for predicates. A predicate is analogous to a slot in an ontology, so the SL0.Context interface might have been defined with methods such as the following:

public Boolean assertPredicate(String name, SL0.Term terms);
public Boolean verifyPredicate(String name, SL0.Term terms);

(The first form is for Inform communicative acts, the second for Query-If acts.) A class implementing SL0.Context could then map the name to a slot. For a knowledge base with many slots, this would be considerably easier than defining a separate method for each predicate (i.e., for each slot).

Finally, the getFunctionParams() method is necessary because there are two forms of function terms:

---

[6] The IDA prototype simulation arbitrarily evaluates the first method found. A more realistic approach for a production system would be to throw an exception.

```
FunctionalTerm    = "(" FunctionSymbol Term* ")"
                  | "(" FunctionSymbol Parameter* ")".
```

The examples have all used the second form, in which parameters are named. FIPA discourages but does not forbid use of the first form, with its unnamed parameters:

```
(position 45.3 40.0)
```

The algorithm that matches methods to functions works with parameter names, so an SL0.Context must be able to determine the name of each parameter. The getFunctionParams() method supplies these names. Its return value is an array of strings, each of which is the name of a parameter. The order of the names in the array determines the assignment of names to values in the SL0 expression.

3.6.1.2. Class SL0Parser

Class SL0Parser converts an SL0 expression into an instance of a nested class in SL0. It is usually used to parse content expressed in SL0 through the static parseContent() method:

```
SL0.Content c;
c = SL0Parser.parseContent( acl.getContentObject().toString() );
```

Class SL0Parser is implemented using Sun's javacc parser generator. Javacc simplifies the construction of top-down parsers by allowing them to be expressed as grammars. For example, the code for recognizing the production

```
Content = "(" ContentExpression+ ")".
```

is:

```
SL0.Content Content():
{    // Declare variables needed to accumulate ContentExpression instances.
     SL0.ContentExpression  contentExpr;
     List                   elements       = new LinkedList();
}
{    // The following derives directly from the grammar.
     <Lparen> ( contentExpr = ContentExpression() { elements.add(contentExpr); } )+ <Rparen>
     {
          try {
               return new SL0.Content(elements);              // Construct and return the instance.
          } catch ( SL0.InvalidContentException e ) {
               throw new ParseException(e.getMessage());      // Oops.
          }
     }
}
```

The grammar in SL0Parser follows the SL0 grammar very closely. There are a few exceptions:

1. The SL0 production for an Agent (i.e., the first operand of an Action Expression) is a Term:

   ```
   Agent = Term.
   ```

   A term is a constant, set, sequence, functional term, or action expression.

   The FIPA-00070 specification places other restrictions on the syntax of an Agent. It must have the form:

"(" "agent-identifier" ":name" word [":addresses" URLSequence]
[":resolvers" AgentIdentifierSequence]
(UserDefinedParameter Expression)* ")".

The parser accepts a Term by default, but can be made to accept only agents that conform to FIPA-00070.

2. FIPA's definition of a double-quoted string is extremely liberal; the ending double quote is optional. This isn't easy to recognize using javacc, so SL0Parser requires matching quotes.

3.6.1.3. Canonical Code for Handling INFORM Messages

The following fragment shows the code an agent uses to handle a message with the INFORM communicative act, assuming the class ContextClass implements the Context interface. Real code would have more sophisticated error handling.

```
public void handleInform(Conversation conv) {
    ACL acl = conv.getACL(conv.getLatestMessageIndex());
    if ( ! acl.getLanguage().equals(SL0Parser.getLanguage()) ) {
        sendNotUnderstood(acl);          // The language isn't SL0.
        return;
    }

    SL0.Content content;
    try {
        content = SL0Parser.parseContent((String)acl.getContentObject());
    } catch ( ParseException e ) {
        sendNotUnderstood(acl);          // The content is syntactically incorrect.
        return;
    }

    if ( ! acl.getOntology().equals(ContextClass.getName()) ) {
        sendNotUnderstood(acl);          // The ontology isn't what's expected.
        return;
    }

    SL0.Context context = new ContextClass(acl, this);
    try {
        content.evaluate(context);       // Here's the evaluation.
    } catch ( SL0.EvaluationException e ) {
        sendNotUnderstood(acl);          // Evaluation failed.
    }
}
```

Note that the ContextClass constructor takes the ACL as an argument. This is usually necessary because, as discussed in Section 3.6.1 above, interpretation of SL0 depends on the communicative act associated with the ACL.

The constructor also takes the task instance as a parameter. Experience with the IDA prototype simulation has shown that having the ContextClass invoke callback methods in its parent is a useful design paradigm.

3.6.2. The Ontology Package

The IDA prototype simulation is based on the GH ontology. This ontology provides the data model. The rules for component behavior trace to the GH ontology, either directly or indirectly.

The GH ontology is implemented in Protégé-2000. Protégé-2000 is one of many tools for creating and editing knowledge bases. There is a distinct possibility of using another tool, perhaps to improve performance, or to gain access to new functionality.

Therefore, the prototype hides the use of Protégé-2000. The only package that is aware of its use is the gh5ontology package (Section 3.6.3). This package must know because:

- It is responsible for loading the knowledge base, which requires knowledge of the tool to perform the loading.
- It has expectations of the capabilities of the knowledge base. It is aware that the GH ontology supports inferencing in Algernon and constraint checking in PAL.

The rest of the IDA prototype simulation has no knowledge that Protégé-2000 is used. Instead, the IDA prototype components access a knowledge base through the ontology package. This package contains a set of interfaces that standardize the model of a knowledge base. The interfaces are drawn from Protégé-2000's package edu.stanford.smi.protege.model package. This simplifies implementation. It also means that interfaces in the ontology package follow OKBC, giving them some authority as a useful model.

The interfaces are:

1. KnowlegeBase, which defines operations for accessing and deleting frames. Each instance of KnowledgeBase represents a single knowledge base.
2. Frame, which denotes a single frame within a knowledge base. The operations provide for accessing own slots.
3. Instance, which denotes a frame that is an instance.
4. Cls, which denotes an Instance that is a class within a knowledge base. Its operations access superclasses and subclasses, and allow the creation of an Instance.
5. Slot, which denotes an Instance specialized to hold values of a specific type. Its operations query the types of values allowed.

The methods in these interfaces are derived almost directly from Protégé-2000. Not all Protégé-2000 methods have been included. The IDA prototype simulation assumes that agents will use existing ontologies, not construct them; therefore, it omits methods such as those that add classes or add template slots to classes.

The ontology package includes two other interfaces:

6. ValidatableCls, an extension of Cls denoting a class that may have associated validation rules. It has a single method, validate(), that when invoked is expected to apply any class-specific validation rules in the knowledge base. The meaning of a validation rule is left up to the class that implements the interface.
7. ATKnowledgeBase, an extension of KnowledgeBase denoting a knowledge base for which "ask" and "tell" queries may be posed. (An "ask" query requests the knowledge base to supply all instances that satisfy the query. A "tell" query requests the knowledge base to store some fact.)

These two interfaces are intended to support PAL and Algernon, respectively, while hiding the validation and query systems a knowledge base uses. The interfaces are experimental and should undergo revision based on experience gained from other knowledge base editors.

The ontology package has a subpackage named impl. This package in turn has subpackages, each of which represents an implementation of the interfaces in ontology with respect to a particular knowledge base tool. The prototype currently implements only one such subpackage: protege, which implements the interfaces for Protégé-2000. Classes in the protege package implement interfaces in ontology; their names are the name of an interface suffixed with Impl. ClsImpl implements ValidatableCls and KnowledgeBaseImpl implements ATKnowledgeBase, reflecting the validation and inferencing capabilities available in Protégé-2000.

The classes in the protege package are wrappers around the corresponding Protégé-2000 class. (Protégé-2000 also uses the interface/class distinction, preferring that applications refer to the interfaces. Extending Protégé-2000 classes is, therefore, not an option.) Each class has a field containing the underlying Protégé-2000 frame. Methods refer to that frame to obtain results.

The classes do not have public constructors. An application may create a frame or knowledge base only through specific methods; this ensures that a knowledge base is aware of all of its frames. The protege package contains a class named ProtegeProject with a single constructor:

```
public ProtegeProject(String kbURL);
```

and a single method:

```
public KnowledgeBase getKnowledgeBase();
```

The ProtegeProject class is a wrapper around the Protégé-2000 class Project. It is used only to test whether the project opened successfully (the constructor throws an exception on failure) and to retrieve the associated knowledge base. This knowledge base is an instance of KnowledgeBaseImpl, i.e., it implements the KnowledgeBase interface. The constructor creates this instance using the non-public constructor of KnowledgeBaseImpl.

3.6.3. The GH ontology Package

The ontology package hides the implementation of knowledge base technology. A GH ontology has additional information that can, and should, be encapsulated. This information includes:

1. *The representation of own slot values.* An agent may be expected to know the elements of the GH ontology (class names, template slot names). It may also be expected to know the underlying domain of a slot, as taken from the corresponding attribute in the Generic Hub (integer, real, string). However, the representation of own slot values is subject to change in the GH ontology; future versions may not use the Type ontology, for example.
2. *Procedural abstractions of rules for accessing GH concepts*. For example, agents often need to know if one organization commands another. The rules for determining command hierarchy stem from Generic Hub coded domains that may be subject to extension. Furthermore, the rules can be implemented several ways (Java, Algernon, etc.). They should be consolidated to simplify agent maintenance in the event of change.

The gh5ontology package encapsulates these and other concepts. Its most important class is GH5KB. The constructor of this class takes as a parameter a KnowledgeBase instance; this in-

stance is expected to be a GH ontology (if it isn't, its use will soon cause exceptions). The instance may then be used to:

- Get and delete frames, as prescribed by the KnowledgeBase interface.
- Get and set slot values in what seems the most natural and direct way. Consider the following fragment:

```
GH5KB kb = new GH5KB(ProtegeProject(gh-ontology-URL).getKnowledgeBase());
Cls materielCls = kb.getCls("Materiel");
Collection allMateriel = materielCls.getInstances();
for ( Iterator mIter = allMateriel. iterator(); mIter.hasNext(); ) {
    Instance materiel = (Instance)mIter.next();
    // Here are some methods defined in GH5KB.
    String name = kb.getStringSlot(materiel, "name");
    String bodyColor = kb.getCodeSlot(materiel, "bodyColourCode");
    System.out.println(name + " has colour " + bodyColor);
}
```

  Methods getStringSlot() and getCodeSlot() hide the details of how the GH ontology represents a string and a coded domain.
- Retrieve an Instance denoting a battlefield object based on its name and class:

```
Instance org = kb.getBattlefieldObject(kb.getCls("Unit"), "Panzerbataillon 433");
```

- Retrieve all battlefield objects, which agents use to search for instances conforming to some property.
- Calculate the speed of a battlefield object based on observations of its position.
- Determine command properties of an organization.

These and other methods encapsulate business rules for the GH ontology.

The gh5ontology package contains the following additional classes:

- Class DateTime converts between GH data model representations of dates and times and Java's java.util.Calendar instances.
- Class NumericExpression encapsulates the Numeric-Expression ontology. It hides how this ontology uses slots to represent numeric expressions. It also contains an evaluate() method that, given an Instance whose class is Numeric-Expression, returns the java.lang.Number resulting from the evaluation of the instance.
- Classes ReportingData, RDComparator, and RDEComparator support comparing a collection of ReportingData instances, or alternately a collection of instances that have associated ReportingData. The comparison is based on either the date and time a ReportingData instance was created, or the effective date and time specified for the instance. The methods can:
    - Compare two instances, returning -1, 0, or 1 depending on whether one instance is less than, equal to, or greater than the other (the usual Java paradigm specified by the java.lang.Comparable interface).
    - Sort a list of instances.
    - Retrieve the most recent instance from a collection of instances.

3.6.4. The Ontology-SQL Binding Package

The Generic Hub was designed with the expectation that it would be implemented as a relational database schema. The prototype can create a knowledge base from a Generic Hub data set accessed via an SQL RDBMS. The IDA prototype simulation can also save changes to a knowledge base to the data set used to create that knowledge base. This functionality is implemented in the osb (Ontology-SQL Binding) package.

The osb package contains functionality that is specific to linking a Generic Hub data set to a GH knowledge base. The IDA prototype simulation does not claim to encapsulate general purpose functionality for translating between an arbitrary knowledge base and an arbitrary database schema. An architecture for such functionality would be useful, but time constraints did not permit a study of the issues.

The osb package contains a class OntologySQLBinding that implements the majority of the package's functionality. It offers the following methods:

1. A constructor that creates a binding. Its signature is:

   ```
   public OntologySQLBinding( SchemaKBMap map,
                              GH5KB kb,
                              Connection dbConnection);
   ```

   Note that the binding involves a GH5KB instance, not an arbitrary ontology. It is expected to be empty when the constructor is invoked. The other two parameters are:

   a) A SchemaKBMap specification of how Generic Hub elements map to classes and slots of the GH ontology. This map is taken from an XML document whose DTD is translated into Java classes using Zeus. Its DTD is discussed in Section 3.6.4.2.
   b) A java.sql.Connection to a DBMS via JDBC.
2. A method to populate the knowledge base given to the constructor using data obtained through the connection, translating data according to the map.
3. Methods to update the knowledge base with all data entered after a certain time, typically the later of the last update or the last save. Each Friendly Organization component has a different knowledge base; these methods ensure that, if two components share the same data set, each component's knowledge base reflects knowledge obtained by the other.
4. Methods to save instances into the underlying data set. A Friendly Organization component's inferences sometimes generate new facts in the knowledge base. These methods perform the reverse of the update methods.

3.6.4.1. Saving and Updating Instances

The save and update operations require some consideration. The desired approach to saving would be to implement a method whose signature is simply:

```
public void save();
```

and whose effect is to save all changes since the last save or update, whichever is more recent.

This, however, is difficult to implement for two reasons. The first is determining which instances in the knowledge base have been saved or modified since the last save operation.

Marking algorithms for this sort of problem are well known, but not especially easy to implement.

The second reason is that instances cannot be saved in an arbitrary order. The standard SQL script to create a Generic Hub data set places integrity constraints on tables. Subtype tables require a corresponding row in their supertype tables. A foreign key in the child table requires the existence of a corresponding row in the linked parent table.

The IDA prototype simulation circumvents these problems by having OntologySQLBinding require that instances to save be explicitly specified, and that they be specified in an order consistent with integrity constraints. An example signature of a method in osb to save instances is:

```
public void saveInstances(List instances);
```

This solution violates information hiding principles, because it relies on packages outside of osb knowing of integrity constraints imposed by the RDBMS. It is not recommended for a production system.

The issue with an update operation is that it potentially requires comparing an entire Generic Hub data set against a knowledge base. This would be unacceptably slow. One solution would be to extend the Generic Hub schema with an additional table (or tables) that record change. Locally extending a standard, however, may negatively impact data interoperability and should be avoided whenever other approaches can suffice.

The osb package instead makes an assumption: All changes to a Generic Hub data set will involve an instance of REPORTING-DATA. This assumption is justifiable, because Generic Hub data comes from observations, and REPORTING-DATA records information about an observation. Moreover, each REPORTING-DATA row contains information on the date and time it was entered. The date and time are used to retrieve all data entered after a specified moment.

Each REPORTING-DATA row is associated with a row from an associative table. This row is presumed to be new. The update method essentially performs a transitive closure, following foreign keys to look for new or modified rows.

Assuming that all updates involve REPORTING-DATA has some flaws. Almost, but not all, tables in the Generic Hub are consistent with this assumption. One counterexample is OBJECT-TYPE-CAPABILITY-NORM. If Generic Hub data set were populated with new object types and their capabilities, the update methods would not detect the change. Another involves actions: neither ACTION-FUNCTIONAL-ASSOCIATION nor ACTION-TEMPORAL-ASSOCATION has associated REPORTING-DATA. REPORTING-DATA is associated with dynamic data, not static data of the sort that is likely to be used in the initial population of a Generic Hub data set. The mutability of tables containing such data is not known. One solution would be to restart decision support systems when these tables are changed, an approach that may or may not be practical. The problem needs more study.

3.6.4.2. The Schema Map

To create knowledge base instances from rows of a table, and conversely to save instances as new rows of a table, requires some knowledge of the relationship between the GH ontology and the Generic Hub schema. This knowledge is encapsulated by an XML document, the format of which is defined by the DTD shown in Figure 28. This document essentially recapitulates all Generic Hub physical elements (tables and columns, not entities and attributes), noting relationships to the GH ontology for each element. For each Generic Hub table, the

```
<!ELEMENT schema-KB-map (table-map+)>
<!ELEMENT table-map (primary-key, organic-column*, relationship*)>
<!ATTLIST table-map
        maps-to-cls      CDATA   #REQUIRED
        table-name       CDATA   #REQUIRED
        discrim-value    CDATA   #IMPLIED
        discrim-column   CDATA   #IMPLIED>
<!ELEMENT primary-key (key-column+)>
<!ELEMENT key-column EMPTY>
<!ATTLIST key-column
        name        CDATA                 #REQUIRED
        type        (integer|float|string) #REQUIRED
        slot-name   CDATA                 #IMPLIED>
<!ELEMENT organic-column EMPTY>
<!ATTLIST organic-column
        name          CDATA          #REQUIRED
        maps-to-slot  CDATA          #REQUIRED
        slot-type     (int|float|string) #IMPLIED
        match         (false|true)   "false"
        trim          (false|true)   "false">
<!ELEMENT relationship (key-column+)>
<!ATTLIST relationship
        slot-name             CDATA   #REQUIRED
        associated-cls-name   CDATA   #REQUIRED>
```

Figure 28. Schema Map DTD

document gives the table's name and notes the class to which it maps. It also categorizes the table's columns: primary keys, organic columns, and relationships. A primary key is specified simply as a column name and a type (one of integer, float, or string). An organic column is specified as a name, a type (one of int, float, or string), and the name of a single-cardinality slot to which it maps; this slot must be a template slot of the class to which the table maps. See Figure 29, which shows how the NETWRK_SERVICE table maps to the NetworkService class.

An organic column has two other attributes. The trim attribute applies to columns whose type is string, and indicates that values should be trimmed prior to being used as the values of instances. The match attribute, which also applies to string-typed columns, indicates that the value must match an existing value of the slot type; new values are not to be created. The trim and match attributes together denote a coded domain. During initial design of the osb package, it seemed probably that trim would be used without match, but this never occurred.

A relationship describes a table's foreign keys. The relationship is, therefore, associated with the child table. The attributes of the relationship are the GH ontology slot that models it (the single cardinality slot with a multiple cardinality inverse) and the name of the class with the multiple cardinality slot. For example, the relationships for table NETWRK_SERVICE describes that class Network has a multiple-cardinality slot provides-NetworkService. Note also that the specification of the primary key for NETWRK_SERVICE shows that column netwrk_id maps to slot netwrk_service-is-provided-by-Network. This is the inverse of slot provides-NetworkService.

```
<table-map table-name="NETWRK_SERVICE" maps-to-cls="NetworkService">
    <primary-key>
        <key-column  name="netwrk_id" type="integer"
                     slot-name="netwrk_service-is-provided-by-Network" />
        <key-column name="netwrk_service_ix" type="integer" />
    </primary-key>
    <organic-column  name="cat_code" maps-to-slot="categoryCode"
                     trim="true" match="true" />
    <organic-column  name="subcat_code" maps-to-slot="subcategoryCode"
                     trim="true" match="true" />
    <relationship slot-name="provides-NetworkService" associated-cls-name="Network">
        <key-column name="netwrk_id" type="integer" />
    </relationship>
</table-map>
```

Figure 29. Schema Map Fragment

The DTD contains two other (related) attributes that merit discussion. Databases implement subtype relationships through a discriminator attribute. The GH ontology only partially preserves this attribute. All discriminators are named categoryCode; unlike the logical model of the Generic Hub, the GH ontology makes no distinction between object-item-category-code and organisation-category-code. This is a reasonable design decision, because the GH ontology contains type operations that make discriminators in superclasses redundant. However, discriminator values are necessary when saving instances. When the prototype saves an instance of AbsolutePoint, for example, it must save rows in tables ABS_POINT, POINT, and LOC. It must know to set the value of the cat_code column of POINT to 'ABS', and the value of the cat_code column of 'LOC' to 'PT'.

This information is encoded in the discrim-value and discrim-column attributes of the table-map element. The discrim-value attribute is the value to which the discrim-column attribute of the supertype table must be set. For example:

```
<table-map table-name="POINT" maps-to-cls="Point" discrim-value="PT" discrim-
column="cat_code">
<table-map table-name="ABS_POINT" maps-to-cls="AbsolutePoint" discrim-value="ABS">
```

The schema map focuses on how tables map to classes. This reflects the decision to implement database load operations before knowledge base save operations.

Finally, the tables must be given in the XML document such that supertypes appear before subtypes. This restriction simplifies parsing and processing.

3.6.5. The GPC Package: A Pseudo Ontology

Expressing facts about the GH ontology can be complex. For example, the description of a battlefield object *o*'s current reported location involves a sequence of relationships involving ObjectItem, ObjectItemLocation, ReportingDataAbsoluteTiming, and AbsolutePoint:

Equation 1. Set-Theoretic Expression for Last Reported Location

$$loc = \{\, p \in \text{AbsolutePoint} \mid \forall\, oil \in \text{ObjectItemLocation} \;\; \forall\, r \in \text{ReportingDataAbsoluteTiming}$$

is-geometrically-defined-through(*o*, *oil*)
∧ is-associated-with(*oil*, *p*)
∧ is-referenced-to(*oil*, *r*)
∧ ¬∃ *r′* ∈ ReportingDataAbsoluteTiming
((effectiveDate(*r′*) = effectiveDate(*r*) ∧ (effectiveTime(*r′*) > effectiveTime(*r*))
∨ effectiveDate(*r′*) > effectiveDate(*r*)) }

The problem as concerns agent development is not the complexity of this expression. Complexity must not be discounted, for it increases both the probability of errors during development and the difficulty developers face during maintenance of comprehending an agent's behavior. Simplicity is always desirable. But the real problem is implementing functionality to interpret these expressions.

Set-theoretic expression interpreters are inherently slow. As mentioned on p. B-26, many expressions are computationally intractable. Agent designers must exercise caution when considering what facts they might add to a knowledge base. Asking an interference engine to evaluate an expression such as the above will strain computational resources.

FIPA recognizes this problem and addresses it through its subsets of SL. These subsets are, however, limited. SL0 cannot express

Equation 1, because it lacks Boolean operators. SL1 cannot either; it lacks quantifiers. SL2 can (though not quite in this form; it does not allow quantifiers to be nested inside anything other than quantifiers). Implementing support for SL2 is effort-intensive, however.

The challenge in the IDA prototype, then, was how to express complex facts. Using SL2 was deemed infeasible because of lack of resources to pursue that approach. Its use also remains unproved in production systems.

The IDA prototype instead introduces the concept of a pseudo-ontology. A pseudo-ontology is built on top of another ontology, rather like the C2-Concepts ontology is built on top of the GH ontology. A pseudo-ontology's distinguishing feature is that it defines an explicit translation between its relationships and its underlying ontology. This translation can be implemented in a language such as Java. The translation can include knowledge of the underlying ontology and therefore can be far more efficient than an SL (or SL2) expression evaluator. Agents can express facts in SL0 or SL1. The prototype uses SL0, expressions of which are always computationally tractable.

Table 1 shows some pseudo-ontologies. The geo-position ontology, for example, concisely expresses the last known recorded location of a battlefield object as an absolute point. (The location is most recent because the ontology defines it to be so).

A pseudo-ontology contains as few classes and slots as are necessary to express facts. Typically only one or two classes, and not many more slots, are needed. Consider the following fact in the geo-position ontology, expressed in SL0:

((has-location "Panzerbattalion 433" (position :latitude 43.20 :longitude -78.6)))

This fact can be modeled using a single class: Geo-Position, with template slots latitude, longitude, and name (because the ontology contains a single class, has-location does not have to be modeled explicitly). When an agent receives the above fact, it translates these slots into classes in the GH ontology. The IDA prototype currently has the SimState agent send geo-position ontology messages to the Friendly Organization component, which assigns the HumanIntelTask to receive them. The HumanIntelTask:

- Creates an instance of AbsolutePoint using the given latitude and longitude.
- Finds the instance of ObjectItem with the given name.
- Creates the necessary additional class instances:
    - ObjectItemLocation, which it relates to the ObjectItem and AbsolutePoint instances.
    - ReportingDataAbsoluteTiming, for which it sets the timing characteristics from the moment when it received the message.

A message using a pseudo-ontology requires support for both the sender and the receivers. Both can benefit from encapsulation of shared concepts. However, each has different needs from the ontology. The sender is concerned with packaging content. The receiver is concerned with understanding it.

The prototype includes a package named gpc. This package implements:

- An interface GeoPosition and its implementation class GeoPositionImpl. This class provides get/set methods for name, latitude, and longitude, and represents a relationship of an object item name to a geodetic position.
- An abstract class AbstractGeoPositionContext. This class implements the SL0.Context interface, and provides:
    - A specification of a function named position. The domain of this function is an ordered pair of floating-point values denoting latitude and longitude, respectively. The range is GeoPosition.
    - The FIPA-OS ACL instance that contained the fact. This instance is maintained as a protected field of the class.

Section 3.6.1.1.2 discussed how a class that implements SL0.Context need not be a knowledge base. AbstractGeoPositionContext is such a class. Pseudo-ontologies in the IDA prototype do not need the full power of an ontology.

Both the SimState and the Friendly Organization components extend class AbstractGeoPositionContext. (The Enemy Organization component does not; it sends messages that use the position function, but it never evaluates them.) The SimState component defines a GeoPositionContext class that adds:

- An implementation of the has-location predicate that records the location of the indicated object.
- An implementation of the sensing action.

The Friendly Organization component, which requests the SimState component to perform the sensing action, receives responses of the form:

```
((result
    (action simstate-agent-name
        (sensing
            :center (position :latitude lat-1 :longitude lon-1)
            :northwest (position :latitude lat-2 :longitude lon-2)))
    (set (position :latitude lat :longitude lon :name n) …)))
```

The action term identifies the message sent. The set term contains the results. The GeoPositionContext class of the Friendly Organization component adds an evaluateResult() method that iterates through each element of the set. GeoPositionContext lets the instantiating instance determine what to do with each element. In the Friendly Organization component, class HumanIntelTask instantiates GeoPositionContext, and for each element it compares the lo-

cation with the previously known location (if any), using this information to determine if the sensed object is moving and posing a threat.

3.6.6. The SQL Package

Package java.sql, a standard component of the Java Runtime Environment, has classes and methods for accessing a data set using SQL. These classes provide most of the functionality needed for database access. However, the IDA prototype contains some common code that has been encapsulated in a package named sql as procedural abstractions:

- SQL has a standard for quoting strings that differs from Java's. In SQL, embedded single quotes must be escaped by doubling them. The sql package provides methods quote() and unquote().
- Of the many data types SQL supports, the Generic Hub uses only three: strings, integers, and floating-point values. Encapsulating this knowledge ensures that all other packages must conform to the Generic Hub's typing structure. The sql package provides public fields that define the range of permissible types.
- New instances require unique keys. The sql package contains methods to support generating keys. (The IDA prototype makes specialized assumptions about key generation.)
- A single knowledge base save operation can executes multiple update and insert queries. To help ensure database integrity, the sql package provides a method to handle these queries as a single transaction.
- Not all SQL RDBMSs implement transactions (Microsoft Access is one example). The sql package encapsulates transaction management to hide the lack of availability of transactions.

3.6.7. The FIPA Package

Apart from SL0, there are many opportunities for encapsulating design decisions regarding the use of FIPA standards and FIPA-OS.

3.6.7.1. Searching for Agents

Locating other agents is a fundamental agent action. Moreover, an agent is seldom satisfied to perform a single search. It must know what agents are available at the time it needs other agents to perform a task.

The FIPA package includes two generic classes for finding agents. They are:

1. Class AgentSearchTask, which is an extension of the FIPA-OS Task class. It is instantiated with a string describing the type of agent for which to search. On completion, it invokes method:

   ```
   public void doneAgentSearchTask(Object results)
   ```

   in its parent task. The results parameter is an array, each element of which describes an agent with the service type specified on instantiation.

2. Class PeriodicAgentSearchTask, which performs an agent search repeatedly. Its constructor takes a service type and an integer:

   ```
   public PeriodAgentSearchTask(String serviceType, int requeryInterval)
   ```

   The effect is to start a search for the specified service type. The search will repeat every requeryInterval seconds until explicitly stopped by the parent.

   Class PeriodicAgentSearchTask is an abstract class. Its subclass must declare a method:

public void utilize(Collection agents, boolean changed)

This method is invoked each time a search completes. The agents parameter gives the agents that match the service type. The changed parameter specifies whether this collection is identical to that given when utilize() was last invoked. (Knowing of no change can save an agent from having to check whether information needs to be resent.)

The FIPA package also contains two classes for finding specific types of agents:

1. Class SimStateAgentSearchTask, which performs a search for a SimState agent. Because each execution of the IDA prototype simulation must have exactly one SimState component, this task searches until it finds the component, then ends. It uses the AgentSearchTask class to perform the actual search.
2. Class AbstractFindOverviewAgentsTask, which searches for all Overview agents. It extends PeriodicAgentSearchTask by providing the service type for which to search and the search interval. It is an abstract class for which a subclass must still implement utilize().

For example, the SimState component implements its search for Overview agents in its IdleTask, as follows:

```
public class IdleTask extends Task {
    private Collection _overviewAgents = new LinkedList();
    …
    /** Callback method invoked automatically by FIPA-OS when the task starts. */
    protected void startTask() {
        // Begin a continuous search for OverviewAgents.
        newTask( new FindOverviewAgentsTask() );
    }
    …
    public class FindOverviewAgentsTask extends AbstractFindOverviewAgentsTask {
        public void utilize(Collection overviewAgents, boolean changed) {
            if ( changed ) {
                synchronized ( _overviewAgents ) {
                    _overviewAgents = new LinkedList(overviewAgents);
                }
            }
        }
    }
```

Other methods in IdleTask may assume that _overviewAgents contains the most recent search results. Note the synchronization, which is generally necessary in applications using FIPA-OS. Messages may be received at any time, and may overlap with the invocation of utilize().

3.6.7.2. The C4IAgent Class

The FIPA-OS paradigms for actions performed on start-up and shutdown vary little between agents:

- All agents extend FIPA-OS classes FIPAOSAgent and must invoke its constructor with parameters that, except for the agent name, don't change.

- Agents register with the AMS and then the DF. This involves catching some exceptions that are noted in a log file but otherwise ignored.
- On shutdown, agents deregister with the DF and then the AMS. As with registration, exceptions entail no response other than noting their presence.

The prototype encapsulates agent start-up and shutdown actions by creating abstract class C4IAgent. This class removes about 50 SLOC from every agent in the prototype.

3.6.7.3. The FIPAACL Class

FIPA specification 00070 describes the string representation of an ACL message. The prototype includes a class FIPAACL whose role is to encapsulate knowledge of that representation. Currently it contains two methods: quote() and unquote(). Callers invoke the quote() method as part of creating an ACL message to ensure that message content is properly formatted. Callers invoke the unquote() method on receiving an ACL message to extract content.

3.6.7.4. The Notification Package

Several agents need to inform other agents of some fact. This action occurs sufficiently often that it pays to encapsulate it. The notification package contains two classes:

1. Class InformAgentTask, which informs a single agent. That is, it creates an ACL message using specified content, language, ontology, sets the message's communicative act to INFORM, and sends that message to a specified recipient. When an instance of InformAgentTask exits, FIPA-OS will invoke the doneInformAgentTask() method in the parent.
2. Class InformAgentTask, which is identical to InformAgent, except that it informs a set of agents rather than a single agent.

Both classes use the No-Protocol protocol, meaning that the sending agent expects no response from receivers. Use of No-Protocol is probably unrealistic in a production environment. More study of the abstraction is needed to determine how sending messages with other protocols could be encapsulated.

3.6.8. Java Properties

Java encourages the specification of runtime configuration parameters through properties. Properties are accessible – and settable – through class java.lang.System. Properties standardize how a Java application can query its environment.

The IDA prototype simulation relies on properties (as do the systems the IDA prototype uses). The IDA prototype makes use of the properties shown in table 2. All properties are prefixed with "ida" to distinguish them from any other properties that might be needed.

Table 2. Runtime Properties Used by the Prototype

Property	Domain	Purpose
ida.app.interactive	0 or 1	If 0, the application is not running interactively. It will have no user interface, and therefore no human controller.
ida.config.file	Any URL	The URL of a configuration file for the loader. See Section 2.6.
ida.datasource	db	The source from which Generic Hub data is to be obtained. Currently this property's value must be the string "db".
ida.db.url	Any URL	The URL of the RDBMS that will be used to access a

		Generic Hub data set. Its format is RDBMS vendor specific.
ida.db.username	Any string	A user name to access the RDBMS, if one is required.
ida.db.password	Any string	A password for the user name, if one is required.
ida.db.sqldriver	Fully qualified Java class name.	The name of the Java class that implements JDBC for the RDBMS.
ida.ontology.url	Any URL	The URL of the GH ontology. It must be a valid Protégé-2000 project name.
ida.schemamap.url	Any URL	The URL of a schema map file. See Section 3.6.4.

Some of these properties (e.g., ida.db.username) are optional. Moreover, some can be specified in a loader configuration. The loader will set properties as specified so that each system component started by the loader finds its properties as expected.

3.6.9. Protocols

FIPA specifies that agents shall conduct conversations according to an interaction protocol. An interaction protocol (protocol for short) defines valid sequences of communicative acts. An agent that adheres to a protocol can infer the state of the conversation (that is, what acts have occurred so far) and the possible responses to a message (that is, what acts can occur next).

FIPA specifies several protocols, and also permits agents to define their own. FIPA-OS supports the predefined FIPA protocols (fipa-request, fipa-query, no-protocol, etc.). Agents should use predefined protocols whenever possible, as the sending agent is assuming the receiving agent understands its protocol.

Many FIPA protocols have been developed to support negotiation (e.g., fipa-brokering, fipa-auction-english, fipa-propose). Such protocols are not suited to the decision support functionality implemented by the IDA prototype simulation. Its agents expect 'yes' or 'no' answers, not barter and compromise.

The IDA prototype simulation uses two protocols. Mostly its agents send messages using protocol no-protocol. Under this protocol, agents send a message but expect no response (see Section 3.1.3.3). These messages use the Inform communicative act.

The IDA prototype simulation also introduces its own protocol, which is named request-reply. Figure 30 shows the protocol as a sequence diagram. The sender (the HumanIntelTask of a Friendly Organization) requests the receiver (the IdleTask of the SimState agent) to provide information; the SimState agent either provides that information, or indicates failure (failure

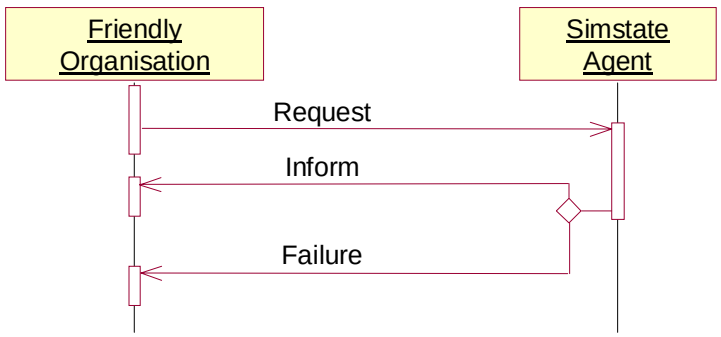

Figure 30. The Request-Reply Protocol

occurs if the Friendly Organization sends a malformed request).

This protocol is a simplification of the fipa-request protocol. It avoids that protocol's extensive error handling. The request-reply protocol, while appropriate for a prototype, is probably not suited to a production system.

It's also worth noting that more advanced decision support systems could make use of bartering protocols. An organization's decision support application might send a request for materiel, facilities, and personnel in support of its current task. The receiving agents would balance the request against current supplies and objectives, and respond accordingly.

References

[Duncan 2003] A. Duncan, private communication.

[Britton 1986] K. Britton, P. Clements, L. Kurokswi, and D. Parnas, A-7E Software Module Guide. NRL Memorandum Report 4702, Revision 2.5, Naval Research Laboratory, Washington, DC (November 1986).

[Enhydra] http://zeus.enhydra.org/.

[FIPA] http://www.fipa.org/.

[FIPA-70] FIPA ACL Message Representation in String Specification. http://www.fipa.org/specs/fipa-00070.

[FIPA-OS] http://fipa-os.sourceforge.net/.

[Java] http://java.sun.com/.

[Parnas 1972] D. Parnas, "On the criteria to be used in decomposing systems into modules." *Comm. ACM* 15(12), pp. 1053–1058 (December 1972).

[Protégé] http://protege.stanford.edu/.

[Smalltalk] A. Goldberg and D. Robson, Smalltalk-80: The Language and Its Implementation. Addison-Wesley, Reading, MA, 1983.

Annex C: Friendly Organization Implementation

This annex presents the implementation of the Friendly Organization component of the IDA prototype simulation. The Friendly Organization component exhibits the most complex behavior of all components. It also makes the most extensive use of agents and ontologies.

Annex B gives a high-level view of the Friendly Organization component's design. This annex complements Annex B with detailed design and implementation information. Annex B breaks components down into packages. This annex continues with classes and their contents.

1. Package and Class Structure

Figure 1 recounts the packages of which the Friendly Organization component consists, and shows the outer classes in each package. Behavior related to a friendly organization is encapsulated within the friendlyorg package. The direct classes of friendlyorg implement a decision support application. Subpackages of friendlyorg implement agent-specific functionality. Subpackages have an outer class named IdleTask that implements the root task of the FIPA-OS task hierarchy. A FriendlyOrg is an application, not an agent, so it has no IdleTask. The class Tasks, however, is implemented as an agent (see Annex B, Section 3.2.2) and has an inner class IdleTask.

Figure 1 illustrates some of the naming conventions used for classes:

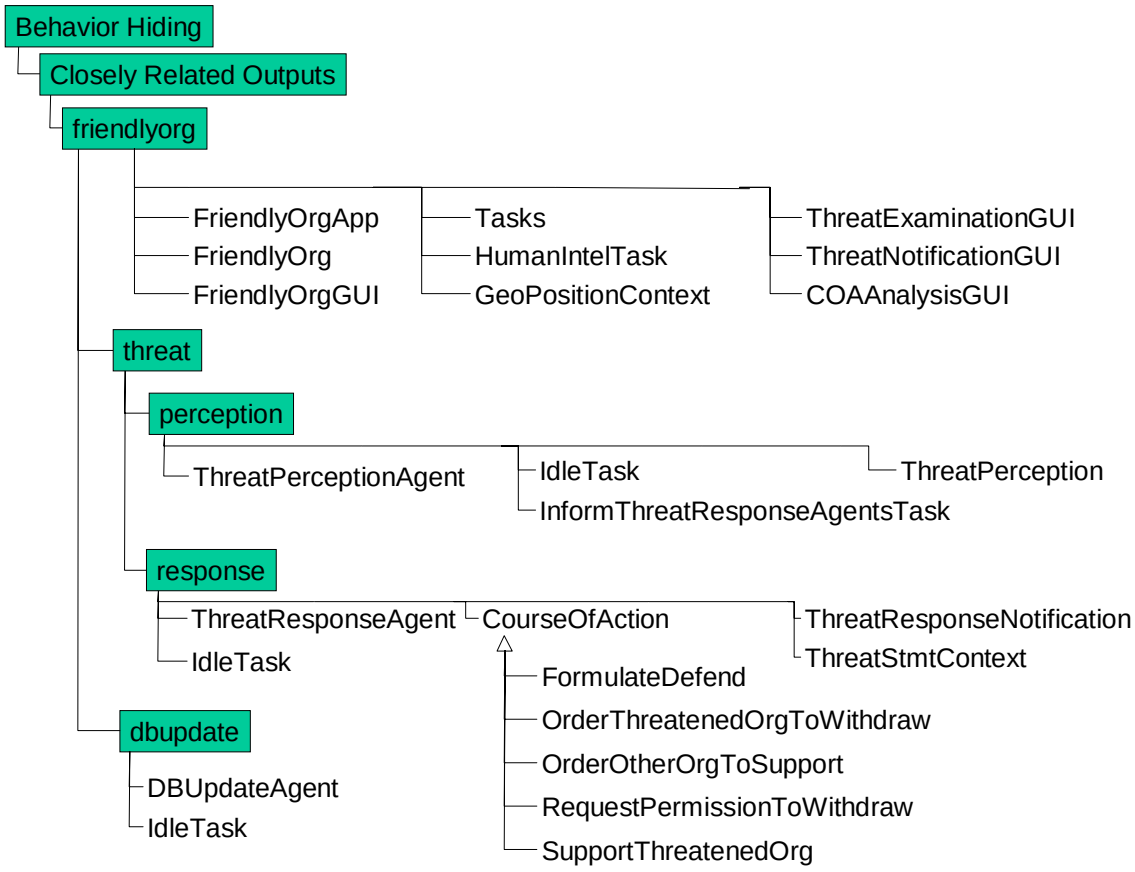

Figure 1. Friendly Organization Component Package and Class Structure

- A class that implements a “true” FIPA-OS agent has a name that ends with Agent. (A true agent performs agent operations, as opposed to a class implemented as an agent to make use of functionality provided by FIPA-OS.)
- A class that encapsulates a graphical user interface has a name that ends with GUI. Note that some packages have several GUIs. Each class denotes a distinct class of window presented to a user.
- A class that implements a FIPA-OS Task has a name that ends with Task.

Note that the response package shows not only classes but class hierarchy relationships among those classes concerned with expressing a course of action that a friendly organization might adopt.

2. Documentation Conventions

The code for the prototype is documented according to the conventions for the javadoc tool. These conventions help to standardize the content and format of API documentation. See http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/ for more information.

Javadoc translates comments of the form /* ... */ into input to an HTML interpreter. These comments often contain HTML elements. The translated version, viewed in a browser, is generally easier to comprehend. The HTML versions are included on the CD that accompanies this document.

The prototype adds one non-standard javadoc tag: “@todo”. This tag indicates an aspect of functionality needed in a production system but beyond the scope of a prototype.

3. Classes in Package FriendlyOrg

3.1. Class FriendlyOrgApp

The FriendlyOrgApp class launches a Friendly Organization component. It consists of a single method, main(), which as per Java standards is a static method whose sole parameter is an array of strings. Fields of the class store static values used for initialization of defaults.

```
package ida.bh.cro.friendlyorg;
import java.io.InputStreamReader;
import java.io.BufferedReader;

/**
 * The <code>FriendlyOrgApp</code> class lets a <code>FriendlyOrg</code> be run as an application.  Usage is:<pre>
    java FriendlyOrgApp [-a N] [-E] [-h] [-s N] [-u N] org-name map-spec</pre>
 * The flags are as follows:
 * <dl compact>
 * <dt><code>-a N</code></dt>
 * <dd>Set the interval between searches for threat response agents to <i>N</i> seconds
 *   (default: {@link FriendlyOrg#DEFAULT_THREAT_RESPONSE_AGENT_SEARCH_INTERVAL})</dd>
 * <dt><code>-e F</code></dt>
 * <dt><code>-E</code></dt>
 * <dd>When a line containing <code>end</code> is received on <code>stdin</code>, exit. Intended for use with the loader.</dd>
 * <dt><code>-h</code></dt>
 * <dd>Hide the user interface, i.e., make the GUI invisible.</dd>
 * <dt><code>-r N</code></dt>
 * <dd>Set the interval between ontology refreshes from the DBMS to <i>N</i> seconds
 *   (default: {@link FriendlyOrg#DEFAULT_ONTOLOGY_REFRESH_INTERVAL}.
 * <dt><code>-s N</code></dt>
 * <dd>Set the interval between searches for threats to <i>N</i> seconds
 *   (default: {@link FriendlyOrg#DEFAULT_THREAT_SEARCH_INTERVAL}).</dd>
 * <dt><code>-u N</code></dt>
 * <dd>Set the interval between searches for db update agents to <i>N</i> seconds
 *   (default: {@link FriendlyOrg#DEFAULT_DB_UPDATE_AGENT_SEARCH_INTERVAL}).</dd>
 * </dl>
 * The two required arguments are:
 * <dl compact>
 * <dt><code>org-name</code></dt>
```

```
 * <dd>The name of the organisation this application is to model.  Must be the name of an organisation in the ontology.</dd>
 * <dt><code>map-spec</code></dt>
 * <dd>The URL of the map specification.</dd>
 * </dl>
 **/
public class FriendlyOrgApp {

    private final static String FLAGS[] = {"[-a threat-response-agent-search-interval]",
                                           "[-E]",
                                           "-h",
                                           "[-r ontology-refresh-interval]",
                                           "[-s threat-search-interval]",
                                           "[-u db-update-agent-search-interval]" };
    private final static String ARGS[] = { "org-name", "map-spec" };

    private static String _usage;
    private static String _pgmName;

    static {
        _pgmName = FriendlyOrgApp.class.getName();
        _usage = "Usage: java " + _pgmName;
        int i;
        for ( i = 0; i < FLAGS.length; i++ )
            _usage += " " + FLAGS[i];
        for ( i = 0; i < ARGS.length; i++ )
            _usage += " " + ARGS[i];
    }

    /**
     * Creates a friendly organization component.  The parameters are described above.
     * @param args Parameters of the component, specified as strings.
     **/
    public static void main(String args[ ]) {
        String orgName;
        String mapSpec;
        Integer threatSearchInterval         = new Integer(FriendlyOrg.DEFAULT_THREAT_SEARCH_INTERVAL);
        Integer responseAgentSearchInterval = new Integer(FriendlyOrg.DEFAULT_THREAT_RESPONSE_AGENT_SEARCH_INTERVAL);
        Integer dbUpdateAgentSearchInterval= new Integer(FriendlyOrg.DEFAULT_DB_UPDATE_AGENT_SEARCH_INTERVAL);
        Integer ontologyRefreshInterval      = new Integer(FriendlyOrg.DEFAULT_ONTOLOGY_REFRESH_INTERVAL);
        boolean exitOnEndInput = false;
        boolean visible = true;
```

```
int i = 0;
while ( i < args.length && args[i].substring(0, 1).equals("-") ) {
    if ( args[i].equals("-help") ) {
        System.out.println(_usage);
    System.exit(0);
    }
    if ( i == args.length - 1 && ! (args[i].equals("-E") || args[i].equals("-h")) ) {
        System.err.println(_pgmName + ": Missing argument to \"" + args[i] + "\" flag.");
        System.err.println(_usage);
        System.exit(1);
    }
    if ( args[i].equals("-a") ) {
        try {
            responseAgentSearchInterval = Integer.valueOf(args[i+1]);
        } catch ( NumberFormatException e ) {
            System.err.println(_pgmName + ": Invalid response agent search interval \"" + args[i+1] + "\".");
            System.err.println(_usage);
            System.exit(1);
        }
        i += 2;
    }
    else if ( args[i].equals("-h") ) {
        visible = false;
        i++;
    }
    else if ( args[i].equals("-E") ) {
        exitOnEndInput = true;
        i++;
    }
    else if ( args[i].equals("-r") ) {
        try {
            ontologyRefreshInterval = Integer.valueOf(args[i+1]);
        } catch ( NumberFormatException e ) {
            System.err.println(_pgmName + ": Invalid ontology refresh interval \"" + args[i+1] + "\".");
            System.err.println(_usage);
            System.exit(1);
        }
        i += 2;
    }
```

```
    else if ( args[i].equals("-s") ) {
        try {
            threatSearchInterval = Integer.valueOf(args[i+1]);
        } catch ( NumberFormatException e ) {
            System.err.println(_pgmName + ": Invalid threat search interval \"" + args[i+1] + "\".");
            System.err.println(_usage);
            System.exit(1);
        }
        i += 2;
    }
    else if ( args[i].equals("-u") ) {
        try {
            dbUpdateAgentSearchInterval = Integer.valueOf(args[i+1]);
        } catch ( NumberFormatException e ) {
            System.err.println(_pgmName + ": Invalid db update agent search interval \"" + args[i+1] + "\".");
            System.err.println(_usage);
            System.exit(1);
        }
        i += 2;
    }
    else {
        System.err.println(_pgmName + ": Unknown flag \"" + args[i] + "\".");
        System.err.println(_usage);
        System.exit(1);
    }
}

if ( i != args.length - 2 ) {
    System.err.println(_pgmName + ": Missing/extra arguments.");
    System.err.println(_usage);
    System.exit(1);
}

orgName = args[i++];
mapSpec = args[i];

FriendlyOrg friendlyOrg;
try {
    friendlyOrg = new FriendlyOrg( orgName,
                                   new java.net.URL(mapSpec),
```

```
                                        threatSearchInterval,
                                        responseAgentSearchInterval,
                                        dbUpdateAgentSearchInterval,
                                        ontologyRefreshInterval,
                                        new Boolean(visible));
        } catch ( java.net.MalformedURLException e ) {
            System.err.println(_pgmName + ": Malformed URL \"" + mapSpec + "\": " + e.getMessage());
            System.exit(1);
        } catch ( FriendlyOrg.UnknownOrgNameException e ) {
            System.err.println(_pgmName + ": \"" + orgName + "\": Unknown organisation.");
            System.exit(1);
            return;
        } catch ( FriendlyOrg.InvalidMapSpecificationException e ) {
            System.err.println(_pgmName + ": \"" + mapSpec + "\": Invalid map specification: " + e.getMessage());
            System.exit(1);
            return;
        }
    }
}
```

3.2. Class FriendlyOrg

The FriendlyOrg class is the decision support application for a Friendly Organization component. Its salient method is its constructor, which the FriendlyOrgApp class invokes to instantiate the class and thereby create the run-time infrastructure for a friendly organization. Other externally visible methods are callback routines invoked from classes to which an instance of FriendlyOrg passes itself. The most important of these methods is update(), invoked as part of the model-view-controller paradigm. The update() method responds to events from Friendly Organization subcomponents. These components are generally implemented as agents. An important exception is the GUI (see Section 3.3), which sends a “shut-down” signal to FriendlyOrg when the user wants to terminate the application.

```
package ida.bh.cro.friendlyorg;

import java.net.URL;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Observable;
import java.util.Set;
```

```
import ida.sd.*;
import ida.sd.adt.OrganisationObserver;
import ida.sd.FIPA.C4IAgent;
import ida.sd.gh5ontology.*;
import ida.sd.gpc.GeoPosition;
import ida.sd.map.mapspec.MapSpec;
import ida.sd.map.mapspec.MapSpecUnmarshaller;
import ida.sd.ontology.*;
import ida.sd.ontology.impl.protege.ProtegeProject;
import ida.sd.osb.*;
import ida.sd.pa.StringPA;

import ida.bh.cro.friendlyorg.dbupdate.DBUpdateAgent;
import ida.bh.cro.friendlyorg.threat.perception.ThreatPerceptionAgent;
import ida.bh.cro.friendlyorg.threat.perception.ThreatPerception;
import ida.bh.cro.friendlyorg.threat.response.*;

/**
 * The <code>FriendlyOrg</code> class implements an instance of a friendly organisation.
 * Each friendly organisation has a map that it displays.  This map shows the organisation's proximate area, its location, and other battlefield
 * objects in the area.  Knowledge of these objects is dynamic, based on communications and intelligence.
 * <p>Each friendly organisation is in contact with other friendly organisations via a communication network.
 * It senses intelligence.  When it receives intelligence, it broadcasts that intelligence to other friendly organisations, and updates its map.
 **/
public class FriendlyOrg
    implements OrganisationObserver {

    /** Default time, in seconds, between searches of the ontology for threats. */
    public final static int DEFAULT_THREAT_SEARCH_INTERVAL                = 50;
    /** Default time, in seconds, between searches for threat response agents. */
    public final static int DEFAULT_THREAT_RESPONSE_AGENT_SEARCH_INTERVAL = 50;
    /** Default time, in seconds, between searches for DBUpdateAgents. */
    public final static int DEFAULT_DB_UPDATE_AGENT_SEARCH_INTERVAL       = 50;
    /** Default time, in seconds, between refreshes of the ontology from the DB. */
    public final static int DEFAULT_ONTOLOGY_REFRESH_INTERVAL             = 60;

    /** If <code>FriendlyOrg</code> receives this message, it terminates. */
    public final static String SHUTDOWN_MESSAGE = "shutdown";
    /** If <code>FriendlyOrg</code> receives this message, it asks all overview agents to flash its location. */
    public final static String FLASH_MESSAGE                              = "flash";
```

```
/** The user interface for this organisation. */
private FriendlyOrgGUI _gui;

/** The friendly organisation {@link Instance} controlling this <code>FriendlyOrg</code>. */
private Instance _org;
/** The name of the organisation controlling this <code>FriendlyOrg</code>. */
private final String _orgName;

/** A list of the agents launched by this application.  Needed to shut them down. */
private List _agents = new java.util.LinkedList();

/** The DB update agent, which is responsible for (what else?) propagating DB updates. */
private DBUpdateAgent _dbUpdateAgent;

/** The Tasks agent, which performs mundane non-ontology related tasks. */
private Tasks _tasksAgent;

/** True if this application was invoked from the command line, false if not. */
private final boolean _isInteractive;
/** True if this instance has a visible GUI, false if not. */
private final boolean _visible;
/** Names of organisations perceived to be a threat. */
private List _threats = new LinkedList();
/** The {@link ThreatPerceptionAgent} launched by this application. */
private ThreatPerceptionAgent _threatPerceptionAgent;

/** The GH5 {@link KnowledgeBase} for this application. */
private final GH5KB _gh5KB;
/** The {@link OntologySQLBinding} for this application. */
private final OntologySQLBinding _osb;
/** The class for an AbsolutePoint.  An ObjectItem can only be loaded onto the map if its
 * Location is an AbsolutePoint instance.  (Eventually we'll handle other Location subclasses.) */
private final Cls _absolutePointCls;

/**
 * Constructs a <code>FriendlyOrg</code> instance.  It is responsible for:
 * <ul>
 * <li>Creating an instance of the GH5 ontology.</li>
 * <li>Populating the ontology from a DBMS, if so specified.</li>
 * <li>Caching battlefield objects.</li>
 * <li>Starting agents:
 *     <ul>
 *      <li>A {@link ThreatPerceptionAgent}.</li>
```

```
 *     <li>A {@link ThreatResponseAgent}.</li>
 *     <li>A {@link DBUpdateAgent}, if a DBMS is being used.</li>
 *     <li>A {@link Tasks} agent, to handle incoming messages.</li>
 *     </ul>
 * </li>
 * <li>Launching the GUI.</li>
 * </ul>
 * @param orgName The name of the organisation controlling this <code>FriendlyOrg</code>.
 * @param mapSpecURL The URL of the map specification.
 * @param threatSearchInterval The interval, in seconds, between times when
 *        threats are to be perceived.  If null, defaults to
 *        {@link #DEFAULT_THREAT_SEARCH_INTERVAL}.
 * @param threatResponseAgentSearchInterval The interval, in seconds, between times
 *        when searches for threat response agents are to occur.  If null, defaults to
 *        {@link #DEFAULT_THREAT_RESPONSE_AGENT_SEARCH_INTERVAL}.
 * @param dbUpdateAgentSearchInterval The interval, in seconds, between times
 *        when searches for {@link DBUpdateAgent}s are to occur.  If null, defaults to
 *        {@link #DEFAULT_DB_UPDATE_AGENT_SEARCH_INTERVAL}.
 * @param ontologyRefreshInterval The interval, in seconds, between times when
 *        the ontology is to be refreshed from the DB.  If null, defaults
 *        to {@link #DEFAULT_ONTOLOGY_REFRESH_INTERVAL}.
 * @param visible If true, show a GUI; if false, keep the GUI invisible.  A value of false
 *        also prevents display of threat responses.  If null, defaults to true.
 */
public FriendlyOrg(String orgName,
                   URL mapSpecURL,
                   Integer threatSearchInterval,
                   Integer threatResponseAgentSearchInterval,
                   Integer dbUpdateAgentSearchInterval,
                   Integer ontologyRefreshInterval,
                   Boolean visible) throws InvalidMapSpecificationException, UnknownOrgNameException {
    _orgName = orgName;
    _visible = (visible == null ? true : visible.booleanValue());

    _isInteractive = System.getProperty(ida.eh.Properties.interactiveApp, "0").equals("1");

    // Open the ontology.
    try {
        _gh5KB = GH5KB.openFromProperties();
    } catch ( OntologyLoadException e ) {
```

```
    System.err.println(e.getMessage());
    if ( _isInteractive )
        System.exit(1);
    return;
}

// Load data into the knowledge base, if so specified.
String dataSource = System.getProperty(ida.eh.Properties.dataSource);
if ( dataSource != null ) {
    if ( dataSource.equals("db") ) {
        try {
            _osb = BindingPA.getBindingUsingSystemProperties(_gh5KB);
            _osb.populateKBfromDB();
        } catch ( BindingException e ) {
            System.err.println("Binding exception: " + e.getMessage());
            return;
        }
    }
    else {
        System.err.println("Unknown external data source \"" + dataSource + "\"");
        return;
    }
}
else {
    _osb = null;
}

_org = _gh5KB.getObjectItem(_orgName);
if ( _org == null )
    throw new UnknownOrgNameException(_orgName);

Cls orgCls = _gh5KB.getCls("Organisation");
Cls orgType = _org.getDirectType();
if ( ! (orgType.equals(orgCls) || orgType.hasSuperclass(orgCls)) )
    throw new UnknownOrgNameException(_orgName + " (a " + orgType.getName() + ", not an Organisation)");

// Note that we create a GUI even if it won't be visible.  We need the GUI's map to test for visible threats.
_gui = new FriendlyOrgGUI(this, orgName, mapSpec(mapSpecURL));

// Launch the agents.
if ( threatSearchInterval == null )
```

```
    threatSearchInterval = new Integer(DEFAULT_THREAT_SEARCH_INTERVAL);
if ( threatResponseAgentSearchInterval == null )
    threatResponseAgentSearchInterval = new Integer(DEFAULT_THREAT_RESPONSE_AGENT_SEARCH_INTERVAL);
_agents.add(_threatPerceptionAgent = new ThreatPerceptionAgent(_gh5KB,
                                                               "tp_" + _orgName,
                                                               _orgName,
                                                               this,
                                                               threatSearchInterval,
                                                               threatResponseAgentSearchInterval));
_agents.add(new ThreatResponseAgent(_gh5KB, "tr_" + _orgName, this));

if ( _osb != null ) {
    if ( dbUpdateAgentSearchInterval == null )
        dbUpdateAgentSearchInterval = new Integer(DEFAULT_DB_UPDATE_AGENT_SEARCH_INTERVAL);
        _dbUpdateAgent = new DBUpdateAgent("dbu_" + _orgName, dbUpdateAgentSearchInterval, _osb);
        _osb.setDBUpdateAgent(_dbUpdateAgent);
        _agents.add(_dbUpdateAgent);

    if ( ontologyRefreshInterval == null )
        ontologyRefreshInterval = new Integer(DEFAULT_ONTOLOGY_REFRESH_INTERVAL);
}
else {
    ontologyRefreshInterval = null;
}

_agents.add(_tasksAgent = new Tasks(_gh5KB, "tasks_" + _orgName, ontologyRefreshInterval, _osb, this));

_absolutePointCls = _gh5KB.getCls("AbsolutePoint");
loadObjectItemsOntoMap();
try {
    _gui.raiseBattlefieldObject(_orgName);
} catch ( FriendlyOrgGUI.UnknownObjectNameException e ) {
    Diagnostics.domainEvent("The map doesn't show the organisation of interest!", this);
}
displayMostRecentTask();

// Make the GUI visible depending on the value of the interactiveApp
// property.  This is necessary if the agent isn't loaded from the AgentLoader.
if ( _isInteractive ) {
    if ( _visible )
    _gui.setVisible(true);
```

```
    }
}
/**
 * Iterates through the list of known object items, placing each one on the map.
 **/
private void loadObjectItemsOntoMap() {
    Iterator objItmIter = _gh5KB.getCls("ObjectItem").getInstances().iterator();

    Cls facilityCls        = _gh5KB.getCls("Facility");
    Cls featureCls         = _gh5KB.getCls("Feature");
    Cls materielCls        = _gh5KB.getCls("Materiel");
    Cls organisationCls = _gh5KB.getCls("Organisation");
    Cls personCls          = _gh5KB.getCls("Person");
    Cls unitCls        = _gh5KB.getCls("Unit");

    while ( objItmIter.hasNext() ) {
        Instance oi = (Instance)objItmIter.next();
        if ( oi.instanceOf(unitCls) )
            loadObjectItemOntoMap(oi, "U");
        else if ( oi.instanceOf(organisationCls) )
            loadObjectItemOntoMap(oi, "O");
        else if ( oi.instanceOf(facilityCls) )
            loadObjectItemOntoMap(oi, "F");
        else if ( oi.instanceOf(featureCls) )
            loadObjectItemOntoMap(oi, "f");
        else if ( oi.instanceOf(materielCls) )
            loadObjectItemOntoMap(oi, "M");
        else if ( oi.instanceOf(personCls) )
            loadObjectItemOntoMap(oi, "P");
        else
            throw new Error("\"" + oi.getDirectType().getName() + "\" not modeled as a subtype of ObjectItem.");
    }
}
/**
 * Places a specified object item on the map.
 * @param oi The battlefield object {@link Instance} to place on the map.
 * @param symbol The display symbol for the object item.
 **/
```

```
private void loadObjectItemOntoMap(Instance oi, String symbol) {
    Instance location = _gh5KB.getCurrentLocation(oi);
    if ( location == null || ! location.getDirectType().equals(_absolutePointCls) )
        return;
    Double lat = _gh5KB.getFloatSlot(location, "latitudeCoordinate");
    Double lon = _gh5KB.getFloatSlot(location, "longitudeCoordinate");
    if ( ! _gui.onMap(lat, lon) )
        return;
    String oiName = _gh5KB.getObjectItemName(oi);
    int affiliation = getAffiliation(oiName);
    _gui.addBattlefieldObject(lat, lon, symbol, oiName, affiliation);
}

/**
 * Displays the most recent <code>ActionTask</code> {@link Instance} associated with the organisation.
 * If the organisation has no associated tasks, clears the display area.
 **/
private void displayMostRecentTask() {
    Instance t = _gh5KB.currentTask(_org);
    if ( t == null ) {
        _gui.setTask("(none)");
        return;
    }
    String taskName = _gh5KB.getStringSlot(t, "name");
    _gui.setTask(taskName == null ? "(no text)" : taskName.trim());
}

/**
 * Returns a constant from {@link FriendlyOrgGUI} that indicates a battlefield object's affiliation.
 * @param objectItemName The name of the battlefield object whose affiliation is to be determined.
 * @return A constant from {@link FriendlyOrgGUI} that indicates a battlefield object's affiliation: {@link FriendlyOrgGUI#FRIENDLY},
 *       {@link FriendlyOrgGUI#HOSTILE}, or {@link FriendlyOrgGUI#SELF}.
 **/
private int getAffiliation(String objectItemName) {
    if ( objectItemName.equals(_orgName) )
        return FriendlyOrgGUI.ObjectItemAffiliation.SELF;
    Instance oi = _gh5KB.getObjectItem(objectItemName);
    return _gh5KB.isHostile(oi) ? FriendlyOrgGUI.ObjectItemAffiliation.HOSTILE : FriendlyOrgGUI.ObjectItemAffiliation.FRIENDLY;
}
```

```
/**
 * Unmarshals a map specification and returns the Java objects for that specification.
 * @param specURL The URL of an XML document containing a {@link MapSpec}.
 * @return A {@link MapSpec} instance unmarshalled from <code>specURL</code>.
 * @throws InvalidMapSpecificationException If <code>specURL</code> does not exist, can't be opened, or contains content that
 *         can't be unmarshalled into a <code>MapSpec</code>.
 **/
private MapSpec mapSpec(URL specURL) throws InvalidMapSpecificationException {
    try {
        MapSpec spec = MapSpecUnmarshaller.unmarshal(specURL.openStream(), true);
        return spec;
    } catch ( java.io.IOException e ) {
        throw new InvalidMapSpecificationException("Cannot unmarshal map specification " + specURL.toString(), e);
    }
}

/**
 * Implements the {@link OrganisationObserver#getOrganisation()} method of the {@link OrganisationObserver} interface.
 * @return The {@linkplain Instance organisation instance} that controls this <code>FriendlyOrg</code>.
 **/
public Instance getOrganisation() {
    return _org;
}

/**
 * Returns the {@link OntologySQLBinding} associated with
 * this instance.
 * @return The {@link OntologySQLBinding} associated with
 *         this instance.
 **/
OntologySQLBinding getOSB() {
    return _osb;
}

/**
 * Returns the latitude of the northwest corner of this organisation's map.
 * @return The latitude of the northwest corner of this organisation's map.
 **/
public double getNWLatitude() {
    return _gui.getNWLatitude();
```

```
}
/**
 * Returns the longitude of the northwest corner of this organisation's map.
 * @return The longitude of the northwest corner of this organisation's map.
 **/
public double getNWLongitude() {
    return _gui.getNWLongitude();
}
/**
 * Returns the latitude of the center of this organisation's map.
 * @return The latitude of the center of this organisation's map.
 **/
public double getCenterLatitude() {
    return _gui.getCenterLatitude();
}
/**
 * Returns the longitude of the center of this organisation's map.
 * @return The longitude of the center of this organisation's map.
 **/
public double getCenterLongitude() {
    return _gui.getCenterLongitude();
}
/**
 * Returns the knowledge base associated with this instance.
 * @return The knowledge base associated with this instance.
 **/
public GH5KB getKB() {
    return _gh5KB;
}
/**
 * Determines whether a battlefield object is visible to this <code>FriendlyOrg</code>.
 * @param objectItemName The name of a battlefield object.
 * @return True if the name refers to an {@link Instance} of an <code>ObjectItem</code> that is visible to this
 *         <code>FriendlyOrg</code> (which currently means on the map, there being no features), false if not.
 **/
public boolean isVisible(String objectItemName) {
```

```
    Instance oi = _gh5KB.getObjectItem(objectItemName);
    if ( oi == null )
        return false;

    Instance location = _gh5KB.getCurrentLocation(oi);
    if ( location == null )
        return false;

    Double lat = _gh5KB.getFloatSlot(location, "latitudeCoordinate");
    Double lon = _gh5KB.getFloatSlot(location, "longitudeCoordinate");
    return _gui.onMap(lat, lon);
}

/** The set of threat response notifications currently visible. */
private Set _TRNsDisplayed = new HashSet();

/**
 * Responds to requests from <code>Observable</code>s.  Currently these requests fall into the following categories:
 * <ul>
 * <li>Threat perceptions from the {@link ThreatPerceptionAgent}.</li>
 * <li>Threat response notifications from the {@link ThreatResponseAgent}.</li>
 * <li>Results of searches for other agents of various types.</li>
 * <li>User interface requests.</li>
 * <li>Requests to shut down the application (which may or may not come from the user interface).</li>
 * </ul>
 **/
public void update(Observable o, Object arg) {
    if ( arg instanceof ThreatResponseNotification ) {
        ThreatResponseNotification trn = (ThreatResponseNotification)arg;
        // If user is already viewing this threat response notification, ignore it.
        synchronized ( _TRNsDisplayed ) {
            if ( _TRNsDisplayed.contains(trn) )
                return;
            _TRNsDisplayed.add(trn);
        }
        if ( _visible )
            new ThreatNotificationGUI(_gui, this, _gh5KB, trn);
    }
    else if ( arg instanceof ThreatPerception ) {
        respondToThreatPerception((ThreatPerception)arg);
    }
```

```
    else if ( arg instanceof Tasks.OverviewAgentsSearch ) {
        _gui.setFlashCapability(((Tasks.OverviewAgentsSearch)arg).agents.size() > 0);
    }
    else if ( arg instanceof ThreatNotificationGUI.Completed ) {
        respondToThreatNotificationGUICompletion((ThreatNotificationGUI.Completed)arg);
    }
    else if ( arg instanceof String && ((String)arg).equals(FLASH_MESSAGE) ) {
        _tasksAgent.sendFlashRequestToOverviewAgents();
    }
    else if ( arg instanceof String && ((String)arg).equals(SHUTDOWN_MESSAGE) ) {
        for ( Iterator aIter = _agents.iterator(); aIter.hasNext(); )
        ((C4IAgent)aIter.next()).stop();

       if ( _isInteractive )
            System.exit(0);
    }
    else if ( arg instanceof String ) {
        _gui.addKBUpdate((String)arg);
    }
    else if ( arg instanceof GeoPosition ) {
        GeoPosition gp = (GeoPosition)arg;
        Double lat = new Double(gp.getLatitude().doubleValue());
        Double lon = new Double(gp.getLongitude().doubleValue());
        String objName = gp.getObjName();
        try {
            _gui.moveBattlefieldObject(objName, lat, lon);
        } catch ( FriendlyOrgGUI.UnknownObjectNameException e ) {
            // Object wasn't on map.  Add it if it should be.
            if ( _gui.onMap(lat, lon) ) {
                // Should determine the symbol.
                _gui.addBattlefieldObject(lat, lon, "U", objName, getAffiliation(objName));
            }
        }
    }
    else {
        throw new Error("Unknown arg to update: " + arg.getClass().getName());
    }
}
private void setCourseOfAction(CourseOfAction coa) {
```

```
    Diagnostics.domainEvent("Accepted COA=" + coa.description(), this);
    Cls actionTaskCls = _gh5KB.getCls("ActionTask");
    List c4IEntities = coa.asC4IModel();
    for ( Iterator eIter = c4IEntities.iterator(); eIter.hasNext(); ) {
        Instance c4Ientity = (Instance)eIter.next();
        if ( c4Ientity.instanceOf(actionTaskCls) ) {
            String vpc = _gh5KB.getCodeSlot(c4Ientity, "activityCode");
            String cc  = _gh5KB.getCodeSlot(c4Ientity, "categoryCode");
            _gui.setTask(cc + ": " + vpc);
            return;
        }
    }
    Diagnostics.error("Can't display COA: No ActionTask", this);
}
private void respondToThreatPerception(ThreatPerception tp) {
    String  orgName  = _gh5KB.getObjectItemName(tp.getOrganisation());
    boolean isThreat = tp.getIsThreatened();
    synchronized( _threats ) {
        if ( isThreat ) {
            if ( ! _threats.contains(orgName) )
                _threats.add(orgName);
        }
        else {
            if ( ! _threats.contains(orgName) ) {
                Diagnostics.error("Received threat cancellation for \"" + orgName + "\", but it isn't recorded as a threat.", this);
                return;
            }
            _threats.remove(orgName);
        }
    }
    _gui.setThreatened(isThreat, orgName);
}
private void respondToThreatNotificationGUICompletion(ThreatNotificationGUI.Completed c) {
    CourseOfAction selectedCoA = c.getCourseOfAction();
    if ( selectedCoA != null ) {
        setCourseOfAction(selectedCoA);
        if ( _osb != null ) {
```

```
            try {
                List entities = selectedCoA.asC4IModel();
                _osb.saveInstances(entities);
                List eClses = new LinkedList();
                for ( Iterator eIter = entities.iterator(); eIter.hasNext(); )
                    eClses.add(((Instance)eIter.next()).getDirectType().getName());
                _gui.addKBUpdate("(" + StringPA.join(eClses, ", ") + ")");
            } catch ( BindingException e ) {
                Diagnostics.noteException(e, this);
            }
        }
    }
    _TRNsDisplayed.remove(c.getThreatResponseNotification());
}

/** Stores the list of holdings of the organisation. */
private List _holdingList = null;

/**
 * Returns the number of holding types possessed by the organisation.
 * @return The number of holding types possessed by the organisation.
 **/
int getHoldingTypesCount() {
    if ( _holdingList == null )
        _holdingList = new LinkedList(_org.getOwnSlotValues(_gh5KB.getSlot("has-Holding")));
    return _holdingList.size();
}

/**
 * Returns the holding quantity for the i'th row in the holding list.
 * @param row An index (zero-based) into the holding list.
 * @return An Integer representing the holding quantity for the i'th row in the holding list.
 **/
Object getHoldingCount(int row) {
    Instance holding = (Instance)_holdingList.get(row);
    return _gh5KB.getIntSlot(holding, "totalQuantity");
}

/**
 * Returns the number of threats this organisation perceives.
 * @return The number of threats this organisation perceives.
```

```
 **/
public int getThreatCount() {
    return _threats.size();
}

/**
 * Returns the name of the <i>i</i>'th threat to this organisation.
 * @param i An index (zero-based) into the threats.
 * @return The name of the <i>i</i>'th threat to this organisation.
 **/
public String getThreat(int i) {
    return (String)_threats.get(i);
}

/**
 * Returns the holding type for the i'th row in the holding list.
 * <p>This implementation isn't great; the parameter should be the
 * holding type.  But I'm still learning about Swing tables.
 * @param row An index (zero-based) into the holding list.
 * @return A String representing the holding type for the i'th row in the holding list.
 **/
Object getHoldingType(int row) {
    Instance holding = (Instance)_holdingList.get(row);
    Instance objType = (Instance)holding.getOwnSlotValue(_gh5KB.getSlot("holding-is-constrained-to-ObjectType"));
    return _gh5KB.getStringSlot(objType, "name");
}

/**
 * Signals that a method has been given an organisation name that is not
 * recognized.
 **/
public class UnknownOrgNameException extends Exception {
    /**
     * Constructs an <code>UnknownOrgNameException</code> instance with
     * a string that by convention is the name of the unknown organisation.
     * @param orgName The name of an unknown organisation.
     **/
    UnknownOrgNameException(String orgName) {
        super(orgName);
    }
```

```
    }
    /**
     * Signals that the map specification was invalid.
     **/
    public class InvalidMapSpecificationException extends Exception {
        public InvalidMapSpecificationException(String mapSpec, Throwable th) {
            super(mapSpec, th);
        }
    public InvalidMapSpecificationException(String mapSpec) {
            super(mapSpec);
        }
    }
}
```

3.3. Class FriendlyOrgGUI

The FriendlyOrgGUI class encapsulates the main window of the decision support application. It is built by extending the Java SWING class Jframe. Its contents maintain the various components of the user interface.

The FriendlyOrgGUI constructor receives as a parameter an instance of a FriendlyOrg. It maintains this instance through a Notifier, which is an abstraction to support one instance notifying another using Java's Observer/Observable implementation of the Model-View-Controller paradigm. (FriendlyOrgGUI cannot itself be Observable, because Observable is a class and FriendlyOrgGUI already extends class Jframe.)

Public methods of FriendlyOrgGUI allow control of the user interface: moving battlefield objects, changing the C2 display area, etc. Note that FriendlyOrgGUI is not a public class, as its functionality is not (and must not be) needed outside the FriendlyOrg package.

```
package ida.bh.cro.friendlyorg;

import java.util.HashMap;

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.AbstractTableModel;

import java.awt.*;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import ida.sd.adt.Notifier;
import ida.sd.pa.GUI;
import ida.sd.map.*;
import ida.sd.map.mapspec.*;

/**
 * The <code>FriendlyOrgGUI</code> class provides the user interface for a friendly organisation.  Each instance of the class represents the GUI
 * for a particular organisation.  The GUI maintains:
 * <ul>
 * <li>A map that displays battlefield objects.</li>
 * <li>A table of the organisation's holdings.</li>
 * </ul>
 * The secrets encapsulated by the GUI are:
 * <ol>
 * <li>The appearance of the map.</li>
 * <li>The appearance of a battlefield object displayed on the map.</li>
 * </ol>
 **/
class FriendlyOrgGUI extends JFrame {
    private final static Integer BO_LAYER   = new Integer(1);
    private final static Font    UNIT_LABEL_FONT = new Font("Serif", Font.BOLD, 18);

    private final static Border ETCHED_BORDER = BorderFactory.createEtchedBorder();

    private final static Border FRIENDLY_OBJ_BORDER = BorderFactory.createRaisedBevelBorder();
    private final static Border HOSTILE_OBJ_BORDER  = BorderFactory.createLoweredBevelBorder();

    /** The pane that stores the map and the battlefield objects.  The map is stored in layer 0.  The objects are in layer 1. */
    private MapPane mapPane;
    /** The actual map. */
    private MapDisplay   map;
    /** A record of the objects on the map.  Each name is mapped to the Swing component that displays it. */
    private HashMap _objectsOnMap = new HashMap();

    /** Displays the most recent task for the organisation. */
    private JTextArea _mostRecentTask = new JTextArea(1, 30);
    /** Displays knowledge base updates the organisation has received. */
```

```
private JTextArea _kbUpdates  = new JTextArea(4 ,30);
/** Lets the user flash the organisation's location. */
private JButton   _flashLoc        = new JButton("Flash Location");

/** Displays the organisation's holdings. */
private JTable _holdings;
/** Displays threats to the organisation. */
private JTable _threats;

/** A wrapper around the observer to be notified when the frame is closed. */
private Notifier _notifier;
/** The friendly organisation that controls this GUI. */
private FriendlyOrg _associatedFriendlyOrg;

/**
 * Constructs a <code>FriendlyOrgGUI</code> instance.
 * @param organisation The friendly organisation that controls this GUI.
 * @param organisationName A human-readable string describing the organisation.
 * @param mapSpec The specification of the map to be used.
 **/
public FriendlyOrgGUI(FriendlyOrg organisation, String organisationName, MapSpec mapSpec) {
    super(organisationName + " Monitor");
    _notifier = new Notifier((java.util.Observer)organisation);
    _associatedFriendlyOrg = organisation;

    Container contentPane = getContentPane();
    contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
    contentPane.add(GUI.VERTICAL_GAP);

    // Create and set up the map pane.
    ImageMap m;
    try {
        m = new ImageMap(mapSpec);
    } catch ( java.net.MalformedURLException e ) {
        throw new java.lang.IllegalArgumentException("Malformed URL \"" + mapSpec.getMapURL() + "\": " + e.getMessage());
    }
    map = new MapDisplay(m);
    mapPane = new MapPane(map);
    mapPane.setBorder(BorderFactory.createEmptyBorder());
    mapPane.setAlignmentY(0.0f);

    // Add map pane, and controls, to frame.
```

```
Box mapBox = new Box(BoxLayout.X_AXIS);
contentPane.add(mapBox);

mapBox.add(mapPane);
mapBox.add(Box.createRigidArea(new Dimension(10, 0)));

Box controlsBox = new Box(BoxLayout.Y_AXIS);
controlsBox.setAlignmentY(0.0f);
mapBox.add(controlsBox);
_flashLoc.setEnabled(false);
_flashLoc.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            _notifier.signal(FriendlyOrg.FLASH_MESSAGE);
        }
    });
_flashLoc.setAlignmentX(0.0f);
controlsBox.add(_flashLoc);
controlsBox.add(Box.createVerticalGlue());
ThreatDisplay td = new ThreatDisplay(this, organisationName);
td.setAlignmentX(0.0f);
controlsBox.add(td);

// Make a new container for the portion of the GUI below the map.
Box bottomDisplay = new Box(BoxLayout.X_AXIS);
contentPane.add(Box.createRigidArea(new Dimension(0, 5)));
contentPane.add(bottomDisplay);

bottomDisplay.add(new HoldingsTable());

// Now add the C4I information in a vertical box with glue that shoves
// the information to the box's top.
Box c4iBox = new Box(BoxLayout.Y_AXIS);
c4iBox.add(new C4IDisplay());
c4iBox.add(Box.createVerticalGlue());

bottomDisplay.add(c4iBox);

// Add listener for window-closing event.
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        _notifier.signal(FriendlyOrg.SHUTDOWN_MESSAGE);
    }
});
```

```
    pack();
}
/**
 * Returns true iff a location is on the map.
 * @param latitude The latitude coordinate, in decimal degrees.
 * @param longitude The latitude coordinate, in decimal degrees.
 **/
public boolean onMap(Double latitude, Double longitude) {
    return map.onMap(latitude, longitude);
}
/**
 * Puts a named object on the map at a specified location.
 * If the location is off the map, ignores the object.
 * @param lat The latitude of the object.
 * @param lon The longitude of the object.
 * @param objType The type of the object.  Currently this is a single-character string, and it's used as the object's label.
 * @param objName The name of the object.  The name is used to refer to the object in subsequent calls to
*                <code>FriendlyOrgGUI</code> methods.
 * @param affiliation A value from class <code>ObjectItemAffliation</code> indicating the object's relationship to the organization
*                controlling the map.
 **/
public void addBattlefieldObject(Double lat, Double lon, String objType, String objName, int affiliation) {
    if ( ! map.onMap(lat, lon) )
        return;
    JLabel battlefieldObject = new JLabel(objType, SwingConstants.LEFT);
    battlefieldObject.setBorder(BorderFactory.createEmptyBorder());
    battlefieldObject.setVerticalAlignment(JLabel.TOP);
    Color backgroundColor;
    switch ( affiliation ) {
    case ObjectItemAffiliation.SELF:
        backgroundColor = Color.BLUE;
        battlefieldObject.setForeground(Color.CYAN);
        battlefieldObject.setBorder(FRIENDLY_OBJ_BORDER);
        break;
    case ObjectItemAffiliation.FRIENDLY:
        backgroundColor = Color.BLUE;
        battlefieldObject.setForeground(Color.WHITE);
        battlefieldObject.setBorder(FRIENDLY_OBJ_BORDER);
```

```
            break;
        case ObjectItemAffiliation.HOSTILE:
            backgroundColor = Color.RED;
            battlefieldObject.setForeground(Color.WHITE);
            battlefieldObject.setBorder(HOSTILE_OBJ_BORDER);
            break;
        default:
            throw new Error("Invalid object item affiliation");
        }
        // The following adjustments to the background color help beveled borders stand out.
        float hsv[] = new float[3];
        Color.RGBtoHSB(backgroundColor.getRed(), backgroundColor.getGreen(), backgroundColor.getBlue(), hsv);
        if ( hsv[1] > 0.5f )
            hsv[1] = 0.5f;
        if ( hsv[2] > 0.7f )
            hsv[2] = 0.7f;
        battlefieldObject.setBackground(new Color(Color.HSBtoRGB(hsv[0], hsv[1], hsv[2])));
        battlefieldObject.setFont(UNIT_LABEL_FONT);
        battlefieldObject.setOpaque(true);
        Point p = map.getPoint(lat, lon);
        FontMetrics m = getFontMetrics(UNIT_LABEL_FONT);
        int w = m.stringWidth(objType);
        int h = m.getAscent() + m.getDescent();
        Insets bfi = battlefieldObject.getInsets();
        battlefieldObject.setBounds(p.x,
                                    p.y,
                                    w + bfi.left + bfi.right,
                                    h + bfi.top + bfi.bottom);
        battlefieldObject.setToolTipText(objName);
        mapPane.add(battlefieldObject, BO_LAYER);
        _objectsOnMap.put(objName, battlefieldObject);
    }
    /**
     * Move a named object to a new location.  If the new location is off the map, remove the object.
     * @param objName The name of the object to move.
     * @param lat The new latitude of the object.
     * @param lon The new longitude of the object.
```

```
 * @throws <code>UnknownObjectNameException</code> if the name doesn't refer to an object on the map.
 **/
public void moveBattlefieldObject(String objName, Double lat, Double lon)
    throws UnknownObjectNameException {
    JLabel obj = (JLabel)_objectsOnMap.get(objName);
    if ( obj == null )
        throw new UnknownObjectNameException(objName);

    if ( ! map.onMap(lat, lon) ) {
        _objectsOnMap.remove(objName);
        mapPane.remove(mapPane.getIndexOf(obj));
        return;
    }
    obj.setLocation(map.getPoint(lat, lon));
}

/**
 * Removes a named object from the map.
 * @param objName The name of the object to move.
 * @throws UnknownObjectNameException If the name doesn't refer to an object on the map.
 **/
public void removeBattlefieldObject(String objName)
    throws UnknownObjectNameException {
    JLabel obj = (JLabel)_objectsOnMap.get(objName);
    if ( obj == null )
        throw new UnknownObjectNameException(objName);

    _objectsOnMap.remove(objName);
    mapPane.remove(mapPane.getIndexOf(obj));
}

/**
 * Raises the named object to the top of the display.  In other words, it becomes on top of any other objects that happen to occupy the same
 * location.
 * @param objName The name of the object to move.
 * @throws UnknownObjectNameException If the name doesn't refer to an object on the map.
 **/
public void raiseBattlefieldObject(String objName)
    throws UnknownObjectNameException {
    JLabel obj = (JLabel)_objectsOnMap.get(objName);
    if ( obj == null )
```

```
            throw new UnknownObjectNameException(objName);
        mapPane.setPosition(obj, 0);
    }

    /**
     * Sets the most-recent-task display area to the specified text.
     * @param task The string that describes the most recent task.
     **/
    public void setTask(String task) {
        _mostRecentTask.setText(task);
    }

    /**
     * Sets the knowledge base update display area to the specified text.
     * @param text The string that describes the update.
     **/
    public void setKBUpdate(String text) {
        _kbUpdates.setText(text);
    }

    private boolean _ousAdded = false;
    /**
     * Appends a string to the knowledge base update display area.
     * @param text The string to append.
     **/
    public void addKBUpdate(String text) {
        if ( _ousAdded )
            _kbUpdates.append("\n");
        else
            _ousAdded = true;
        _kbUpdates.append(text);
    }

    /**
     * Sets the threat state of the organisation.
     * @param threatened If true, <code>org</code> is a threat. If false, <code>org</code> is not a threat.
     * @param org The name of the organisation in question.
     * @todo Prioritize the threats.
     * @todo Do we need parameters?
     **/
    public void setThreatened(boolean threatened, String org) {
```

```
    // Update the threat table display.
    ((ThreatTableModel)_threats.getModel()).fireTableDataChanged();
}

/**
 * Returns the threat state of the organisation.
 * @return True if the organisation's GUI is showing that it's threatened, false if not.
 **/
public boolean getThreatened() {
    return _threats.getRowCount() > 0;
}

/**
 * Sets whether the user can use the flash capability.
 * @param enabled If true, the user can use the flash capability.
 *                If false, he cannot.
 **/
public void setFlashCapability(boolean enabled) {
    _flashLoc.setEnabled(enabled);
}

/**
 * Returns the latitude of the northwest corner of this map.
 **/
public double getNWLatitude() {
    return map.getNorthLat();
}

/**
 * Returns the longitude of the northwest corner of this map.
 **/
public double getNWLongitude() {
    return map.getWestLon();
}

/**
 * Returns the latitude of the center of this map.
 **/
public double getCenterLatitude() {
    return map.getCenterLat();
}
```

```
/**
 * Returns the longitude of the center of this map.
 **/
public double getCenterLongitude() {
    return map.getCenterLon();
}

/**
 * The <code>UnknownObjectNameException</code> class signals that a method that
 * takes the name of a battlefield object as an argument has been invoked with
 * a name that hasn't been added, or has been removed.
 **/
public class UnknownObjectNameException extends Exception {
    UnknownObjectNameException(String message) {
        super(message);
    }
}

/** The column headings for the holdings table. */
private static String[] _columns = { "Holding Type", "Total Quantity" };

/**
 * The <code>HoldingsTableModel</code> class implements a table model based
 * on the current holdings of the associated friendly organisation.
 **/
private class HoldingsTableModel extends AbstractTableModel {
    public int getRowCount() {
        return _associatedFriendlyOrg.getHoldingTypesCount();
    }
    public int getColumnCount() {
        return _columns.length;
    }
    public Object getValueAt(int row, int column) {
        if ( column == 0 )
            return _associatedFriendlyOrg.getHoldingType(row);
        else
            return _associatedFriendlyOrg.getHoldingCount(row);
    }

    public String getColumnName(int column) {
        return _columns[column];
```

```
    }
}
/**
 * The <code>ThreatTableModel</code> class implements a table model based
 * on the known threats to the associated friendly organisation.
 **/
private class ThreatTableModel extends AbstractTableModel {
    public int getRowCount() {
        return _associatedFriendlyOrg.getThreatCount();
    }
    public int getColumnCount() {
        return 1;
    }
    public Object getValueAt(int row, int column) {
        return _associatedFriendlyOrg.getThreat(row);
    }
}
/**
 * The <code>MapPane</code> class is used to hold the map and the battlefield objects it contains.  The class is an extension of the
 * <code>JLayeredPane</code> class.
 * <p>Having the map grow or shrink is undesirable. This class overrides the get*Size() methods such that the size -- minimum, maximum,
 *  or preferred -- is always the same.
 **/
private class MapPane extends JLayeredPane {
    Dimension _size;
    /**
     * Constructs a <code>MapPane</code> instance.  The size of the instance is taken from the size of a parameter label.
     * @param map A <code>mapDisplay</code> whose size is used as the fixed size of the <code>MapPane</code>.
     **/
    MapPane(MapDisplay map) {
        super();
        _size = map.getSize();
        add(map, new Integer(0));
    }
    public Dimension getMinimumSize() {
        return _size;
```

```
    }
    public Dimension getPreferredSize() {
        return _size;
    }
    public Dimension getMaximumSize() {
        return _size;
    }
}
/**
 * The <code>ObjectItemAffiliation</code> class provides the set of valid values for the <code>affiliation</code> parameter of
 *  <code>addBattlefieldObject()</code>.
 **/
public class ObjectItemAffiliation {
    public final static int SELF = 1;
    public final static int FRIENDLY = 2;
    public final static int HOSTILE = 3;
}
/**
 * The <code>HoldingsTable</code> class encapsulates the display of the holdings table.
 **/
private class HoldingsTable extends JScrollPane {
    /**
     * Creates a <code>HoldingsTable</code>.  Has the side effect
     * of setting {@link #_holdings} to a newly-created {@link JTable}.
     **/
    HoldingsTable() {
        super();
        _holdings = new JTable(new HoldingsTableModel());
        setViewportView(_holdings);
        _holdings.getColumnModel().getColumn(0).setMinWidth(180);
        _holdings.getColumnModel().getColumn(1).setMinWidth(100);
        setPreferredSize(new Dimension(300, 200));
        setBorder(BorderFactory.createTitledBorder(ETCHED_BORDER, "Organisation Holdings"));
    }
}
/**
 * The <code>C4IDisplay</code> class extends <code>JPanel</code> to provide a panel whose maximum height is always the same is the
```

```
 * preferred height, which in turn is always computed from the sum of the heights of its non-label components.
 **/
private class C4IDisplay extends JPanel {
    /**
     * Creates a <code>C4IDisplay</code> containing:
     * <ul>
     * <li>The most recent task.</li>
     * <li>Ontology updates received.</li>
     * </ul>
     **/
    C4IDisplay() {
        super(new GridBagLayout());
        setBorder(BorderFactory.createTitledBorder(ETCHED_BORDER, "C4I Display"));

        GUI.addLabeledComponent(this, 0, "Most Recent Task: ", _mostRecentTask);

        JScrollPane kbUpdatesScrollPane = new JScrollPane(_kbUpdates);
        kbUpdatesScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
        GUI.addLabeledComponent(this, 1, "KB Updates: ", kbUpdatesScrollPane);
    }

    /**
     * Gets the preferred size.  The height is computed as the sum of all non-label components, plus a small amount for
     * inter-component spacing.
     **/
    public Dimension getPreferredSize() {
        int height = 0;
        Component[] components = getComponents();
        for ( int i = 0; i < components.length; i++ ) {
            if ( ! (components[i] instanceof JLabel) )
                height += ((JComponent)components[i]).getPreferredSize().height;
        }
        Insets i = getInsets();
        height += i.top + i.bottom;
        return new Dimension(super.getPreferredSize().width, height + GUI.ROW_GAP*(components.length - 1));
    }

    /**
     * Gets the maximum size using the preferred size as height and the maximum size as width.
     **/
    public Dimension getMaximumSize() {
```

```
        return new Dimension(super.getMaximumSize().width, getPreferredSize().height);
    }
}

// The following section concerns the display of threats.
/** Maximum number of threats a scroll pane should show. */
private final static int THREATS_DISPLAYED = 5;
/** Maximum number of characters to show; used to determine scroll pane width. */
private final static int THREAT_NAME_MAX_LENGTH = 20;

private static final String NO_THREAT_LABEL      = "None";
private static final Color  NO_THREAT_COLOR      = Color.BLACK;
private static final String THREAT_PRESENT_LABEL = "Yes";
private static final Color  THREAT_PRESENT_COLOR = Color.RED;

/**
 * The <code>ThreatDisplay</code> class encapsulates the display of threats. The list of known threats is displayed in the
 * {@link #_threats} table. Other classes are responsible for setting and modifying the rows in this table.  This class is responsible for
 * organizing and displaying the table and supporting components.  The primary secret of this class is the format used to display
 * the components.
 **/
private class ThreatDisplay extends Box {

    /** Shows succinctly whether a threat exists. */
    private JLabel _threatPresent;
    /** Lets the user view a threat in detail. */
    private JButton _viewThreat = new JButton("View");

    /**
     * Creates a <code>ThreatDisplay</code>.  Has the following side effects:
     * <ul>
     * <li>Sets {@link #_threats} to a newly-created {@link JTable}.</li>
     * <li>Sets {@link #_threatPresent} to {@link #NO_THREAT_LABEL} and {@link #NO_THREAT_COLOR}.
     * </ul>
     * @param parent      The parent GUI.  Needed for control of dialogs.
     * @param exampleName A "typical" organisation name.  Used to determine
     *                    the width of the table.
     * @todo The View button should be enabled only when threats are
     *       present.
     **/
    ThreatDisplay(final FriendlyOrgGUI parent, String exampleName) {
```

```
super(BoxLayout.Y_AXIS);

setBorder(BorderFactory.createTitledBorder(ETCHED_BORDER, "Threats"));

_threatPresent = new JLabel(NO_THREAT_LABEL);
add(GUI.VERTICAL_GAP);
add(_threatPresent);

// Set up the threat table.
_threats = new JTable(new ThreatTableModel());
_threats.setTableHeader(null);
_threats.setRowSelectionAllowed(true);
_threats.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
if ( exampleName.length() > THREAT_NAME_MAX_LENGTH )
    exampleName = exampleName.substring(0, THREAT_NAME_MAX_LENGTH);
int w = getFontMetrics(_threats.getFont()).stringWidth(exampleName);
int h = _threats.getRowHeight()*THREATS_DISPLAYED;
_threats.setPreferredSize(new Dimension(w, h));

JScrollPane scrollPane = new JScrollPane(  _threats,
                                           JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                                           JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
add(GUI.VERTICAL_GAP);
add(scrollPane);

// Add a button to allow a detailed view of a threat.
_viewThreat.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int selectedRow = _threats.getSelectedRow();
            if ( selectedRow == -1 ) {
                JOptionPane.showMessageDialog(parent, "Select the threat.", "Missing Input", JOptionPane.ERROR_MESSAGE);
                return;
            }
            new ThreatExaminationGUI(_associatedFriendlyOrg, (String)_threats.getModel().getValueAt(selectedRow, 0));
        }
    });
add(GUI.VERTICAL_GAP);
add(_viewThreat);

Insets i;
i = scrollPane.getInsets();
Dimension scrollPaneSize = new Dimension(w + i.left + i.right, h + i.top + i.bottom);
```

```
        scrollPane.setPreferredSize(scrollPaneSize);

        i = getInsets();
        Dimension threatDisplaySize = new Dimension(scrollPaneSize.width + i.left + i.right, scrollPaneSize.height + i.top + i.bottom);
        setPreferredSize(threatDisplaySize);

        // Add a listener to handle enabling/disabling the View button and setting the _threatPresent label.
        _threats.getModel().addTableModelListener(new TableModelListener() {
                public void tableChanged(TableModelEvent e) {
                    if ( _threats.getRowCount() > 0 ) {
                        _threatPresent.setText(THREAT_PRESENT_LABEL);
                        _threatPresent.setForeground(THREAT_PRESENT_COLOR);
                        _viewThreat.setEnabled(_threats.getSelectedRow() != -1);
                    }
                    else {
                        _threatPresent.setText(NO_THREAT_LABEL);
                        _threatPresent.setForeground(NO_THREAT_COLOR);
                        _viewThreat.setEnabled(false);
                    }
                }
            });
        // Add a listener to enable/disable the View button based on whether a row of the _threats table is selected.
        _threats.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
                public void valueChanged(ListSelectionEvent e) {
                    _viewThreat.setEnabled(_threats.getSelectedRow() != -1);
                }
            });

        // Initialize values by signaling a change.
        ((ThreatTableModel)_threats.getModel()).fireTableDataChanged();
    }
  }
}
```

3.4. Class GeoPositionContext

The GeoPositionContext class supplies that portion of context functionality needed by a friendly organization. The SimState agent will supply a Friendly Organization with information detected by sensors. The information is supplied as an SL0 “result” atomic formula. Thus GeoPositionContext provides an implementation of evaluateResult() that extracts the information in the formula and uses it to update the Friendly Organization’s knowledge base. The update is performed using callback methods found in class FriendlyOrg.

```
package ida.bh.cro.friendlyorg;

import java.util.Iterator;
import java.util.Set;

import fipaos.ont.fipa.ACL;
import fipaos.ont.fipa.FIPACONSTANTS;

import ida.sd.Diagnostics;
import ida.sd.FIPA.sl0.SL0;
import ida.sd.gpc.*;
import ida.sd.adt.Notifier;

/**
 * Class <code>GeoPositionContext</code> extends {@link AbstractGeoPositionContext} to fill in those properties necessary in the context of a
 * friendly organisation.  This class adds an implementation of the <code>evaluateResult()</code> method that effectively implements sensing by
 * iterating through the result, which must be a set of positions, and invoking the owner's <code>testForLocationChange</code> method.
 **/
public class GeoPositionContext
    extends AbstractGeoPositionContext {

    /** The task that contains the testForLocationChange() callback method. */
    private final HumanIntelTask owner;
    /** Used to notify an observer of GeoPosition events. */
    private final Notifier       notifier;

    public final static String SENSING_FUNCTION = "sensing";

    public GeoPositionContext(ACL acl, HumanIntelTask owner, FriendlyOrg friendlyOrg) {
        super(acl);
        this.owner    = owner;
        this.notifier = new Notifier(friendlyOrg);
    }

    public Boolean evaluateResult(Object term, Object equivalentTerm) throws SL0.EvaluationException {
        // Should verify that left-hand term is the sensing function.

        Set positions;
        try {
            positions = (Set)equivalentTerm;
        } catch ( ClassCastException e ) {
            throw new SL0.EvaluationException("Cannot evaluate result: Expected Set as right-hand term", e);
        }
        Diagnostics.domainEvent("Sensed " + positions.size() + " position(s).", this);
```

```
        // Iterate through the set.  Let the owning task test for location changes of all object items in the set.
        for ( Iterator pIter = positions.iterator(); pIter.hasNext(); ) {
            GeoPosition p = (GeoPosition)pIter.next();
            owner.testForLocationChange(p);
            if ( p.getObjName() != null )
                notifier.signal(p);
        }
        return new Boolean(true);
    }
    /**
     * Implements the truth method. Asserting or verifying truth makes no sense in this context, so this method always
     * throws an EvaluationException.
     **/
    public Boolean truth(Boolean truth) throws SL0.EvaluationException {
        throw new SL0.EvaluationException("Can't handle the truth.");
    }
}
```

3.5. Class Tasks

The Tasks class is a FIPA-OS agent started by a FriendlyOrg instance. It isn't a true agent. It encapsulates asynchronous functionality, for which FIPA-OS has support superior to Java's thread mechanism.

Tasks invokes an IdleTask as the root task. This task is implemented as a nested class of Tasks. IdleTask is a public class (as is the Tasks agent, and all other nested classes implementing tasks). This visibility is necessary to use FIPA-OS's callback structure. FIPA-OS will attempt to invoke a method named doneSimStateSearchAgentTask() when a SimStateSearchAgentTask completes. For the invocation to succeed, both the method and its containing class must be public.

Tasks also contains nested class OverviewAgentsSearch. This class is local to the package. It is used in MVC notification.

```
package ida.bh.cro.friendlyorg;

import java.io.IOException;
import java.io.StringReader;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;

// Import the fipa classes
import fipaos.ont.fipa.*;
```

```
// We will also need to import agent classes
import fipaos.agent.*;
import fipaos.agent.task.Task;
import fipaos.agent.task.WaitTask;
import fipaos.ont.fipa.fipaman.AgentID;

import fipaos.agent.conversation.Conversation;

import ida.sd.Diagnostics;

import ida.sd.adt.Notifier;
import ida.sd.adt.OrganisationObserver;
import ida.sd.FIPA.C4IAgent;
import ida.bh.ss.fipa.AbstractFindOverviewAgentsTask;
import ida.bh.ss.fipa.SimStateAgentSearchTask;
import ida.bh.ss.fipa.FlashRequest;
import ida.sd.FIPA.fipaPA;
import ida.sd.gh5ontology.GH5KB;
import ida.sd.osb.BindingException;
import ida.sd.osb.OntologySQLBinding;

/**
 * The <code>Tasks</code> class implements an agent that performs various tasks for a {@link FriendlyOrg}.
 * <p>The agent isn't a true agent, meaning that it isn't examining the ontology and helping in decision-making.  FIPA-OS just happens to be the
 * best way to implement the necessary functionality.
 * <p>The tasks this agent performs are as follows:
 * <ol>
 * <li>It initiates all appropriate sensor tasks.</li>
 * <li>It initiates periodic searches for {@link ida.bh.cro.overview.OverviewAgent}s.</li>
 * <li>On request, it sends "flash location" messages to the {@link ida.bh.cro.overview.OverviewAgent}s.</li>
 * </ol>
 * The assumption in this model is that sensors aren't implemented by agents.  Sensing is a low-level activity that feeds directly into a GH5
 * data set.  Rather, agents examine a GH5 knowledge base, looking for patterns of interest.  Sensing is therefore implemented through this task,
 * rather than as a separate agent.
 **/
public class Tasks
    extends C4IAgent {

    /** The agent type used when registering with the DF. */
    public final static String AGENT_TYPE = "fo-tasks-agent";

    /** Used to notify the parent class of events. */
```

```
private final Notifier _notifier;
/** A link between the GH5 ontology and an underlying DB. */
private final OntologySQLBinding _osb;
/** The interval, in milliseconds, between the times when the ontology is to be refreshed from an underlying DB. */
private final int _kbRefreshInterval;

/** The name of the organisation being modeled. */
private final String _orgName;

/** {@link ida.bh.cro.overview.OverviewAgent}s active as of the last search. */
private Collection _overviewAgents = new LinkedList();

/** The ID of the {@link ida.bh.cro.simstate.SimState} agent for this simulation. */
private AgentID _simStateAgent;

/** The instance that invoked this instance. */
private final OrganisationObserver _orgObserver;
/** The underlying knowledge base. */
private final GH5KB _gh5KB;

/**
 * Constructs a <code>Tasks</code> instance.  Initializes fields, registers this agent with the AMS and DF, then starts an
 * {@link Tasks.IdleTask} as the listener task.
 * @param gh5KB A GH5 knowledge base.
 * @param name The name of the agent, which must be unique across the platform.
 * @param knowledgeBaseRefreshInterval  The interval, in seconds, between refreshes of the knowledge base from the underlying DBMS.
 *        Ignored if <code>osb</code> is null.
 * @param osb A binding between the knowledge base and an underlying DBMS being used to populate the knowledge base.  If  null, no
 *        DBMS underlies the knowledge base.
 * @param o The {@link FriendlyOrg} that invoked this agent.
 */
public Tasks(GH5KB gh5KB, String name, Integer knowledgeBaseRefreshInterval, OntologySQLBinding osb, OrganisationObserver o) {
    super(name);
    _gh5KB = gh5KB;
    _orgObserver = o;
    _notifier = new Notifier(o);
    _orgName = _gh5KB.getObjectItemName(o.getOrganisation());
    if ( (_osb = osb) != null )
        _kbRefreshInterval = knowledgeBaseRefreshInterval.intValue() * 1000;
    else
        _kbRefreshInterval = 0;     // Needed 'cause _kbRefreshInterval is final.
```

```
    startPushing();
    registerWithAMS();
    registerWithDF(AGENT_TYPE);
    setListenerTask(new IdleTask());
}
/**
 * Requests {@link ida.bh.cro.overview.OverviewAgent}s to flash the location of the friendly organisation this application is modeling.
 **/
public void sendFlashRequestToOverviewAgents() {
    Collection overviewAgents = new LinkedList();
    synchronized ( _overviewAgents ) {
        overviewAgents.addAll(_overviewAgents);
    }
    FlashRequest.send(this, _orgName, overviewAgents);
}
/**
 * The <code>IdleTask</code> class is the listener task for the <code>Task</code> agent.  Its roles are to:
 * <ol>
 * <li>Initiate all appropriate sensor tasks.</li>
 * <li>Initiate the searches for {@link ida.bh.cro.overview.OverviewAgent}s.</li>
 * <li>Initiate the search for the {@link ida.bh.cro.simstate.SimState} agent.</li>
 * <li>Initiate periodic refreshes of the knowledge base from the database, if one is being used.</li>
 * </ol>
 **/
public class IdleTask extends Task {
    /** Creates a new <code>IdleTask</code>. */
    public IdleTask() {}
    /**
     * Callback method invoked when the task is initialised.  Starts three tasks: a {@link Tasks.FindOverviewAgentsTask},
     * a {@link SimStateAgentSearchTask}, and a {@link Tasks.KnowledgeBaseRefreshTask}.  This last task is
     * only started if a DBMS is being used.
     **/
    protected void startTask() {
        newTask( new FindOverviewAgentsTask() );
        newTask( new SimStateAgentSearchTask() );
        if ( _osb != null )
```

```
            newTask( new KnowledgeBaseRefreshTask() );
    }
    /**
     * Callback method invoked when the {@link SimStateAgentSearchTask} completes. Starts the tasks that simulate sensors.
     * (Currently this is limited to a {@link HumanIntelTask}.)  Note that it doesn't make sense to start sensing until the <code>SimState</code>
     * agent is found, because there  wouldn't be anyone to receive or report ground truth.
     * @param result The agent ID of the <code>SimState</code> agent.
     **/
    public void doneSimStateAgentSearchTask(Object result) {
        Diagnostics.fipaEvent("SimStateAgentSearchTask completed.", this);
        _simStateAgent = (AgentID)result;
        newTask( new HumanIntelTask(_gh5KB, (FriendlyOrg)_orgObserver, _simStateAgent) );
        // Should also search the ontology for other sensors this org has.
    }
    /**
     * Callback method invoked when the {@link Tasks.FindOverviewAgentsTask} completes.  Does nothing.
     * @param t The completed task.
     **/
    public void doneTasks_FindOverviewAgentsTask(Task t) {
        Diagnostics.fipaEvent("FindOverviewAgentsTask completed.", this);
    }
    /**
     * Callback method invoked when the {@link Tasks.KnowledgeBaseRefreshTask} completes. Does nothing.
     * @param t The completed task.
     **/
    public void doneTasks_FindKnowledgeBaseRefreshTask(Task t) {
        Diagnostics.fipaEvent("KnowledgeBaseRefreshTask completed.", this);
    }
}
/**
 * The <code>FindOverviewAgentsTask</code> periodically updates the set of {@link ida.bh.cro.overview.OverviewAgent}s known
 * to be active.
 **/
public class FindOverviewAgentsTask extends AbstractFindOverviewAgentsTask {
    /**
     * Callback method invoked each time a search for {@link ida.bh.cro.overview.OverviewAgent}s  completes.  If the set of known
     * <code>OverviewAgent</code>s has changed since the last invocation, updates that set, and signals the {@link OrganisationObserver}
```

```
     * given to this {@link Task} of the agents found.
     * @param overviewAgents The <code>OverviewAgent</code>s found in the search.
     * @param changed True if <code>overviewAgents</code> differs from the last time this method was invoked, false if not or if
     *             this method is being invoked for the first time.
     **/
    public void utilize(Collection overviewAgents, boolean changed) {
        if ( changed ) {
            synchronized( _overviewAgents ) {
                _overviewAgents.clear();
                _overviewAgents.addAll(overviewAgents);
            }
            _notifier.signal(new OverviewAgentsSearch(_overviewAgents));
        }
    }
}
/**
 * The <code>KnowledgeBaseRefreshTask</code> class implements a {@link Task} that periodically refreshes the knowledge base from
 * the underlying database.
 **/
public class KnowledgeBaseRefreshTask extends Task {
    /** Creates a <code>KnowledgeBaseRefreshTask</code>. */
    public KnowledgeBaseRefreshTask() {}

    /**
     * Callback method invoked when the task is initialised.  Starts a {@link WaitTask}.  When the <code>WaitTask</code> ends, the
     * knowledge base gets refreshed.
     **/
    protected void startTask() {
        Diagnostics.fipaEvent("KnowledgeBaseRefreshTask started.  Starting WaitTask.", this);
        newTask ( new WaitTask(_kbRefreshInterval) );
    }

    /**
     * Callback method invoked when the {@link WaitTask} started by this task ends.  Refreshes the knowledge base from the underlying DB,
     * then starts another <code>WaitTask</code>.
     * @param t The <code>WaitTask</code> that has completed.
     **/
    public void doneWaitTask(Task t) {
        Diagnostics.fipaEvent("WaitTask completed.  Updating knowledge base.", this);
```

```
            try {
                _osb.updateKBfromDB();
            } catch ( BindingException e ) {
                Diagnostics.noteException(e, this);
            }
            Diagnostics.fipaEvent("KB refresh completed.  Starting new WaitTask.", this);
            newTask ( new WaitTask(_kbRefreshInterval) );
        }
    }
    /**
     * The <code>OverviewAgentsSearch</code> class signals the results of a search for {@link OverviewAgent}s.
     **/
    class OverviewAgentsSearch {
        /** The {@link OverviewAgent}s found in a search. */
        Collection agents;
        /**
         * Creates an <code>OverviewAgentsSearch</code>.  The results of the search are visible through the {@link agents} field.
         * @param agents The {@link OverviewAgent}s found in the search.
         **/
        OverviewAgentsSearch(Collection agents) {
            this.agents = agents;
        }
    }
}
```

3.6. Class HumanIntelTask

The HumanIntelTask class implements a simulation of a human observing his immediate battle space. A HumanIntelTask is invoked by the Tasks agent. Unlike other tasks that agent invokes, HumanIntelTask is not nested. The intelligence gathering role will eventually expand in operational applications that expand the current IDA prototype. A decision was made to keep HumanIntelTask separate from other classes to encourage encapsulation of design decisions.

```
package ida.bh.cro.friendlyorg;

import java.util.Calendar;
import java.util.Collection;
import java.util.Collections;
import java.util.GregorianCalendar;
import java.util.Iterator;
```

```
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

// Import the fipa classes
import fipaos.ont.fipa.*;

// We will also need to import agent classes
import fipaos.agent.*;
import fipaos.agent.task.Task;
import fipaos.agent.task.WaitTask;
import fipaos.ont.fipa.fipaman.AgentID;

import fipaos.agent.conversation.Conversation;

import ida.sd.Diagnostics;

import ida.bh.cro.simstate.*;
import ida.bh.ss.protocol.RequestReplyProtocol;

import ida.sd.adt.Notifier;
import ida.sd.Earth;
import ida.sd.FIPA.C4IAgent;
import ida.sd.FIPA.sl0.ParseException;
import ida.sd.FIPA.sl0.SL0;
import ida.sd.FIPA.sl0.SL0Parser;
import ida.sd.gh5ontology.*;
import ida.sd.gpc.GeoPosition;
import ida.sd.osb.BindingException;
import ida.sd.osb.OntologySQLBinding;
import ida.sd.ontology.*;
import ida.sd.pa.MathPA;
import ida.sd.pa.PoissonDistribution;

/**
 * The <code>HumanIntelTask</code> class implements a task that simulates human observers scanning the visible terrain and reporting
 * movement of enemy organisations.  The limits of "visible terrain" are considered to be the limits the map, and since we don't have any feature
 * information, we consider the entire map visible.
 * <p>This task is local to the <code>friendlyorg</code> package.  It is declared public to meet fipaos implementation requirements.
 **/
public class HumanIntelTask
    extends Task {
```

```
private final GH5KB _gh5KB;
private final PoissonDistribution _delay;
private final FriendlyOrg _friendlyOrg;
private final AgentID _simStateAgent;
private final static int MEAN_DELAY = 30;
/**
 * Constructs a <code>HumanIntel</code> instance.
 * @param friendlyOrg    The {@link FriendlyOrg} that invoked this task.
 */
public HumanIntelTask(GH5KB gh5KB, FriendlyOrg friendlyOrg, AgentID simStateAgent) {
    _gh5KB = gh5KB;
    _friendlyOrg = friendlyOrg;
    _simStateAgent = simStateAgent;
    _delay = new PoissonDistribution(MEAN_DELAY);
}
/**
 * Callback method invoked when the task is initialised.  Starts a <code>WaitTask</code>,
 * after which the first map examination is performed.
 **/
protected void startTask() {
    Diagnostics.fipaEvent("HumanIntelTask started OK.  Starting WaitTask", this);
    newTask( new WaitTask(_delay.next() * 1000) );
}
/**
 * Callback method invoked when the <code>WaitTask</code> completes.
 * Sends a request to the SimState agent for all visible battlefield objects.
 **/
public void doneWaitTask(Task t) {
    Diagnostics.fipaEvent("WaitTask done.  Asking SimState agent for intel.", this);
    ACL request = getNewConversation(RequestReplyProtocol.getProtocolName());
    request.setPerformative(FIPACONSTANTS.REQUEST);
    request.setOntology(GeoPositionContext.getName());
    request.setLanguage(SL0Parser.getLanguage());
    String nw     = "(position :latitude " + _friendlyOrg.getNWLatitude()     + " :longitude " + _friendlyOrg.getNWLongitude()     + ")";
    String center = "(position :latitude " + _friendlyOrg.getCenterLatitude() + " :longitude " + _friendlyOrg.getCenterLongitude() + ")";
```

```
    String reportRequest =  "(" +
                            GeoPositionContext.SENSING_FUNCTION +
                            " :center " +
                            center +
                            " :northwest " +
                            nw +
                            " :report-org-name \"true\")";
    SL0.Content content;
    try {
        content = SL0Parser.parseContent("((action " + _simStateAgent.toString() + " " + reportRequest + "))", true);
    } catch ( ParseException e ) {
        throw new Error(e);
    }
    request.setContentObject(content.toString());
    request.setReceiverAID(_simStateAgent);
    Diagnostics.fipaEvent("Sending intel request " + (String)request.getContentObject(), this);
    forward(request);
}

/**
 * Callback method invoked when the agent receives a message with the INFORM performative.
 * Handles the message according to its ontology.  (Currently only {@link GeoPositionContext} is recognized.)
 * @param conv The conversation containing the message.
 **/
public void handleInform(Conversation conv) {
    Diagnostics.reportHandle(conv, this);
    ACL acl = conv.getACL(conv.getLatestMessageIndex());

    if ( acl.getOntology().equals(GeoPositionContext.getName()) ) {
        handleGeoPositionInform(acl);
    }
    else {
        Diagnostics.error("Unexpected ontology " + acl.getOntology(), this);
        sendNotUnderstood(acl);
    }

    int delay = _delay.next();
    Diagnostics.fipaEvent("Starting WaitTask for " + delay + " seconds.", this);
    newTask( new WaitTask(delay * 1000) );
```

```
}
/**
 * Handles an ACL whose ontology is geo-position.  Extracts, parses, and evaluates the ACL's content, which results in the invocation of
 * {@link testForLocationChange}.
 * @param acl An ACL whose ontology indicates it's geoposition information.
 * @throws ParseException If the content is syntactically invalid.
 * @throws SL0.EvaluationException If the content is semantically invalid.
 **/
private void handleGeoPositionInform(ACL acl) {
    if ( ! acl.getLanguage().equals(SL0Parser.getLanguage()) ) {
        Diagnostics.error("Unexpected language " + acl.getLanguage(), this);
        sendNotUnderstood(acl);
        return;
    }
    try {
        SL0.Content content = SL0Parser.parseContent((String)acl.getContentObject());
        Diagnostics.fipaEvent("Received content " + content.toString(), this);
        content.evaluate(new GeoPositionContext(acl, this, _friendlyOrg));
    } catch ( ParseException e ) {
        Diagnostics.noteException(e, this);
        sendNotUnderstood(acl);
        return;
    } catch ( SL0.EvaluationException e ) {
        Diagnostics.noteException(e, this);
        sendNotUnderstood(acl);
        return;
    }
}
private final static int PRECISION = 5;
/**
 * Callback method invoked in response to evaluation of
 * <code>(result (sensing ...) ;(set  (</code>{@link GeoPosition}<code>1{@link GeoPosition}<code>2...)))</code>
 * atomic formulae of the {@link GeoPositionContext} ontology. This method is invoked with a single {@link GeoPosition} <i>p</i>.
 * that contains the name of a battlefield object.  It tests whether the last recorded position of that object is different from the one in <i>p</i>.
 * If so (or if the object has no known position) it {@linkplain noteLocationChange notes the change of location}.
 * @param p The {@link GeoPosition} of some battlefield object.
 **/
```

```
void testForLocationChange(GeoPosition p) {
    float sensedLatitude = p.getLatitude().floatValue();
    float sensedLongitude = p.getLongitude().floatValue();

    Instance objItem = _gh5KB.getObjectItem(p.getObjName());
    Instance lastRecordedLocation = _gh5KB.getCurrentLocation(objItem);
    if ( lastRecordedLocation == null ) {
        noteLocationChange(objItem, sensedLatitude, sensedLongitude);
        return;
    }

    float lastRecordedLatitude = _gh5KB.getFloatSlot(lastRecordedLocation, "latitudeCoordinate").floatValue();
    float lastRecordedLongitude = _gh5KB.getFloatSlot(lastRecordedLocation, "longitudeCoordinate").floatValue();

    if ( ! ( MathPA.almostEqual(lastRecordedLatitude, sensedLatitude, PRECISION) &&
             MathPA.almostEqual(lastRecordedLongitude, sensedLongitude, PRECISION) ) )
        noteLocationChange(objItem, sensedLatitude, sensedLongitude);
}
/**
 * Creates a specification of a change in location of some battlefield object, and saves that specification in the underlying DBMS.
 * The specification is encapsulated using GH5 entities:
 * <ol>
 * <li>An ObjectItemLocation.</li>
 * <li>A ReportingDataAbsoluteTiming:
 *    <ol>
 *    <li>reportingTime and reportingDate are set to the current time.</li>
 *    <li>effectiveTime and EffectiveDate are set to the current time.</li>
 *    <li>The duration is set to 1000 seconds.  This value is arbitrary, and not realistic.  Perhaps it should be based on perceived speed of
 *       the object.</li>
 *    <li>effectiveTimePrecisionCode is set to SECOND.  Given that this is human intelligence, MINUTE might be better.</li>
 *    <li>reliabilityCode is set to B (usually reliable).</li>
 *    <li>sourceTypeCode is set to EYOBSN (human observation).</li>
 *    <li>credabilityCode is set to RPTPLA (possible or probable).</li>
 *    </ol>
 * </li>
 * <li>An AbsolutePoint, with latitude and longitude set from parameters.</li>
 * </ol>
 * Establishes entity relationships (e.g., <code>ObjectItemLocation</code> <code>is-referenced-to</code> <code>ReportingData</code>).
 * @param obj The battlefield object whose location has changed.
 * @param latitude The new latitude of the battlefield object.
```

```
 * @param longitude The new longitude of the battlefield object.
 **/
private void noteLocationChange(Instance obj, float latitude, float longitude) {
    Calendar now = Calendar.getInstance();
    try {
        // Create the associative entity.
        Instance objItemLoc = _gh5KB.getCls("ObjectItemLocation").createDirectInstance();

        setObjectItemLocationAttributes(obj, objItemLoc, (double)latitude, (double)longitude, now);

        // Create the reporting data that's associated with the
        // ObjectItemLocation.
        Instance report = _gh5KB.getCls("ReportingDataAbsoluteTiming").createDirectInstance();
        _gh5KB.setDateTimeSlots(report, "reportingDate", "reportingTime", now);
        _gh5KB.setDateTimeSlots(report, "effectiveDate", "effectiveTime", now);
        _gh5KB.setIntSlot(report, "duration", new Integer(1000));   // 1000 is arbitrary.  There are some interesting possibilities here.
        _gh5KB.setCodeSlot(report, "effectiveTimePrecisionCode", "SECOND");
        _gh5KB.setCodeSlot(report, "reliabilityCode", "B");
        _gh5KB.setCodeSlot(report, "sourceTypeCode", "EYOBSN");
        _gh5KB.setCodeSlot(report, "credibilityCode", "RPTPLA");
        _gh5KB.setCodeSlot(report, "categoryCode", "REP");
        _gh5KB.setCodeSlot(report, "entityCategoryCode", "OILOCA");
        _gh5KB.setCodeSlot(report, "countingIndicatorCode", "NO");
        report.setOwnSlotValue(_gh5KB.getSlot("reporting-data-is-reported-by-Organisation"), _friendlyOrg.getOrganisation());

        // Create the new location.
        Instance pt = _gh5KB.getCls("AbsolutePoint").createDirectInstance();
        _gh5KB.setFloatSlot(pt, "latitudeCoordinate", new Double(latitude));
        _gh5KB.setFloatSlot(pt, "longitudeCoordinate", new Double(longitude));
        _gh5KB.setCodeSlot(pt, "angularPrecisionCode", "SECOND");

        // Create the relationships.
        objItemLoc.setOwnSlotValue(_gh5KB.getSlot("obj_item_loc-is-referenced-to-ReportingData"), report);
        obj.addOwnSlotValue(_gh5KB.getSlot("is-geometrically-defined-through-ObjectItemLocation"), objItemLoc);
        pt.addOwnSlotValue(_gh5KB.getSlot("provides-geometric-definition-for-ObjectItemLocation"), objItemLoc);
        Diagnostics.domainEvent("Created new location for " + _gh5KB.getObjectItemName(obj), this);

        // Save the instances.  Because of DB integrity constraints, order matters.
        // It would be better to have osb determine the correct order, thereby
        // removing knowledge of the DB from this class.
        OntologySQLBinding osb = _friendlyOrg.getOSB();
```

```
        if ( osb == null )
            return;

        osb.saveInstances(new Instance[] { pt, report, objItemLoc });

        // Place an item in the KB update area.
        new Notifier(_friendlyOrg).signal("(" +
                                                pt.getDirectType().getName() + ", " +
                                                report.getDirectType().getName() + ", " +
                                                objItemLoc.getDirectType().getName() +
                                                ")");
    } catch ( InvalidCodeException e ) {
        throw new Error(e);
    } catch ( BindingException e ) {
        Diagnostics.noteException(e, this);
    }
}

private void setObjectItemLocationAttributes(Instance objItem, Instance objItemLoc, double latitude, double longitude, Calendar now) {
    // If the object has a last known location, compute speed and bearing angle.
    Collection prevLocs = objItem.getOwnSlotValues(_gh5KB.getSlot("is-geometrically-defined-through-ObjectItemLocation"));
    if ( prevLocs.size() == 0 )
        return;

    List prevLocsList = new LinkedList(prevLocs);
    Collections.sort(prevLocsList, new RDComparator(_gh5KB, "provides-applicable-information-for-ObjectItemLocation"));
    Instance prevObjItemLoc = (Instance)prevLocsList.get(prevLocsList.size() - 1);
    Instance prevLocation   = (Instance)prevObjItemLoc.getOwnSlotValue(_gh5KB.getSlot("obj_item_loc-is-associated-with-Location"));

    double prevLat = _gh5KB.getFloatSlot(prevLocation, "latitudeCoordinate").doubleValue();
    double prevLon = _gh5KB.getFloatSlot(prevLocation, "longitudeCoordinate").doubleValue();

    _gh5KB.setFloatSlot(objItemLoc, "bearingAngle", new Double(Earth.angle(prevLat, prevLon, latitude, longitude)));

    Instance prevLocationRD = (Instance)prevObjItemLoc.getOwnSlotValue(_gh5KB.getSlot("obj_item_loc-is-referenced-to-ReportingData"));
    if ( prevLocationRD == null )
        return;     // Without timing, we can't compute speed.
    if ( ! prevLocationRD.instanceOf(_gh5KB.getCls("ReportingDataAbsoluteTiming") ) )
        return;     // We don't handle relative timing yet.

    String prevEffectiveDate = _gh5KB.getStringSlot(prevLocation, "effectiveDate");
    String prevEffectiveTime = _gh5KB.getStringSlot(prevLocation, "effectiveTime");
    if ( prevEffectiveDate == null || prevEffectiveTime == null )
```

```
            return;
        String etpc = _gh5KB.getCodeSlot(prevLocationRD, "effectiveTimePrecisionCode");
        if ( etpc == null )
            etpc = "SECOND";
        // timeDiff is in seconds.
        long timeDiff = (now.getTime().getTime() - DateTime.getCalendar(prevEffectiveDate, prevEffectiveTime).getTime().getTime())/1000;
        // distance is in kilometers.
        double distance = Earth.distance(latitude, longitude, prevLat, prevLon)/1000.0;
        // speed is in km/hr.
        double speed = distance/timeDiff * 3600.0;
        _gh5KB.setFloatSlot(objItemLoc, "speedRate", new Double(speed));
        try {
            _gh5KB.setCodeSlot(objItemLoc, "useCategoryCode", "CEOFMA");
        } catch ( InvalidCodeException e ) {
            throw new Error(e);
        }
        // In the following, 100.0 is arbitrary.  What would be a realistic
        // accuracy value for human observation?
        _gh5KB.setFloatSlot(objItemLoc, "accuracyQuantity", new Double(100.0));
    }
}
```

3.7. Class COAAnalysisGUI

The decision support application pops up certain windows automatically or by user request. One that users may request is an analysis of a course of action. Class COAAnalysisGUI implements a user interface for that analysis. A course-of-action analysis is displayed in response to a user request from a ThreatNotificationGUI.

The class is implemented by extending JDialog. The implication is that a user will want to examine an analysis before continuing to assess the consequences of a threat and selecting a response.

```
package ida.bh.cro.friendlyorg;

import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Font;
```

```
import java.awt.Frame;

import java.util.Iterator;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;

import ida.sd.pa.GUI;

import ida.bh.cro.friendlyorg.threat.response.CourseOfAction;

/**
 * The <code>COAAnalysisGUI</code> class provides the user interface for display of detailed course of action rationale in response to a threat.
 * The GUI presents a tabular listing of the steps in the rationale for a CoA.
 **/
class COAAnalysisGUI
    extends JDialog {

    /** The {@link ida.bh.agent.threat.response.CourseOfAction} associated with this instance. */
    private CourseOfAction _coa;

    /**
     * Constructs a <code>COAAnalysisGUI</code>, which is a user interface for showing the rationale behind a course of action.  Creates and
     * assembles the components of the GUI and presents them to the user.
     * @param owner The dialog that owns this GUI.
     * @param coa The course of action to display.
     **/
    public COAAnalysisGUI(JDialog owner, CourseOfAction coa) {
        super(owner, "Course of Action Rationale");

        _coa = coa;

        Container contentPane = getContentPane();
        contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
        contentPane.add(GUI.VERTICAL_GAP);

        JLabel stmt = new JLabel(coa.description(), SwingConstants.CENTER);
        stmt.setAlignmentX(0.5f);
        stmt.setFont(new Font("Serif", Font.BOLD, (int)(getFont().getSize()*1.25)));
        contentPane.add(stmt);
        contentPane.add(GUI.VERTICAL_GAP);

        JTable coaTable = new JTable(new AbstractTableModel() {
                public int getRowCount() {
                    return _coa.getRationale().size();
```

```
            }
            public int getColumnCount() {
                return 1;
            }
            public Object getValueAt(int row, int column) {
                return _coa.getRationale().get(row);
            }
        });
    coaTable.setRowSelectionAllowed(false);
    coaTable.setColumnSelectionAllowed(false);
    coaTable.setTableHeader(null);

    int maxRationaleWidth = 0;
    for ( Iterator rIter = _coa.getRationale().iterator(); rIter.hasNext(); ) {
        int w = getFontMetrics(coaTable.getFont()).stringWidth((String)rIter.next());
        if ( w > maxRationaleWidth )
            maxRationaleWidth = w;
    }
    coaTable.setPreferredSize(new Dimension(maxRationaleWidth, coaTable.getPreferredSize().height));

    JScrollPane coaTablePane = GUI.wrapTableInScrollPane(coaTable, "Rationale", null);
    coaTablePane.setAlignmentX(0.5f);
    contentPane.add(coaTablePane);
    contentPane.add(GUI.VERTICAL_GAP);

    JButton _close = new JButton("OK");
    _close.addActionListener(new CloseDialog(this));
    _close.setAlignmentX(0.5f);
    contentPane.add(_close);

    pack();
    setVisible(true);
}

/**
 * The <code>CloseDialog</code> class implements Java's {@link java.awt.event.ActionListener ActionListener} interface to
 * allow the {@link #_close} button to close the dialog.
 **/
private class CloseDialog implements ActionListener {
    private JDialog _dialog;
```

```
        CloseDialog(JDialog d) {
            _dialog = d;
        }

        public void actionPerformed(ActionEvent e) {
            _dialog.dispose();
        }
    }
}
```

3.8. Class ThreatExaminationGUI

The decision support application pops up certain windows automatically or by user request. Whenever it displays a threat, a user may request a detailed examination of that threat. Class ThreatExaminationGUI implements a user interface for that examination.

```
package ida.bh.cro.friendlyorg;

import java.awt.Container;
import java.awt.GridBagLayout;

import java.text.DecimalFormat;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;

import ida.sd.Earth;
import ida.sd.gh5ontology.*;
import ida.sd.ontology.*;
import ida.sd.pa.GUI;

/**
 * The <code>ThreatExaminationGUI</code> class displays a window that describes a threat.  The primary secrets of this class are the
 * information presented and the manner in which it is presented.
 **/
public class ThreatExaminationGUI
    extends JFrame {
    /**
     * Creates a <code>ThreatExaminationGUI</code> and displays it to the user.
     * @param friendlyOrg The friendly organisation that has a threat.
     * @param threatName  The name of the organisation that is a threat.
```

```
 **/
public ThreatExaminationGUI(FriendlyOrg friendlyOrg, String threatName) {
    super("Threat Examination of " + threatName);

    Container contentPane = getContentPane();
    contentPane.setLayout(new GridBagLayout());

    GH5KB kb = friendlyOrg.getKB();

    Instance threat = kb.getObjectItem(threatName);

    String value;

    GUI.addLabeledComponent(contentPane, 0, "Name:", new JLabel(threatName, SwingConstants.LEFT));

    Instance threatLocation = kb.getCurrentLocation(threat);
    if ( threatLocation != null ) {
        value = Earth.image(kb.getFloatSlot(threatLocation, "latitudeCoordinate").doubleValue(),
                            kb.getFloatSlot(threatLocation, "longitudeCoordinate").doubleValue());
    }
    else {
        value = "(none recorded)";
    }
    GUI.addLabeledComponent(contentPane, 1, "Location:", new JLabel(value, SwingConstants.LEFT));

    Instance fo = friendlyOrg.getOrganisation();
    Instance foLocation = kb.getCurrentLocation(fo);
    if ( foLocation != null ) {
        try {
            int duration = kb.estimatedTimeToReach(threat, foLocation).intValue();
            if ( duration >= 86400 )
                value = ">= 1 day";
            else {
                long hrs  = duration/3600;
                long remSecs = duration%3600;
                long mins = remSecs/60;
                long secs = remSecs%60;
                DecimalFormat twoD = new DecimalFormat("00");
                value = twoD.format(hrs) + ":" + twoD.format(mins) + ":" + twoD.format(secs);
            }
        } catch ( NoLocationException e ) {
            value = "(threat location unknown)";
        } catch ( SpeedIndeterminateException e ) {
```

```
                value = "(threat speed indeterminate)";
            }
        }
        else {
            value = "(" + kb.getObjectItemName(friendlyOrg.getOrganisation()) + " location unknown)";
        }
        GUI.addLabeledComponent(contentPane, 2, "Time to attack:", new JLabel(value, SwingConstants.LEFT));
        pack();
        setVisible(true);
    }
}
```

3.9. Class ThreatNotificationGUI

The decision support application pops up certain windows automatically or by user request. It automatically displays an instance of the ThreatNotificationGUI class in response to messages it receives from its Threat Notification Agent.

```
package ida.bh.cro.friendlyorg;

import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.Font;
import java.awt.Insets;

import javax.swing.AbstractCellEditor;
import javax.swing.BorderFactory;
import javax.swing.border.TitledBorder;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
```

```
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.SwingConstants;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableCellEditor;
import javax.swing.table.TableColumn;
import javax.swing.table.TableColumnModel;

import ida.bh.cro.friendlyorg.threat.response.CourseOfAction;
import ida.bh.cro.friendlyorg.threat.response.ThreatResponseNotification;

import ida.sd.Diagnostics;
import ida.sd.adt.GUI.ButtonCellEditor;
import ida.sd.adt.Notifier;
import ida.sd.adt.OrganisationObserver;
import ida.sd.gh5ontology.GH5KB;
import ida.sd.pa.GUI;

/**
 * The <code>ThreatNotificationGUI</code> class provides the user interface for display of threat notifications.  The GUI presents:
 * <ul>
 * <li>A statement of the threat.</li>
 * <li>A table of prioritized candidate courses of action.</li>
 * <li>The ability to analyze a course of action.  (The analysis is to examine the rationale underlying it.)</li>
 * <li>The ability to choose a course of action.</li>
 * </ul>
 **/
class ThreatNotificationGUI
    extends JDialog {
    /** The {@link ida.bh.agent.threat.response.ThreatResponseNotification}
     * being shown to the user. */
    private ThreatResponseNotification _trn;

    // The following are the fields of this dialog.
    /** Lets use select whether to show all courses of action, or only those with non-zero effectiveness. */
    private JCheckBox _showImpossibleCoAs = new JCheckBox("Show Impossible Courses of Action");
    /** The buttons a user clicks to analyze a COA in detail. */
```

```
private JButton[] _selectionButtons;
/** The table that holds the courses of action. */
private JTable       _coaTable;
/** Supplies the data for {@link #_coaTable}. */
private COATableModel _coaTableModel;
/** Never set {@link #_coaTable}'s height to show more than this number of rows.
 * (Instead, let a {@linkplain #_tablePane scroll pane} do the work.) */
private final static int MAX_ROWS = 10;
/** Keeps {@link #_coaTable} to a fixed maximum size. */
private JScrollPane   _tablePane;

/** Font for title. */
private final static Font TITLE_FONT       = new Font("Serif", Font.BOLD, 14);
/** Font for threat statement. */
private final static Font THREAT_STMT_FONT = new Font("Serif", Font.BOLD, 14);

private Notifier _notifier;

/**
 * Constructs a <code>ThreatNotificationGUI</code> that presents a {@link ThreatResponseNotification} to the user.
 * @param ownerFrame The frame that will own this dialog.
 * @param org The {@link FriendlyOrg} presenting this dialog to a user.
 * @param gh5kb The underlying knowledge base.
 * @param tr The {@link ThreatResponseNotification} to be shown to the user.
 **/
public ThreatNotificationGUI(JFrame ownerFrame, OrganisationObserver org, GH5KB gh5kb, ThreatResponseNotification tr) {
    super(ownerFrame, gh5kb.getObjectItemName(org.getOrganisation()) + ": Courses of Action");

    _notifier = new Notifier(org);
    _trn = tr;

    int nCoAs = _trn.size();
    int contentPanePreferredWidth = 0;
    int w;
    int contentPanePreferredHeight = 0;

    Container contentPane = getContentPane();
    contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));
    contentPane.add(GUI.VERTICAL_GAP);
    contentPanePreferredHeight += 10;

    JLabel windowTitle = new JLabel(gh5kb.getObjectItemName(org.getOrganisation()) + ": Threat Response Courses of Action",
                                    SwingConstants.CENTER);
```

```
windowTitle.setAlignmentX(0.5f);
windowTitle.setFont(TITLE_FONT);
contentPane.add(windowTitle);
if ( (w = windowTitle.getPreferredSize().width) > contentPanePreferredWidth )
    contentPanePreferredWidth = w;
contentPanePreferredHeight += windowTitle.getPreferredSize().height;
JLabel threatStmt = new JLabel(tr.threatStatement(), SwingConstants.CENTER);
threatStmt.setAlignmentX(0.5f);
threatStmt.setFont(THREAT_STMT_FONT);
contentPane.add(threatStmt);
contentPane.add(GUI.VERTICAL_GAP);
if ( (w = threatStmt.getPreferredSize().width) > contentPanePreferredWidth )
    contentPanePreferredWidth = w;
contentPanePreferredHeight += threatStmt.getPreferredSize().height + GUI.VERTICAL_GAP.getHeight();

if ( nCoAs == 0 ) {
    contentPane.add(new JLabel("No available courses of action!"));
    pack();
    setVisible(true);
    return;
}

_selectionButtons = new JButton[nCoAs];
for ( int i = 0; i < nCoAs; i++ ) {
    JButton b = new JButton("View");
    b.addActionListener(new PopUpAnalysisGUI(this, _trn.get(i)));
    _selectionButtons[i] = b;
}

_coaTable = new JTable(_coaTableModel = new COATableModel());
_coaTable.setRowSelectionAllowed(true);        // Let user select a course of action.
_coaTable.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
_coaTable.setColumnSelectionAllowed(false);

TableColumnModel columnModel = _coaTable.getColumnModel();
columnModel.getColumn(0).setPreferredWidth(getFontMetrics(_coaTable.getFont()).stringWidth(_columnNames[0]) + 10);
columnModel.getColumn(1).setPreferredWidth(getFontMetrics(_coaTable.getFont()).stringWidth(_columnNames[1]) + 200);

TableCellRenderer buttonRenderer = new TableCellRenderer() {
        public Component getTableCellRendererComponent(        JTable table,
                                                               Object value,
```

```
                                                        boolean isSelected,
                                                        boolean hasFocus,
                                                        int row,
                                                        int column) {
            return _selectionButtons[row];
        }
    };
TableColumn buttonCol = _coaTable.getColumnModel().getColumn(_coaTableModel.getButtonColumn());
buttonCol.setCellRenderer(buttonRenderer);
buttonCol.setPreferredWidth(_selectionButtons[0].getPreferredSize().width + 10);
buttonCol.setCellEditor(new ButtonCellEditor());

GUI.setWidthProperties(columnModel.getColumn(0));
GUI.setWidthProperties(buttonCol);

int preferredTableWidth = 0;
for ( int i = 0; i < 3; i++ )
    preferredTableWidth += columnModel.getColumn(i).getPreferredWidth();
_coaTable.setPreferredSize(new Dimension(preferredTableWidth, coaTableHeight()));

_tablePane = GUI.wrapTableInScrollPane(_coaTable, coaTableTitle(), new Integer(MAX_ROWS < _trn.size() ? MAX_ROWS : _trn.size()));
_tablePane.setAlignmentX(0.5f);
contentPane.add(_tablePane);
contentPane.add(GUI.VERTICAL_GAP);
if ( (w = _tablePane.getPreferredSize().width) > contentPanePreferredWidth )
    contentPanePreferredWidth = w;
contentPanePreferredHeight += _tablePane.getPreferredSize().height + GUI.VERTICAL_GAP.getHeight();

_showImpossibleCoAs.setHorizontalAlignment(SwingConstants.CENTER);
_showImpossibleCoAs.addItemListener(new ItemListener() {
        /**
         * Arranges for the table to show/hide "impossible" courses-of-action (those with zero effectiveness).  Does so by:
         * <ol>
         * <li>Resetting the table's preferred size based on the number of rows to show.</li>
         * <li>Firing a table-data-changed event.</li>
         * </ol>
         * Also resets the title of the surrounding scroll pane.
         **/
        public void itemStateChanged(ItemEvent e) {
            int rowsToShow = (e.getStateChange() == ItemEvent.SELECTED ? _trn.size() : _trn.possibleCoASize());
            _coaTable.setPreferredSize(new Dimension(_coaTable.getPreferredSize().width, _coaTable.getRowHeight() * rowsToShow));
```

```
                _coaTableModel.fireTableDataChanged();
                GUI.resetTitle(_tablePane, coaTableTitle());
            }
        });
    contentPane.add(_showImpossibleCoAs);
    if ( (w = _showImpossibleCoAs.getPreferredSize().width) > contentPanePreferredWidth )
        contentPanePreferredWidth = w;
    contentPanePreferredHeight += _showImpossibleCoAs.getPreferredSize().height + GUI.VERTICAL_GAP.getHeight();

    JButton select = new JButton("Select");
    JButton cancel = new JButton("Cancel");
    select.addActionListener(new CloseButtonListener(this, true));
    cancel.addActionListener(new CloseButtonListener(this, false));
    JPanel selectButtonsPane = new JPanel();
    selectButtonsPane.setAlignmentX(0.5f);
    selectButtonsPane.setLayout(new BoxLayout(selectButtonsPane, BoxLayout.X_AXIS));
    selectButtonsPane.add(Box.createHorizontalGlue());
    selectButtonsPane.add(select);
    selectButtonsPane.add(Box.createRigidArea(new Dimension(10, 0)));
    selectButtonsPane.add(cancel);
    selectButtonsPane.add(Box.createHorizontalGlue());
    selectButtonsPane.setPreferredSize(new Dimension( select.getPreferredSize().width +
                                                      10 +
                                                      cancel.getPreferredSize().width,
                                                      selectButtonsPane.getPreferredSize().height));
    contentPane.add(selectButtonsPane);
    if ( (w = selectButtonsPane.getPreferredSize().width) > contentPanePreferredWidth )
        contentPanePreferredWidth = w;
    contentPanePreferredHeight += selectButtonsPane.getPreferredSize().height + GUI.VERTICAL_GAP.getHeight();

    contentPane.setSize(new Dimension(contentPanePreferredWidth, contentPanePreferredHeight));

    pack();
    setLocationRelativeTo(ownerFrame);
    setVisible(true);
}

/**
 * Returns a string that provides a title for the course-of-action table.  The purpose is to standardize the title throughout this module.
 * @return A string that provides a title for the course-of-action table.
 **/
```

```
private String coaTableTitle() {
    int maxSize = _trn.size();
    int currentSize = _showImpossibleCoAs.isSelected() ? maxSize : _trn.possibleCoASize();
    return "Courses of Action (" + currentSize + " of " + maxSize + ")";
}

/**
 * Returns the height of the course-of-action table.  This value varies depending on whether the user has selected
 * {@link _showImpossibleCoAs}.
 * @return The height of the course-of-action table, in pixels.
 **/
private int coaTableHeight() {
    return _coaTable.getRowHeight() * coaTableRows();
}

/**
 * Returns the number of rows currently available for display in thecourse-of-action table.  This value varies depending on whether the
 * user has selected {@link _showImpossibleCoAs}.
 * @return The number of rows in the course-of-action table.
 **/
private int coaTableRows() {
    return _showImpossibleCoAs.isSelected() ? _trn.size() : _trn.possibleCoASize();
}

private final static String[] _columnNames = { "Rating", "Course of Action", "" };
private class COATableModel extends AbstractTableModel {
    private final static int VALUE_COL  = 0;
    private final static int DESCR_COL  = 1;
    private final static int BUTTON_COL = 2;

    public int getRowCount() {
        if ( _showImpossibleCoAs.isSelected() ) {
            return _trn.size();
        }
        else {
            for ( int i = _trn.size() - 1; i >= 0; i-- ) {
                if ( _trn.get(i).value() > 0.0 )
                    return i + 1;
            }
            return 0;
        }
```

```
}
public int getColumnCount() {
    return 3;
}
public Object getValueAt(int row, int column) {
    switch ( column ) {
    case VALUE_COL:
        return new Float(_trn.get(row).value());
    case DESCR_COL:
        return _trn.get(row).description();
    case BUTTON_COL:
        return _selectionButtons[row];
    default:
        throw new Error("Impossible column " + column);
    }
}
public String getColumnName(int column) {
    return _columnNames[column];
}
public boolean isCellEditable(int row, int column) {
    return column == BUTTON_COL;
}
public Class getColumnClass(int column) {
    switch ( column ) {
    case VALUE_COL:
        return Float.class;
    case DESCR_COL:
        return String.class;
    case BUTTON_COL:
        return JButton.class;
    default:
        throw new Error("Impossible column " + column);
    }
}
public int getButtonColumn() {
    return BUTTON_COL;
```

```
    }
}
/**
 * The <code>CloseButtonListener</code> class extends {@link java.awt.event.ActionListener}
 * to implement functionality needed for the buttons that close the <code>ThreatNotificationGUI</code>.
 **/
private class CloseButtonListener implements ActionListener {
    private JDialog _parent;
    private boolean _includeSelection;

    /**
     * Constructs a <code>CloseButtonListener</code> that specifieswhat to do when an action listener receives an event.  The
     * minimal action is to close the <code>ThreatNotificationGUI</code>.
     * @param d The {@link javax.swing.JDialog} containing the buttons for this listener.
     * @param includeSelection If true, the observer of the <code>ThreatNotificationGUI</code> will be notified.
     *          If false, the observer will not be notified.
     **/
    CloseButtonListener(JDialog d, boolean includeSelection) {
        _parent = d;
        _includeSelection = includeSelection;
    }

    /**
     * Callback method invoked when an {@link java.awt.event.ActionEvent} occurs.  Uses {@link #_notifier} to signal the observer.
     *  If {@link #_includeSelection} is true, the course of action selected by the user is included in the signal.  In any case, closes
     * the <code>ThreatNotificationGUI</code>.
     * @param e The {@link java.awt.event.ActionEvent}.
     **/
    public void actionPerformed(ActionEvent e) {
        if ( _includeSelection ) {
            ListSelectionModel lsm = _coaTable.getSelectionModel();
            if ( lsm.isSelectionEmpty() ) {
                JOptionPane.showMessageDialog(  _parent,
                                                "Select the course of action to be performed (or press Cancel)",
                                                "Missing Input",
                                                JOptionPane.ERROR_MESSAGE);
                return;
            }
            _notifier.signal(new Completed(_trn, _trn.get(lsm.getMinSelectionIndex())));
```

```
        }
        else {
            _notifier.signal(new Completed(_trn));
        }
        _parent.setVisible(false);
    }
}
private class PopUpAnalysisGUI implements ActionListener {
    private JDialog       _dialog;
    private CourseOfAction _coa;

    PopUpAnalysisGUI(JDialog d, CourseOfAction coa) {
        _dialog = d;
        _coa = coa;
    }

    public void actionPerformed(ActionEvent e) {
        new COAAnalysisGUI(_dialog, _coa);
    }
}

/**
 * The <code>Completed</code> class is used to signal that a <code>ThreatNotificationGUI</code> has completed.  An instance
 * contains the course of action a user has selected.
 **/
class Completed {
    private CourseOfAction _coa;
    private ThreatResponseNotification _trn;

    /**
     * Constructs a <code>Completed</code> in which there is noassociated {@link ida.bh.agent.threat.response.CourseOfAction}.
     **/
    Completed(ThreatResponseNotification trn) {
        _coa = null;
        _trn = trn;
    }

    /**
     * Constructs a <code>Completed</code> with an associated {@link ida.bh.agent.threat.response.CourseOfAction}.
     **/
    Completed(ThreatResponseNotification trn, CourseOfAction coa) {
```

```
            _coa = coa;
            _trn = trn;
        }
        /**
         * Returns the {@link ida.bh.agent.threat.response.CourseOfAction} associated with this instance.
         * @return The course of action associated with this instance, or null if no course of action was associated when the instance
         *         was created.
         **/
        CourseOfAction getCourseOfAction() {
            return _coa;
        }
        ThreatResponseNotification getThreatResponseNotification() {
            return _trn;
        }
    }
}
```

4. Classes in Package DBUpdate

4.1. Class DBUpdateAgent

The DBUpdateAgent class implements the agent a Friendly Organization component invokes to maintain consistency between different Generic Hub data sets that different Friendly Organization components use in the course of a simulation. Each Friendly Organization component has one of these agents. They send updates to each other.

This class has almost the minimum functionality of an agent. It registers itself, starts an IdleTask, waits until that task terminates, then deregisters itself and shuts down.

The agent's constructor takes an instance of OntologySQLBinding as a parameter. Through this instance, the agent communicates with the RDBMS accessing the relevant Generic Hub data set for this Friendly Organization component.

This class provides one method, propagateChanges(), that other classes invoke. It initiates the sending of messages that communicate changes to other Friendly Organization components.

This agent uses a service description that is detailed in comparison to those used by other agents. The URL of the underlying RDBMS is given to let these agents determine to whom changes should be propagated. Changes are only sent to agents that access other Generic Hub data sets.

```
package ida.bh.cro.friendlyorg.dbupdate;
```

```
import java.util.List;

import fipaos.agent.task.Task;
import fipaos.ont.fipa.fipaman.PropertyTemplate;
import fipaos.ont.fipa.fipaman.ServiceDescription;

import ida.sd.Diagnostics;
import ida.sd.FIPA.C4IAgent;
import ida.sd.osb.BindingException;
import ida.sd.osb.OntologySQLBinding;

/**
 * The <code>DBUpdateAgent</code> class provides an agent that:
 * <ol>
 * <li>Informs other <code>DBUpdateAgent</code>s of updates to the DB.</li>
 * <li>Listens for messages from other <code>DBUpdateAgent</code>s of updates to the DB, and performs those updates.</li>
 * </ol>
 * The primary secrets of this class are the algorithms for updating the DB, and the algorithms for informing other agents of updates to
 * the DB.
 **/
public class DBUpdateAgent
    extends C4IAgent {

    /** The registered type for this agent. */
    public final static String AGENT_TYPE = "db-upd";
    /** The registered name of the ontology for this agent. */
    public final static String DB_UPDATE_ONTOLOGY = "db-update";
    /** The registered name of the language this agent uses. */
    public final static String DB_UPDATE_LANGUAGE = "binary";
    /** The property name in this agent's service description stating the DB URL of this agent. */
    public final static String DB_URL_PROPERTY    = "db-url";
    /** The IdleTask spawned by this agent. */
    private IdleTask _idleTask;

    /**
     * Constructs a <code>DBUpdateAgent</code> instance that will initiate update messages to other agents, and will respond to update
     * messages from other <code>DBUpdateAgent</code>s.
     * @param name The name of the agent, which must be unique across the platform.
     * @param dbUpdateAgentSearchInterval The interval, in seconds, between searches for <code>DBUpdateAgent</code>s.
     * @param osb A specification of an Ontology/SQL binding.
     **/
```

```
public DBUpdateAgent(String name, Integer dbUpdateAgentSearchInterval, OntologySQLBinding osb) {
    super(name);

    startPushing();

    registerWithAMS();

    try {
        // Register with the local DF, giving a ServiceDescription with a property that states the URL of the database we're accessing.
        // FIPA 00023 states that a Service Description may specify languages, but fipa-os doesn't support that.
        ServiceDescription sd = new ServiceDescription();
        sd.setServiceType(AGENT_TYPE);
        sd.setProtocols(new String[] { "no-protocol" });
        sd.setOntologies(new String[] { DB_UPDATE_ONTOLOGY });
        sd.setProperties(new PropertyTemplate[] { new PropertyTemplate(DB_URL_PROPERTY, osb.getDBURL()) });
        registerWithDF(sd);

        // Define an IdleTask that will act as the Listener task.
        setListenerTask( _idleTask = new IdleTask(dbUpdateAgentSearchInterval, osb) );
    } catch ( Exception e ) {
        // Possible exceptions: BindingException and fipaos.parser.ParserException.
        // In either case, we stop the agent.
        Diagnostics.noteException(e, this);
        Diagnostics.error("Stopping DBUpdateAgent.", this);
        shutdown();
    }
}

/**
 * Stops this agent and all tasks it's spawned.
 **/
public void stop() {
    _idleTask.stop();
    shutdown();
}

/**
 * Callback method invoked when {@link IdleTask IdleTask} completes.
 **/
public void doneIdleTask(Task t) {
    Diagnostics.fipaEvent("IdleTask completed.", this);
}
```

```
    /**
     * Sends a message to other DBUpdateAgents that a change to the DB is in order.
     * <p>The message consists of a {@link java.util.List} of SQL queries.
     * Each query is either an <code>INSERT</code> or an <code>UPDATE</code>.
     * @param queries The queries.
     **/
    public void propagateChanges(List queries) {
        _idleTask.propagateChanges(queries);
    }
}
```

4.2. Class IdleTask

The IdleTask class listens for and handles database updates. Note that it updates a Generic Hub data set, not a GH knowledge base. The KnowledgeBaseRefreshTask of the Tasks agent performs that chore.

The IdleTask class also maintains the set of currently known DBUpdateAgents. It uses this set to determine to whom updates should be sent.

```
package ida.bh.cro.friendlyorg.dbupdate;

import java.sql.Connection;
import java.sql.SQLException;

import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

import fipaos.ont.fipa.*;
import fipaos.ont.fipa.fipaman.AgentID;
import fipaos.ont.fipa.fipaman.DFAgentDescription;
import fipaos.ont.fipa.fipaman.PropertyTemplate;
import fipaos.ont.fipa.fipaman.ServiceDescription;
import fipaos.agent.*;
import fipaos.agent.conversation.Conversation;
import fipaos.agent.task.Task;
import fipaos.agent.task.WaitTask;

import ida.sd.Diagnostics;
```

```
import ida.sd.FIPA.AgentSearchTask;
import ida.sd.FIPA.notification.InformAgentsTask;
import ida.sd.osb.BindingException;
import ida.sd.osb.OntologySQLBinding;
import ida.sd.sql.sqlPA;
/**
 * The <code>IdleTask</code> class implements a task that is the root task of a {@link DBUpdateAgent} instance.  Its roles are to:
 * <ol>
 * <li>Respond to <code>INFORM</code> messages from other <code>DBUpdateAgent</code>s by updating the underlying DB based on
 *      message content.</li>
 * <li>Periodically search for other <code>DBUpdateAgent</code>s and maintain a list of them.</li>
 * <li>Implement the <code>DBUpdateAgent</code>'s ability to send updates.</li>
 * </ol>
 **/
public class IdleTask
    extends Task {

    /** The interval, in milliseconds, between the times when the ontology is searched for db update agents. */
    private final int _dbUpdateAgentSearchInterval;

    /** The list of known db update agents. */
    private Set _dbUpdateAgentIDs = new HashSet();
    /** The binding to an SQL DBMS. */
    private final OntologySQLBinding _osb;
    /** The URL of the underlying DBMS. */
    private final String _our_DB_URL;

    /**
     * Constructs an <code>IdleTask</code> instance.
     * @param dbUpdateAgentSearchInterval The interval, in seconds, between the times when searches for db update agents are initiated.
     * @param osb A specification of an Ontology/SQL binding.
     * @throws BindingException If the URL of the DB specified in <code>osb</code> can't be determined.
     */
    public IdleTask(Integer dbUpdateAgentSearchInterval, OntologySQLBinding osb) throws BindingException {
        _dbUpdateAgentSearchInterval = dbUpdateAgentSearchInterval.intValue()*1000;
        _osb = osb;
        _our_DB_URL = _osb.getDBURL();
    }

    /**
```

```
 * Callback method invoked when FIPA-OS successfully initialises the <code>IdleTask</code>.
 * Starts a task to search for other {@link DBUpdateAgent}s.
 **/
protected void startTask() {
    Diagnostics.fipaEvent("IdleTask started OK.  Starting DBUpdateAgent search task.", this);
    newTask( new DBUpdateAgentSearchTask() );
}

/**
 * Callback method invoked in response to an INFORM request. The request is expected to be a list of queries.
 **/
public void handleInform(Conversation conv) {
    Diagnostics.reportHandle(conv, this);

    ACL acl = conv.getACL(conv.getLatestMessageIndex());
    if ( ! acl.getOntology().equals(DBUpdateAgent.DB_UPDATE_ONTOLOGY) ) {
        Diagnostics.error("Unexpected ontology \"" + acl.getOntology() + "\"", this);
        sendNotUnderstood(acl);
        return;
    }
    if ( ! acl.getLanguage().equals(DBUpdateAgent.DB_UPDATE_LANGUAGE) ) {
        Diagnostics.error("Unexpected language \"" + acl.getLanguage() + "\"", this);
        sendNotUnderstood(acl);
        return;
    }
    List queries = (List)acl.getContentObject();
    Diagnostics.domainEvent("Received " + queries.size() + " queries.", this);
    for ( Iterator qIter = queries.iterator(); qIter.hasNext(); ) {
        String qry = (String)qIter.next();
        try {
            Connection dbConnection = _osb.getConnection();
            synchronized ( dbConnection ) {
                sqlPA.executeUpdateQuery(dbConnection, qry, "DBUA_SP");
            }
        } catch ( SQLException e ) {
            Diagnostics.noteException(e, this);
        }
    }
}
```

```
/**
 * Sends a message to other {@link DBUpdateAgent}s signaling that a change to the DB is in order.  The change is stated as a list of SQL
 * queries.  Starts an {@link InformAgentsTask}.
 * @param queries A list of UPDATE and INSERT queries.
 **/
public void propagateChanges(List queries) {
    newTask( new InformAgentsTask(   new LinkedList(_dbUpdateAgentIDs),
                                     (java.io.Serializable)queries,
                                     DBUpdateAgent.DB_UPDATE_ONTOLOGY,
                                     DBUpdateAgent.DB_UPDATE_LANGUAGE) );
}

/**
 * Callback method invoked when the {@link InformAgentsTask} completes.
 * @param t The completed task.
 **/
public void doneInformAgentsTask(Task t) {
    Diagnostics.fipaEvent("InformAgentsTask done.", this);
}

/**
 * Stops this task.
 * @todo Stop all subtasks.
 **/
public void stop() {
    done();
}

/**
 * Class <code>DBUpdateAgentSearchTask</code> performs a periodic search for {@link DBUpdateAgent}s.  It maintains a set of those
 * agents.  Excluded from the set are:
 * <ol>
 * <li>This instance.</li>
 * <li>Agents whose {@link DBUpdateAgent#DB_URL_PROPERTY} has the same value as the URL of the DB for this instance.</li>
 * </ol>
 **/
public class DBUpdateAgentSearchTask extends Task {

    /**
     * Callback method invoked when the task starts successfully.  Starts a search for <code>DBUpdateAgent</code>s.
     **/
```

```
protected void startTask() {
    Diagnostics.fipaEvent("DBUpdateAgentSearchTask started OK.  Starting search for agents.", this);
    newTask( new AgentSearchTask(DBUpdateAgent.AGENT_TYPE) );
}
/**
 * Callback method invoked with the agent search task completes.  Adds to the set of known {@link DBUpdateAgent}s all agents not
 * previously known, except those that use the same db URL as this instance.  It then starts a <code>WaitTask</code> that, on completion,
 * will start a new search.
 * <p>This method does not account for the possibility of a <code>DBUpdateAgent</code> terminating.
 * @param results An array of {@link DFAgentDescription} instances, each of which is expected to be a <code>DBUpdateAgent</code>
 *            found by the search.
 **/
public void doneAgentSearchTask(Object results) {
    Diagnostics.fipaEvent("AgentSearchTask done.", this);
    DFAgentDescription descs[] = (DFAgentDescription [])results;
    int agentsAdded = 0;
    synchronized ( _dbUpdateAgentIDs ) {
        DESCRIPTION: for ( int i = 0; i < descs.length; i++ ) {
            AgentID id = descs[i].getAgentID();
            if ( id.equals(_owner.getAID()) || _dbUpdateAgentIDs.contains(id) )
                continue;
            Set agentServices = descs[i].getAgentServices();
            for ( Iterator asIter = agentServices.iterator(); asIter.hasNext(); ) {
                String dbURL;
                try {
                    dbURL = dbURLProperty((ServiceDescription)asIter.next());
                } catch ( Exception e ) {
                    Diagnostics.error(id.getName() + ": " + e.getMessage(), this);
                    continue DESCRIPTION;
                }
                if ( dbURL.equals(_our_DB_URL) )
                    continue;
                _dbUpdateAgentIDs.add(id);
                agentsAdded++;
            }
        }
    }
    Diagnostics.domainEvent("Added " + agentsAdded + " DB Update Agent(s) to known set.", this);
```

```
            Diagnostics.fipaEvent("Starting WaitTask.", this);
            newTask( new WaitTask(_dbUpdateAgentSearchInterval) );
        }
        /**
         * Returns the {@link #DB_URL_PROPERTY} from a service description.
         * @param serviceDescription The ServiceDescription from which to obtain the <code>DB_URL_PROPERTY</code>.
         * @return The <code>DB_URL_PROPERTY</code> from a service description.
         * @throws Exception If the service description doesn't have the <code>DB_URL_PROPERTY</code>.
         **/
        private String dbURLProperty(ServiceDescription serviceDescription) throws Exception {
            Set properties = serviceDescription.getProperties();
            if ( properties == null || properties.size() == 0 )
                throw new Exception("Service description has no properties.");
            for ( Iterator pIter = properties.iterator(); pIter.hasNext(); ) {
                PropertyTemplate p = (PropertyTemplate)pIter.next();
                if ( p.getName().equals(DBUpdateAgent.DB_URL_PROPERTY) )
                    return p.getTerm();
            }
            throw new Exception("Service description lacks " + DBUpdateAgent.DB_URL_PROPERTY + " property.");
        }
        /**
         * Callback method invoked when a {@link WaitTask} ends.  Starts a new {@link AgentSearchTask} for agents of type
         * {@link DBUpdateAgent#AGENT_TYPE AGENT_TYPE}.
         * @param t The <code>WaitTask</code> that completed.
         **/
        public void doneWaitTask(Task t) {
            Diagnostics.fipaEvent("WaitTask done.  Starting new search for DBUpdateAgents.", this);
            newTask( new AgentSearchTask(DBUpdateAgent.AGENT_TYPE) );
        }
    }
}
```

5. Classes in Package Threat.Perception

5.1. Class ThreatPerceptionAgent

The ThreatPerceptionAgent implements a FIPA-OS agent that periodically checks for the presence of threats in the battle space using facts in the GH knowledge base. Functionality in this class is the standard agent functionality: register, invoke root task, wait for root task termination, and then shut down.

The class includes method notifyOfIndependentlyIdentifiedThreat(). This method takes a battlefield object as a parameter. Its purpose is to simulate the same actions that would occur if the ThreatPerceptionAgent had identified that object as a threat. It was developed to satisfy a requirement that was later removed from the implemented version. No other classes in the IDA prototype simulation invoke it.

```
package ida.bh.cro.friendlyorg.threat.perception;

import fipaos.agent.task.Task;
import fipaos.ont.fipa.fipaman.ServiceDescription;

import ida.sd.adt.OrganisationObserver;
import ida.sd.Diagnostics;
import ida.sd.FIPA.C4IAgent;
import ida.sd.gh5ontology.GH5KB;
import ida.sd.ontology.*;

/**
 * The <code>ThreatPerceptionAgent</code> is responsible for identifying threats to an organisation.  Threat identification involves searches
 * through the ontology for all known enemies, and checks to see if their recent position indicates movement towards an organisation. (This
 * artificial definition of a threat is introduced for the sake of the example.  The point is that threat identification can be encapsulated in an agent.)
 * When a threat is found, the agent notifies all
 * {@link ida.bh.cro.friendlyorg.threat.response.ThreatResponseAgent ThreatResponseAgent}</code>s.
 **/
public class ThreatPerceptionAgent
    extends C4IAgent {

    /** The root task for this agent. */
    private IdleTask _idleTask;
    /** The string that identifies the agent type for the DF. */
    public final static String AGENT_TYPE = "threat-perception";

    private Instance _us;
    private GH5KB _gh5KB;

    /**
```

```
 * Constructs a <code>ThreatPerceptionAgent</code> that will assume the task of perceiving threats to an organisation.
 * @param agentName The name of the agent.  Must be unique across the platform.
 * @param organisationName The name of the organisation for which threats are to be perceived.
 * @param friendlyOrg The organisation that controls this threat response agent.
 * @param threatSearchInterval The interval, in seconds, between times when threats are to be perceived.
 * @param threatResponseAgentSearchInterval The interval, in seconds, between times when searches for response agents are to occur.
 * @throws UnknownOrgException If organisationName does not refer to the name of an organisation in the ontology.
 **/
public ThreatPerceptionAgent(GH5KB gh5KB,
                             String agentName,
                             String organisationName,
                             OrganisationObserver friendlyOrg,
                             Integer threatSearchInterval,
                             Integer threatResponseAgentSearchInterval) {
    super(agentName);
    _gh5KB = gh5KB;
    _us = friendlyOrg.getOrganisation();
    startPushing();

    // Register with the local AMS (i.e. the platform) as the  first action our agent takes.
    registerWithAMS();

    // Now we can register with the local DF.
    ServiceDescription foService = new ServiceDescription();
    foService.setServiceName(agentName);
    foService.setServiceType(AGENT_TYPE);
    registerWithDF(foService);

    // Make an IdleTask responsible for listening to all incoming messages.
    setListenerTask(_idleTask = new IdleTask(_gh5KB, friendlyOrg, threatSearchInterval, threatResponseAgentSearchInterval));
}

/**
 * Simulates the behavior of a threat to us being perceived.  Creates an {@link InformThreatResponseAgentsTask}, informing all threat
 * response agents known to the {@link IdleTask}.<p>This method may be invoked when a threat is identified by a source other than the
 * <code>ThreatPerceptionAgent</code>, which is periodic.  The intent is to allow for non-periodic identification of threats.
 * @param objItem The battlefield object that is perceived to be a threat.
 * @todo Add a corresponding method to cancel a threat.
 **/
public void notifyOfIndependentlyIdentifiedThreat(Instance objItem) {
```

```
        if ( objItem.instanceOf(_gh5KB.getCls("Organisation")) )
            _tm.newTask( new InformThreatResponseAgentsTask(_gh5KB, _idleTask.getKnownThreatResponseAgents(), objItem, _us, true) );
        else
            Diagnostics.error("Not prepared to handle " + objItem.getDirectType().getName(), this);
    }

    /**
     * Callback method invoked when an {@link InformThreatResponseAgentsTask} completes. Does nothing.
     **/
    public void doneInformThreatResponseAgentsTask(Task t) {
        Diagnostics.fipaEvent("Aperiodic InformThreatResponseAgentsTask completed.", this);
    }

    /**
     * Stops this agent and its tasks. */
    public void stop() {
        shutdown();
    }
}
```

5.2. Class IdleTask

The IdleTask class of the threat perception agent encapsulates the functionality that identifies a threat. In other words, the meaning of "threat" is in this class. It is specifically supplied by the private method searchForTheatsAndNotifyOfAny(), which is part of the subtask OntologySearchTask. A more descriptive name should be given to this method to facilitate its use.

The IdleTask class also maintains the current set of active threat response agents. The searchForTheatsAndNotifyOfAny() method uses this set each time it finds a threat. The actual notification is performed by an instance of the InformThreatResponseAgentsTask class, which is started as a subtask of OntologySearchTask.

```
package ida.bh.cro.friendlyorg.threat.perception;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

// Import the fipa classes
import fipaos.ont.fipa.*;

// Import the classes for DF searches
import fipaos.ont.fipa.fipaman.DFAgentDescription;
```

```
// We will also need to import agent classes
import fipaos.agent.*;
import fipaos.agent.task.Task;
import fipaos.agent.task.WaitTask;

import ida.sd.adt.Notifier;
import ida.sd.adt.OrganisationObserver;
import ida.sd.Diagnostics;
import ida.sd.FIPA.AgentSearchTask;
import ida.sd.FIPA.notification.InformAgentsTask;
import ida.sd.gh5ontology.*;
import ida.sd.ontology.*;

/**
 * The <code>IdleTask</code> class implements a task that is the root task of a ThreatPerceptionAgent instance.  Its roles are to:
 * <ol>
 * <li>Periodically examine the ontology for hostile organisations and, for each such organisation, identify all that are perceived to be
 *     a threat.</li>
 * <li>Notify all ThreatResponseAgents of all threats.</li>
 * </ol>
 * The primary secrets of this class are the algorithms for carrying out the required functionality.
 **/
public class IdleTask
    extends Task {

    private GH5KB _gh5KB;

    /** The organisation that is the focus of concern. */
    private Instance _concernedOrganisation;

    /** The interval, in milliseconds, between the times when the ontology is searched for threats. */
    private int _threatSearchInterval;

    private int _threatResponseAgentSearchInterval;

    /** The set of enemy agents known to present a threat. */
    private Set _threateningOrganisations = new java.util.HashSet();

    /** The list of known threat response agents. */
    private List _knownThreatResponseAgentIDs = new LinkedList();

    private Notifier _notifier;

    /**
```

```
 * Constructs an <code>IdleTask</code> instance.
 * @param organisation The organisation that is the focus of concern; that is, we determine whether something threatens
 *                this particular organisation.
 * @param threatSearchInterval The interval, in seconds, between the times when searches for threats are initiated.
 * @param threatResponseAgentSearchInterval The interval, in seconds, between the times when searches for threat response agents
 *                are initiated.
 */
public IdleTask( GH5KB gh5KB,
                 OrganisationObserver organisation,
                 Integer threatSearchInterval,
                 Integer threatResponseAgentSearchInterval) {
    _gh5KB = gh5KB;
    _concernedOrganisation = organisation.getOrganisation();
    _threatSearchInterval = threatSearchInterval.intValue()*1000;
    _threatResponseAgentSearchInterval = threatResponseAgentSearchInterval.intValue()*1000;
    _notifier = new Notifier(organisation);
}

/**
 * Callback method invoked when FIPA-OS successfully initialises the <code>IdleTask</code>.  Starts two tasks:
 * <ol>
 * <li>A task to search for threat response agents.</li>
 * <li>A task to search the ontology for threats.</li>
 * </ol>
 **/
protected void startTask() {
    Diagnostics.fipaEvent("IdleTask started OK.  Starting tasks.", this);
    newTask( new OntologySearchTask() );
    newTask( new ThreatResponseAgentSearchTask() );
}

/**
 * The <code>OntologySearchTask</code> class periodically searches the ontology for organisations that are threats, or were but are no
 * longer.  Whenever it finds such an organisation, it notifies all known threat response agents.
 **/
public class OntologySearchTask extends Task {

    /** The number of threats about which this task is currently sending notifications. */
    private Integer _notifications = new Integer(0);

    /**
```

```
 * Callback method invoked when the task starts successfully.  Starts a search for threats.
 **/
protected void startTask() {
    searchForThreatsAndNotifyOfAny();
}

/**
 * Searches the knowledge base for anything that looks like a threat. Also searches for things that were threats, but aren't now.
 * Sends a message of each threat and non-threat to every known {@link ida.bh.agent.threat.response.ThreatResponseAgent}.
 **/
private void searchForThreatsAndNotifyOfAny() {
    Cls orgCls = _gh5KB.getCls("Organisation");
    for ( Iterator instIter = _gh5KB.getBattlefieldObjects().iterator(); instIter.hasNext(); ) {
        Instance candidate = (Instance)instIter.next();

        // Only consider organisations.
        Cls candidateType = candidate.getDirectType();
        if ( ! (candidateType.equals(orgCls) || candidateType.hasSuperclass(orgCls)) )
            continue;

        boolean isInPathOfCandidate;
        try {
            isInPathOfCandidate = _gh5KB.inPathOf(candidate, _concernedOrganisation);
        } catch ( NoLocationException e ) {
            continue;
        } catch ( SpeedIndeterminateException e ) {
            continue;
        }

        if ( isInPathOfCandidate ) {
            StringBuffer msg = new StringBuffer()
                .append(_gh5KB.getStringSlot(candidate, "name"))
                .append(" is moving toward ")
                .append(_gh5KB.getStringSlot(_concernedOrganisation, "name"));

            if ( _threateningOrganisations.contains(candidate) ) {
                Diagnostics.domainEvent(msg.append(" (previously detected)").toString(), this);
                continue;
            }
            Diagnostics.domainEvent(msg.append(" (new threat)").toString(), this);
            _threateningOrganisations.add(candidate);
```

```
            _notifier.signal(new ThreatPerception(true, candidate));
            synchronized( _knownThreatResponseAgentIDs ) {
                newTask( new InformThreatResponseAgentsTask(_gh5KB,
                                                            _knownThreatResponseAgentIDs,
                                                            candidate,
                                                            _concernedOrganisation, true)
                );
            }
            incrementNotificationCount();
        }
        else {
            if ( _threateningOrganisations.contains(candidate) ) {
                Diagnostics.domainEvent(_gh5KB.getStringSlot(candidate, "name") + " is no longer a threat", this);
                _threateningOrganisations.remove(candidate);
                _notifier.signal(new ThreatPerception(false, candidate));
                synchronized( _knownThreatResponseAgentIDs ) {
                    newTask( new InformThreatResponseAgentsTask( _gh5KB,
                                                                 _knownThreatResponseAgentIDs,
                                                                 candidate,
                                                                 _concernedOrganisation,
                                                                 false)
                    );
                }
                incrementNotificationCount();
            }
        }
    }

    // If no threats were found, wait for an interval, then search again.
    synchronized( _notifications ) {
        if ( _notifications.intValue() == 0 ) {
            Diagnostics.domainEvent("No threats found.  Starting WaitTask.", this);
            newTask( new WaitTask(_threatSearchInterval) );
        }
    }
}

/**
 * Callback method invoked when a {@link InformThreatResponseAgentsTask} ends.
 * When all have ended, wait for an interval before searching again.
```

```
     **/
    public void doneInformThreatResponseAgentsTask(Task t) {
        Diagnostics.fipaEvent("InformThreatResponseAgentsTask completed.", this);
        synchronized( _notifications ) {
            int notifications;
            notifications = _notifications.intValue() - 1;
            _notifications = new Integer(notifications);
            if ( notifications == 0 ) {
                Diagnostics.domainEvent("All threat response agents informed.  Starting WaitTask.", this);
                newTask( new WaitTask(_threatSearchInterval) );
            }
        }
    }

    /**
     * Callback method invoked when a <code>WaitTask</code> ends.  Starts a new search of the ontology for threats.
     **/
    public void doneWaitTask(Task t) {
        Diagnostics.fipaEvent("WaitTask done.  Starting new search for threats.", this);
        searchForThreatsAndNotifyOfAny();
    }

    /**
     * Safely increments the _notifications count.
     **/
    private void incrementNotificationCount() {
        synchronized ( _notifications ) {
            _notifications = new Integer(_notifications.intValue() + 1);
        }
    }
}

/**
 * The <code>ThreatResponseAgentSearchTask</code> class periodically searches for all threat response agents.  Its responsibility is to
 * update the <code>_knownThreatResponseAgentIDs</code> field.
 **/
public class ThreatResponseAgentSearchTask extends Task {

    protected void startTask() {
        Diagnostics.fipaEvent("ThreatResponseAgentSearchTask started OK.  Starting search for agents.", this);
        newTask( new AgentSearchTask("threat-response") );
```

```
        }
        public void doneAgentSearchTask(Object results) {
            Diagnostics.fipaEvent("AgentSearchTask done.", this);
            DFAgentDescription descs[] = (DFAgentDescription [])results;
            Diagnostics.domainEvent("Found " + descs.length + " threat-response agent(s).", this);
            synchronized ( _knownThreatResponseAgentIDs ) {
                _knownThreatResponseAgentIDs.clear();
                for ( int i = 0; i < descs.length; i++ )
                    _knownThreatResponseAgentIDs.add(descs[i].getAgentID());
            }
            Diagnostics.fipaEvent("Starting WaitTask.", this);
            newTask( new WaitTask(_threatResponseAgentSearchInterval) );
        }
        public void doneWaitTask(Task t) {
            Diagnostics.fipaEvent("WaitTask done.  Starting new search for agents.", this);
            newTask( new AgentSearchTask("threat-response") );
        }
    }
    /**
     * Returns the list of currently known threat response agent IDs.
     * @return The list of currently known threat response agent IDs.
     **/
    public List getKnownThreatResponseAgents() {
        return _knownThreatResponseAgentIDs;
    }
}
```

5.3. Class InformThreatResponseAgentsTask

The InformThreatResponseAgentsTask class carries out the basic function of sending threat messages to a given list of agents. The class's constructor is given a subject/object pair (each is a battlefield object). It formulates an SL0 message from this information, and passes that message to the generic InformAgentsTask class.

```
package ida.bh.cro.friendlyorg.threat.perception;

import java.util.LinkedList;
import java.util.List;

// Import the fipa classes
```

```
import fipaos.ont.fipa.*;

// We will also need to import agent classes
import fipaos.agent.*;
import fipaos.agent.task.Task;

import ida.sd.Diagnostics;
import ida.sd.FIPA.notification.InformAgentsTask;
import ida.sd.FIPA.sl0.SL0;
import ida.sd.FIPA.sl0.SL0Parser;
import ida.sd.gh5ontology.*;
import ida.sd.ontology.Instance;

/**
 * The <code>InformThreatResponseAgentsTask</code> class implements a task whose role is to inform
 * {@link ida.bh.cro.friendlyorg.threat.response.ThreatResponseAgent}s of a threat.  The primary secret of this class is the algorithms for
 * informing threat response agents.
 **/
public class InformThreatResponseAgentsTask
    extends Task {

    private final GH5KB    _gh5KB;
    private final List     _agentIDs;
    private final Instance _subject;
    private final Instance _object;
    private final boolean  _isThreat;

    /**
     * Creates an <code>InformThreatResponseAgentsTask</code> instance that encapsulates a list of agents to inform, and threat information of
     * which to inform them.
     * @param threatResponseAgentIDs The list of {@link ida.bh.cro.friendlyorg.threat.response.ThreatResponseAgent}s
     *         that this task shall inform.
     * @param subject The organisation {@link ida.sd.ontology.Instance} that is (not) a threat.
     * @param object  The organisation {@link ida.sd.ontology.Instance} that is (not) threatened.
     * @param isThreat True if <code>subject</code> threatens
     *         <code>object</code>, false if not.
     **/
    public InformThreatResponseAgentsTask(GH5KB gh5KB, List threatResponseAgentIDs, Instance subject, Instance object, boolean isThreat) {
        _gh5KB    = gh5KB;
        _agentIDs = threatResponseAgentIDs;
        _subject  = subject;
```

```
        _object   = object;
        _isThreat = isThreat;
    }

    /**
     * Callback method invoked by FIPA-OS when this task has initialized successfully.  Formulates the threat statement and creates an
    *  {@link ida.sd.FIPA.notification.InformAgentsTask} to carry out the notification.
     **/
    protected void startTask() {
        List predicateTerms = new LinkedList();
        predicateTerms.add(new SL0.Strng(_gh5KB.getStringSlot(_subject, "name")));
        predicateTerms.add(new SL0.Strng(_gh5KB.getStringSlot(_object,  "name")));
        try {
            // When I implement SL1, I'll change this to (threat ...) or (not (threat ...)).
            SL0.AtomicFormula threatStmt = new SL0.PredicateAtomicFormula(_isThreat ? "threat" : "not-threat", predicateTerms);
            newTask( new InformAgentsTask(_agentIDs, "(" + threatStmt.toString() + ")", "threat", SL0Parser.getLanguage()) );
        } catch ( SL0.InvalidContentException e ) {
            throw new Error(e); // Shouldn't happen, as we construct the stmt identically each time.
        }
    }

    /**
     * Callback method invoked when the {@link ida.sd.FIPA.notification.InformAgentsTask} ends. Ends this task, notifying its parent.
     * @param t The <code>InformAgentsTask</code> that has completed.
     **/
    public void doneInformAgentsTask(Task t) {
        Diagnostics.fipaEvent("InformAgentsTask completed.", this);
        done();
    }
}
```

5.4. Class ThreatPerception

The ThreatPerception class is a simple data abstraction. Its constructor takes an instance (from the knowledge base) of a battlefield object, plus a Boolean statement of whether that object currently poses a threat. Its methods allow read-only access to the constructor's parameters. It is used to communicate threat information among the packages and subpackages of the Friendly Organization components.

```
package ida.bh.cro.friendlyorg.threat.perception;

import ida.sd.ontology.Instance;
```

```
public class ThreatPerception {
    private boolean  _isThreat;
    private Instance _org;

    public ThreatPerception(boolean isThreat, Instance org) {
        _isThreat = isThreat;
        _org = org;
    }

    public boolean getIsThreatened() {
        return _isThreat;
    }

    public Instance getOrganisation() {
        return _org;
    }
}
```

6. Classes in Package Threat.Response

6.1. Class ThreatResponseAgent

The ThreatResponseAgent class implements a FIPA-OS agent that formulates courses of action in response to threats detected by the threat perception agent. No code to shut down the agent cleanly was implemented in the IDA prototype, but should be added in a production-level implementation.

```
package ida.bh.cro.friendlyorg.threat.response;

import fipaos.ont.fipa.fipaman.ServiceDescription;

import ida.sd.adt.OrganisationObserver;
import ida.sd.FIPA.C4IAgent;
import ida.sd.gh5ontology.GH5KB;

/**
 * The <code>ThreatResponseAgent</code> class is responsible for responding to threat information messages.  The primary secrets of the class
 * are the algorithms and data structures for responding to those messages.
 **/
public class ThreatResponseAgent
    extends C4IAgent {

    /** The type of agent we use when registering with the DF. */
```

```
    public final static String AGENT_TYPE = "threat-response";

    /**
     * Constructs a <code>ThreatResponseAgent</code> that will begin listening for detected threats and formulating responses.
     * @param gh5KB The knowledge base the agent is to use.
     * @param agentName The name to use for the agent.  Must be unique across the platform.
     * @param organisationObserver An observer that will be notified of the threat responses.
     **/
    public ThreatResponseAgent(GH5KB gh5KB, String agentName, OrganisationObserver organisationObserver) {
        super(agentName);

        startPushing();

        // Register with the local AMS (i.e. the platform) as the
        // first action our agent takes.
        registerWithAMS();

        // Now we can register with the local DF.
        ServiceDescription foService = new ServiceDescription();
        foService.setServiceName(agentName);
        foService.setServiceType(AGENT_TYPE);
        registerWithDF(foService);

        // Make an IdleTask responsible for listening to all incoming messages.
        setListenerTask(new IdleTask(gh5KB, organisationObserver));
    }
}
```

6.2. Class IdleTask

The IdleTask class, the root task of a threat response agent, listens for messages from a threat perception agent. These messages are sent in SL0, with the INFORM performative. The handleInform() method receives and parses each message to yield SL0 content. It then evaluates the content using an instance of ThreatStmtContext (see Section 6.10) as the context. Callback methods in the IdleTask class (respondToThreat() and respondToThreatCancellation()) link the context with the IdleTask, enabling the IdleTask to notify the FriendlyOrganisation instance that launched the threat response agent.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
```

```
import java.util.List;
import java.util.Map;
import java.util.Set;

// Import the fipa classes
import fipaos.ont.fipa.*;

// Import the classes for DF searches
import fipaos.ont.fipa.fipaman.DFAgentDescription;

// We will also need to import agent classes
import fipaos.agent.*;
import fipaos.agent.task.Task;
import fipaos.agent.task.WaitTask;

import fipaos.agent.conversation.Conversation;

import ida.sd.adt.Notifier;
import ida.sd.adt.OrganisationObserver;
import ida.sd.Diagnostics;
import ida.sd.FIPA.AgentSearchTask;
import ida.sd.FIPA.notification.InformAgentsTask;
import ida.sd.FIPA.sl0.ParseException;
import ida.sd.FIPA.sl0.SL0;
import ida.sd.FIPA.sl0.SL0Parser;
import ida.sd.gh5ontology.GH5KB;
import ida.sd.ontology.Cls;
import ida.sd.ontology.Instance;

/**
 * The <code>IdleTask</code> class implements a task that is the root task of a <code>ThreatResponseAgent</code> instance.  Its role is to
 * listen for threat perceptions.  When it receives one, it formulates a response and notifies the associated organisation of that response.
 **/
public class IdleTask
    extends Task {

    /** An instance of the {@link ida.sd.adt.Notifier Notifier} class.  Used for notification. */
    private final Notifier _notifier;

    /** The organisation associated with the observer that is to be notified in response to threats. */
    private final Instance _organisation;

    private final GH5KB _gh5KB;
```

```
/**
 * Constructs an <code>IdleTask</code> instance.
 * @param organisation The Observer to be notified of our response to threats.
 */
public IdleTask(GH5KB gh5KB, OrganisationObserver organisation) {
    _gh5KB = gh5KB;
    _notifier = new Notifier(organisation);
    _organisation = organisation.getOrganisation();
}

/**
 * Callback method invoked when FIPA-OS successfully initialises the <code>IdleTask</code>.
 **/
protected void startTask() {
    Diagnostics.fipaEvent("IdleTask started OK.", this);
}

/**
 * Callback method invoked when an INFORM request is received.  Receives and interprets the request, which should be threat information.
 * @param conv The {@link fipaos.agent.conversation.Conversation Conversation} containing the INFORM request.
 **/
public void handleInform(Conversation conv) {
    Diagnostics.reportHandle(conv, this);

    ACL acl = conv.getACL(conv.getLatestMessageIndex());
    if ( ! (acl.getOntology().equals("threat") && acl.getLanguage().equals(SL0Parser.getLanguage())) ) {
        Diagnostics.error("Unexpected ACL " + new String(fipaos.parser.acl.string.Parser.deparse(acl)), this);
        sendNotUnderstood(acl);
        return;
    }

    try {
        SL0.Content threatFact = SL0Parser.parseContent(acl.getContentObject().toString());
        Diagnostics.domainEvent("Received threat stmt " + threatFact.toString(), this);
        threatFact.evaluate(new ThreatStmtContext(acl, this));
    } catch ( ParseException e ) {
        Diagnostics.noteException(e, this);
        sendNotUnderstood(acl);
        return;
    } catch ( SL0.EvaluationException e ) {
        Diagnostics.noteException(e, this);
```

```
            sendNotUnderstood(acl);
            return;
        }
    }
    /**
     * Callback method invoked through {@link ThreatStmtContext} in response to a message noting a newly preceived threat (not necessarily to
     * us). We formulate and prioritize responses, then signal our controlling organisation of those responses.
     * @param threateningOrganisationName The name of the organisation that's threatening <code>threatenedOrganisationName</code>.
     * @param threatenedOrganisationName  The name of the organisation that's threatened by <code>threateningOrganisationName</code>.
     **/
    void respondToThreat(String threateningOrganisationName, String threatenedOrganisationName) {
        Diagnostics.domainEvent("Responding to threat of " + threateningOrganisationName + " to " + threatenedOrganisationName, this);

        Cls orgCls = _gh5KB.getCls("Organisation");

        Instance threatened = _gh5KB.getObjectItem(threatenedOrganisationName);
        Instance threat     = _gh5KB.getObjectItem(threateningOrganisationName);

        List candidateCoAs = new LinkedList();
        candidateCoAs.add(new FormulateDEFEND(_gh5KB, _organisation, threatened, threat));
        candidateCoAs.add(new RequestPermissionToWithdraw(_gh5KB, _organisation, threatened, threat));
        candidateCoAs.add(new OrderThreatenedOrgToWithdraw(_gh5KB, _organisation, threatened, threat));
        candidateCoAs.add(new SupportThreatenedOrg(_gh5KB, _organisation, threatened, threat));

        // For each organisation that we command, add appropriate candidate CoAs.
        Collection OIAs = _organisation.getOwnSlotValues(_gh5KB.getSlot("is-the-subject-of-ObjectItemAssociation"));
        for ( Iterator oaIter = OIAs.iterator(); oaIter.hasNext(); ) {
            Instance oia = (Instance)oaIter.next();

            // We want an organisation that we command.
            Instance subordinateOrg = (Instance)oia.getOwnSlotValue(_gh5KB.getSlot("object-is-associated-with-ObjectItemAssociation"));
            if ( ! subordinateOrg.instanceOf(orgCls) )
                continue;
            if ( ! "CMDCTL".equals(_gh5KB.getCodeSlot(oia, "categoryCode")) )
                continue;

            if ( _gh5KB.getObjectItemName(subordinateOrg).equals(threatenedOrganisationName) ||
                _gh5KB.commands(subordinateOrg, threatened) )
                continue;

            Diagnostics.domainEvent("Adding candidate CoAs for subordinate " + _gh5KB.getObjectItemName(subordinateOrg), this);
            candidateCoAs.add(new OrderOtherOrgToSupport(_gh5KB, _organisation, threatened, threat, subordinateOrg));
```

```
            candidateCoAs.add(new OrderThreatenedOrgToWithdraw(_gh5KB, _organisation, threatened, threat, subordinateOrg));
        }
        Diagnostics.domainEvent( "Prioritizing " + candidateCoAs.size() + " CoA(s) and notifying " + _gh5KB.getObjectItemName(_organisation),
                                 this);
        Collections.sort(candidateCoAs);
        String trs = "\"" + threateningOrganisationName + "\" is threatening \"" + threatenedOrganisationName + "\"";
        _notifier.signal(new ThreatResponseNotification(trs, candidateCoAs));
    }
    /**
     * Callback method invoked through {@link ThreatStmtContext} in response to a message noting a cancelled threat.  If we are the organisation
     * that's no longer threatened, we can take appropriate action.
     * @param threateningOrganisationName The name of the organisation that's no longer threatening
     *            <code>threatenedOrganisationName</code>.
     * @param threatenedOrganisationName  The name of the organisation that's no longer threatened by
                <code>threateningOrganisationName</code>.
     * @todo Implement this method!
     **/
    void respondToThreatCancellation(String threateningOrgName, String threatenedOrgName) {
        Diagnostics.domainEvent("Cancelling threat by " + threateningOrgName + " to " + threatenedOrgName, this);
        if ( _gh5KB.getObjectItemName(_organisation).equals(threatenedOrgName) ) {
            // We are no longer threatened by the org.
            // Of course, we may still be threatened by other orgs.
        }
        else {
        }
    }
}
```

6.3. Class CourseOfAction

Class CourseOfAction is the superclass of all classes that denote a possible course of action a threat response agent might formulate. Most of its methods are procedural abstractions of functionality used in subclasses.

A course of action can be evaluated. Evaluation is responsible for determining the appropriateness of a specific course of action in response to a threat. Appropriateness is measured by a floating-point value between 0 and 1, inclusive. 0 means the course of action is inappropriate. 1 means the course of action is appropriate, that is, it has no unforeseen risks.

The class also contains two abstract methods that subclasses must implement. The first, asC4IModel(), must state the course of action in terms of the GH ontology. In other words, it must return a list of related instances that express the course of action to the best of the GH ontology's ability.

The second, description(), states the course of action textually. The asC4IModel() method is used to build information for the knowledge base; the description() method provides information for consumption by a human.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.text.SimpleDateFormat;

import java.util.Calendar;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import ida.sd.gh5ontology.*;
import ida.sd.ontology.*;

/**
 * The <code>CourseOfAction</code> class is the superclass of all courses of action in response to a threat.
 * <p>Evaluating a course of action depends on at least three organisations:
 * <ol>
 * <li>The organisation that is formulating a response to a threat notification (that is, us).</li>
 * <li>The organisation that is determined to be threatened.  This may or may not be us.</li>
 * <li>The organisation that is the threat.</li>
 * </ol>
 * The secret of this class is the data structures for representing course-of- action information.  The secret of subclasses includes the algorithms
 * for implementing the {@link #value value} method.
 * <p>A subclass has the responsibility to:
 * <ol>
 * <li>Define the nature of a specific course of action.</li>
 * <li>{@linkplain #asC4IModel Translate} an instance of itself into a set of {@linkplain ida.sd.ontology.Instance instances} that are a GH5
 *       expression of the course of action the instance denotes.</li>
 * </ol>
 **/
public abstract class CourseOfAction
    implements Comparable {

    protected final GH5KB _gh5KB;
    /** The organisation formulating a response to a threat notification. */
    protected final Instance _us;
```

```
/** An organisation that's been determined as threatened. */
protected final Instance _threatened;
/** The organisation that's threatening the {@linkplain #_threatened threatened organisation}. */
protected final Instance _threat;

/** Holds textual statements of rationale behind estimating a course of action's effectiveness. */
protected List _rationale = new LinkedList();

/** The name of the threatened organisation.  All subclasses tend to use it at some point. */
protected final String _threatenedOrgName;

// The following are some constants useful for stating rationales.
protected final static String NOT_THREATENED = "We are not threatened.";
protected final static String THREATENED     = "We are threatened.";

protected final static String NOT_TASKED = "We are not otherwise tasked.";
protected final static String TASKED     = "We are otherwise tasked.";

protected final static String SUPPLIES_ADEQUATE   = "Our supplies are adequate for defense against the threat.";
protected final static String SUPPLIES_INADEQUATE = "Our supplies are not adequate for defense against the threat.";

/**
 * Constructs a <code>CourseOfAction</code>, storing the arguments in protected fields.  A subclass must extend the constructor to set the
 * {@link #_value} field.
 * @param us The organisation formulating a response to a threat notification.
 * @param threatened An organisation that's been determined as threatened.
 * @param threat The organisation that's a threat.
 **/
public CourseOfAction(GH5KB gh5KB, Instance us, Instance threatened, Instance threat) {
    _gh5KB = gh5KB;
    _us = us;
    _threatened = threatened;
    _threat = threat;
    _threatenedOrgName = _gh5KB.getObjectItemName(_threatened);
}

/** The efficiency of this course of action.  Subclass constructors must set its value. */
protected float _value;

/**
 * Returns a value between 0.0 and 1.0 (inclusive) denoting the level of effectiveness of a course of action in response to a threat.
 **/
public float value() {
```

```
    return _value;
}

/**
 * Returns a textual description of the course of action.
 * @return A textual description of the course of action.
 **/
public abstract String description();

/**
 * Gets statements of rationale that have been added.
 **/
public List getRationale() {
    return _rationale;
}

/**
 * Converts this course of action into a {@link java.util.List} of {@link ida.sd.ontology.Instance}s.  These instances
 * express this course of action in terms of the GH5.
 * <p>An implementation of this method is not responsible for saving  these actions into the underlying DB, if one is present.  The caller has
 * that responsibility.
 * <p>The list must be sorted such that saving it into the underlying DB will not violate integrity constraints.  It is hoped that this
 * restriction will be removed in the future.
 * @return A list of entities and relationships that express this course of action.
 **/
public abstract List asC4IModel();

// Many courses of action need to know about the threatened org's location and the threat's time to reach it.  The following fields
// and methods support such calculations.

private float _timingRatio;
private String _orgName;

/**
 * Finds the times needed for two organisations -- one of which is {@link #_threat _threat} -- to reach a location, and returns the ratio of those
 * times.  Has two side effects:
 * <ol>
 * <li>Stores the timing ratio and the name of <code>org</code>, which is useful in a subsequent use of {@link #timingRatioMessage()}.</li>
 * <li>If an exception is thrown, adds an explanation of why to {@link #_rationale _rationale}.</li>
 * </ol>
 * @param org An {@link ida.sd.ontology.Instance} of Organisation.
 * @param loc An {@link ida.sd.ontology.Instance} of Location.
```

```
 * @return The time <code>org</code> needs to reach <code>loc</code> divide by the time {@link #_threat _threat} needs to reach
 *         <code>loc</code>, or 1.0, whichever is smaller.
 * @throws IncomputableException If either organisation lacks a location or a speed.
 **/
protected float timingRatio(Instance org, Instance loc)
    throws IncomputableException {
    float timeForOrgToReachLoc;
    try {
        timeForOrgToReachLoc = _gh5KB.estimatedTimeToReach(org, loc).floatValue();
    } catch ( NoLocationException e ) {
        _rationale.add("Can't determine location of " + _gh5KB.getObjectItemName(org) + ".");
        throw new IncomputableException(e);
    } catch ( SpeedIndeterminateException e ) {
        _rationale.add("Can't determine speed of " + _gh5KB.getObjectItemName(org) + ".");
        throw new IncomputableException(e);
    }

    float timeForThreatToReachLoc;
    try {
        timeForThreatToReachLoc = _gh5KB.estimatedTimeToReach(_threat, loc).floatValue();
    } catch ( NoLocationException e ) {
        _rationale.add("Can't determine location of " + _gh5KB.getObjectItemName(_threat) + ".");
        throw new IncomputableException(e);
    } catch ( SpeedIndeterminateException e ) {
        _rationale.add("Can't determine speed of " + _gh5KB.getObjectItemName(_threat) + ".");
        throw new IncomputableException(e);
    }

    _timingRatio = timeForOrgToReachLoc/timeForThreatToReachLoc;
    _orgName = _gh5KB.getObjectItemName(org);
    return _timingRatio <= 1.0 ? _timingRatio : 1.0f;
}

/**
 * Like {@link #timingRatio(Instance, Instance) timingRatio}, but also can add a message to {@link #_rationale _rationale}.
 * The message is constructed from {@link #timingRatioMessage()}.
 * @param org See {@link #timingRatio(Instance, Instance) timingRatio}.
 * @param loc See {@link #timingRatio(Instance, Instance) timingRatio}.
 * @param addMessage If true, adds the message to {@link #_rationale}.  If false, does not add the message.
 * @return See {@link #timingRatio(Instance, Instance) timingRatio}.
```

```
     * @throws IncomputableException See {@link #timingRatio(Instance, Instance) timingRatio}.
     **/
    protected float timingRatio(Instance org, Instance loc, boolean addMessage)
        throws IncomputableException {
        float r = timingRatio(org, loc);
        if ( addMessage )
            _rationale.add(timingRatioMessage());
        return r;
    }

    /**
     * Standardizes the rationale message for timing ratios.  Only valid for the ratio computed from the last invocation of {@link #timingRatio}.
     **/
    protected String timingRatioMessage() {
        if ( _timingRatio > 1.0f )
            return _orgName + " can't reached threatened location before " + _gh5KB.getObjectItemName(_threat) + ".";
        else
            return _orgName + " will reach threatened location in " + _timingRatio + "% of time that " + _gh5KB.getObjectItemName(_threat) +
"will.";
    }

    // The following fields and methods support the construction of ontology updates.

    protected final static String ORDER_AT_CC = "ORD";
    protected final static String PLAN_AT_CC = "PLAN";

    protected final static String DEFEND_AT_VPC = "DEFEND";
    protected final static String MOVE_AT_VPC = "MOVE";

    protected final static String ORDER_ATS_CC = "ORD";
    protected final static String PLAN_ATS_CC = "PLAN";

    private final static SimpleDateFormat DATE_FORMAT = new SimpleDateFormat("yyyyMMdd");

    /**
     * Provides a <code>ReportingData {@link ida.sd.ontology.Instance instance} that describes a planned <code>ActionTask</code>.  A
     * procedural abstraction that supports the {@link #asC4IModel()} method.
     * @param day The day to use for the reporting date.  If null, use today.
     * @param time The time to use for the reporting date.  If null, use now.
     **/
    protected Instance updateReportingData(Calendar dateTime) {
        Instance rd = _gh5KB.getCls("ReportingDataAbsoluteTiming").createDirectInstance();
        try {
```

```
        _gh5KB.setCodeSlot(rd, "timingCategoryCode",     "RDABST");
        _gh5KB.setCodeSlot(rd, "reliabilityCode",        "A");
        _gh5KB.setCodeSlot(rd, "sourceTypeCode",         "HUMINT");
        _gh5KB.setCodeSlot(rd, "credibilityCode",        "RPTFCT");
        _gh5KB.setCodeSlot(rd, "categoryCode",           "REP");
        _gh5KB.setCodeSlot(rd, "countingIndicatorCode", "NO");
        _gh5KB.setCodeSlot(rd, "accuracyCode",           "3");
        _gh5KB.setCodeSlot(rd, "entityCategoryCode",     "ACTTST");
        if ( dateTime != null ) {
            _gh5KB.setDateSlot(rd, "reportingDate", dateTime);
            _gh5KB.setTimeSlot(rd, "reportingTime", dateTime);
            _gh5KB.setDateSlot(rd, "effectiveDate", dateTime);
            _gh5KB.setTimeSlot(rd, "effectiveTime", dateTime);
        }
        _gh5KB.setCodeSlot(rd, "effectiveTimePrecisionCode", "SECOND");
        _gh5KB.setIntSlot(rd, "duration", new Integer(30)); // 30 is arbitrary.  Should probably be a parameter.
    } catch ( InvalidCodeException e ) {
        throw new Error(e);
    }
    return rd;
}
/**
 * Provides an OrganisationActionAssociation {@link ida.sd.ontology.Instance}  that can relate {@link #_us} to an action.  The caller may want
 * to set the IntentText field.
 * @param day The day to use for the reporting date.  If null, use today.
 * @param time The time to use for the reporting date.  If null, use now.
 **/
protected Instance updateOAA(Calendar dateTime) {
    Instance oaa = _gh5KB.getCls("OrganisationActionAssociation").createDirectInstance();
    try {
        _gh5KB.setCodeSlot(oaa, "categoryCode", "REQUST");
    } catch ( InvalidCodeException e ) {
        throw new Error(e);
    }
    if ( dateTime != null ) {
        _gh5KB.setDateSlot(oaa, "effectiveDate", dateTime);
        _gh5KB.setTimeSlot(oaa, "effectiveTime", dateTime);
    }
```

```
    return oaa;
}
/**
 * Provides an <code>ActionTaskStatus</code> {@link ida.sd.ontology.Instance} that is suited to a proposed course of action.
 * @param categoryCode The category code for the <code>ActionTaskStatus</code>.
 **/
protected Instance updateActionTaskStatus(String categoryCode) {
    Instance ts = _gh5KB.getCls("ActionTaskStatus").createDirectInstance();
    try {
        _gh5KB.setCodeSlot(ts, "categoryCode", categoryCode);
        _gh5KB.setCodeSlot(ts, "timingCode",   "RQESAT");
        _gh5KB.setCodeSlot(ts, "progressCode", "NST");
    } catch ( InvalidCodeException e ) {
        throw new Error(e);
    }
    _gh5KB.setFloatSlot(ts, "completionFraction", new Double(0.0));
    return ts;
}
protected Instance updateActionTask(String name, String categoryCode, String activityCode, String detailIntro) {
    Instance t = _gh5KB.getCls("ActionTask").createDirectInstance();
    try {
        _gh5KB.setStringSlot(t, "name", "Support");
        _gh5KB.setCodeSlot(t, "categoryCode",       categoryCode);
        _gh5KB.setCodeSlot(t, "activityCode",       activityCode);
        _gh5KB.setCodeSlot(t, "priorityCode",       "1");
        _gh5KB.setCodeSlot(t, "startQualifierCode", "ASAP");

        // Consider formatting this in HTML.
        StringBuffer detail = new StringBuffer(detailIntro);
        for ( int i = 0; i < _rationale.size(); i++ )
            detail.append("\n").append(i).append(") ").append(_rationale.get(i));
        _gh5KB.setStringSlot(t, "detailText", detail.toString());
    } catch ( InvalidCodeException e ) {
        throw new Error(e);
    } catch ( StringLengthException e ) {
        throw new Error(e);
    }

    return t;
```

```
}
protected void establishRelations(Instance targetOrg,
                                  Instance reportingOrg,
                                  Instance reportingData,
                                  Instance task,
                                  Instance taskStatus,
                                  Instance oaa) {
    task.addOwnSlotValue(_gh5KB.getSlot("has-ActionTaskStatus"), taskStatus);
    reportingData.addOwnSlotValue(_gh5KB.getSlot("provides-applicable-information-for-ActionTaskStatus"), taskStatus);
    targetOrg.addOwnSlotValue(_gh5KB.getSlot("has-its-role-specified-through-OrganisationActionAssociation"), oaa);
    task.addOwnSlotValue(_gh5KB.getSlot("is-acted-upon-as-specified-by-OrganisationActionAssociation"), oaa);
    reportingOrg.addOwnSlotValue(_gh5KB.getSlot("is-the-reporting-agent-for-ReportingData"), reportingData);
}
protected List collectedEntities(Instance reportingData, Instance actionTask, Instance actionTaskStatus, Instance oaa) {
    List c4IEntities = new LinkedList();
    c4IEntities.add(reportingData);
    c4IEntities.add(actionTask);
    c4IEntities.add(actionTaskStatus);
    c4IEntities.add(oaa);

    return c4IEntities;
}
/**
 * Signals that a timing ratio is not computable.
 **/
protected class IncomputableException extends Exception {
    /**
     * Constructs an <code>IncomputableException</code>.
     * @param e The reason why the timing ratio can't be computed.  The intent is that it come from the method that couldn't
     *          compute the ratio.
     **/
    public IncomputableException(Exception e) {
        super("Timing ratio incomputable", e);
    }
}
/**
 * Implements the {@link java.lang.Comparable} interface for this class.
```

```
     * @param o Something that had better be a <code>CourseOfAction</code>.
     * @return See {@link java.lang.Comparable}.
     * @see java.lang.Comparable
     **/
    public int compareTo(Object o) {
        // We negate the comparison: we want largest values first.
        float otherValue = ((CourseOfAction)o).value();
        if ( _value < otherValue )
            return 1;
        else if ( _value > otherValue )
            return -1;
        else
            return 0;
    }
}
```

6.4. Class FormulateDefend

The FormulateDefend class is one possible course of action in response to a threat. The functionality in this class evaluates whether holding one's position to defend against a threat is an appropriate course of action.

A value of 0 does not necessarily mean that the threat is overwhelming. 0 is also assigned if the organization evaluating the threat is not the one that's threatened.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.Calendar;
import java.util.List;

import ida.sd.gh5ontology.GH5KB;
import ida.sd.gh5ontology.StringLengthException;
import ida.sd.ontology.Instance;

public class FormulateDEFEND
    extends CourseOfAction {

    /**
     * Constructs a <code>FormulateDEFEND</code> which will evaluate the effectiveness of <code>us</code> formulating a DEFEND action.
     * @param us  The organisation that is evaluating whether defending its current location is an effective course of action.
     * @param threatened An organisation that has been determined to be threatened.
     * @param threat An organisation that threatens <code>threatened</code>.
     **/
```

```
public FormulateDEFEND(GH5KB gh5KB, Instance us, Instance threatened, Instance threat) {
    super(gh5KB, us, threatened, threat);

    if ( ! _us.getName().equals(threatened.getName()) ) {
        _rationale.add(NOT_THREATENED);     // Nothing to defend against.
        _value = 0.0f;
        return;
    }
    _rationale.add(THREATENED);

    if ( _gh5KB.currentTask(_us) != null ) {
        _rationale.add(TASKED);             // We already have a task.
        _value = 0.0f;
        return;
    }
    _rationale.add(NOT_TASKED);

    double supplyAdequacy = _gh5KB.supplyAdequacy(_us);
    if ( supplyAdequacy >= 0.5 ) {
        _rationale.add(SUPPLIES_ADEQUATE);
        _value = (float)supplyAdequacy;
        return;
    }
    else {
        _rationale.add(SUPPLIES_INADEQUATE);
        _value = 0.0f;
        return;
    }
}

public String description() {
    return "We formulate a DEFEND action in response to the threat.";
}

/**
 * Creates the list of C4I entities that reflect a DEFEND task.
 * @return The list of C4I entities that reflect a DEFEND task.
 **/
public List asC4IModel() {
    Calendar now = Calendar.getInstance();

    Instance reportingData = updateReportingData(now);
```

```
        Instance task = updateActionTask("DEFEND", PLAN_AT_CC, DEFEND_AT_VPC,
                                         "We shall defend in response to the threat from " + _threat.getName() + ":");
        _gh5KB.setDateSlot(task, "plannedStartDate", now);

        Instance taskStatus = updateActionTaskStatus(PLAN_ATS_CC);

        Instance oaa = updateOAA(now);
        try {
            _gh5KB.setStringSlot(oaa, "intentText", "We intend to defend our position.");
        } catch ( StringLengthException e ) {
            throw new Error(e);
        }

        establishRelations(_us, _us, reportingData, task, taskStatus, oaa);

        return collectedEntities(reportingData, task, taskStatus, oaa);
    }
}
```

6.5. Class OrderOtherOrgToSupport

The OrderOtherOrgToSupport class is one possible course of action in response to a threat. The functionality in this class evaluates whether issuing an order to another organization to support the threatened organization is appropriate. Assessing appropriateness in this case includes determining whether the organization evaluating the course of action has authority to issue orders to other organizations.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.Calendar;
import java.util.List;

import ida.sd.gh5ontology.GH5KB;
import ida.sd.gh5ontology.StringLengthException;
import ida.sd.ontology.Instance;

public class OrderOtherOrgToSupport
    extends CourseOfAction {

    private final Instance _subordinate;

    /**
     * Constructs an <code>OrderOtherOrgToSupport</code> which will evaluate the effectiveness of us ordering a subordinate organisation to
     * support a threatened organisation.
     * @param us  The organisation that is evaluating whether requesting permission to withdraw is an effective course of action.
```

```
 * @param threatened An organisation that has been determined to be threatened.
 * @param threat An organisation that threatens <code>org</code>.
 * @param subordinate An organisation commanded by us.
 **/
public OrderOtherOrgToSupport(GH5KB gh5KB, Instance us, Instance threatened, Instance threat, Instance subordinate) {
    super(gh5KB, us, threatened, threat);
    _subordinate = subordinate;

    if ( _gh5KB.getObjectItemName(_us).equals(_threatenedOrgName) ) {
        _rationale.add("We are the threatened organisation.");        // Actually, if we command other orgs, we
        _value = 0.0f;                                                // could still have them support us.  Later.
        return;
    }
    _rationale.add("We are not the threatened organisation.");

    if ( ! _gh5KB.commands(_us, _threatened) ) {
        _rationale.add("We do not command the threatened organisation.");
        _value = 0.0f;
        return;
    }
    _rationale.add("We command the threatened organisation.");

    Instance subordinateTask = _gh5KB.currentTask(_subordinate);
    if ( subordinateTask == null ) {
        _rationale.add(_gh5KB.getObjectItemName(_subordinate) + " is not otherwise tasked.");

        _value = (float)_gh5KB.supplyAdequacy(_subordinate);
        if ( _value <= 0.5f ) {
            _rationale.add("Supplies of " + _gh5KB.getObjectItemName(_subordinate) + " are inadequate.");
            return;
        }
        _rationale.add("Supplies of " + _gh5KB.getObjectItemName(_subordinate) + " are adequate.");

        float timingRatio;
        try {
            Instance threatenedLocation = _gh5KB.getCurrentLocation(_threatened);
            if ( threatenedLocation == null ) {
                _rationale.add("Can't determine location of " + _threatenedOrgName);
                _value = 0.0f;
                return;
            }
```

```
            timingRatio = timingRatio(_subordinate, threatenedLocation);
        } catch ( IncomputableException e ) {
            _rationale.add("Can't determine timing ratio for " + _gh5KB.getObjectItemName(_subordinate));
            _value = 0.0f;
            return;
        }
        _rationale.add(timingRatioMessage());
        _value *= 1.0f - timingRatio;
        return;
    }
    else {
        _rationale.add(_gh5KB.getObjectItemName(_subordinate) + " is otherwise tasked.");
        // Ultimately, evaluate the subordinate's task.  For now, assume it's critical.
        _value = 0.0f;
        return;
    }
}

public String description() {
    return "We order " + _gh5KB.getObjectItemName(_subordinate) + " to support " + _threatenedOrgName + ".";
}

/**
 * Creates the list of C4I entities that reflect an order for a subordinate organisation to support a friendly organisation identified as threatened.
 * This is formulated as an order to move.
 * @return The list of C4I entities that reflect the request.
 **/
public List asC4IModel() {
    Calendar now = Calendar.getInstance();

    Instance reportingData    = updateReportingData(now);
    Instance actionTaskStatus = updateActionTaskStatus(ORDER_ATS_CC);
    Instance oaa              = updateOAA(now);

    String subordinateName    = _gh5KB.getObjectItemName(_subordinate);

    try {
        _gh5KB.setStringSlot(oaa, "intentText", "We intend " + subordinateName + " to support " + _threatenedOrgName);
    } catch ( StringLengthException e ) {
        throw new Error(e);
    }
```

```
        Instance actionTask = updateActionTask("Support", ORDER_AT_CC, MOVE_AT_VPC,
                                               "We shall move to the location of " +
                                               subordinateName +
                                               " in response to the threat from " +
                                               _gh5KB.getObjectItemName(_threat) +
                                               " to " +
                                               _threatenedOrgName +
                                               ".");
        _gh5KB.setDateTimeSlots(actionTask, "plannedStartDate", "plannedStartTime", now);

        establishRelations(_subordinate, _us, reportingData, actionTask, actionTaskStatus, oaa);

        return collectedEntities(reportingData, actionTask, actionTaskStatus, oaa);
    }
}
```

6.6. Class OrderThreatenedOrgToWithdraw

The OrderThreatenedOrgToWithdraw class is one possible course of action in response to a threat. The functionality in this class evaluates whether ordering the organization that is threatened to withdraw from its current position is an appropriate course of action. Assessing appropriateness in this case includes determining whether the organization evaluating the course of action has authority to issue orders to other organizations.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.Calendar;
import java.util.List;

import ida.sd.gh5ontology.GH5KB;
import ida.sd.gh5ontology.StringLengthException;
import ida.sd.ontology.Instance;

public class OrderThreatenedOrgToWithdraw
    extends CourseOfAction {

    private final Instance _subordinate;

    /**
     * Constructs an <code>OrderThreatenedOrgToWithdraw</code> that will evaluate the effectiveness of us giving a threatened organisation an
     * order to withdraw.
     * @param us  The organisation that is evaluating whether requesting permission to withdraw is an effective course of action.
     * @param threatened An organisation that has been determined to be threatened.
     * @param threat An organisation that threatens <code>org</code>.
```

```
 * @param subordinate An organisation commanded by us.  We evaluate whether this organisation is available to support
 *                    the threatened organisation.
 **/
public OrderThreatenedOrgToWithdraw(GH5KB gh5KB, Instance us, Instance threatened, Instance threat, Instance subordinate) {
    super(gh5KB, us, threatened, threat);
    _subordinate = subordinate;

    if ( _gh5KB.getObjectItemName(_us).equals(_threatenedOrgName) ) {
        _rationale.add("We are the threatened organisation.");
        _value = 0.0f;
        return;
    }
    _rationale.add("We are not the threatened organisation.");

    if ( ! _gh5KB.commands(_us, _threatened) ) {
        _rationale.add("We do not command the threatened organisation.");
        _value = 0.0f;
        return;
    }
    _rationale.add("We command the threatened organisation.");

    float supplyAdequacy;
    if ( _subordinate != null ) {
        _rationale.add("We can command " + _gh5KB.getObjectItemName(_subordinate));

        float timingRatio;
        try {
            Instance threatenedLocation = _gh5KB.getCurrentLocation(_threatened);
            if ( threatenedLocation == null ) {
                _rationale.add("Can't determine location of " + _threatenedOrgName);
                _value = 0.0f;
                return;
            }
            timingRatio = timingRatio(_subordinate, threatenedLocation);
        } catch ( IncomputableException e ) {
            _value = 0.0f;
            return;
        }

        Instance subordinateTask = _gh5KB.currentTask(_subordinate);
        if ( subordinateTask == null ) {
```

```
            _rationale.add(_gh5KB.getObjectItemName(_subordinate) + " is not otherwise tasked.");
            supplyAdequacy = (float)_gh5KB.supplyAdequacy(_subordinate);
            if ( supplyAdequacy < 0.5f ) {
                _rationale.add(_gh5KB.getObjectItemName(   _subordinate) +
                                                           " cannot support " +
                                                           _threatenedOrgName +
                                                           " (inadequate supplies).");
                _value = 1.0f - supplyAdequacy;
                return;
            }
            _rationale.add(_gh5KB.getObjectItemName(_subordinate) + " has adequate supplies to support " + _threatenedOrgName + ".");
            _rationale.add(timingRatioMessage());
            _value *= timingRatio;
            return;
        }
        else {
            _rationale.add(_gh5KB.getObjectItemName(_subordinate) + " is otherwise tasked.");
            // Ultimately, evaluate the subordinate's task.  For now, assume it's critical.
            _rationale.add(timingRatioMessage());
            _value *= timingRatio;
            return;
        }
    }
    else {
        _rationale.add("No subordinate organisation to command to support " + _threatenedOrgName + ".");
        supplyAdequacy = (float)_gh5KB.supplyAdequacy(_threatened);
        _value = 1.0f - supplyAdequacy;
        if ( supplyAdequacy < 0.6f ) {
            _rationale.add(_threatenedOrgName + " has inadequate supplies to defend itself.");
            return;
        }
        else {
            _rationale.add(_threatenedOrgName + " has adequate supplies to defend itself.");
            return;
        }
    }
}
```

```
/**
 * Constructs an <code>OrderThreatenedOrgToWithdraw</code> which will
 * evaluate the effectiveness of us giving a threatened organisation an order to withdraw.
 * @param us  The organisation that is evaluating whether requesting permission to withdraw is an effective course of action.
 * @param threatened An organisation that has been determined to be threatened.
 * @param threat An organisation that threatens <code>org</code>.
 **/
public OrderThreatenedOrgToWithdraw(GH5KB gh5KB, Instance us, Instance threatened, Instance threat) {
    this(gh5KB, us, threatened, threat, null);
}

public String description() {
    return "We order subordinate organisation " + _threatenedOrgName + " to withdraw.";
}

/**
 * Creates the list of C4I entities that reflect an order to withdraw.  This is formulated as an order to move.
 * <p>N.B. I believe that in the GH5 "withdraw" means to disengage from contact with hostile forces.  Here, the organisation need not
 * actually be in contact with the forces; it must only be aware of them.
 * @return The list of C4I entities that reflect the withdraw order.
 **/
public List asC4IModel() {
    Calendar now = Calendar.getInstance();

    Instance reportingData    = updateReportingData(now);
    Instance actionTaskStatus = updateActionTaskStatus(ORDER_ATS_CC);
    Instance oaa              = updateOAA(now);
    try {
        _gh5KB.setStringSlot(oaa, "intentText", "We intend to move to a safe location.");
    } catch ( StringLengthException e ) {
        throw new Error(e);
    }

    Instance actionTask = updateActionTask( "Withdraw",
                                            ORDER_AT_CC,
                                            MOVE_AT_VPC,
                                            "We shall move to a safe location in response to the threat from " +
                                            _gh5KB.getObjectItemName(_threat) + ":");
    _gh5KB.setDateSlot(actionTask, "plannedStartDate", now);

    establishRelations(_us, _us, reportingData, actionTask, actionTaskStatus, oaa);
```

```
        return collectedEntities(reportingData, actionTask, actionTaskStatus, oaa);
    }
}
```

6.7. Class RequestPermissionToWithdraw

The RequestPermissionToWithdraw class is one possible course of action in response to a threat. The functionality in this class evaluates whether the organization that is threatened should request permission from its superior organization to withdraw from its current position.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.Calendar;
import java.util.List;

import ida.sd.gh5ontology.GH5KB;
import ida.sd.gh5ontology.StringLengthException;
import ida.sd.ontology.Instance;

public class RequestPermissionToWithdraw
    extends CourseOfAction {

    /**
     * Constructs a <code>RequestPermissionToWithdraw</code> which will evaluate the effectiveness of us requesting permission to withdraw.
     * @param us  The organisation that is evaluating whether requesting permission to withdraw is an effective course of action.
     * @param threatened An organisation that has been determined to be threatened.
     * @param threat An organisation that threatens <code>org</code>.
     **/
    public RequestPermissionToWithdraw(GH5KB gh5KB, Instance us, Instance threatened, Instance threat) {
        super(gh5KB, us, threatened, threat);

        if ( ! _gh5KB.getObjectItemName(_us).equals(_threatenedOrgName) ) {
            _rationale.add(NOT_THREATENED);
            _value = 0.0f;
            return;
        }
        _rationale.add(THREATENED);

        float supplyAdequacy = (float)_gh5KB.supplyAdequacy(_us);
        Instance currentTask = _gh5KB.currentTask(_us);
        if ( currentTask != null ) {
            _rationale.add(TASKED);
```

```
    String activityCode = _gh5KB.getCodeSlot(currentTask, "activityCode");
    if ( "DEFEND".equals(activityCode) ) {
        _rationale.add("We are tasked to DEFEND.");
        // Make the value inversely proportional to supply adequacy: If we are tasked to defend, we should reject withdrawal
        // if our supplies are adequate.
        _value = 1.0f - supplyAdequacy;
        if ( supplyAdequacy > 0.6f ) {
            _rationale.add(SUPPLIES_ADEQUATE);
            return;
        }
        else {
            _rationale.add(SUPPLIES_INADEQUATE);
            return;
        }
    }
    else {
        _rationale.add("We have a task that is not DEFEND.");
        _value = supplyAdequacy;
        if ( supplyAdequacy > 0.5f ) {
            _rationale.add(SUPPLIES_ADEQUATE);
            return;
        }
        else {
            _rationale.add(SUPPLIES_INADEQUATE);
            return;
        }
    }
}
else {
    _rationale.add(NOT_TASKED);
    // If we aren't tasked, let's stay and fight if our supplies are high.  So again, make _value inversely proportional to supplies.
    _value = 1.0f - supplyAdequacy;
    if ( supplyAdequacy > 0.8f ) {
        _rationale.add(SUPPLIES_ADEQUATE);
        return;
    }
    else {
```

```
                _rationale.add(SUPPLIES_INADEQUATE);
                return;
            }
        }
    }
    public String description() {
        return "We request permission to withdraw from our location in response to the threat.";
    }
    /**
     * Creates the list of C4I entities that reflect a request to withdraw.  This is formulated as a plan to move.
     * <p>N.B. I believe that in the GH5 "withdraw" means to disengage from contact with hostile forces.  Here, the organisation need not
     * actually be in contact with the forces; it must only be aware of them.
     * @return The list of C4I entities that reflect the withdraw request.
     **/
    public List asC4IModel() {
        Calendar now = Calendar.getInstance();

        Instance reportingData    = updateReportingData(now);
        Instance actionTaskStatus = updateActionTaskStatus(PLAN_ATS_CC);
        Instance oaa              = updateOAA(now);
        try {
            _gh5KB.setStringSlot(oaa, "intentText", "We intend to move to a safe location.");
        } catch ( StringLengthException e ) {
            throw new Error(e);
        }

        Instance actionTask = updateActionTask( "Withdraw",
                                                PLAN_AT_CC,
                                                MOVE_AT_VPC,
                                                "We shall move to a safe location in response to the threat from " +
                                                _gh5KB.getObjectItemName(_threat) + ":");
        _gh5KB.setDateSlot(actionTask, "plannedStartDate", now);

        establishRelations(_us, _us, reportingData, actionTask, actionTaskStatus, oaa);

        return collectedEntities(reportingData, actionTask, actionTaskStatus, oaa);
    }
}
```

6.8. Class SupportThreatenedOrg

The SupportThreatenedOrg class is one possible course of action in response to a threat. The functionality in this class evaluates whether abandoning one's current objectives to provide support to the threatened organization is an appropriate course of action. Assessing appropriateness of this course of action includes determining whether the organization evaluating the course of action has other objectives. In a production system the designers would want to weigh the feasibility of summarily changing those objectives; the IDA prototype currently does not implement this functionality.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.Calendar;
import java.util.List;

import ida.sd.gh5ontology.GH5KB;
import ida.sd.gh5ontology.StringLengthException;
import ida.sd.ontology.Instance;

public class SupportThreatenedOrg
    extends CourseOfAction {

    /**
     * Constructs a <code>SupportThreatenedOrg</code> which will evaluate the effectiveness of us supporting a threatened organisation.
     * @param us  The organisation that is evaluating whether requesting permission to withdraw is an effective course of action.
     * @param threatened An organisation that has been determined to be threatened.
     * @param threat An organisation that threatens <code>org</code>.
     **/
    public SupportThreatenedOrg(GH5KB gh5KB, Instance us, Instance threatened, Instance threat) {
        super(gh5KB, us, threatened, threat);

        if ( _gh5KB.getObjectItemName(_us).equals(_threatenedOrgName) ) {
            _rationale.add("We are the threatened organisation and so can't support ourselves.");
            _value = 0.0f;
            return;
        }
        _rationale.add("We are not the threatened organisation.");

        if ( _gh5KB.commands(_us, _threatened) ) {
            _rationale.add("We command the threatened organisation and can't provide direct support.");
            _value = 0.0f;
            return;
        }
        _rationale.add("We don't command the threatened organisation.");
```

```
    Instance currentTask = _gh5KB.currentTask(_us);
    if ( currentTask != null ) {
        _rationale.add("We are otherwise tasked.");
        _value = 0.0f;
        return;
    }
    _rationale.add("We are not otherwise tasked.");

    double supplyAdequacy = _gh5KB.supplyAdequacy(_us);
    if ( supplyAdequacy < 0.8 ) {
        _rationale.add("Our supplies are inadequate.");
        _value = (float)supplyAdequacy;
        return;
    }
    _rationale.add("Our supplies are adequate.");

    Instance location = _gh5KB.getCurrentLocation(_threatened);
    if ( location == null ) {
        _rationale.add("Location of " + _threatenedOrgName + " can't be determined.");
        _value = 0.0f;
        return;
    }

    try {
        _value *= 1.0f - timingRatio(_us, location, true);
        return;
    } catch ( IncomputableException e ) {
        _value = 0.0f;
        return;
    }
}

public String description() {
    return "We provide direct support to " + _threatenedOrgName;
}

/**
 * Creates the list of C4I entities that reflect a request to support a friendly organisation identified as threatened.  This is formulated as
 * a plan to move.
 * @return The list of C4I entities that reflect the request.
 **/
```

```
    public List asC4IModel() {
        Calendar now = Calendar.getInstance();

        Instance reportingData    = updateReportingData(now);
        Instance actionTaskStatus = updateActionTaskStatus(PLAN_ATS_CC);
        Instance oaa              = updateOAA(now);
        try {
            _gh5KB.setStringSlot(oaa, "intentText", "We intend to support " + _threatenedOrgName);
        } catch ( StringLengthException e ) {
            throw new Error(e);
        }

        Instance actionTask = updateActionTask( "Support",
                                                PLAN_AT_CC,
                                                MOVE_AT_VPC,
                                                "We shall request to support " +
                                                _threatenedOrgName +
                                                " in response to the threat from " +
                                                _gh5KB.getObjectItemName(_threat) +
                                                ":");

        establishRelations(_us, _us, reportingData, actionTask, actionTaskStatus, oaa);

        return collectedEntities(reportingData, actionTask, actionTaskStatus, oaa);
    }
}
```

6.9. Class ThreatResponseNotification

The ThreatResponseNotification class is a data abstraction for passing a set of courses of action between classes.

```
package ida.bh.cro.friendlyorg.threat.response;

import java.util.List;

/**
 * The <code>ThreatResponseNotification</code> class encapsulates candidate courses of action in response to a threat.  The primary secret of
 * this class is the algorithms and data structures used to package the CoAs.
 **/
public class ThreatResponseNotification {
    private CourseOfAction[] _prioritizedCoursesOfAction;

    private String _threatStatement;
```

```
/**
 * Constructs a <code>ThreatResponseNotification</code> based on a prioritized list of
 * {@link ida.bh.cro.friendlyorg.threat.response.CourseOfAction CourseOfAction} instances.
 * @param threatStatement A textual statement of the threat.
 * @param prioritizedCoursesOfAction Course of action, from highest priority to lowest priority.
 * @todo The threat statement should not be textual.  Expect it to change to a data structure of some sort.
 **/
public ThreatResponseNotification(String threatStatement, List prioritizedCoursesOfAction) {
    _threatStatement = threatStatement;
    int nCoAs = prioritizedCoursesOfAction.size();
    _prioritizedCoursesOfAction = new CourseOfAction[nCoAs];
    for ( int i = 0; i < nCoAs; i++ )
        _prioritizedCoursesOfAction[i] = (CourseOfAction)prioritizedCoursesOfAction.get(i);
}

/**
 * Gets a textual statement of the threat associated with this threat response notification.
 * @return A textual statement of the threat.
 **/
public String threatStatement() {
    return _threatStatement;
}

/**
 * Determines if this <code>ThreatResponseNotification</code> has a possible course of action.
 * @return True if at least one course of action is possible (i.e., has a value &gt;0), false if not.
 **/
public boolean hasPossibleCoA() {
    return possibleCoASize() > 0;
}

/**
 * Gets the number of courses of action associated with this instance.
 * @return The number of courses of action associated with this instance.
 **/
public int size() {
    return _prioritizedCoursesOfAction.length;
}

/**
 * Gets the number of <em>possible</em> courses of action associated with this instance.
```

```
 * @return The number of possible courses of action associated with this instance.
 **/
public int possibleCoASize() {
    for ( int i = _prioritizedCoursesOfAction.length - 1; i >= 0; i-- ) {
        if ( _prioritizedCoursesOfAction[i].value() > 0.0 )
            return i + 1;
    }
    return 0;
}

/**
 * Gets the <i>i</i>'th {@link CourseOfAction} associated with this instance.  The courses of action are in priority order, from highest to lowest.
 * @return The <i>i</i>'th {@link CourseOfAction} associated with this instance.
 **/
public CourseOfAction get(int i)
    throws IndexOutOfBoundsException {
    if ( i < 0 || i >= _prioritizedCoursesOfAction.length )
        throw new IndexOutOfBoundsException(new Integer(i).toString());
    return _prioritizedCoursesOfAction[i];
}

/**
 * Overrides the {@link java.lang.Object#equals} method to base equality on comparison of the {@linkplain #threatStatement threat statement}.
 * @param o The object to be tested for equality.  Should be a <code>ThreatResponseNotification</code>.
 * @return <code>False</code> if <code>o</code> is not a <code>ThreatResponseNotification</code>;
 *         otherwise, the result of testing equality of this instance's threat statement with that of <code>o</code>'s.
 **/
public boolean equals(Object o) {
    ThreatResponseNotification trn;
    try {
        trn = (ThreatResponseNotification)o;
    } catch ( java.lang.ClassCastException e ) {
        // o wasn't a ThreatResponseNotification.
        return false;
    }
    try {
        return _threatStatement.equals(trn.threatStatement());
    } catch ( java.lang.NullPointerException e ) {
        // _threatStatement was null.
        return trn.threatStatement() == null;
```

```
        }
    }
    /**
     * Overrides {@link java.lang.Object#hashCode} consistent with the overridden definition of {@linkplain #equals equality} for this class.
     * @return The hash code for the {@linkplain #threatStatement threat statement}.
     **/
    public int hashCode() {
        return _threatStatement.hashCode();
    }
}
```

6.10. Class ThreatStmtContext

The ThreatStmtContext class implements a context in which SL0 content can be evaluated. The ontology used to send threat statements has two predicates, threat and not-threat. This class provides methods that implement each.

```
package ida.bh.cro.friendlyorg.threat.response;

import fipaos.ont.fipa.ACL;
import fipaos.ont.fipa.FIPACONSTANTS;

import ida.sd.FIPA.sl0.SL0;

/**
 * Class <code>ThreatStmtContext</code> implements a {@link ida.sd.FIPA.sl0.SL0.Context Context} that provides two predicates:
 * <code>(threat&amp;nbsp;org-1&amp;nbsp;org-2)</code> and <code>(not-threat&amp;nbsp;org-1&amp;nbsp;org-2)</code>.
 * Using SL1 would be preferrable. Oh well, as soon as it's implemented....
 **/
public class ThreatStmtContext
    implements SL0.Context {

    /** The ACL for this instance. */
    private ACL      _acl;
    /** The IdleTask instance that creates this instance. */
    private IdleTask _task;

    /**
     * Constructs a <code>ThreatStmtContext</code>.
     * @param acl An ACL for this instance.
     * @param task The {@link IdleTask} instance that creates this instance.
     **/
```

```
public ThreatStmtContext(ACL acl, IdleTask task) {
    _acl  = acl;
    _task = task;
}

/**
 * Implements the <code>threat</code> predicate, which states that one organisation is a threat to another organisation.
 * @param threatener The name of the organisation that's a threat.
 * @param threatened The name of the organisation that's threatened by <code>threatener</code>.
 * @return True.
 **/
public Boolean threat(String threatener, String threatened) {
    if ( _acl.getPerformative().equalsIgnoreCase(FIPACONSTANTS.INFORM) )
        _task.respondToThreat(threatener, threatened);

    return new Boolean(true);
}

/**
 * Implements the <code>not-threat</code> predicate, which states that one organisation (the subject) is <em>not</em> a threat to another
 * organisation (the object).
 * @param subjectOrg The subject organisation's name.
 * @param objectOrg The object organisation's name.
 * @return True.
 **/
public Boolean notThreat(String subjectOrg, String objectOrg) {
    if ( _acl.getPerformative().equalsIgnoreCase(FIPACONSTANTS.INFORM) )
        _task.respondToThreatCancellation(subjectOrg, objectOrg);

    return new Boolean(true);
}

// The methods to implement the context are trivial, since this context has no functions and doesn't expect a result.
private final static String eMsg = "Unimplemented context facet";

/**
 * Throws an {@link java.lang.Error} indicating this method shouldn't be invoked.
 * @throws Error If this method is invoked.
 **/
public Boolean evaluateResult(Object term, Object equivalentTerm) {
    throw new Error(eMsg);
}
```

```
    /**
     * Throws an {@link java.lang.Error} indicating this method shouldn't be invoked.
     * @throws Error If this method is invoked.
     **/
    public String[] getFunctionParams(String functionName) {
        throw new Error(eMsg);
    }
    /**
     * Throws an {@link java.lang.Error} indicating this method shouldn't be invoked.
     * @throws Error If this method is invoked.
     **/
    public Boolean truth(Boolean truth) {
        throw new Error(eMsg);
    }
}
```

Annex D: GH5 Ontology Implementation

This annex presents the implementation of the GH5 Ontology component of the prototype. The GH5 Ontology component encapsulates many important design decisions regarding how other components access a GH knowledge base.

Annex B gives a high-level view of the GH5 Ontology component's design. This annex complements Annex B with detailed design and implementation information. Annex B breaks components down into packages. This annex continues with classes and their contents.

1. Package and Class Structure

The GH5 Ontology consists of a single package named gh5ontology. Figure 1 shows the outer classes in the package. The package has three kinds of classes:

1. Classes in the top group provide the package's behavior.
2. Classes in the lower left group are the exceptions thrown by methods in the package.
3. Classes in the lower right are the errors thrown by methods in the package. It is not common practice to throw errors rather than exceptions in Java programs. The IDA prototype simulation does so in response to mistakes in external definitions, in particular those in the GH ontology. If one of these errors is thrown, it indicates that the ontology needs to be fixed before the prototype can execute correctly. The expectation, of course, is that these errors will never be thrown.

The five classes in the top group may be further categorized. Class GH5KB is the main class of the package, providing most of the functionality. The other classes encapsulate some portion of ontology behavior:

- Class DateTime encapsulates how the GH Ontology represents dates and times, translating betweens these representations and Java standard classes.
- Class NumericExpression encapsulates the Numeric-Expression ontology.
- Class ReportingData encapsulates the GH Ontology notion of the ReportingData class,

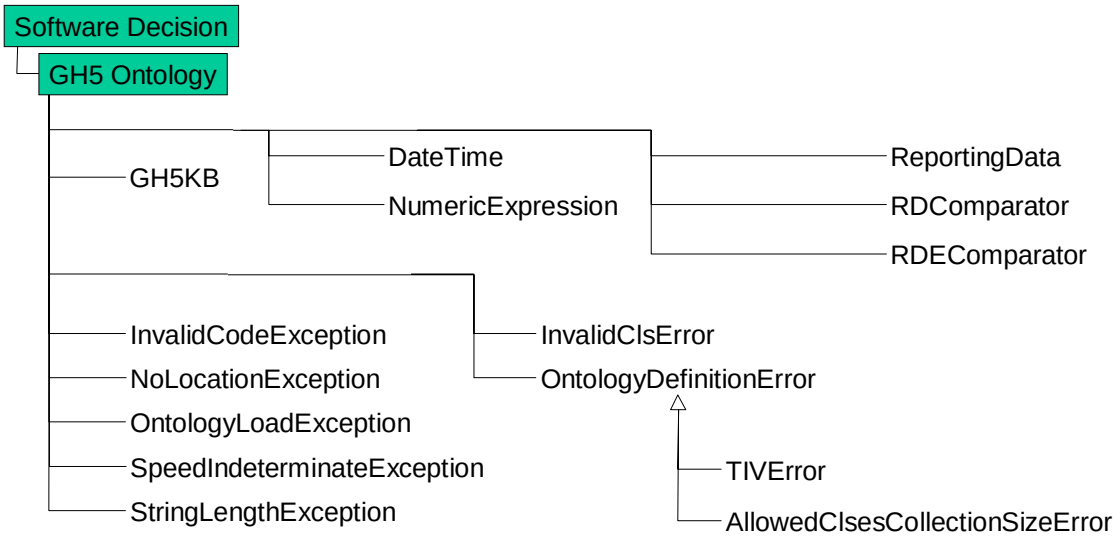

Figure 1. Friendly Organization Component Package and Class Structure

particularly with regard to ordering a set of instances of reporting data by some criterion. The class is supported by the two classes RDComparator and RDEComparator, which implement comparisons of two instances of ReportingData according to reporting date/time and effective date/time, respectively.

1. Documentation Conventions

The code for the prototype is documented according to the conventions for the javadoc tool. These conventions help to standardize the content and format of API documentation. See http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/ for more information.

Javadoc translates comments of the form /* ... */ into input to an HTML interpreter. These comments often contain HTML elements. The translated version, viewed in a browser, is generally easier to comprehend. The HTML versions are included on the CD that accompanies this document.

The IDA prototype simulation adds one non-standard javadoc tag: "@todo". This tag indicates an aspect of functionality needed in a production system but beyond the scope of the current prototype.

2. Class GH5KB

The GH5KB class is the main class of the gh5ontology package. The FriendlyOrg class (Annex C) creates an instance of GH5KB. It uses this instance to access a knowledge base instead of directly accessing the prototype's underlying model of a knowledge base in package ida.sd.ontology (Annex B, Section 3.6.2) or the Protégé-2000 implementation beneath that model. Indeed, the instance of GH5KB *is* the knowledge base so far as a FriendlyOrg is concerned.

Class GH5KB implements interface ida.sd.ontology.KnowledgeBase and so provides all the operations expected of a knowledge base in the context of the IDA prototype. However, class GH5KB has some knowledge of the structure of a GH ontology, in particular how slot values are represented. Instance methods of the class translate Java types (e.g., int, float) and classes (e.g., String, Calendar) into values appropriate to the slots, namely the GH ontology's type-instance-value format (Annex A). Note that some facets of slots – their names and cardinality, for instance – are *not* encapsulated by the class. A friendly organization must understand the ontology to a certain degree.

Class GH5KB also contains some methods that encapsulate procedural abstractions. Examples include inPathOf(), which determines whether a moving object is on a linear path that intersects the area near a specified point, and isHostile(), which tests whether a battlefield object is hostile to a friendly organization. Including these methods in the class standardizes the meaning of concepts such as hostility. Arguably, these methods belong in distinct classes within the gh5ontology package, as the concepts they encapsulate are distinct.

```
package ida.sd.gh5ontology;

import java.text.SimpleDateFormat;

import java.util.Calendar;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;

import ida.sd.Earth;
import ida.sd.ontology.*;
import ida.sd.ontology.impl.protege.ProtegeProject;
```

```
import ida.sd.osb.BindingException;
import ida.sd.osb.BindingPA;
import ida.sd.Track;

import ida.sd.Diagnostics;

/**
 * Class <code>GH5KB</code> implements a {@link ida.sd.ontology.KnowledgeBase} specific to the GH5.  It extends the canonical set of
 * methods for accessing a knowledge base with methods specific to the structure of the GH5, as well as useful procedural abstractions.
 * <p>The primary secrets of this class are the representation of slot values in the GH5, and the algorithms used to implement access methods.
 **/
public class GH5KB
    implements KnowledgeBase {
    private final static int ORG_TRACK_SIZE = 50;
    private final static double OBJ_HALF_WIDTH = (ORG_TRACK_SIZE/2.0)/Earth.METERS_PER_DEGREE;
    /** The underlying {@link ida.sd.ontology.KnowledgeBase}. */
    private KnowledgeBase _kb;

    // The following fields are ontology elements that are accessed in many methods, and are therefore initialized and maintained to prevent having
    // to look them up repeatedly.
    private final Cls _numericExpressionCls;
    private final Cls _fixedLengthStringCls;

    private final Cls _objItemCls;
    private final Cls _orgCls;
    private final Cls _mobilityCapabilityCls;

    private final Slot _nameSlot;
    private final Slot _objItemLocSlot;
    private final Slot _objItemLocRDRefSlot;

    private final Slot _tivSlot;
    private final Slot _eelSlot;

    private final Slot _stringLengthSlot;

    private final NumericExpression _numericExpression;

    /**
     * Constructs a GH5KB that is a wrapper around a {@link ida.sd.ontology.KnowledgeBase} that models the GH5.  No formal check is made as
     * to whether the knowledge base models the GH5, but if it doesn't attempts to use the instance will quickly yield exceptions.
     * @param kb A <code>KnowledgeBase</code> that models the GH5.
     **/
    public GH5KB(KnowledgeBase kb) {
```

```
    _kb = kb;
    _numericExpressionCls   = _kb.getCls("Numeric-Expression");
    _fixedLengthStringCls   = _kb.getCls("String-Type-Fixed");
    _mobilityCapabilityCls  = kb.getCls("MobilityCapability");
    _objItemCls             = kb.getCls("ObjectItem");
    _orgCls                 = kb.getCls("Organisation");
    _tivSlot                = kb.getSlot("type-instance-value");
    _nameSlot               = kb.getSlot("name");
    _objItemLocRDRefSlot    = kb.getSlot("obj_item_loc-is-referenced-to-ReportingData");
    _objItemLocSlot         = kb.getSlot("is-geometrically-defined-through-ObjectItemLocation");
    _eelSlot                = _kb.getSlot("enumerated-element-label");
    _stringLengthSlot       = _kb.getSlot("textual-class-string-length");
    _numericExpression = new NumericExpression(kb);
}
/**
 * Returns the named {@link ida.sd.ontology.Cls}.
 * @param name The name of a <code>Cls</code> in the GH5 ontology.
 * @return The named {@link ida.sd.ontology.Cls}.
 **/
public Cls getCls(String name) {
    return _kb.getCls(name);
}
/**
 * Returns the named {@link ida.sd.ontology.Slot}.
 * @param name The name of a <code>Slot</code> in the GH5 ontology.
 * @return The named {@link ida.sd.ontology.Slot}.
 **/
public Slot getSlot(String name) {
    return _kb.getSlot(name);
}
/**
 * Returns the named {@link ida.sd.ontology.Instance}.
 * @param name The name of an <code>Instance</code> in the GH5 knowledge base.
 * @return The named {@link ida.sd.ontology.Instance}.
 **/
```

```
public Instance getInstance(String name) {
    return _kb.getInstance(name);
}

/**
 * Deletes an {@link ida.sd.ontology.Instance} from the knowledge base.
 * @param inst The instance to delete from the knowledge base.
 **/
public void deleteInstance(Instance inst) {
    _kb.deleteInstance(inst);
}

/**
 * Returns an {@link ida.sd.ontology.Instance} of an <code>ObjectItem</code> as identified by its <code>name</code>
 * {@link ida.sd.ontology.Slot}.  If  multiple <code>ObjectItem</code>s have the same name, this method randomly chooses one.
 * @param name A string.
 * @return An {@link ida.sd.ontology.Instance} of an <code>ObjectItem</code> whose name slot has the value <code>name</code>,
 *       or <code>null</code> if no such <code>Instance</code> exists.
 **/
public Instance getObjectItem(String name) {
    return getBattlefieldObject(_objItemCls, name);
}

/**
 * Returns an {@link ida.sd.ontology.Instance} of a {@link ida.sd.ontology.Cls} (including its subclasses) with a <code>name</code>
 * {@link ida.sd.ontology.Slot} where the value of the <code>name</code> slot matches a given string.  If multiple <code>ObjectItem</code>s
 * have the same name, this method randomly chooses one.
 * @param cls A {@link ida.sd.ontology.Cls} with a <code>name</code> slot.
 * @param name A string.
 * @return An {@link ida.sd.ontology.Instance} of <code>cls</code> whose name slot has the value <code>name</code>,
 *       or <code>null</code> if no such <code>Instance</code> exists.
 **/
public Instance getBattlefieldObject(Cls cls, String name) {
    if ( ! (cls.equals(_objItemCls) || cls.hasSuperclass(_objItemCls) ) )
        throw new InvalidClsError(_objItemCls, cls);

    for ( Iterator oiIter = cls.getInstances().iterator(); oiIter.hasNext(); ) {
        Instance oiInst = (Instance)oiIter.next();
        if ( name.equals(getStringSlot(oiInst, _nameSlot)) )
            return oiInst;
    }
```

```
    return null;
}
/**
 * Returns the value of the <code>name</code> {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance} of
 * <code>ObjectItem</code> or one of its subclasses.  Throws an error if the given instance isn't of the correct <code>cls</code>.
 * @param objItem A direct or indirect instance of <code>ObjectItem</code>.
 * @return The value of the <code>name</code> slot of <code>objItem</code>.
 **/
public String getObjectItemName(Instance objItem) {
    if ( ! objItem.instanceOf(_objItemCls) )
        throw new InvalidClsError(_objItemCls, objItem.getDirectType());
    return getStringSlot(objItem, "name");
}
/**
 * Returns all direct and indirect instances of the <code>ObjectItem</code> {@linkplain ida.sd.ontology.Cls class}.
 * @return All direct and indirect instances of the <code>ObjectItem</code> {@linkplain ida.sd.ontology.Cls class}.
 **/
public Collection getBattlefieldObjects() {
    return _objItemCls.getInstances();
}
/**
 * Returns the current (i.e., most recent according to <code>ReportingData</code>) location of an <code>ObjectItem</code>
 * {@linkplain ida.sd.ontology.Instance instance}.
 * @param objectItem An instance of the <code>ObjectItem</code> class.
 * @return The {@linkplain ida.sd.ontology.Instance instance} of <code>Location</code> associated with <code>objectItem</code> that has
 *       the most recent reporting date, or <code>null</code> if no instances of <code>Location</code> are associated with
 *       <code>ObjectItem</code>.
 **/
public Instance getCurrentLocation(Instance objectItem) {
    if ( ! objectItem.instanceOf(_objItemCls) )
        throw new InvalidClsError(_objItemCls, objectItem.getDirectType());
    Instance mostRecentObjItemLoc = ReportingData.getMostRecent(this,
                                                                objectItem.getOwnSlotValues(_objItemLocSlot),
                                                                _objItemLocRDRefSlot);
    if ( mostRecentObjItemLoc == null )
        return null;
```

```
    return (Instance)mostRecentObjItemLoc.getOwnSlotValue(_kb.getSlot("obj_item_loc-is-associated-with-Location"));
}

/**
 * Returns true if one battlefield object {@link ida.sd.ontology.Instance instance} is moving such that its track intersects the location of another
 * {@link ida.sd.ontology.Instance instance}.  The second instance is assumed not to be moving.
 * <p>Currently, each instance is assumed to be ORG_TRACK_SIZE metres wide. The moving object's tract intersects the static object's track
 * if there's any overlap.
 * @param movingObj The {@link ida.sd.ontology.Instance} that's moving.
 * @param staticObj The {@link ida.sd.ontology.Instance} at rest.
 **/
public boolean inPathOf(Instance movingObj, Instance staticObj)
    throws NoLocationException, SpeedIndeterminateException {
    if ( ! movingObj.instanceOf(_objItemCls) )
        throw new InvalidClsError(_objItemCls, movingObj.getDirectType());
    if ( ! staticObj.instanceOf(_objItemCls) )
        throw new InvalidClsError(_objItemCls, staticObj.getDirectType());

    List objItemLocations;
    int size;
    objItemLocations = new LinkedList(movingObj.getOwnSlotValues(_objItemLocSlot));
    // If movingObj doesn't have any location information, we can't determine its path.
    if ( (size = objItemLocations.size()) == 0 )
        throw new NoLocationException(getObjectItemName(movingObj));

    Instance mostRecentObjItemLoc = ReportingData.getMostRecent(this, objItemLocations, _objItemLocRDRefSlot);

    double latCur;
    double lonCur;
    Slot   locSlot = _kb.getSlot("obj_item_loc-is-associated-with-Location");

    // Get the current location of the staticObj.
    Instance staticObjectItemLoc = getCurrentLocation(staticObj);
    double staticObjectItemLatitude = getFloatSlot(staticObjectItemLoc, "latitudeCoordinate").doubleValue();
    double staticObjectItemLongitude = getFloatSlot(staticObjectItemLoc, "longitudeCoordinate").doubleValue();

    // First, see if the ObjectItemLocation has a bearing angle. If so, use it.
    Instance bearingAngleInst = (Instance)mostRecentObjItemLoc.getOwnSlotValue(_kb.getSlot("bearingAngle"));
    if ( bearingAngleInst != null ) {
        Instance numericExpr = (Instance)bearingAngleInst.getOwnSlotValue(_tivSlot);
        double bearingAngle = ((Float)_numericExpression.evaluate(numericExpr)).doubleValue();
        Instance movingLoc = (Instance)mostRecentObjItemLoc.getOwnSlotValue(locSlot);
```

```
        latCur = getFloatSlot(movingLoc, "latitudeCoordinate").doubleValue();
        lonCur = getFloatSlot(movingLoc, "longitudeCoordinate").doubleValue();
        return Track.overlaps(lonCur, latCur, bearingAngle, staticObjectItemLongitude, staticObjectItemLatitude, OBJ_HALF_WIDTH);
    }
    else {
        // Use the last two objItemLocations, and extrapolate.
        if ( size == 1 )
            throw new SpeedIndeterminateException(getObjectItemName(movingObj));          // Only 1 location.
        ReportingData.sortByReportingTime(this, objItemLocations, _objItemLocRDRefSlot);  // Sorts on reportingDate/reportingTime;
                                                                                          // arguably, effectiveDate/effectiveTime is
                                                                                          // what's wanted.

        // Get the last two points in the list, and extrapolate.
        double latPrev, lonPrev;
        Instance objItemLoc, loc;

        loc = (Instance)((Instance)objItemLocations.get(size - 2)).getOwnSlotValue(locSlot);
        latPrev = getFloatSlot(loc, "latitudeCoordinate").doubleValue();
        lonPrev = getFloatSlot(loc, "longitudeCoordinate").doubleValue();

        loc = (Instance)((Instance)objItemLocations.get(size - 1)).getOwnSlotValue(locSlot);
        latCur = getFloatSlot(loc, "latitudeCoordinate").doubleValue();
        lonCur = getFloatSlot(loc, "longitudeCoordinate").doubleValue();

        double distPrev = Earth.distance(latPrev, lonPrev, staticObjectItemLatitude, staticObjectItemLongitude);
        double distCur = Earth.distance(latCur, lonCur, staticObjectItemLatitude, staticObjectItemLongitude);

        if ( distPrev <= distCur )      // Is the moving object coming closer?
            return false;               // If not, return false.

        if ( lonCur != lonPrev ) {
            double slope = (latCur - latPrev)/(lonCur - lonPrev);
            double y_intercept = latPrev - slope*lonPrev;
            return Track.overlaps(slope, y_intercept, staticObjectItemLongitude, staticObjectItemLatitude, OBJ_HALF_WIDTH);
        }
        else {
            return Track.overlaps(lonCur, latCur, 0.0, staticObjectItemLongitude, staticObjectItemLatitude, OBJ_HALF_WIDTH);
        }
    }
}
/**
```

```
 * Determines whether an {@linkplain ida.sd.ontology.Instance instance} of an <code>ObjectItem</code> is hostile.  Hostility is
 * specified by the hostility code associated with the <code>ObjectItem</code>'s most recent <code>ObjectItemStatus</code> instance.
 * If the <code>ObjectItem</code> has no status, it is assumed to be hostile.
 * @param oi The <code>ObjectItem</code> whose hostility is to be determined.
 * @return True if the <code>ObjectItem</code> is hostile, false if not.
 **/
public boolean isHostile(Instance objItem) {
    if ( ! objItem.instanceOf(_objItemCls) )
        throw new InvalidClsError(_objItemCls, objItem.getDirectType());

    Instance mostRecentStatus = ReportingData.getMostRecent( this,
                                                             objItem.getOwnSlotValues(_kb.getSlot("has-ObjectItemStatus")),
                                                             _kb.getSlot("obj-item-status-is-referenced-to-ReportingData"));
    if ( mostRecentStatus == null )
        return true;

    String hostilityCode = getCodeSlot(mostRecentStatus, "hostilityCode");
    return hostilityCode == null || hostilityCode.equals("HO");
}

/**
 * Determines the time needed for an organisation to reach a location.  Works by first checking mobility capability.  If the organisation
 * has none, checks past movement history to determine maximum speed.  If there's no movement history, throws an exception.
 * @param org The organisation whose capability to move is to be determined.
 * @param destination The location to which we might like the organisation to move.
 * @return The estimated duration, in seconds, needed for the the organisation to move from its current location to the destination.
 * @throws NoLocationException If the organisation has no current location.
 * @throws SpeedIndeterminateException If the organisation's speed can't be estimated.
 **/
public synchronized Double estimatedTimeToReach(Instance org, Instance destination)
    throws  NoLocationException,
            SpeedIndeterminateException {
    double orgSpeed;            // The speed at which the organisation can travel, in km/hr.

    // Check org's mobility capabilities.
    List mobilityCapabilities = new LinkedList();

    // First check through ObjectItemCapability.
    Collection capabilities = org.getOwnSlotValues(_kb.getSlot("is-specified-with-ObjectItemCapability"));
    if ( capabilities.size() > 0 ) {
        extractMobilityCapabilities(capabilities, mobilityCapabilities, _kb.getSlot("obj-item-cap-quantifies-Capability"));
```

```
}
else {
    // Try through ObjectTypeCapabilityNorm.
    Collection objectItemTypes = org.getOwnSlotValues(_kb.getSlot("is-classified-as-ObjectItemType"));
    if ( objectItemTypes.size() > 0 ) {
        for ( Iterator oitIter = objectItemTypes.iterator(); oitIter.hasNext(); ) {
            Instance objType = (Instance)oitIter.next();
            capabilities = objType.getOwnSlotValues(_kb.getSlot("is-specified-as-having-ObjectTypeCapabilityNorm"));
            if ( capabilities.size() > 0 )
                extractMobilityCapabilities(capabilities, mobilityCapabilities, _kb.getSlot("obj-type-cap-norm-quantifies-Capability"));
        }
    }
}
if ( mobilityCapabilities.size() > 0 ) {
    // We ignore the day/night codes, terrain attributes, etc. for now.
    orgSpeed = 0.0;
    for ( Iterator cIter = mobilityCapabilities.iterator(); cIter.hasNext(); ) {
        Instance capabilitySpec[] = (Instance[])cIter.next();
        Instance capabilityAssoc = (Instance)capabilitySpec[0];
        Instance mobilityCapability = (Instance)capabilitySpec[1];

        double quantity = getFloatSlot(capabilityAssoc, "quantity").doubleValue();
        String UoM = getCodeSlot(mobilityCapability, "unitOfMeasureCode");
        if ( ! UoM.equals("KPH") )
            throw new Error("Not prepared to deal with unit " + UoM);
        if ( quantity > orgSpeed )
            orgSpeed = quantity;
    }
}
else {
    // Try through movement history.
    orgSpeed = battlefieldObjectSpeedThroughHistory(org);
}
if ( orgSpeed == 0.0 )
    throw new SpeedIndeterminateException(getObjectItemName(org));

Instance currentLocation = getCurrentLocation(org);
double distanceToDestination = Earth.distance( getFloatSlot(currentLocation, "latitudeCoordinate").doubleValue(),
                                               getFloatSlot(currentLocation, "longitudeCoordinate").doubleValue(),
```

```
                                                    getFloatSlot(destination, "latitudeCoordinate").doubleValue(),
                                                    getFloatSlot(destination, "longitudeCoordinate").doubleValue());
    return new Double(3600.0 * (distanceToDestination/1000.0)/ orgSpeed);
}

/**
 * Takes a {@link java.util.Collection} of associative entities that represent associations between either an <code>ObjectItem</code> or
 * an <code>ObjectType</code>, and identifies which of the associative entities represent mobility capabilities.  For each mobility capability,
 * constructs a two-element array (<code>Instance[2]</code>) containing the associative entity and the related mobility capability.
 * Each such array is added to the <code>mobilityCapabilities</code> parameter.
 * @param capabilityAssocs A homogeneous collection of {@linkplain ida.sd.ontology.Instance instances}
 *        of either <code>ObjectItemCapability</code> or <code>ObjectTypeCapabilityNorm</code>.
 * @param mobilityCapabilities A {@link java.util.List} that, on return from this method,
 *        will have all mobility capabilities identified from <code>capabilityAssocs</code> added.
 * @param assocSlot A {@link ida.sd.ontology.Slot} present in the members of <code>capabilityAssocs</code>.  It must be an inverse slot of
 *        either <code>is-quantified-in-ObjectItemCapability</code> or <code><code>is-quantified-in-ObjectTypeCapabilityNorm</code>.
 **/
private void extractMobilityCapabilities(Collection capabilityAssocs, List mobilityCapabilities, Slot assocSlot) {
    for ( Iterator caIter = capabilityAssocs.iterator(); caIter.hasNext(); ) {
        Instance cAssoc = (Instance)caIter.next();
        Instance capability = (Instance)cAssoc.getOwnSlotValue(assocSlot);
        if ( capability.getDirectType().equals(_mobilityCapabilityCls) )
            mobilityCapabilities.add(new Instance[] { cAssoc, capability });
    }
}

/**
 * Returns the speed of a battlefield object as determined by its last two observed locations.
 * @param battlefieldObject The object whose speed is to be computed.
 * @return The speed of the battlefield object, in km/hr.
 * @throws NoLocationException If the organisation has no current location.
 * @throws SpeedIndeterminateException If the object has only one location, or if the locations were reported at the same time.
 * @todo Support relative reporting data.
 **/
public synchronized double battlefieldObjectSpeedThroughHistory(Instance battlefieldObject)
    throws NoLocationException,
           SpeedIndeterminateException {
    List objItemLocationsList = new LinkedList(battlefieldObject.getOwnSlotValues(_objItemLocSlot));

    // If there's no location, we conclude the object isn't moving.
```

```
int size = objItemLocationsList.size();
if ( size == 0 )
    throw new NoLocationException(getObjectItemName(battlefieldObject));

ReportingData.sortByReportingTime(this, objItemLocationsList, _objItemLocRDRefSlot);

// If the last point in the list has a speed, use that.
Instance currentObjItemLocInst = (Instance)objItemLocationsList.get(size - 1);
Double speed;
if ( (speed = getFloatSlot(currentObjItemLocInst, "speedRate")) != null )
    return speed.doubleValue();

// If the list has only 1 item, the object isn't moving.
if ( size == 1 )
    throw new SpeedIndeterminateException(getStringSlot(battlefieldObject, "name"));

// Use the last 2 points to compute the speed.
double prevLat, prevLon;
double curLat, curLon;
String date, time;
Instance reportingData;

Slot oil2Location = _kb.getSlot("obj_item_loc-is-associated-with-Location");

Instance prevObjItemLocInst = (Instance)objItemLocationsList.get(size - 2);
Instance prevLocation = (Instance)prevObjItemLocInst.getOwnSlotValue(oil2Location);
prevLat = getFloatSlot(prevLocation, "latitudeCoordinate").doubleValue();
prevLon = getFloatSlot(prevLocation, "longitudeCoordinate").doubleValue();

reportingData = (Instance)prevObjItemLocInst.getOwnSlotValue(_objItemLocRDRefSlot);
// The following assumes reportingData is absolute.
Calendar prevTime = getDateTimeSlots(reportingData, "effectiveDate", "effectiveTime");
if ( prevTime == null )
    throw new SpeedIndeterminateException(getObjectItemName(battlefieldObject));

Instance currentLocation = (Instance)currentObjItemLocInst.getOwnSlotValue(oil2Location);
curLat = getFloatSlot(currentLocation, "latitudeCoordinate").doubleValue();
curLon = getFloatSlot(currentLocation, "longitudeCoordinate").doubleValue();
reportingData = (Instance)currentObjItemLocInst.getOwnSlotValue(_objItemLocRDRefSlot);    // Should check to see if it's absolute
                                                                                         // or relative...
Calendar curTime = getDateTimeSlots(reportingData, "effectiveDate", "effectiveTime");;
if ( curTime == null )
    throw new SpeedIndeterminateException(getStringSlot(battlefieldObject, "name"));
```

```
    long timeSpan = (curTime.getTime().getTime() - prevTime.getTime().getTime())/1000;
    if ( timeSpan == 0 )
        throw new SpeedIndeterminateException(getStringSlot(battlefieldObject, "name"));

    double distance = Earth.distance(prevLat, prevLon, curLat, curLon);
    return (distance/timeSpan)*3.6;         // 3.6 multiplier converts m/s to km/hr.
}

/**
 * A {@link java.util.Comparator} suitable for comparing two {@linkplain ida.sd.ontology.Instance instances}
 * that have <code>effectiveDate</code> and <code>effectiveTime</code> slots.
 **/
public final java.util.Comparator dtComparator = new java.util.Comparator() {
        public int compare(Object o1, Object o2) {
            Calendar c1 = getDateTimeSlots((Instance)o1, "effectiveDate", "effectiveTime");
            Calendar c2 = getDateTimeSlots((Instance)o2, "effectiveDate", "effectiveTime");
            if ( c1 == null )
                return c2 == null ? 0 : -1;
            else if ( c2 == null )
                return 1;

            if ( c1.before(c2) )
                return -1;
            else if ( c1.after(c2) )
                return 1;
            else
                return 0;
        }
    };

/**
 * Returns the task currently associated with an organisation.
 * <p>Currently, just returns the most recently created task.  Does not check if the task applies to the current time.
 * @return The task currently associated with an organisation, or null if no task is associated.
 **/
public synchronized Instance currentTask(Instance org) {
    Collection orgActionTaskAssocList = org.getOwnSlotValues(_kb.getSlot("has-its-role-specified-through-OrganisationActionAssociation"));
    if ( orgActionTaskAssocList == null || orgActionTaskAssocList.size() == 0 )
        return null;
    List l = new LinkedList(orgActionTaskAssocList);
    Collections.sort(l, dtComparator);
```

```
    Instance currentOAA = (Instance)l.get(l.size()-1);
    return (Instance)currentOAA.getOwnSlotValue(_kb.getSlot("org_act_assoc-is-associated-with-Action"));
}

/**
 * Returns the task currently associated with an organisation or, if the organisation has no task, the task associated with its superiors.
 * <p>Currently, just returns the last task; does not check if the task applies to the current time.
 * @return The task currently associated with an organisation, or null if no task is associated.
 **/
public synchronized Instance currentTaskInOrgHierarchy(Instance org) {
    Set orgsExamined = new HashSet();       // Just in case the org-org assocs have a circularity.
    for (;;) {
        List orgActionTaskAssocList =
            new LinkedList(org.getOwnSlotValues(_kb.getSlot("has-its-role-specified-through-OrganisationActionAssociation")));
        if ( orgActionTaskAssocList != null && orgActionTaskAssocList.size() > 0 ) {
            List l = new LinkedList(orgActionTaskAssocList);
            Collections.sort(l, dtComparator);
            Instance currentOAA = (Instance)l.get(l.size()-1);
            return (Instance)currentOAA.getOwnSlotValue(_kb.getSlot("org-action-task-assoc-is-associated-with-Action"));
        }
        Instance commandingOrganisation = commandingOrganisation(org);
        if ( commandingOrganisation == null || orgsExamined.contains(commandingOrganisation) )
            return null;
        orgsExamined.add(commandingOrganisation);
        org = commandingOrganisation;
    }
}

/**
 * Returns the organisation that commands an organisation -- i.e., is the subject of a CMDCTL association where the specified organisation is
 * an object.
 * <p>Currently admits of no more than one commanding organisation.
 * @param org The organisation whose commanding organisation is to be determined.
 * @return The commanding organisation, or <code>null</code> if there is none.
 **/
public synchronized Instance commandingOrganisation(Instance org) {
    Collection OIAs = org.getOwnSlotValues(_kb.getSlot("is-the-object-of-ObjectItemAssociation"));
    if ( OIAs == null || OIAs.size() == 0 )
        return null;
```

```
    Slot subjectObjItemSlot = _kb.getSlot("subject-is-associated-with-ObjectItem");
    for ( Iterator oiaIter = OIAs.iterator(); oiaIter.hasNext(); ) {
        Instance oia = (Instance)oiaIter.next();
        String categoryCode = getCodeSlot(oia, "categoryCode");
        if ( categoryCode == null || ! categoryCode.equals("CMDCTL") )
            continue;
        Instance subjectObjItem = (Instance)oia.getOwnSlotValue(subjectObjItemSlot);
        if ( subjectObjItem.instanceOf(_orgCls) )
            return subjectObjItem;
    }
    return null;
}

/**
 * Determines whether an {@linkplain ida.sd.ontology.Instance} of one <code>Organisation</code> commands another, directly or indirectly.
 * The semantics of commanding are as specified by the {@link #commandingOrganisation} method.
 * @param org1 An organisation.
 * @param org2 An organisation.
 * @return True if <code>org1</code> commands <code>org2</code>, false if not.
 **/
public synchronized boolean commands(Instance org1, Instance org2) {
    Instance commandingOrg;
    commandingOrg = commandingOrganisation(org2);
    while ( commandingOrg != null ) {
        if ( getObjectItemName(commandingOrg).equals(getObjectItemName(org1)) )
            return true;
        commandingOrg = commandingOrganisation(commandingOrg);
    }
    return false;
}

/** Supports the {@link #supplyAdequacy} method. */
private Map _supplyAdequacyMap = new HashMap();
/**
 * Determines if an organisation's supplies are adequate.  Very simple-minded.  Should be connected to the organisation's
 * ability to achieve some objective.
 **/
public synchronized double supplyAdequacy(Instance org) {
    // The current implementation works as follows:  Each organisation  for which this method is invoked gets a randomly assigned value.
    // The distribution is weighted such that the probability of adequacy is 3/4.
```

```
    String orgName = getObjectItemName(org);
    if ( ! _supplyAdequacyMap.containsKey(orgName) ) {
        double r = java.lang.Math.random();
        _supplyAdequacyMap.put(orgName, new Double(r <= 0.25 ? r*0.75 : r*0.25 + 0.75));
    }
    return ((Double)_supplyAdequacyMap.get(orgName)).doubleValue();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// The following methods encapsulate how the GH5 ontology stores values in slots.
//
// We provide two forms of each method.  One uses a slot name.  The other uses an actual Slot.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/**
 * Returns the value of the named code-valued {@link Slot} of an {@link Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @return The value of the named {@link Slot} of an {@link Instance}.
 **/
public String getCodeSlot(Instance inst, String slotName) {
    return getCodeSlot(inst, _kb.getSlot(slotName));
}

/**
 * Returns the value of a code-valued {@link Slot} of an {@link Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName An own slot of <code>inst</code>.
 * @return The value of <code>slot</code>.
 **/
public String getCodeSlot(Instance inst, Slot slot) {
    Instance codeTypedInst = (Instance)inst.getOwnSlotValue(slot);
    if ( codeTypedInst == null )
        return null;
    Instance tiv = (Instance)codeTypedInst.getOwnSlotValue(_tivSlot);
    if ( tiv == null )
        throw new Error("Null tiv for " + inst.getDirectType().getName() + "." + slot.getName() + ", cti=" + codeTypedInst.getName());
    return (String)((Instance)codeTypedInst.getOwnSlotValue(_tivSlot)).getOwnSlotValue(_eelSlot);
}

/**
```

```
 * Sets the value of the named code-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @param code The code to which the named slot will be set.
 * @throws InvalidCodeException If <code>code</code> is not a valid code for <code>slot</code>.
 **/
public void setCodeSlot(Instance inst, String slotName, String code)
    throws InvalidCodeException {
    setCodeSlot(inst, _kb.getSlot(slotName), code);
}

/**
 * Sets the value of a code-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName An own slot of <code>inst</code>.
 * @param code The code to which the slot will be set.  If <code>null</code>, the slot's value becomes <code>null</code>.
 * @throws InvalidCodeException If <code>code</code> is not a valid code  for <code>slot</code>.
 **/
public void setCodeSlot(Instance inst, Slot slot, String code)
    throws InvalidCodeException {
    if ( code == null ) {
        inst.setOwnSlotValue(slot, null);
        return;
    }

    Cls enumCls = allowedCls(inst, slot);
    for ( Iterator eiIter = enumCls.getInstances().iterator(); eiIter.hasNext(); ) {
        Instance enumInst = (Instance)eiIter.next();
        Instance tiv = (Instance)enumInst.getOwnSlotValue(_tivSlot);
        if ( tiv.getOwnSlotValue(_eelSlot).equals(code) ) {
            inst.setOwnSlotValue(slot, enumInst);
            return;
        }
    }
    throw new InvalidCodeException(inst.getDirectType(), slot, code);
}

/**
 * Returns the value of the named floating-point {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
```

```
 * @param slotName The name of an own slot of <code>inst</code>.
 * @return The value of the named {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 **/
public Double getFloatSlot(Instance inst, String slotName) {
    return getFloatSlot(inst, _kb.getSlot(slotName));
}

/**
 * Returns the value of a floating-point {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName An own slot of <code>inst</code>.
 * @return The value of <code>slot</code>.
 **/
public Double getFloatSlot(Instance inst, Slot slot) {
    Instance floatTypedInst = (Instance)inst.getOwnSlotValue(slot);
    if ( floatTypedInst == null )
        return null;
    Instance tiv = (Instance)(floatTypedInst.getOwnSlotValue(_tivSlot));

    if ( tiv.getDirectType().hasSuperclass(_numericExpressionCls) ) {
        Number n = _numericExpression.evaluate(tiv);
        return n instanceof Double ? (Double)n : new Double(n.doubleValue());
    }
    else {
        throw new TIVError(inst, slot, _numericExpressionCls, tiv.getDirectType());
    }
}

/**
 * Sets the value of the named floating-point {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @param value The value to which the named slot will be set.
 **/
public void setFloatSlot(Instance inst, String slotName, Number value) {
    setFloatSlot(inst, _kb.getSlot(slotName), value);
}

/**
 * Sets the value of a floating-point {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
```

```
 * @param slotName An own slot of <code>inst</code>.
 * @param value The value to which the slot will be set.  If <code>null</code>, the slot's value becomes <code>null</code>.
 **/
public void setFloatSlot(Instance inst, Slot slot, Number value) {
    if ( value == null ) {
        inst.setOwnSlotValue(slot, null);
        return;
    }

    setNumericSlot(inst, slot, _numericExpression.createLiteral(value.floatValue()));
}

/**
 * Returns the value of the named integer-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @return The value of the named {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 **/
public Integer getIntSlot(Instance inst, String slotName) {
    return getIntSlot(inst, _kb.getSlot(slotName));
}

/**
 * Returns the value of an integer-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slot An own slot of <code>inst</code>.
 * @return The value of <code>slot</code>.
 **/
public Integer getIntSlot(Instance inst, Slot slot) {
    Instance intTypedInst = (Instance)inst.getOwnSlotValue(slot);
    if ( intTypedInst == null )
        return null;

    Instance tiv = (Instance)(intTypedInst.getOwnSlotValue(_tivSlot));
    if ( tiv.getDirectType().hasSuperclass(_numericExpressionCls) ) {
        Number n = _numericExpression.evaluate(tiv);
        return n instanceof Integer ? (Integer)n : new Integer(n.intValue());
    }
    else {
        throw new TIVError(inst, slot, _numericExpressionCls, tiv.getDirectType());
    }
```

```
}

/**
 * Sets the named integer-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance} to a specified value.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @param value The value to which the named slot will be set.
 **/
public void setIntSlot(Instance inst, String slotName, Integer value) {
    setIntSlot(inst, _kb.getSlot(slotName), value);
}

/**
 * Sets an integer-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance} to a given value.
 * @param inst An instance in this knowledge base.
 * @param slot An own slot of <code>inst</code>.
 * @param value The value to which the slot will be set.  If <code>null</code>, the slot's value becomes <code>null</code>.
 **/
public void setIntSlot(Instance inst, Slot slot, Integer value) {
    if ( value == null ) {
        inst.setOwnSlotValue(slot, null);
        return;
    }

    setNumericSlot(inst, slot, _numericExpression.createLiteral(value));
}

/**
 * Sets the value of a {@link ida.sd.ontology.Slot} that takes a numeric value.
 * @param inst An {@link ida.sd.ontology.Instance}
 * @param slot An own slot of <code>inst</code> that takes a numeric value.
 * @param slotValue An instance of the <code>Numeric-Literal</code> {@link ida.sd.ontology.Cls}.
 **/
private void setNumericSlot(Instance inst, Slot slot, Instance slotValue) {
    Cls tivCls = allowedTIVCls(inst, slot);
    if ( ! tivCls.equals(_numericExpressionCls) )
        throw new TIVError(inst, slot, _numericExpressionCls, tivCls);

    Instance slotInst = allowedCls(inst, slot).createDirectInstance();
    slotInst.setOwnSlotValue(_tivSlot, slotValue);

    inst.setOwnSlotValue(slot, slotInst);
```

```
}
/**
 * Returns a {@link java.util.Calendar} for the date and time denoted by
 * named {@linkplain ida.sd.ontology.Slot slots} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param dateSlotName The name of an own slot of <code>inst</code> that denotes a date.
 * @param timeSlotName The name of an own slot of <code>inst</code> that denotes a time.
 * @return A {@link java.util.Calendar} for the date and time denoted by the named slots.
 **/
public Calendar getDateTimeSlots(Instance inst, String dateSlotName, String timeSlotName) {
    return getDateTimeSlots(inst, _kb.getSlot(dateSlotName), _kb.getSlot(timeSlotName));
}
/**
 * Returns a {@link java.util.Calendar} for the date and time denoted by {@linkplain ida.sd.ontology.Slot slots} of an
 * {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param dateSlot A own slot of <code>inst</code> that denotes a date.
 * @param timeSlot An own slot of <code>inst</code> that denotes a time.
 * @return A {@link java.util.Calendar} for the date and time denoted by the slots.
 **/
public Calendar getDateTimeSlots(Instance inst, Slot dateSlot, Slot timeSlot) {
    Instance dateTIV = (Instance)inst.getOwnSlotValue(dateSlot);
    Instance timeTIV = (Instance)inst.getOwnSlotValue(timeSlot);
    if ( dateTIV == null || timeTIV == null )
        return null;
    Integer time = (Integer)_numericExpression.evaluate((Instance)timeTIV.getOwnSlotValue(_tivSlot));
    return DateTime.getCalendar((String)dateTIV.getOwnSlotValue(_tivSlot), time.toString());
}
/**
 * Given the names of two {@linkplain ida.sd.ontology.Slot slots} of an {@link ida.sd.ontology.Instance}  that model a date and a time,
 * sets the slots to the date and time of a specified moment.
 * @param inst An instance in this knowledge base.
 * @param dateSlotName The name of an own slot of <code>inst</code> that denotes a date.
 * @param timeSlotName The name of an own slot of <code>inst</code> that denotes a time.
 * @param moment A moment in time.
 **/
public void setDateTimeSlots(Instance inst, String dateSlotName, String timeSlotName, Calendar moment) {
```

```
    setDateTimeSlots(inst, _kb.getSlot(dateSlotName), _kb.getSlot(timeSlotName), moment);
}

/**
 * Given two {@linkplain ida.sd.ontology.Slot slots} of an {@link ida.sd.ontology.Instance}that model a date and a time, sets the slots to the
 * date and time of a specified moment.
 * @param inst An instance in this knowledge base.
 * @param dateSlot An own slot of <code>inst</code> that denotes a date.
 * @param timeSlot An own slot of <code>inst</code> that denotes a time.
 * @param moment A moment in time.
 **/
public void setDateTimeSlots(Instance inst, Slot dateSlot, Slot timeSlot, Calendar moment) {
    setDateSlot(inst, dateSlot, moment);
    setTimeSlot(inst, timeSlot, moment);
}

/**
 * Returns a {@link java.util.Calendar} for the date denoted by the named {@linkplain ida.sd.ontology.Slot slot} of an
 * {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param dateSlotName The name of an own slot of <code>inst</code> that denotes a date.
 * @return A {@link java.util.Calendar} for the date denoted by <code>dateSlotName</code>.
 **/
public Calendar getDateSlot(Instance inst, String dateSlotName) {
    return getDateSlot(inst, _kb.getSlot(dateSlotName));
}

/**
 * Returns a {@link java.util.Calendar} for the date denoted  by an own {@linkplain ida.sd.ontology.Slot slot} of an
 * {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param dateSlot An own slot of <code>inst</code> that denotes a date.
 * @return A {@link java.util.Calendar} for the date denoted by <code>dateSlot</code>.
 **/
public Calendar getDateSlot(Instance inst, Slot dateSlot) {
    Instance dateTIV = (Instance)inst.getOwnSlotValue(dateSlot);
    if ( dateTIV == null )
        return null;

    return DateTime.getDateCalendar((String)dateTIV.getOwnSlotValue(_tivSlot));
}
```

```
/**
 * Given the name of a {@linkplain ida.sd.ontology.Slot slot} of an {@link ida.sd.ontology.Instance}  that models a date,
 * sets that slot to the date of a specified moment.
 * @param inst An instance in this knowledge base.
 * @param dateSlotName The name of an own slot of <code>inst</code> that denotes a date.
 * @param date A moment in time.  Time fields are ignored; only date information is used.
 **/
public void setDateSlot(Instance inst, String dateSlotName, Calendar date) {
    setDateSlot(inst, _kb.getSlot(dateSlotName), date);
}

/**
 * Given a {@linkplain ida.sd.ontology.Slot slot} of an {@link ida.sd.ontology.Instance}  that models a date,
 * sets that slot to the date of a specified moment.
 * @param inst An instance in this knowledge base.
 * @param dateSlot An own slot of <code>inst</code> that denotes a date.
 * @param date A moment in time.  Time fields are ignored; only date information is used.
 **/
public void setDateSlot(Instance inst, Slot dateSlot, Calendar date) {
    if ( date == null ) {
        inst.setOwnSlotValue(dateSlot, null);
        return;
    }

    // The following assumes a date is represented as a string.
    Instance dateTIV = allowedCls(inst, dateSlot).createDirectInstance();
    dateTIV.setOwnSlotValue(_tivSlot, DateTime.dateToString(date));
    inst.setOwnSlotValue(dateSlot, dateTIV);
}

/**
 * Given the name of a {@linkplain ida.sd.ontology.Slot slot} of an {@link ida.sd.ontology.Instance}  that models a time,
 * returns that slot's value as the number of seconds since midnight.
 * @param inst An instance in this knowledge base.
 * @param timeSlotName The name of an own slot of <code>inst</code> that denotes a time.
 * @return The value of <code>timeSlotName</code> as the number of seconds since midnight.
 **/
public Integer getTimeSlot(Instance inst, String timeSlotName) {
    return getTimeSlot(inst, _kb.getSlot(timeSlotName));
}
```

```
/**
 * Given a {@linkplain ida.sd.ontology.Slot slot} of an {@link ida.sd.ontology.Instance}  that models a time,
 * returns that slot's value as the number of seconds since midnight.
 * @param inst An instance in this knowledge base.
 * @param timeSlot An own slot of <code>inst</code> that denotes a time.
 * @return The value of <code>timeSlot</code> as the number of seconds since midnight.
 **/
public Integer getTimeSlot(Instance inst, Slot timeSlot) {
    Instance timeTIV = (Instance)inst.getOwnSlotValue(timeSlot);
    if ( timeTIV == null )
        return null;

    return (Integer)_numericExpression.evaluate((Instance)timeTIV.getOwnSlotValue(_tivSlot));
}

/**
 * Given the name of a {@linkplain ida.sd.ontology.Slot slot} of an {@link ida.sd.ontology.Instance}  that models a time,
 * sets that slot's value to a specified time.
 * @param inst An instance in this knowledge base.
 * @param timeSlotName The name of an own slot of <code>inst</code> that denotes a time.
 * @param time The time to which <code>timeSlotName</code>'s value will be set.  Only the time information is used;
 *        date information is ignored.
 **/
public void setTimeSlot(Instance inst, String timeSlotName, Calendar time) {
    setTimeSlot(inst, _kb.getSlot(timeSlotName), time);
}

/**
 * Given a {@linkplain ida.sd.ontology.Slot slot} of an {@link ida.sd.ontology.Instance} that models a time,
 * sets that slot's value to a specified time.
 * @param inst An instance in this knowledge base.
 * @param timeSlot An own slot of <code>inst</code> that denotes a time.
 * @param time The time to which <code>timeSlot</code>'s value will be set.  Only the time information is used; date information is ignored.
 **/
public void setTimeSlot(Instance inst, Slot timeSlot, Calendar time) {
    if ( time == null ) {
        inst.setOwnSlotValue(timeSlot, null);
        return;
    }

    // A time is represented as a integral numeric expression
```

```
    // denoting seconds past midnight.
    Instance timeTIV = allowedCls(inst, timeSlot).createDirectInstance();
    timeTIV.setOwnSlotValue(_tivSlot, _numericExpression.createLiteral(DateTime.timeToInt(time)));
    inst.setOwnSlotValue(timeSlot, timeTIV);
}

/**
 * Returns the value of the named string-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @return The value of the named {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 **/
public String getStringSlot(Instance inst, String slotName) {
    return getStringSlot(inst, _kb.getSlot(slotName));
}

/**
 * Returns the value of a string-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance}.
 * @param inst An instance in this knowledge base.
 * @param slotName An own slot of <code>inst</code>.
 * @return The value of <code>slot</code>.
 **/
public String getStringSlot(Instance inst, Slot slot) {
    Instance stringTypedInst = (Instance)inst.getOwnSlotValue(slot);
    return stringTypedInst != null ? (String)stringTypedInst.getOwnSlotValue(_tivSlot) : null;
}

/**
 * Sets the named string-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance} to a specified value.
 * @param inst An instance in this knowledge base.
 * @param slotName The name of an own slot of <code>inst</code>.
 * @param value The value to which the named slot will be set.
 * @throws StringLengthException If <code>value</code> is longer than the maximum length allowed for the named slot.
 **/
public void setStringSlot(Instance inst, String slotName, String value) throws StringLengthException {
    setStringSlot(inst, _kb.getSlot(slotName), value);
}

/**
 * Sets a string-valued {@link ida.sd.ontology.Slot} of an {@link ida.sd.ontology.Instance} to a given value.
 * @param inst An instance in this knowledge base.
```

```
 * @param slot An own slot of <code>inst</code>.
 * @param value The value to which the slot will be set.  If <code>null</code>, the slot's value becomes <code>null</code>.
 * @throws StringLengthException If <code>value</code> is longer than  the maximum length allowed for <code>slot</code>.
 **/
public void setStringSlot(Instance inst, Slot slot, String value) throws StringLengthException {
    if ( value == null ) {
        inst.setOwnSlotValue(slot, null);
        return;
    }
    Cls stringType = allowedCls(inst, slot);
    Instance vInst = stringType.createDirectInstance();
    int length = ((Integer)stringType.getOwnSlotValue(_stringLengthSlot)).intValue();
    if ( value.length() > length )
        throw new StringLengthException(value, length);
    if ( stringType.hasSuperclass(_fixedLengthStringCls) ) {
        StringBuffer vb = new StringBuffer(length).append(value);
        for ( int i = value.length(); i < length; i++ )
            vb.append(' ');
        vInst.setOwnSlotValue(_tivSlot, vb.toString());
    }
    else {
        vInst.setOwnSlotValue(_tivSlot, value);
    }
    inst.setOwnSlotValue(slot, vInst);
}

private Cls allowedCls(Instance inst, Slot slot) {
    Collection allowedClses = inst.getDirectType().getTemplateSlotAllowedClses(slot);
    if ( allowedClses.size() != 1 )
        throw new AllowedClsesCollectionSizeError(inst, slot, allowedClses);
    return (Cls)allowedClses.iterator().next();
}

private Cls allowedTIVCls(Instance inst, Slot slot) {
    Cls allowedCls = allowedCls(inst, slot);
    Collection allowedTIVClses = allowedCls.getTemplateSlotAllowedClses(_tivSlot);
    if ( allowedTIVClses.size() != 1 )
        throw new AllowedClsesCollectionSizeError(allowedCls, _tivSlot, allowedTIVClses);
    return (Cls)allowedTIVClses.iterator().next();
```

```
}
/**
 * Uses system properties to construct a <code>GH5KB</code> instance, and returns that instance.
 * These properties are as follows:
 * <ul>
 * <li>The knowledge base to open is the value of {@link ida.eh.Properties#ontologyURL}.</li>
 * <li>If the {@link ida.eh.Properties#dataSource} is <code>"db"</code>, the knowledge base is populated from a database.
 *       See {@link ida.sd.osb.BindingPA} for details.</li>
 * </ul>
 * This method explicitly binds the knowledge base to a
 * {@linkplain ida.sd.ontology.impl.protege.ProtegeProject Protege-2000 based implementation}.
 * @return A <code>GH5KB</code> instance created from the value of the <code>ontologyURL</code> property,
 *       and maybe loaded with data from the <code>dataSource</code> property.
 * @throws OntologyLoadException If the knowledge base cannot be created for any reason.
 **/
public static GH5KB loadFromProperties() throws OntologyLoadException {
    // Open the ontology.
    GH5KB kb = openFromProperties();

    // Load data into the knowledge base, if so specified.
    String dataSource = System.getProperty(ida.eh.Properties.dataSource);
    if ( dataSource != null ) {
        if ( dataSource.equals("db") ) {
            try {
                BindingPA.loadDBUsingSystemProperties(kb);
            } catch ( BindingException e ) {
                throw new OntologyLoadException(e);
            }
        }
        else {
            throw new OntologyLoadException("Unknown external data source \"" + dataSource + "\"");
        }
    }

    return kb;
}
/**
 * Creates a <code>GH5KB</code> using the value of system property {@link ida.eh.Properties#ontologyURL} as the ontology.  This method
 * explicitly binds the knowledge base to a
```

```
     * {@linkplain ida.sd.ontology.impl.protege.ProtegeProject Prot&eacute;g&eacute;-2000 based implementation}.
     * @return A <code>GH5KB</code> instance created from the value of the <code>ontologyURL</code> property.
     * @throws OntologyLoadException If the <code>ontologyURL</code> property is missing, or if Prot&eacute;g&eacute;-2000 fails to load the
     *       ontology from the specified URL.
     **/
    public static GH5KB openFromProperties() throws OntologyLoadException {
        String ontologyURL;
        if ( (ontologyURL = System.getProperty(ida.eh.Properties.ontologyURL)) == null )
            throw new OntologyLoadException("Missing property \"" + ida.eh.Properties.ontologyURL + "\" (ontology URL)");
        try {
            return new GH5KB(new ProtegeProject(ontologyURL).getKnowledgeBase());
        } catch ( ProtegeProject.ProjectException e ) {
            throw new OntologyLoadException(e);
        }
    }
}
```

3. Class DateTime

GH-conformant data sets have their own set of rules for representing dates and times. Dates are usually represented as 8-character strings of the form YYYYMMDD. Times are usually represented as 6-character strings of decimal digits interpreted as seconds since midnight.

The GH ontology uses the Generic Hub's format to represent dates. For an instance of class A.D. Date, the value type of own slot type-instance-value is a string. The GH ontology represents times as instances of the Numeric-Expression class.

These representations were chosen mainly for convenience in loading Generic Hub data sets. They might change in the future, so it's desirable to hide them from other classes. This is the purpose of class DateTime. It provides methods that convert between GH ontology representations of dates and times and Java representations.

```
package ida.sd.gh5ontology;

import java.text.DecimalFormat;
import java.text.SimpleDateFormat;

import java.util.Calendar;
import java.util.GregorianCalendar;

/**
 * The <code>DateTime</code> class provides procedural abstractions for manipulating dates and times in a GH5 knowledge base.
```

```
 * These abstractions convert GH5-formatted dates and times to Java class instances and vice versa. The primary secrets of this class are:
 * <ol>
 * <li>The format of dates and times in a GH5 knowledge base (but <em>not</em> their representation in a database; that secret is
 *       more general).</li>
 * <li>The algorithms for converting between GH5 formats and Java formats.</li>
 * </ol>
**/
public class DateTime {
    /** Ensures that no instances of this class are constructed. */
    private DateTime() {}

    /**
     * Converts a date/time pair expressed in GH5 format into a {@link Calendar}.
     * <p>The sample data set we have includes times that are right-padded with blanks.  I don't know whether this is strictly legal, but
     * we allow it in this method.
     * @param date An 8-character string of the form <code>yyyymmdd</code>.
     * @param time A 6-character string denoting seconds since midnight.
     * @return A {@link Calendar} expressing the given date/time pair.
     **/
    public static Calendar getCalendar(String date, String time) {
        int year        = Integer.valueOf(date.substring(0, 4)).intValue();
        int month       = Integer.valueOf(date.substring(4, 6)).intValue() - 1;
        int day         = Integer.valueOf(date.substring(6, 8)).intValue();
        int timeSeconds = Integer.valueOf(time.trim()).intValue();
        int hour        = timeSeconds/3600;
        int mins        = (timeSeconds - hour*60)/60;
        int secs        = timeSeconds - hour*3600 - mins*60;
        return new GregorianCalendar(year, month, day, hour, mins, secs);
    }

    /**
     * Converts a date expressed in GH5 format into a {@link java.util.Calendar}.  Time information is not set.
     * @param date An 8-character string of the form <code>yyyymmdd</code>.
     * @return A {@link java.util.Calendar} expressing the given date.
     **/
    public static Calendar getDateCalendar(String date) {
        int year  = Integer.valueOf(date.substring(0, 4)).intValue();
        int month = Integer.valueOf(date.substring(4, 6)).intValue() - 1;
        int day   = Integer.valueOf(date.substring(6, 8)).intValue();
        return new GregorianCalendar(year, month, day);
```

```
    }

    /** Encapsulates the <code>CHAR(8)</code> format of GH5 date attributes. */
    private final static SimpleDateFormat DATE_FORMAT = new SimpleDateFormat("yyyyMMdd");

    /**
     * Returns the date information associated with a {@link Calendar} as a GH5-formatted date.
     * @param calendar A calendar.
     * @return The date information of <code>calendar</code> as an 8-character string of the form <code>yyyymmdd</code>.
     **/
    public static String dateToString(Calendar calendar) {
        return DATE_FORMAT.format(calendar.getTime());
    }

    /**
     * Returns the time information associated with a {@link Calendar} as seconds since midnight.
     * @param calendar A calendar.
     * @return The time information of <code>calendar</code> as seconds since midnight.
     **/
    public static Integer timeToInt(Calendar calendar) {
        int hours = calendar.get(Calendar.HOUR_OF_DAY);
        int mins  = calendar.get(Calendar.MINUTE);
        int secs  = calendar.get(Calendar.SECOND);
        return new Integer(hours*3600 + mins*60 + secs);
    }

    /** Encapsulates the <code>CHAR(6)</code> format of GH5 time attributes. */
    private final static DecimalFormat TIME_FORMAT = new DecimalFormat("000000");

    /**
     * Returns the time information associated with a {@link Calendar} as a GH5-formatted time.
     * @param calendar A calendar.
     * @return The time information of <code>calendar</code> as a 6-character string denoting seconds since midnight.
     **/
    public static String timeToString(Calendar calendar) {
        return TIME_FORMAT.format(timeToInt(calendar).longValue());
    }
}
```

4. Class NumericExpression

GH ontology slots that model numeric quantities have an instance of class Numeric-Expression as the value type of their type-instance-value own slot. This representation aids inferencing by allowing functional expression. Specifying a bearing angle as 45° is fine, but stating it as the arcsine of some expression that equals 0.707106 may help agents understand how the bearing angle was derived. The GH ontology allows both forms.

Most of the time, an agent will simply want the numeric value of a slot rather than its structure. The NumericExpression class encapsulates how an instance of the Numeric-Expression class in the GH ontology can be translated into a numeric value. It does so through the method evaluate(), which when applied to an instance of Numeric-Expression yields a Java Number.

```
package ida.sd.gh5ontology;

import java.util.Iterator;

import ida.sd.ontology.*;

/**
 * The <code>NumericExpression</code> class encapsulates the <code>Numeric-Expression</code> {@link Cls} in the GH ontology.  It hides
 * decisions on the representation of that class. The primary secrets of this class are the representation (slots and their facets) of the
 * <code>Numeric-Expression</code> class, and the algorithms for manipulating instances of <code>NumericExpression</code>.
 **/
public class NumericExpression {

    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // The following fields are used to cache classes and frames that are commonly used when evaluating a NumericExpresison.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    private final Cls _integerLiteralCls;
    private final Cls _realLiteralCls;

    private final Slot _valueSlot;
    private final Slot _baseSlot;
    private final Slot _descrSlot;

    private final Cls _productCls;
    private final Cls _sumCls;
    private final Cls _differenceCls;
    private final Cls _divisionCls;

    private final Slot _operandsSlot;

    private final static Integer _TEN = new Integer(10);

    /**
```

```
 * Constructs a <code>NumericExpression</code> instance.
 * @param gh5KB A GH5 knowledge base.
 **/
public NumericExpression(KnowledgeBase gh5KB) {
    _integerLiteralCls = gh5KB.getCls("Integer-Literal");
    _realLiteralCls    = gh5KB.getCls("Real-Literal");

    _valueSlot  = gh5KB.getSlot("value");
    _baseSlot   = gh5KB.getSlot("base");
    _descrSlot  = gh5KB.getSlot("description");

    _productCls      = gh5KB.getCls("Product");
    _sumCls          = gh5KB.getCls("Sum");
    _differenceCls   = gh5KB.getCls("Difference");
    _divisionCls     = gh5KB.getCls("Division");

    _operandsSlot = gh5KB.getSlot("operands");
}

/**
 * Creates an {@link Instance} of an <code>Integer-Literal</code> from an {@link Integer}.  The base is 10, and the description is the string
 * image of the parameter.
 * @param i The value for the literal.
 **/
public Instance createLiteral(Integer i) {
    Instance inst = _integerLiteralCls.createDirectInstance();
    inst.setOwnSlotValue(_valueSlot, i);
    inst.setOwnSlotValue(_baseSlot, _TEN);
    inst.setOwnSlotValue(_descrSlot, i.toString());
    return inst;
}

/**
 * Creates an {@link Instance} of an <code>Integer-Literal</code> from an <code>int</code>.
 * The base is 10, and the description is the string image of the parameter.
 * @param i The value for the literal.
 **/
public Instance createLiteral(int i) {
    return createLiteral(new Integer(i));
}

/**
```

```
 * Creates an {@link Instance} of a <code>Real-Literal</code> from a {@link Float}.
 * The base is 10, and the description is the string image of the parameter.
 * @param f The value for the literal.
 **/
public Instance createLiteral(Float f) {
    Instance inst = _realLiteralCls.createDirectInstance();
    inst.setOwnSlotValue(_valueSlot, f);
    inst.setOwnSlotValue(_baseSlot, _TEN);
    inst.setOwnSlotValue(_descrSlot, f.toString());
    return inst;
}

/**
 * Creates an {@link Instance} of a <code>Real-Literal</code> from a <code>float</code>.
 * The base is 10, and the description is the string image of the parameter.
 * @param f The value for the literal.
 **/
public Instance createLiteral(float f) {
    return createLiteral(new Float(f));
}

/**
 * Returns the {@link java.lang.Number} that results from evaluating an {@link ida.sd.ontology.Instance} of {@link ida.sd.ontology.Cls}
 * <code>NumericExpression</code>.  The type of the result will be either {@link java.lang.Integer} or {@link java.lang.Double}.
 * @param i The instance to evaluate.
 * @return The <code>Number</code> that results from evaluating the instance.
 **/
public Number evaluate(Instance i) {
    if ( i.getDirectType().equals(_integerLiteralCls) )
        return (Integer)i.getOwnSlotValue(_valueSlot);
    else if ( i.getDirectType().equals(_realLiteralCls) )
        return new Double(((Float)i.getOwnSlotValue(_valueSlot)).doubleValue());
    else if ( i.getDirectType().equals(_productCls) ) {
        Iterator oIter = i.getOwnSlotValues(_operandsSlot).iterator();
        if ( ! oIter.hasNext() )
            return null;
        Number product = evaluate((Instance)oIter.next());
        while ( oIter.hasNext() ) {
            Number nextTerm = evaluate((Instance)oIter.next());
            if ( nextTerm instanceof Double || product instanceof Double )
```

```
            product = new Double(product.doubleValue() * nextTerm.doubleValue());
        else
            product = new Integer(product.intValue() * nextTerm.intValue());
    }
    return product;
}
else if ( i.getDirectType().equals(_sumCls) ) {
    Iterator oIter = i.getOwnSlotValues(_operandsSlot).iterator();
    if ( ! oIter.hasNext() )
        return null;
    Number sum = evaluate((Instance)oIter.next());
    while ( oIter.hasNext() ) {
        Number nextTerm = evaluate((Instance)oIter.next());
        if ( nextTerm instanceof Double || sum instanceof Double )
            sum = new Double(sum.doubleValue() + nextTerm.doubleValue());
        else
            sum = new Integer(sum.intValue() + nextTerm.intValue());
    }
    return sum;
}
else if ( i.getDirectType().equals(_differenceCls) ) {
    Iterator oIter = i.getOwnSlotValues(_operandsSlot).iterator();
    if ( ! oIter.hasNext() )
        return null;
    Number difference = evaluate((Instance)oIter.next());
    while ( oIter.hasNext() ) {
        Number nextTerm = evaluate((Instance)oIter.next());
        if ( nextTerm instanceof Double || difference instanceof Double )
            difference = new Double(difference.doubleValue() - nextTerm.doubleValue());
        else
            difference = new Integer(difference.intValue() - nextTerm.intValue());
    }
    return difference;
}
else if ( i.getDirectType().equals(_divisionCls) ) {
    Iterator oIter = i.getOwnSlotValues(_operandsSlot).iterator();
    if ( ! oIter.hasNext() )
        return null;
    Number quotient = evaluate((Instance)oIter.next());
```

```
            while ( oIter.hasNext() ) {
                Number nextTerm = evaluate((Instance)oIter.next());
                if ( nextTerm instanceof Double || quotient instanceof Double )
                    quotient = new Double(quotient.doubleValue() / nextTerm.doubleValue());
                else
                    quotient = new Integer(quotient.intValue() / nextTerm.intValue());
            }
            return quotient;
        }
        else {
            throw new Error("Not yet handling " + i.getDirectType().getName() + " in numeric expressions");
        }
    }
}
```

5. Class ReportingData

Reporting data is a crucial concept in the Generic Hub, and therefore in the GH ontology as well. In particular, it implements time-sequenced observations. Many agents in the IDA prototype simulation need to know about the most recent observation of some battlefield object property.

The ReportingData class contains a collection of static methods that support orderings of sets of instances based on the associated values of REPORTING-DATA. Note that, aside from getEffectiveTime(), the methods work with instances of classes that have associated REPORTING-DATA (e.g., ObjectItemLocation) rather than with instances of ReportingData. This is generally what a user needs.

```
package ida.sd.gh5ontology;

import java.util.Calendar;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import ida.sd.ontology.*;

/**
 * The <code>ReportingData</code> class contains procedural abstractions for dealing with instances of the GH5 <code>ReportingData</code>
 * class.  Each instance of <code>ReportingData</code> has slots that specify a moment in time.  The methods in this class understand those
 * slots and how they can be used to order instances.  The primary secrets of this class are the slots that order reporting data, and the algorithms
 * for ordering a collection of instances.
```

```
 **/
public class ReportingData {

    /** Ensures that no instances of this class are created. */
    private ReportingData() {}

    /**
     * Given a homogeneous {@link Collection} of {@link Instance}s  that are associated with <code>ReportingData</code>, returns the most
     * recent instance of the collection as determined by reporting date/time.  An instance of class {@link RDComparator} determines which is most
     * recent.
     * @param gh5KB A GH5 knowledge base.
     * @param instances A homogeneous (i.e., all of the same {@link Cls}) collection of {@linkplain Instance instances}.
     * @param relatedReportingDataSlot A template slot of the {@link Cls} of all members of <code>instances</code>.
     * @return The member of <code>instances</code> whose <code>ReportingData</code> instance, as obtained from
     *           <code>relatedReportingDataSlot</code>, is more recent than all other members.  If <code>instances</code> has no members,
     *           returns <code>null</code>.
     **/
    public static Instance getMostRecent(GH5KB gh5KB, Collection instances, Slot relatedReportingDataSlot) {
        if ( instances.size() == 0 )
            return null;

        java.util.Comparator c = new RDComparator(gh5KB, relatedReportingDataSlot);

        Iterator instIter = instances.iterator();
        Instance mostRecentInst = (Instance)instIter.next();
        Instance mostRecentRD = (Instance)mostRecentInst.getOwnSlotValue(relatedReportingDataSlot);

        while ( instIter.hasNext() ) {
            Instance candidateInst = (Instance)instIter.next();
            Instance candidateRD   = (Instance)candidateInst.getOwnSlotValue(relatedReportingDataSlot);
            if ( candidateRD == null )
                continue;
            else if ( mostRecentRD == null || c.compare(candidateRD, mostRecentRD) > 0 ) {
                mostRecentInst = candidateInst;
                mostRecentRD   = candidateRD;
            }
        }
        return mostRecentInst;
    }

    /**
     * Sorts a list of {@linkplain Instance instances} according to the  reporting date/times of their associated <code>ReportingData</code>.
```

```
 * @param gh5kb A GH5 knowledge base.
 * @param instances A homogeneous list of instances.  On return, will be ordered according to related reporting data.
 * @param relatedReportingDataSlot A template slot of the {@link Cls} of all members of <code>instances</code>.
 **/
public static void sortByReportingTime(GH5KB gh5kb, List instances, Slot relatedReportingDataSlot) {
    Collections.sort(instances, new RDComparator(gh5kb, relatedReportingDataSlot));
}

/**
 * Sorts a list of {@linkplain Instance instances} according to the  effective date/times of their associated <code>ReportingData</code>.
 * @param gh5kb A GH5 knowledge base.
 * @param instances A homogeneous list of instances.  On return, will be ordered according to related reporting data.
 * @param relatedReportingDataSlot A template slot of the {@link Cls} of all members of <code>instances</code>.
 **/
public static void sortByEffectiveTime(GH5KB gh5kb, List instances, Slot relatedReportingDataSlot) {
    Collections.sort(instances, new RDEComparator(gh5kb, relatedReportingDataSlot));
}

/**
 * Returns the effective time of an instance of <code>ReportingData</code>.
 * @param gh5kb A GH5 knowledge base.
 * @param rd An {@link Instance} of <code>ReportingData</code>.
 * @return The effective time of an instance of <code>ReportingData</code>, or <code>null</code> if the effective time cannot be computed.
 **/
public static Calendar getEffectiveTime(GH5KB gh5kb, Instance rd) {
    if ( rd.instanceOf(gh5kb.getCls("ReportingDataAbsoluteTiming")) ) {
        return gh5kb.getDateTimeSlots(rd, "effectiveDate", "effectiveTime");
    }
    else if ( rd.instanceOf(gh5kb.getCls("ReportingDataRelativeTiming")) ) {
        // ReportingDataRelativeTiming is w.r.t. an ActionTask.  We use the ActionTask's planned start date and startPrecisionCode.
        // It's possible we need to consider startQualifierCode, but I'm not sure how.
        Integer duration = gh5kb.getIntSlot(rd, gh5kb.getSlot("offsetDuration"));
        if ( duration == null )
            return null;
        Instance associatedActionTask = (Instance)rd.getOwnSlotValue(gh5kb.getSlot("rdrt-uses-as-timing-ref-ActionTask"));
        Calendar actionTaskPlannedStart = gh5kb.getDateTimeSlots(associatedActionTask, "plannedStartDate", "plannedStartTime");
        if ( actionTaskPlannedStart == null )
            return null;
        String precisionCode = gh5kb.getCodeSlot(associatedActionTask, "startPrecisionCode");
        if ( precisionCode == null || precisionCode.equals("NKN") )
```

```
            return null;
        int d = duration.intValue();
        if ( precisionCode.equals("DAY") )
            actionTaskPlannedStart.add(Calendar.DATE, d);
        else if ( precisionCode.equals("HR") )
            actionTaskPlannedStart.add(Calendar.HOUR_OF_DAY, d);
        else if ( precisionCode.equals("MINUTE") )
            actionTaskPlannedStart.add(Calendar.MINUTE, d);
        else if ( precisionCode.equals("MON") )
            actionTaskPlannedStart.add(Calendar.MONTH, d);
        else if ( precisionCode.equals("SECOND") )
            actionTaskPlannedStart.add(Calendar.SECOND, d);
        else if ( precisionCode.equals("WEK") )
            actionTaskPlannedStart.add(Calendar.WEEK_OF_MONTH, d);
        else if ( precisionCode.equals("YEA") )
            actionTaskPlannedStart.add(Calendar.YEAR, d);
        else
            throw new IllegalArgumentException(         "Instance " + associatedActionTask.getName() +
                                                    ": Unknown startPrecisionCode \"" + precisionCode + "\"");
        return actionTaskPlannedStart;
    }
    else {
        return null;
    }
}
/**
 * Compares two <code>ReportingData</code> instances according to their  reporting date/time slots.  The return value follows Java's
 * {@linkplain java.util.Comparator#compare comparator protocol}.
 * <p>Reporting date is a required slot; reporting time is not.  Lack of reporting time interpreted to be prior to presence thereof.  If both
 * <code>i1</code> and <code>i2</code> lack reporting time, they are considered equal.
 * @param kb A {@link GH5KB} in which <code>i1</code> and <code>i2</code> exist.
 * @param i1 An {@link Instance} of <code>ReportingData</code>.
 * @param i2 An {@link Instance} of <code>ReportingData</code>.
 * @return A negative integer if the moment specified by <code>i1</code> is before the moment specified by <code>i2</code>; 0 if they
 *                 specify the same moment; a positive integer if the moment specified by <code>i1</code> is after the moment specified by
 *                 <code>i2</code>.
 * @throws IllegalArgumentException If <code>i1</code> and <code>i2</code> are not both instances of the
```

```
 *                <code>ReportingData</code> class.
 **/
public static int compareByReportingDateTime(GH5KB kb, Instance i1, Instance i2) {
    Cls reportingDataCls = kb.getCls("ReportingData");
    if ( ! i1.instanceOf(reportingDataCls) )
        throw new IllegalArgumentException(i1.getName() + " is not an instance of ReportingData");
    if ( ! i2.instanceOf(reportingDataCls) )
        throw new IllegalArgumentException(i2.getName() + " is not an instance of ReportingData");

    Slot reportingDateSlot = kb.getSlot("reportingDate");
    Slot reportingTimeSlot = kb.getSlot("reportingTime");
    Calendar reportingDT1 = kb.getDateTimeSlots(i1, reportingDateSlot, reportingTimeSlot);
    Calendar reportingDT2 = kb.getDateTimeSlots(i2, reportingDateSlot, reportingTimeSlot);
    if ( reportingDT1.after(reportingDT2) )
        return 1;
    else if ( reportingDT1.equals(reportingDT2) )
        return 0;
    else
        return -1;
}

/**
 * Compares two <code>ReportingData</code> instances according  to their effective date/time slots.  The return value follows Java's
 * {@linkplain java.util.Comparator#compare comparator protocol}.
 * <p>Effective date/time has a precision.  It is used to form a range.  Overlap of ranges indicates equality.
 * <p>Effective date is a required slot; effective time is not.  Lack of effective time interpreted to be prior to presence thereof.  If both
 * <code>i1</code> and <code>i2</code> lack reporting time, they are considered equal.
 * @param kb A {@link GH5KB} in which <code>i1</code> and <code>i2</code> exist.
 * @param i1 An {@link Instance} of <code>ReportingData</code>.
 * @param i2 An {@link Instance} of <code>ReportingData</code>.
 * @return A negative integer if the moment specified by <code>i1</code> is before the moment specified by <code>i2</code>; 0 if they
 *                specify the same moment; a positive integer if the moment specified by <code>i1</code> is after the moment specified by
 *                <code>i2</code>.
 * @throws IllegalArgumentException If <code>i1</code> and <code>i2</code> are not both instances of the
 *                <code>ReportingData</code> class.
 **/
public static int compareByEffectiveDateTime(GH5KB kb, Instance i1, Instance i2) {
    Cls reportingDataCls = kb.getCls("ReportingData");
    if ( ! i1.instanceOf(reportingDataCls) )
        throw new IllegalArgumentException(i1.getName() + " is not an instance of ReportingData");
```

```
    if ( ! i2.instanceOf(reportingDataCls) )
        throw new IllegalArgumentException(i2.getName() + " is not an instance of ReportingData");

    Slot precisionCodeSlot = kb.getSlot("precisionCode");

    EffectiveDT effectiveDT1 = minMax(kb, i1, precisionCodeSlot);
    EffectiveDT effectiveDT2 = minMax(kb, i2, precisionCodeSlot);

    if ( effectiveDT1.max.before(effectiveDT2.min) )
        return -1;
    else if ( effectiveDT1.min.after(effectiveDT2.max) )
        return 1;
    else
        return 0;
}

private static EffectiveDT minMax(GH5KB kb, Instance i, Slot precisionCodeSlot) {
    Calendar effectiveDT = getEffectiveTime(kb, i);
    EffectiveDT dt = new EffectiveDT();
    dt.min = (Calendar)effectiveDT.clone();
    dt.max = (Calendar)effectiveDT.clone();

    if ( ! i.instanceOf(kb.getCls("ReportingDataAbsoluteTiming")) )
        return dt;     // More likely, we should be using start-precision-code.

    String precision = kb.getCodeSlot(i, precisionCodeSlot);
    if ( precision.equals("DAY") ) {
        dt.min.add(Calendar.HOUR_OF_DAY, -12);
        dt.max.add(Calendar.HOUR_OF_DAY, 12);
    }
    else if ( precision.equals("HR") ) {
        dt.min.add(Calendar.MINUTE, -30);
        dt.max.add(Calendar.MINUTE, 30);
    }
    else if ( precision.equals("MINUTE") ) {
        dt.min.add(Calendar.SECOND, -30);
        dt.max.add(Calendar.SECOND, 30);
    }
    else if ( precision.equals("MON") ) {
        int days = effectiveDT.getActualMaximum(Calendar.DAY_OF_MONTH)/2;
        dt.min.add(Calendar.DAY_OF_MONTH, -days);
        dt.max.add(Calendar.DAY_OF_MONTH, days);
```

```
        }
        else if ( precision.equals("NKN") ) {
            // No effect.
        }
        else if ( precision.equals("SECOND") ) {
            dt.min.add(Calendar.MILLISECOND, -500);
            dt.max.add(Calendar.MILLISECOND, 500);
        }
        else if ( precision.equals("WEK") ) {
            // 84 == half the hours in a week.
            dt.min.add(Calendar.HOUR_OF_DAY, -84);
            dt.max.add(Calendar.HOUR_OF_DAY, 84);
        }
        else if ( precision.equals("YEA") ) {
            int days = effectiveDT.getActualMaximum(Calendar.DAY_OF_YEAR)/2;
            dt.min.add(Calendar.DAY_OF_YEAR, -days);
            dt.max.add(Calendar.DAY_OF_YEAR, days);
        }
        else {
            throw new IllegalArgumentException("Instance " + i.getName() + " has invalid precision code \"" + precision + "\"");
        }
        return dt;
    }
    private static class EffectiveDT {
        Calendar min;
        Calendar max;
    }
}
```

6. Class RDComparator

There are two ways to compare two instances of REPORTING-DATA. Instances may be compared based on reporting date/time (the moment when an observation is made) or effective date/time (the moment when the observation becomes, became, or will become effective). Class RDComparator supports comparison based on reporting date/time.

The class supports the standard Java paradigm for comparison by implementing interface java.util.Comparator. This requires an implementation of method compare(), which, given two objects, returns an integer value denoting whether one object is equal to, less than,

or greater than the second. The IDA prototype implementation of compare() makes some arguable choices based on whether one or both instances have missing slot values. A production system will require a thorough examination of these choices.

Note that the comparison is of instances that have associated instances of REPORTING-DATA, not on instances of ReportingData.

```
package ida.sd.gh5ontology;

import ida.sd.ontology.*;

/**
 * The <code>RDComparator</code> class supports comparing GH5 class {@link Instance instances} that have an associated
 * <code>ReportingData</code> item based on reporting date/time.  Examples of  such classes include <code>ObjectItemLocation</code>
 * and <code>Holding</code>.
 * <p>Two instances are compared based on the values of their <code>reportingDate</code> and <code>reportingTime</code> slots.
 * <p>The primary secret of this class is the algorithm for comparison.
 **/
public class RDComparator
    implements java.util.Comparator {

    private final GH5KB _gh5KB;

    private final Slot _relatedReportingDataSlot;
    private final Slot _reportingDateSlot;
    private final Slot _reportingTimeSlot;
    private final Slot _tivSlot;

    /**
     * Constructs an <code>RDComparator</code> from a {@link Slot}.
     * @param gh5KB A GH5 knowledge base.
     * @param relatedReportingDataSlot A {@link Slot} whose value type is {@link Instance}, whose allowed class is
                <code>ReportingData</code>, and whose cardinality is 1.  It should be an inverse slot of a slot in <code>ReportingData</code>.
     **/
    public RDComparator(GH5KB gh5KB, Slot relatedReportingDataSlot) {
        _gh5KB = gh5KB;
        _relatedReportingDataSlot = relatedReportingDataSlot;
        _reportingDateSlot = gh5KB.getSlot("reportingDate");
        _reportingTimeSlot = gh5KB.getSlot("reportingTime");
        _tivSlot                = gh5KB.getSlot("type-instance-value");
    }

    /**
     * Constructs an <code>RDComparator</code> from a {@link Slot} name.
     * @param gh5KB A GH5 knowledge base.
```

```
 * @param relatedReportingDataSlotName The name of a {@link Slot} whose value type is {@link Instance}, whose allowed class is
 *           <code>ReportingData</code>, and whose cardinality is 1.  It should be an inverse slot of a slot in <code>ReportingData</code>.
 **/
public RDComparator(GH5KB gh5KB, String relatedReportingDataSlotName) {
    this(gh5KB, gh5KB.getSlot(relatedReportingDataSlotName));
}

/**
 * Implements the {@link java.util.Comparator#compare} method to compare two instances in a GH5 ontology.  It is assumed that:
 * <ol>
 * <li>Both <code>o1</code> and <code>o2</code> are instances of {@link Instance}.</li>
 * <li><code>((Instance)o1).getDirectType().equals((Instance)o2).getDirectType()</code> is true.</li>
 * <li>The slot (or slot name) given to the constructor of this instance is a slot of class <code>((Instance)o1).getDirectType()</code>.</li>
 * </ol>
 * @param o1 An object that is an instance of {@link Instance}.
 * @param o2 An object that is an instance of {@link Instance}.
 * @return Let <i>r</i><sub>1</sub> be the <code>ReportingData</code> associated with <code>o1</code>, and let <i>r</i><sub>2</sub>
 *           be the <code>ReportingData</code> associated with <code>o2</code>.  This method returns:
 *           <ul>
 *           <li>A negative integer if the <code>reportingDate</code> and <code>reportingTime</code> slots of <i>r</i><sub>1</sub> denote a
 *               moment before the <code>reportingDate</code> and <code>reportingTime</code> slots of <i>r</i><sub>2</sub>.</li>
 *           <li>0 if these slots denote the same moment.</li>
 *           <li>A positive integer if the <code>reportingDate</code> and <code>reportingTime</code> slots of <i>r</i><sub>1</sub> denote a
 *               moment after the <code>reportingDate</code> and <code>reportingTime</code> slots of <i>r</i><sub>2</sub>.</li>
 *           </ul>
 *           Lack of reporting data, or lack of reporting date or reporting time, is interpreted to be prior to presence thereof.  If both
 *           <code>o1</code> and <code>o2</code> lack reporting data, date, or time, they are considered equal.
 **/
public int compare(Object o1, Object o2) {
    Instance i1 = (Instance)o1;
    Instance i2 = (Instance)o2;
    Instance rdInst1 = (Instance)i1.getOwnSlotValue(_relatedReportingDataSlot);
    Instance rdInst2 = (Instance)i2.getOwnSlotValue(_relatedReportingDataSlot);

    if ( rdInst1 == null )
        return rdInst2 == null ? 0 : -1;
    else if ( rdInst2 == null )
        return 1;

    return ReportingData.compareByReportingDateTime(_gh5KB, rdInst1, rdInst2);
```

```
    }
}
```

7. Class RDEComparator

There are two ways to compare two instances of REPORTING-DATA. Instances may be compared based on reporting date/time (the moment when an observation is made) or effective date/time (the moment when the observation becomes, became, or will become effective). Class RDEComparator supports comparison based on effective date/time.

The class supports the standard Java paradigm for comparison by implementing interface java.util.Comparator. This requires an implementation of method compare(), which, given two objects, returns an integer value denoting whether one object is equal to, less than, or greater than the second. The implementation of compare() makes some arguable choices based on whether one or both instances have missing slot values. A production system would want to examine these choices.

Note that the comparison is of instances that have associated instances of REPORTING-DATA, not on instances of ReportingData.

```
package ida.sd.gh5ontology;

import java.util.Calendar;
import java.util.Comparator;

import ida.sd.ontology.*;

/**
 * The <code>RDEComparator</code> class supports comparing GH5 class {@link Instance instances} that have an associated
 * <code>ReportingData</code> item based on effective date/time.  Examples of such classes include <code>ObjectItemLocation</code>
 * and <code>Holding</code>.
 * <p>The primary secret of this class is the algorithm for comparison.
 **/
public class RDEComparator
    implements Comparator {

    private final GH5KB _gh5kb;

    private final Slot _relatedReportingDataSlot;

    /**
     * Constructs an <code>RDEComparator</code> from a {@link Slot}.
     * @param gh5KB A GH5 knowledge base.
     * @param relatedReportingDataSlot A {@link Slot} whose value type is {@link Instance}, whose allowed class is
     *          <code>ReportingData</code>, and whose cardinality is 1.  It should be an inverse slot of a slot in <code>ReportingData</code>.
     **/
```

```
public RDEComparator(GH5KB gh5KB, Slot relatedReportingDataSlot) {
    _gh5kb = gh5KB;
    _relatedReportingDataSlot = relatedReportingDataSlot;
}
/**
 * Constructs an <code>RDEComparator</code> from a {@link Slot} name.
 * @param gh5KB A GH5 knowledge base.
 * @param relatedReportingDataSlotName The name of a {@link Slot} whose value type is {@link Instance}, whose allowed class is
 *          <code>ReportingData</code>, and whose cardinality is 1.  It should be an inverse slot of a slot in <code>ReportingData</code>.
 **/
public RDEComparator(GH5KB gh5KB, String relatedReportingDataSlotName) {
    this(gh5KB, gh5KB.getSlot(relatedReportingDataSlotName));
}
/**
 * Implements the {@link Comparator#compare} method to compare two instances in a GH5 ontology.  It is assumed that:
 * <ol>
 * <li>Both <code>o1</code> and <code>o2</code> are instances of {@link Instance}.</li>
 * <li><code>((Instance)o1).getDirectType().equals((Instance)o2).getDirectType()</code> is true.</li>
 * <li>The slot (or slot name) given to the constructor of this instance is a slot of class <code>((Instance)o1).getDirectType()</code>.</li>
 * </ol>
 * @param o1 An object that is an instance of {@link Instance}.
 * @param o2 An object that is an instance of {@link Instance}.
 * @return Let <i>r</i><sub>1</sub> be the <code>ReportingData</code> associated with <code>o1</code>, and let <i>r</i><sub>2</sub>
 *          be the <code>ReportingData</code> associated with <code>o2</code>.  This method returns:
 *          <ul>
 *          <li>A negative integer if the <code>effectiveDate</code> and <code>effectiveTime</code> slots of <i>r</i><sub>1</sub> denote a
 *              moment before the <code>effectiveDate</code> and <code>effectiveTime</code> slots of <i>r</i><sub>2</sub>
 *          (if <i>r</i><sub>1</sub> is relative timing data, its absolute effective time is computed).</li>
 *          <li>0 if these slots denote the same moment.</li>
 *          <li>A positive integer if the <code>effectiveDate</code> and <code>effectiveTime</code> slots of <i>r</i><sub>1</sub> denote a
 *              moment after the <code>effectiveDate</code> and <code>effectiveTime</code> slots of <i>r</i><sub>2</sub>.</li>
 *          </ul>
 *          Lack of reporting data, or lack of effective date or effective time, is interpreted to be prior to presence thereof.  If both
 *          <code>o1</code> and <code>o2</code> lack reporting data, date, or time, they are considered equal.
 **/
public int compare(Object o1, Object o2) {
    Instance i1 = (Instance)o1;
    Instance i2 = (Instance)o2;
```

```
        Instance rdInst1 = (Instance)i1.getOwnSlotValue(_relatedReportingDataSlot);
        Instance rdInst2 = (Instance)i2.getOwnSlotValue(_relatedReportingDataSlot);

        if ( rdInst1 == null )
            return rdInst2 == null ? 0 : -1;
        else if ( rdInst2 == null )
            return 1;

        return ReportingData.compareByEffectiveDateTime(_gh5kb, rdInst1, rdInst2);
    }
}
```

8. Exceptions

Methods in the package gh5ontology encounter a common set of erroneous situations. The following subsections characterize these situations as exceptions.

8.1. Class InvalidCodeException

An InvalidCodeException signals an attempt to use the GH ontology improperly. For example, the caller is trying to set the value of an own slot whose value type is a coded domain, but has used a code that is not valid for that domain.

```
package ida.sd.gh5ontology;

import ida.sd.ontology.*;

/**
 * The <code>InvalidCodeException</code> class signals an attempt to set a code-valued slot to a code that is not in the legal set of values.
 **/
public class InvalidCodeException
    extends Exception {
    /**
     * Constructs an <code>InvalidCodeException</code> with a standard message.
     * @param cls A <code>Cls</code> for which an attempt was made to set the code.
     * @param slot The template slot of <code>cls</code> whose code was attempted to be set.
     * @param code The value to which <code>slot</code> was requested to be set.
     **/
    public InvalidCodeException(Cls cls, Slot slot, String code) {
        super(cls.getName() + "." + slot.getName() + ": Invalid code \"" + code + "\"");
    }
}
```

8.2. Class NoLocationException

Several methods in the GH5KB class expect a battlefield object to have a location. The NoLocationException class is used to signal that no location has ever been associated with a given battlefield object; that is, the value of the object's own slot is-geometrically-defined-through-ObjectItemLocation is null.

```
package ida.sd.gh5ontology;

/**
 * The <code>NoLocationException</code> class signals a situation where a  battlefield object was expected to have a location, but didn't.
 **/
public class NoLocationException
    extends Exception {
    /**
     * Constructs a <code>NoLocationException</code>.
     * @param objItemName The name of a battlefield object.
     **/
    public NoLocationException(String objItemName) {
        super(objItemName);
    }
}
```

8.3. Class OntologyLoadException

The GH5KB class contains some methods that assist in loading a GH knowledge base. The OntologyLoadException class is used to signal that loading failed for some reason.

```
package ida.sd.gh5ontology;

public class OntologyLoadException extends Exception {
    public OntologyLoadException(String message) {
        super(message);
    }

    public OntologyLoadException(Throwable th) {
        super(th);
    }
}
```

8.4. Class SpeedIndeterminateException

Several methods in the GH5KB class expect a battlefield object to have a speed. The SpeedIndeterminateException class is used to signal that no speed can be determined. It should be noted that speed can be determined in two ways. One is the value of the speedRate own slot of the most recent ObjectItemLocation instance (as determined by reporting data)ObjectItemLocation instance associated with a battlefield object. This value is preferred; but if absent, speed can also be computed from the last two locations of the object.

```
package ida.sd.gh5ontology;

/**
 * The <code>SpeedIndeterminateException</code> class signals an attempt to compute the speed of a battlefield object in a context where the
 * speed is indeterminate.
 **/
public class SpeedIndeterminateException
    extends Exception {
    /**
     * Constructs a <code>SpeedIndeterminateException</code>.
     * @param objItemName The name of a battlefield object.
     **/
    public SpeedIndeterminateException(String objItemName) {
        super(objItemName);
    }
}
```

8.5. Class StringLengthException

String-valued columns in the Generic Hub have a predefined maximum length. The GH ontology adheres to that length; it is enforced by classes in the gh5ontology package. A StringLengthException is thrown if the maximum length is violated.

```
package ida.sd.gh5ontology;

/**
 * The <code>StringLengthException</code> class signals an attempt to set a string-typed slot to a length longer than permitted.
 **/
public class StringLengthException
    extends Exception {
    /**
     * Constructs a <code>StringLengthException</code> with a standard message.
     * @param s The string that is too long.
     * @param maxLength The maximum number of characters permitted.
```

```
     **/
    public StringLengthException(String s, int maxLength) {
        super("\"" + s + "\" is longer than " + maxLength + " characters");
    }
}
```

9. Errors

The gh5ontology package contains several classes that extend Error.These classes denote situations that were deemed valuable in detecting implementation errors during debugging. The class OntologyDefinitionError, for example (which three classes in this section extend) indicates a problem in the ontology that must be corrected.

An important purpose of these classes is to standardize error reporting. The constructors often have combinations of parameters that are unusual for subclasses of Throwable.

On reflection, it would be better to make these classes extend Exception. A production system might encounter them through, say, a configuration error in which an old version of the GH ontology is used.

9.1. Class AllowedClsesCollectionSizeError

Some of the slots in the GH ontology have single cardinality; others have multiple cardinality. Code in the gh5ontology package tests whether slots have the expected cardinality, and throw an AllowedClsesCollectionSizeError if they do not.

```
package ida.sd.gh5ontology;

import java.util.Collection;

import ida.sd.ontology.Cls;
import ida.sd.ontology.Instance;
import ida.sd.ontology.Slot;

/**
 * Class <code>AllowedClsesCollectionSizeError</code> signals that the GH5 ontology has a slot that is expected to allow instances of only one
 * class but allows instances of multiple classes.  The purpose of this class is to encapsulate and standardize the messages associated with this
 * error.
 **/
public class AllowedClsesCollectionSizeError
    extends OntologyDefinitionError {
    /**
     * Constructs an <code>AllowedClsesCollectionSizeError</code> for a  situation where the {@link Slot} of some {@link Instance} was expected
     * to allow only one class but allows multiple.
```

```
     * @param inst The {@link Instance} being accessed when the error occurred.
     * @param slot The {@link Slot} being accessed when the error occurred.
     * @param allowedClses The collection of classes allowed by <code>slot</code>.
     **/
    public AllowedClsesCollectionSizeError(Instance inst, Slot slot, Collection allowedClses) {
        super(inst, slot, allowedClses.size() + " allowed classes (expected 1)");
    }

    /**
     * Constructs an <code>AllowedClsesCollectionSizeError</code> for a situation where the {@link Slot} of some {@link Cls} was expected to
     * allow only one class but allows multiple.
     * @param cls The {@link Cls} being accessed when the error occurred.
     * @param slot The {@link Slot} being accessed when the error occurred.
     * @param allowedClses The collection of classes allowed by <code>slot</code>.
     **/
    public AllowedClsesCollectionSizeError(Cls cls, Slot slot, Collection allowedClses) {
        super(cls, slot, allowedClses.size() + " allowed classes (expected 1)");
    }
}
```

9.2. Class InvalidClsError

Many methods in the gh5ontology package take a value of type Instance as a parameter. Because the ontology model is dynamic, the class of an instance can't be checked until runtime. Methods throw the InvalidClsError if the class is not what was expected.

```
package ida.sd.gh5ontology;

import ida.sd.ontology.Cls;

/**
 * The <code>InvalidClsError</code> class constructs an {@link java.lang.Error} denoting that some method was given a
 * {@link ida.sd.ontology.Cls}, or an {@link ida.sd.ontology.Instance} of a <code>Cls</code>, it didn't expect.
 * This is a programming error that must be corrected for execution to succeed.
 **/
public class InvalidClsError extends Error {
    /**
     * Constructs an InvalidClsError with a standard message.
     * @param expected The {@link ida.sd.ontology.Cls} that was expected.
     * @param received The {@link ida.sd.ontology.Cls} that was received.
     **/
    public InvalidClsError(Cls expected, Cls received) {
```

```
        super("Expected " + expected.getName() + ", received " + received.getName());
    }
}
```

9.3. Class OntologyDefinitionError

The OntologyDefinitionError class is the superclass of all classes that indicate an error in the definition of the GH ontology. In other words, these errors indicate a problem in the GH ontology (or knowledge base) as opposed to the code.

```
package ida.sd.gh5ontology;

import ida.sd.ontology.Cls;
import ida.sd.ontology.Instance;
import ida.sd.ontology.Slot;

/**
 * Class <code>OntologyDefinitionError</code> signals that the GH5 ontology does not conform to some expectation.
 **/
public class OntologyDefinitionError
    extends Error {
    /**
     * Constructs an <code>OntologyDefinitionError</code> based on a message.
     * @param message The message that is to be associated with the error.
     **/
    public OntologyDefinitionError(String message) {
        super(message);
    }

    /**
     * Constructs an <code>OntologyDefinitionError</code> for a {@link Slot} of a given {@link Instance}.
     * @param inst The instance that caused the error.
     * @param slot The slot of <code>inst</code> that caused the error.
     * @param message A description of the error.
     **/
    public OntologyDefinitionError(Instance inst, Slot slot, String message) {
        this(inst.getDirectType(), slot, message);
    }

    /**
     * Constructs an <code>OntologyDefinitionError</code> for a {@link Slot} of a given {@link Cls}.
     * @param cls The class that caused the error.
     * @param slot The slot of <code>cls</code> that caused the error.
```

```
     * @param message A description of the error.
     **/
    public OntologyDefinitionError(Cls cls, Slot slot, String message) {
        this(cls.getName() + "." + slot.getName() + ": " + message);
    }
}
```

9.4. Class TIVError

Many own slot values of GH ontology classes have a subclass of Type as their value type. Class Type has a template slot type-instance-value. Subclasses of Type specialize the value type of this slot, either into a built-in type (integer, float, string) or another class (e.g., Numeric-Expression). If a method finds that an instance of Type has a type-instance-value with some value type other than that expected, it will throw an instance of TIVError.

```
package ida.sd.gh5ontology;

import ida.sd.ontology.*;

/**
 * Class <code>TIVError</code> signals an error in the definition of a GH5 ontology related specifically to a misuse of the
 * <code>type-instance-value</code> slot.  The purpose of this class is to standardize the information associated with the error.
 **/
public class TIVError
    extends OntologyDefinitionError {
    /**
     * Constructs a <code>TIVError</code> for a situation where some class was expected to be the allowed class of the
     * <code>type-instance-value</code> slot, but the <code>type-instance-value</code> slot in question allows some other class.
     * @param inst The {@link Instance} being accessed when the error occurred.
     * @param slot The {@link Slot} being accessed when the error occurred.
     * @param expected The {@link Cls} that was expected to be allowed.
     * @param received The {@link Cls} that the <code>type-instance-value</code> slot of <code>slot</code> allows.
     **/
    public TIVError(Instance inst, Slot slot, Cls expected, Cls received) {
        super(inst, slot, " Expected " + expected.getName() + ", received " + received.getName());
    }
}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YY)	2. REPORT TYPE	3. DATES COVERED (From – To)
March 2005	Study	

4. TITLE AND SUBTITLE

A GH-Based Ontology to Support Applications for Automating Decision Support

5a. CONTRACT NUMBER

DASW01-98-C-0067
DASW01-02-C-0012

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBERS

6. AUTHOR(S)

Steven P. Wartik, Francisco L. Loaiza, Task Leader

5d. PROJECT NUMBER

5e. TASK NUMBER

BC-1-2445

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES

Institute for Defense Analyses
4850 Mark Center Drive
Alexandria, VA 22311-1882

8. PERFORMING ORGANIZATION REPORT NUMBER

IDA Paper P-3934

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQDA CIO/G-6
107 Army Pentagon, Room 1C670
Washington, DC 20310-0107

10. SPONSOR'S / MONITOR'S ACRONYM

ASD/CIO

11. SPONSOR'S / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release, unlimited distribution: 13 July 2005.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The United States Department of Defense is pursing net-centricity as its data strategy. A fundamental goal of net-centricity is that data be understandable to agents searching the Global Information Grid. IDA has studied the feasibility of making Command and Control data understandable by representing it using a formal ontology, and also by developing a prototype application that makes use of this ontology to search for and detect threats to a battlefield unit. The ontology is based on the Generic Hub, a NATO-standard information exchange data model; the data elements it contains are therefore real. Analysis of these elements illustrates both what can be represented using an ontology and the advantages and limitations of the Generic Hub as the basis for automated search. Many Generic Hub data elements model physical quantities. Such properties of these elements as units of measure and conceptual meaning can be formalized and interrelated through an ontology. Other elements, principally those that model abstract concepts, will require extensive input from subject matter experts to formalize. The prototype application shows how agents can use C2 information to assist decision makers. It was developed to execute using open-source systems. It simultaneously illustrates the potential of state-of-the-art technology and the relative immaturity of certain aspects of this technology. In particular, rule-based reasoning, which is seen as necessary for net-centricity, is not yet adequately supported by production-quality tools.

15. SUBJECT TERMS

C2IEDM, Generic Hub, Ontology, Automated Decision Support Application, Course of Action Analysis, Agents, Protégé.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Bruce Haberkamp
Unclassified	Unclassified	Unclassified	Unlimited	368	19b. TELEPHONE NUMBER (Include Area Code) (703) 697-1744