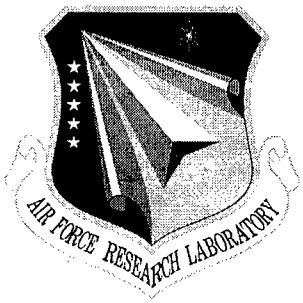
AFRL-IF-RS-TR-2000-12
Final Technical Report
February 2000



REUSABLE TOOLS FOR KNOWLEDGE BASE AND ONTOLOGY DEVELOPMENT

SRI International

Vinay K. Chaudhri

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-12 has been reviewed and is approved for publication.

APPROVED: Craig S. Anken

CRAIG S. ANKEN
Project Engineer

FOR THE DIRECTOR: Northrup Fowler

NORTHRUP FOWLER
Technical Advisor
Information Technology Division

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2000	Final Sep 96 - Jul 99

4. TITLE AND SUBTITLE
REUSABLE TOOLS FOR KNOWLEDGE BASE AND ONTOLOGY DEVELOPMENT

5. FUNDING NUMBERS
C - F30602-96-C-0332
PE - 63728F
PR - 2532
TA - 01
WU - 51

6. AUTHOR(S)
Vinay K. Chaudhri

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-12

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Craig S. Anken/IFTD/(315) 330-4833

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In the construction of a new knowledge base (KB), significant productivity gains can be obtained by reusing existing knowledge components. These components include pieces of domain knowledge and KB development tools. To support reuse of domain knowledge, the knowledge sharing community has undertaken various efforts, including the development of shared portable ontologies and the development of well-defined languages for knowledge interchange. There has been, however, less emphasis on the reuse of KB development tools. A significant amount of effort is invested in building customized tools for specific knowledge representation systems (KRSs). These tools work only with a single KRS, and the development effort is wasted if the KRS is no longer used. A KRS developer usually does not have the choice of using off-the-shelf tools and is forced to develop custom tools.

Open knowledge base connectivity (OKBC) is an application programming interface (API) for KPRSs that has been developed to address the problem of KB tools reusability. The name OKBC was chosen to be analogous to ODBC (Open Database Connectivity), as used in the database community.

This work experimented with several KB development tools to test the ability of OKBC to enable the construction of reusable tools. These tools included GKB-editor, a graphical tool for browsing and editing KBs, and PERK, a system for storing KBs in Oracle and for controlling multiuser access to KBs.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Knowledge Base, Artificial Intelligence, Reuse	40
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

List of Figures

List of Tables

Preamble

This project has investigated techniques for constructing reusable knowledge base (KB) development tools. A KB development tool is reusable if it can be used with multiple knowledge representation systems (KRSs). The primary technique employed for this purpose was Open Knowledge Base Connectivity (OKBC), which is an application programming interface (API) for accessing KRSs.

OKBC has improved upon its predecessor, the Generic Frame Protocol (GFP), in several significant ways. Some of our contributions to this improvement include the design for an assertional view of a KRS, the design of constraint checking operations, and a comparison with schema querying facilities of object-oriented databases.

We experimented with several KB development tools to test the ability of OKBC to enable the construction of reusable tools. These tools included GKB-Editor, a graphical tool for browsing and editing KBs, and PERK, a system for storing KBs in Oracle and for controlling multiuser access to KBs. We were able to more easily enable the reusability of GKB-Editor than of PERK. PERK was more difficult to reuse, because it needs to access many of the internal data structures of a KRS that are not exposed by OKBC. The most powerful result of this work was the OKBC knowledge model, which is reusable not just across KRSs but also across a range of object-oriented applications.

We conducted several other engineering efforts during the project. GKB-Editor was enhanced by providing a spreadsheet viewer and a World Wide Web (WWW) interface. We developed an initial prototype of a "conflict arbitration interface" for the collaboration subsystem of PERK. A substantial revision of the OKBC specification was undertaken during July 1997. The revision required us to invest a significant effort in upgrading our KB development tools that not only included GKB-Editor and Ocelot [PLK97], but also PERK and the collaboration system [KCP99]. These tools were upgraded during the current project.

The results of the project represent a substantial research and development activity. The tools developed are being extensively used in DARPA's High Performance Knowledge Bases (HPKB) project. GKB-Editor is being extensively employed to enable KB comprehension and reuse. The KB for project Genoa is being developed using GKB-Editor. OKBC is being used as an API by several participants in the HPKB program. We expect collaboration capabilities of PERK to be increasingly important in future projects.

The following publications were prepared under this project.

- Vinay K. Chaudhri and Peter D. Karp. "Querying Schema Information," 4th International Workshop on Knowledge Representation Meets Databases, August 1998 [CK97].
- Vinay K. Chaudhri,Adam Farquhar, Richard Fikes, Peter D. Karp and James P. Rice. "OKBC: A Foundation for Knowledge Base Interoperability," *Proceedings of the National Conference on Artificial Intelligence* [CFF+98].
- Alex Borgida, Vinay K. Chaudhri, and Martin Staudt. Report on the 5th International Workshop on Knowledge Representation Meets Databases, *ACM SIGMOD RECORD*. (27)3:10-15, September 1998 [BCS98].
- "A Collaborative Environment for Authoring Large Knowledge Bases and Ontologies." A draft of this paper was prepared under a previous contract from the Air Force Research Laboratory. Under the current contract, the draft was polished for submission to the *Journal of Intelligent Information System*, and the corrections requested by reviewers were incorporated [KCP99].

1 Introduction

In the construction of a new knowledge base (KB), significant productivity gains can be obtained by reusing existing knowledge components. These components include pieces of domain knowledge (for example, theories of economics or fault diagnosis) and KB development tools (for example, editors and theorem provers). To support reuse of domain knowledge, the knowledge sharing community has undertaken various efforts, including the development of shared portable ontologies [FFR97] and the development of well-defined languages for knowledge interchange [GF92]. There has been, however, less emphasis on the reuse of KB development tools. A significant amount of effort is invested in building customized tools for specific knowledge representation systems (KRSs). These tools work only with a single KRS, and the development effort is wasted if the KRS is no longer used. A KRS developer usually does not have the choice of using off-the-shelf tools and is forced to develop custom tools.

Open Knowledge Base Connectivity (OKBC) is an application programming interface (API) for KRSs that has been developed to address the problem of KB tools reusability. The name OKBC was chosen to be analogous to ODBC (Open Database Connectivity), as used in the database community [Gei95].

An API specifies the *operations* that can be used to access a system by an application program. When specifying an API for a KRS, some assumptions must be made about the representation used by that KRS. Such assumptions are made explicit in the *OKBC knowledge model*. As it can be too restrictive to enforce the same semantics for all operations in an API across all KRSs, OKBC supports *behaviors* to allow for differences among KRSs. Behaviors are a tool to achieve flexibility in specifying OKBC operations. Thus, the OKBC specification consists of three components: a knowledge model, a collection of operations to access a KRS, and a collection of behaviors.

A KRS can be *bound* to OKBC by defining a mapping from OKBC to the native API of that KRS. To achieve interoperability, a KB tool accesses a KRS using only OKBC operations. Such a tool is isolated from the peculiarities of the KRS and can be used with any KRS that has been bound to OKBC. The interoperability achieved by using OKBC is at the level of the OKBC knowledge model. For example, the OKBC knowledge model defines the concept of a *class* that has the same interpretation across all OKBC bindings. OKBC does not guarantee, however, that a particular class (e.g., `Person`) defined in KBs residing in two different KRSs represents identical concepts.

OKBC is a successor to the Generic Frame Protocol (GFP) [KMG95] and improves upon GFP in two significant ways. First, OKBC supports a larger class of KRSs because its knowledge model includes an *assertional view* of a KRS, provides an explicit treatment of entities that are not frames, and has a much better way of controlling inference and specifying default values. Second, OKBC can be used on practically any platform and with a substantially larger range of applications because it supports network transparency, multiple programming languages, and a remote procedure language.

We experimented with several KB development tools to test the ability of OKBC to enable the construction of reusable tools. These tools included GKB-Editor, a graphical tool for browsing and editing KBs, and PERK, a system for storing KBs in Oracle and for controlling multiuser access to KBs. We were able to more easily enable the reusability of GKB-Editor, because PERK needs to access many of the internal data structures of a KRS that are not exposed by OKBC. The most powerful result of this work was the OKBC knowledge model, which is reusable not just across KRSs but also across a range of object-oriented applications.

We conducted several other engineering efforts were undertaken during the project. GKB-Editor was enhanced by providing a spreadsheet viewer, and a World Wide Web (WWW) interface.

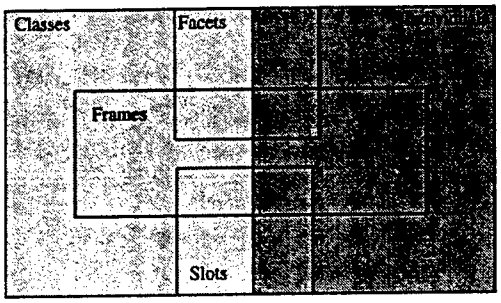


Figure 1: The OKBC knowledge model defines that classes and individuals form disjoint partitions of a KB. It does not commit to whether classes, individuals, slots, and facets are represented as frames. It also does not commit to whether slots and facets should be represented as classes or individuals.

We begin this report with a discussion of the technical issues faced in the design of OKBC. Then we describe the technical enhancements to GKB-Editor and PERK that were undertaken during this project. Finally, we discuss the ability of OKBC to enable the construction of reusable tools.

2 OKBC

The OKBC knowledge model is designed to include representational features supported by several KRSs [Kar92]. It includes constants, frames, slots, facets, classes, individuals, and knowledge bases. Classes and individuals form two disjoint partitions of a KB (see Figure 1). A *class* is defined as a set of entities. Each of the entities in a class is said to be an *instance* of that class. An *individual* is an entity that is not a set.

Any entity has associated with it a collection of *own* slots. Own slots describe the direct properties of an entity. For example, if the age of Fred is 42, then **age** is an own slot of **Fred**. Own slots and their values are not inherited. A class has associated with it a collection of *template* slots. Template slots describe properties of the instances of a class (own slots of a class describe the properties of the class itself). Template slots are inherited by subclasses of a class; a template slot on a class becomes an own slot on each instance of the class.

Own facets describe the properties of slots associated with an entity — for example, cardinality or range. A template slot of a class has associated with it a collection of *template* facets that describe own facets for the corresponding own slot of each instance of the class.

Orthogonal to the knowledge-level distinction between classes and individuals is the notion of a *frame*. A frame is a data structure that is typically used to represent a single entity and the slots and facets associated with it. The decision as to which entities are represented as frames is driven primarily by implementation considerations; historically, KRSs have made different decisions about it. The OKBC knowledge model does not legislate which entities are frames (see Figure 1). For example, in a given KRS, classes may or may not be represented as frames. Even if classes are generally represented as frames, OKBC allows for a subset of classes not to be represented as frames. Indeed, it is common for KRSs to exclude unnamed sets, such as {1, 2, 4}, and primitive data structures, such as numbers and strings, from the set of frames.

The OKBC knowledge model does not legislate whether slots and facets should be represented as classes or individuals. In some KRSs, a slot (or more generally, a relation) denotes a set of

tuples. In such systems, a slot is therefore also a class. In other KRSs, the spaces of classes and slots are disjoint. In such systems, a slot is also an individual. As shown in Figure 1, the knowledge model allows for a slot or a facet to be either a class or an individual.

A KB is a collection of classes, individuals, frames, slots, slot values, facets, facet values, frame-slot associations, and frame-slot-facet associations and sentences. Multiple KBs may be represented in a KRS.

OKBC supports operations that apply to specific frames in a KB (for example, querying the values of a slot of a frame), operations that apply to a KRS but not to any specific KB (for example, getting a list of all the KBs defined using a specific KRS), and operations that apply neither to a KRS nor to a KB (for example, establishing a connection to a knowledge server).

Although GFP was successfully used in several projects at Stanford University's Knowledge Systems Laboratory (KSL) [FFR97, FIFB96] and at SRI International [PLK97, KRPPT96], it lacked the power and flexibility needed in a generic API. Most of the enhancements considered here address the deficiencies encountered while GFP was used at KSL and SRI. In the rest of this section, we describe those enhancements.

2.1 Support for assertions

GFP was found to be inadequate for use with KRSs that prefer to view a KB as a collection of logical sentences, as well as systems that have a knowledge model more expressive than the knowledge model of GFP. To address these problems, we introduced a *tell/ask* interface that supports an *assertional view* of a KB.

The design approach for supporting OKBC was analogous to the one adopted in KRYPTON [BFL83]. An OKBC KB supports two alternative and isomorphic views of a KB: a frame-oriented view and an assertional view. (The frame-oriented view was called the *terminological component* in KRYPTON.) While defining the assertional view of a KB, we took a lowest common denominator approach: an assertion language with an expressive power roughly equivalent to an object-oriented frame language is defined. For other assertions, support is provided, but no portability claims are made.

2.1.1 Assertion language

OKBC defines an assertion language (AL) for declarative specification of knowledge. The AL is a first-order language with conjunction and predicate symbols, but without disjunction, explicit quantifiers, function symbols, negation, or equality. The predicate symbols of the OKBC AL are `class`, `individual`, `primitive`, `instance-of`, `type-of`, `subclass-of`, `slot-of`, `facet-of`, `template-slot-of`, `template-facet-of`, `own-slot-value`, `own-facet-value`, `template-slot-value`, and `template-facet-value`. For example, (`instance-of John Person`) means that `John` is an instance of the class `Person`. For convenience, (`instance-of John Person`) may be written as (`Person John`).

A well-formed formula (WFF) of the AL is an atomic formula constructed by enclosing one of the predicate symbols followed by a number of terms in parentheses. The terms of the AL are constants and variables. The conjunction of two WFFs of the AL is a WFF.

OKBC provides the **tell**, **ask**, and **untell** operations to query and update a KB using the AL. OKBC guarantees only that ground WFFs can be **tell**ed or **untell**ed. Any WFF may be **ask**ed.

OKBC specifies the effect of **tell**ing any WFF of the AL to a KB by identifying an equivalent set of OKBC operations that does not include **tell**. For example, the operation (**tell** (`instance-of`

frame class)), which asserts *frame* to be an instance of *class*, is equivalent to the operation (**add-instance-type** frame class). **Asking** any WFF of the AL is similarly equivalent to a set of OKBC operations not including **ask**. For example, the operation (**ask** (`instance-of` ?x class)) is equivalent to the operation (**get-class-instances** class).

2.1.2 Assertions not guaranteed to be supported by OKBC

To handle assertions outside of the AL, OKBC defines the operations **tellable** and **askable**. The OKBC operation **tellable** determines which sentences may be acceptable to **tell** for a specific KB. Before using **tell** with an arbitrary formula, an application can check whether a formula is **tellable**. If the formula is not **tellable**, the application cannot safely assert that formula using **tell**.

For example, consider the WFFs (`age John` 30) and (`friend John Sally`). It is straightforward to assert them using either **tell** or **add-slot-value**. An application may, however, wish to assert the disjunction, (`or` (`age` John 30) (`friend` John Sally)), which is not a WFF of the AL. An OKBC binding for a KRS is free to accept this formula, and an application can check for this by using the **tellable** operation. If a formula is **tellable** for a KRS, the **tell** operation can be used to communicate that formula to the KRS. Using this mechanism, a KB may accept formulae that contain quantifiers, functions, or higher-arity predicates.

There is no equivalence between using **tell** with arbitrary formulae and a set of OKBC operations that do not use **tell**. Use of such formulae may, therefore, not be portable across different OKBC bindings.

2.2 Handling entities that are not frames

As discussed above, KRSs make different assumptions about which entities are represented as frames. These differences influence the semantics of operations that systematically process frames in a KB (for example, **get-kb-frames**, **get-kb-individuals**, **get-kb-classes** that respectively return all the frames, classes, and individuals in a KB). These operations could be specified by saying that they respectively return all the frames, classes, and individuals in a KB. Because of differences in which entities are represented as frames, this simplicity can be deceptive.

2.2.1 Not all entities are frames

As shown in Figure 1, not all classes in a KB are necessarily represented as frames. Given such differences, it is not obvious how to define the operation **get-kb-classes**. Should it return only those classes that are frames? Should it return all sets in a KB?

Returning only those classes that are frames is a problem for KRSs that do not represent all classes as frames. Some of the non-frame classes can be important to a client application. Defining **get-kb-classes** to return all the sets is also problematic because the results of one OKBC operation cannot necessarily be passed to another operation, making an application program more complex. The complexity occurs because it is generally not possible to perform operations such as creating slots and adding slot values to entities that are not frames. Thus, if **get-kb-classes** were to return classes that are not frames, an application program would need to identify those classes that are not frames and treat them differently. A possible solution to this problem is to require a KRS to appear as if it represents every class as a frame. This is not reasonable, however, because it is unnatural and can make the implementation extremely inefficient.

To address this problem, we introduced an extra argument, **selector**, to **get-kb-classes** and similar operations. When the value of **selector** is `:frames`, only those classes that are frames are

returned, and when its value is `:all`, all classes are returned. For a system in which all classes are represented by frames, **get-kb-classes** returns identical results for these two values of **selector**. We expect many applications to use `:frames` as a value for the **selector** argument, because it has the desirable property that the union of **get-kb-classes** and **get-kb-individuals** equals the result of **get-kb-frames**. A third legal value for this argument is `:system-default`, which gives a KRS the freedom to use the most efficient or natural method of computing **get-kb-classes**.

2.2.2 Not all entity categories are frames

As shown in Figure 1, not all KRSs represent all categories of entities as frames. Consider two KRSs: KRS1, which represents slots as frames, and KRS2, which does not. Furthermore, consider KB1, stored in KRS1, and KB2, stored in KRS2, which were created using identical sets of OKBC creation operations. Calling an operation such as **get-kb-frames** will return different results on KB1 and KB2. This may make it more difficult for an application to work portably with both KBs, but it is acceptable if OKBC provides a mechanism to detect the difference. The `:are-frames` behavior allows a KRS to indicate which categories of entities are represented as frames.

The values of the `:are-frames` behavior constitute a set of keywords: `:class`, `:slot`, `:facet`, and `:individual`. If the values of `:are-frames` contain an entity category, it implies that frames may be used to represent them. In most KRSs, classes and instances of those classes are represented as frames, and therefore we expect the most common set of values for `:are-frames` to be at least {`:class`, `:individual`}. If two KRSs have different values for the behavior `:are-frames`, we can expect to get a different list of frames by executing **get-kb-frames** on these KBs.

2.3 Systems that do not support named frames

Many KRSs, for example, LOOM [MB91] provide unique symbolic names for every frame in a KRS. Other KRSs, for example, Ontolingua [FFR97], require unique object identifiers in a KB, but not unique symbolic names. Still other KRSs allow for anonymous frames [FIFB96]. A KRS may advertise the support for frame names by the `:frame-names-required` behavior. When the value of the `:frame-names-required` behavior is *true*, it means that each frame in the KRS has a name, each frame name is unique in the KB, and the frame name supplied at the time of creating a frame can be used at a later time to locate that frame. When the value of `:frame-names-required` is *false*, frames are not required to be named, and a frame name, if supplied, may not necessarily be unique in a KB. One may not be able to locate a frame unambiguously by using the name that was supplied when the frame was created.

Any portable application whose behavior depends on the existence of unique frame names should first query the value of the `:frame-names-required` behavior and, if it is found *false*, should provide an alternative implementation for functionality that depends on frame names.

2.4 Controlling KRS inference

One area in which KRSs differ widely is in the inference mechanisms that they support and in the methods available to control the inference mechanisms. It is critical for applications to have some means of controlling the type and cost of inferences that a KRS performs in response to a retrieval operation. Unfortunately, there is not yet widespread agreement on either the inference mechanisms or the parameters used to control them. This makes it impossible for OKBC to provide a rich KRS-independent method for controlling inference. Instead, OKBC provides a restricted method for specifying which inferences should be performed in retrieval operations, as

well as methods for a KRS to indicate the degree to which the specifications have been satisfied. OKBC does not provide means to specify limits on computing time in performing those inferences.

OKBC retrieval operations support an **inference-level** argument that takes one of the three values: `:direct`, `:taxonomic`, or `:all-inferable`. When **inference-level** is `:direct`, *at least* the directly asserted nonredundant values are returned. When **inference-level** is `:taxonomic`, *at least* the directly asserted and inherited values are returned. The inherited values are computed using at least the taxonomic inheritance axioms defined by the knowledge model. For example, a taxonomic inheritance axiom for slot values states that if a template slot S of a class C has value V, then for all instances of C, the own slot S has value V, and for all subclasses of C, the template slot S has value V. Similar inheritance axioms are defined for facet values, and for the class/subclass and class/instance relationships. When **inference-level** is `:all-inferable`, values inferable by any means supported by the KRS are returned, including any values inferable at the `:taxonomic` inference level.

With an **inference-level** value of `:direct`, returning exactly the directly asserted values may impose a high burden on some systems, such as forward chaining systems that do not maintain a distinction between directly asserted and inferred values. To permit flexibility in such cases, we use the following two techniques.

First, the **inference-level** argument defines the lower bound on the values that may be returned. For example, when the **inference-level** is `:direct`, *at least* the directly asserted values are returned, but a KRS is not prevented from returning additional values. Second, any OKBC operation accepting the **inference-level** argument returns two additional values, called **exact-p** and **more-status**. The value of **exact-p** is *true* if it is known that exactly the `:direct` (or `:taxonomic`) values are returned. An OKBC implementation that always returns *false* as the value of **exact-p** is compliant. The value of **more-status** is either *false*, which indicates that there are known to be no more results, or `:more`, which indicates that there may still be more results but the KRS was unable to find out how many more, or an integer, which indicates how many more values exist.

By specifying the inference level in terms of the lower bound on the result and returning two additional values, **exact-p** and **more-status**, we were able to permit flexibility in the specification and also be accurate.

2.5 Handling defaults

In the absence of any widely accepted model of defaults [BDK97], OKBC incorporates only simple provisions for default values of slots and facets. Template slots and template facets have a set of *default values* associated with them. Intuitively, these default values inherit to instances unless the inherited values are logically inconsistent with other assertions in the KB, the values have been removed, for example, at the instance, or the default values have been explicitly overridden by other default values. OKBC does not require a KRS to determine the logical consistency of a KB, nor does it guarantee a means of explicitly overriding default values. Instead, OKBC leaves the inheritance of default values unspecified. That is, no requirements are imposed on the relationship between default values of template slots and facets and the values of the corresponding own slots and facets. The default values on a template slot or template facet are simply available to the KRS to use in whatever way it chooses when determining the values of own slots and facets. The slot or facet values that are not default values are referred to as "known true" values. Operations on slot and facet values take a **value-selector** argument that allows a user to choose between only default values and monotonic ("known true") values.

2.6 OKBC compliance

Many OKBC back end implementors are interested in implementing only a subset of the functionality specified by OKBC. Two such examples are the CLIPS back end being developed at the Section of Medical Informatics (SMI) at Stanford, and the plan monitoring system being developed by Jon Doyle at MIT.

We studied the requirements of Jon Doyle's plan monitoring system. It is a system with fixed schema — that is, no schema changes are allowed. The systems with fixed schema form a useful class of applications, because schema changes are inherently difficult, and many systems choose not to deal with them. Motivated by this, we have proposed a new compliance class called `:fixed-schema`. Adding a new compliance class means allowing an additional legal value for the behavior `:compliance`. No new operations need to be added.

An application that wishes to be compliant in the fixed schema compliance class will specify `:fixed-schema` as one of the values of the `:compliance` behavior. For an OKBC implementation to be compliant in the `:fixed-schema` class, none of the schema change operations is required. A system compliant in the `:fixed-schema` class, however, is free to implement any schema change operations it wishes to support. The `:fixed-schema` compliance class reduces the number of mandatory methods that need to be implemented for full compliance.

The `:fixed-schema` compliance class is more general than the `:read-only` compliance class, because it allows updates on slot and facet values that are not permitted under the `:read-only` compliance class. But it is not more general than the `:monotonic` compliance class, as retractions are allowed but new schema information cannot be added. The inclusion relationships between various compliance classes are shown in Figure 2.

The `:fixed-schema` compliance class was presented at the meeting of the OKBC working group held in conjunction with the December 1997 meeting of the HPKB program, where concerns were expressed regarding the way compliance classes are currently specified. The current specification allows a compliance class to be flexible by allowing more operations than it promises. For example, a system in the `:fixed-schema` compliance class is defined to support at least those operations that do not involve any schema change. The `:read-only` compliance class is defined analogously, that is, the system supports at least all the read-only operations. Such specification allows flexibility, but it makes the behavior of a system in a compliance class less predictable. For example, a system in the `:read-only` compliance class may support updates. The relative merits of these two specifications remain unexplored.

2.7 Support for polymorphism

The Interface Definition Language (IDL) supported by CORBA supports slot polymorphism, that is, it allows two slots representing different things, but belonging to different classes, to have the same name. For example, the slot "sharpness" can mean taste of food for a class of foods, as well as the sharpness for a class of knives. OKBC handles such cases by requiring an implementor to set the `:frame-names-required` behavior to false. In IDL, slot names are not necessarily unique, but slots can be uniquely identified in the context of a frame. Furthermore, frame names are always unique. Therefore, the current solution offered by OKBC is inadequate for IDL.

Since IDL is being actively used in DARPA's JFACC program, we undertook a preliminary investigation to provide support for polymorphism. Our solution requires three extensions to the OKBC specification.

1. Generalizing the value of `:frame-names-required` behavior

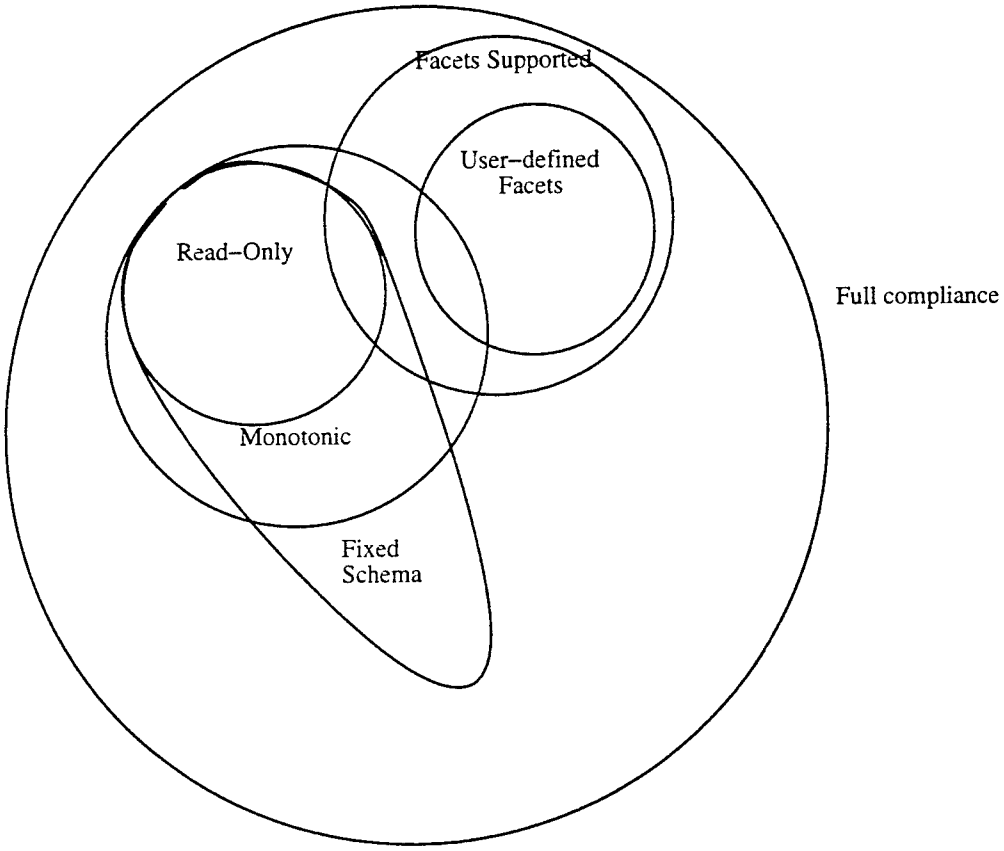


Figure 2: Inclusion relationship among compliance classes

Recall that the :frame-names-required behavior allows a KRS to advertise its support for frame names. In the current design, when the value of :frame-names-required is **true**, each frame is required to have a name, each frame name is unique in the KB, and the frame name supplied at the time of creating a frame can be used at a later time to locate the frame by using **coerce-to-frame** until changed by **put-frame-name**. When the value of :frame-names-required is **false**, frame names are not required, and may be nonunique in a KB. One may not be able to locate a frame by using the name that was supplied when the frame was created.

In the proposed design, the :frame-names-required behavior should have values from the set of entity types: :class, :individual, :slot, :facet. If an entity type is a value of this behavior, frame names are required for the frames of that entity type.

For IDL, one can set :frame-names-required to {:class, :individual}. Class frames and individual frames will be required to have unique names, but slots and facet frames may not have unique names.

2. Introducing a new behavior :coercion-type

   The new behavior :coercion-type will specify how an OKBC implementation coerces the frame, slot, and facet arguments. The behavior can have three values.

   - :none - Arguments are not coerced.
   - :with-context - A slot argument is coerced in the context of frame. The context of coercion is discussed in more detail below.
   - :non-context - All arguments are coerced without any context. So, slot name is coerced to a slot without the frame context. This setting makes sense only when frame names are required for all entity types.

   The context of coercion is specified by an additional argument to **coerce-to-slot** operation. For example,

```
(coerce-to-slot 'slot-1 :context c1 :slot-type :template :error-p nil)
#<slot slot-1 in c1>
  T
```

   When :coercion-type is :with-context, an application does not have to necessarily supply slot handles or slot objects as a value for the slot argument, thus providing a natural support for slot name polymorphism.

3. Adding a context argument to **coerce-to-slot** operation

   The operation **coerce-to-slot** will take a frame argument. When a frame argument is supplied, the slot object or slot handle representing that slot in the given frame is returned.

   With the frame argument, one can invoke the operations

```
(coerce-to-slot ''S'' :frame  ''A'')
(coerce-to-slot ''S'' :frame ''B'')
```

and get different results, which will be the slot object or slot handle representing the slot "S" in the two frames.

If the `:frame-names-required` does not contain `:slot` as a value, the above coercion will be required to work, but not the following coercion:

```
(coerce-to-slot ''S'')
```

The acceptability of the above proposal is yet to be seen, and remains open for future work.

2.8 Checking constraints

Many KRSs provide runtime slot-value constraint checking. Each time a slot value is changed (either locally, or through a change to inherited values), the KRS evaluates constraints that have been defined by the user to specify what values are allowable for a given slot.

Constraint checking is described by two behaviors: `:constraint-checking-time` controls when constraint checking should be performed, and `:constraint-report-time` controls when constraint violations should be reported to the user, as follows.

Allowable values of `:constraint-checking-time` are

- `:immediate` — Constraints are checked as soon as any sideeffect causing OKBC operation is executed.
- `:deferred` — Constraint checking does not occur when a sideeffect causing OKBC operation is executed, but is delayed until either the **check-constraints** operation is executed explicitly or the **constraint-checking-type** is changed to `:background` or `:immediate`.
- `:background` — Constraint checking is performed as a background process. Violated constraints may be retrieved using the **get-pending-constraint-violations** operation.
- `:never` — Constraints are never checked.

OKBC allows control of the constraint checking facilities provided by the underlying KRS. A KRS defines a `:constraint-checking-time` behavior, whose value is a list of possible constraint checking policies including `:immediate`, `:deferred`, `:never`, and `:background`. The operation **check-constraints** may be used to trigger constraint checking, and **set-constraint-checking-time** may be used to modify the policy, if allowed by the KRS.

For systems that support deferred constraint checking, we propose the following OKBC operations. If any of the following operations is invoked on a system that does not support user-controlled constraint checking, it does not have any effect when **error-p** is **nil**, and returns an error when **error-p** is **true**.

set-constraint-checking-time checking-time &key kb error-p kb-local-only-p

Using **set-constraint-checking-time**, a user can query or update the time of constraint checking. The valid values for **checking-time** are `:immediate`, `:deferred`, `:background`, and `:never`. When **error-p** is **t**, an error is signaled when called with an unsupported value of **checking-time**.

get-constraint-checking-time &key kb error-p

The **get-constraint-checking-time** operation returns the current value of constraint checking time.

check-constraints &key frame slot facet kb (checking-time :now) (error-reporting-time :immediate) (error-p t)

The **check-constraints** operation initiates the checking of constraints in the **kb**. If the **frame** argument is specified, only those constraints are checked which are *attached* to that frame. The constraints attached to a frame are either those expressed using the facet mechanism or the ones that are explicitly attached to that frame in a **tell** operation. If the **slot** argument is specified, constraints on only that slot are checked. If the **facet** argument is specified, only that facet constraint is checked. Any combination of **frame**, **slot**, and **facet** may be specified for better control of constraint checking. For example, if **slot** and **facet** are supplied, for all frames in the KB, **slot** is checked for the constraint specified by **facet**. The **checking-time** can be `:now` or `:background`. If the **checking-time** is `:background`, a background process may be initiated to check the constraints on an ongoing basis. The constraint violations are signaled as condition objects (see below). The possible values for **error-reporting-time** are `:immediate` and `:deferred`. If **:error-reporting-time** is **:deferred**, constraint violations are stored for later access using the OKBC operation **get-pending-constraint-violations** described later. If **checking-time** is **:background**, **:error-reporting-time** defaults to **:deferred**. The **error-p** argument is used to control the behavior of **check-constraints** in case any error other than constraint violation is encountered.

with-deferred-constraint-checking &key frames slots facets kb &body body
This macro is equivalent to the following:

```
(let ((constraint-checking-time (get-constraint-checking-time)))
   (unwind-protect
     (progn
      (set-constraint-checking-time :checking-time :deferred ...)
      body
      (check-constraints  :checking-time :now ...))
    (set-constraint-checking-time :checking-time constraint-checking-time)))
```

It evaluates the forms in **body** by switching off the constraint checking. At the end of the execution of the body, the constraints are checked. The arguments **frames**, **slots**, and **facets** have the effect as described with the **check-constraints** operation. On exit from the macro, the current value of constraint checking time is restored to its original value.

After working with the KB for some time, a user may want to determine the pending constraint violations. To return pending constraint violations, we propose the following OKBC operations.

get-pending-constraint-violations &key kb

Pending constraint violations is the set of violations that were detected at the time of executing a previous **check-constraints** operation or any violations discovered by the background constraint checking process. The function returns a list of condition objects representing the pending constraint violations.

remove-pending-constraint-violations &key condition-object kb

This operation removes the **condition-object** from the list of pending violations. If **condition-object** is **nil**, all pending violations are removed.

When a constraint is violated, KRSs usually signal conditions. We propose three kinds of conditions to signal constraint violations: `constraint-violation`, `valuetype-violation`, and `cardinality-violation`. The most general kind of signal for constraint violation is the `constraint-violation` condition. LOOM detects constraint violations only when slot values are added using the functions `set-value` and `fset-value`. When slot values are added using `tell` or `tellm`, a constraint violation is signaled by displaying a warning. Classic has an extensive set of conditions that are signaled whenever there is a constraint violation. For example, an `inconsistent-interval-conflict` condition is signaled when a host concept has an inconsistent interval (i.e., the value of the `:numeric-minimum` facet is greater than the value of the `:numeric-maximum` facet).

We propose that all update OKBC operations should signal an appropriate condition whenever an error is encountered. For example, if `add-slot-value` operation violates a maximum cardinality constraint, the `maximum-cardinality-violated` condition should be signaled. OKBC conditions are objects that in case of constraint violation contain the frame, slot, value, and an informational message describing the error.

We believe that conditions provide a useful service to the user, and therefore, we should expand the current set of conditions that are signaled when a constraint is violated. We propose to define a condition for each facet. For example, when the `:same-values` facet is violated, the `:same-values-violated` condition is signaled.

The constraint checking operations proposed here are not yet part of the OKBC specification. We need to spend more time studying their properties and acquire some experience in using them before they can be considered ready for inclusion in OKBC.

2.9 Comparing OKBC knowledge model with object-oriented databases

Even though the OKBC knowledge model is not as expressive as many knowledge representation languages, it supports many features not yet common in database systems. For example, schema information can play an important role for formulating a query and while information is retrieved from multiple sources. Yet, most database systems have a primitive support for querying the schema information. We undertook a small study to compare the schema querying capabilities supported by OKBC to those found in object-oriented databases.

2.9.1 Schema queries in OKBC

OKBC defines a collection of methods to query schema information. These methods can be classified into three broad categories: taxonomic queries, frame structure queries, and constraint queries. A fourth class of schema queries is the *class comparison query* which was supported in GFP but was dropped from OKBC (for the reasons explained below). Table 1 lists a subset of OKBC/GFP methods in each category.

The *taxonomic* queries allow us to query the class-subclass relationships. For example, `get-class-subclasses` allows us to determine all the subclasses of a class. The root classes can be determined using `get-kb-root-classes`.

The *frame structure* queries retrieve the slots and facets associated with a frame. For example, `get-frame-slots` returns all the slots associated with a frame. (A formal definition of what it means for a slot to be associated with a frame can be found elsewhere [CFF+97].)

GFP Method	Brief Description
Taxonomic Queries	
`get-class-subclasses`	Returns a list of direct subclasses of a class
`get-class-superclasses`	Returns a list of direct superclasses of a class
`get-root-classes`	Returns a list of those classes that have no superclass
Frame Structure Queries	
`get-frame-slots`	Returns a list of all the slots of a frame
`get-frame-facets`	Returns a list of all the facets of a frame
Constraint Queries	
`get-facet-value`	Returns the value of a facet
`get-slot-facets`	Returns the list of facets applicable to a frame
Class Comparison Queries	
`equivalent-p`	Given classes class1 and class2, returns true when the extensions of class1 and class2 are the same
`consistent-classes-p`	Given classes class1 and class2, returns true if an instance could satisfy the definition of both classes simultaneously
`class-disjoint-p`	Given classes class1 and class2, returns true if they are incompatible, that is, an instance could not satisfy the definition of both classes simultaneously

Table 1: Querying Schema Information by Using OKBC/GFP

The *constraint* queries allow us to query the facet information. As stated earlier, facets are used to represent constraints on slot values. Currently supported facet names are `:VALUE-TYPE`, `:CARDINALITY`, `:MINIMUM-CARDINALITY`, `:MAXIMUM-CARDINALITY`, `:INVERSE`, `:NUMERIC-MINIMUM`, `:NUMERIC-MAXIMUM`, `:SAME-VALUES`, `:SOME-VALUES`, `:NOT-SAME-VALUES`, `:SUBSET-OF-VALUES`, and `:COLLECTION-TYPE`. an informal definition of the facets is shown in Table 2.9.1. More formal definitions of facets are available elsewhere [CFF+97]. By using `get-facet-values` on the facets `:numeric-minimum` and `:numeric-maximum`, the range constraints on a slot can be obtained.

The *class comparison* queries support the inferences usually available only in description logic systems, such as LOOM [Mac91] and Classic [BBMR89]. For example, `equivalent-p` returns true for two classes class1 and class2 when the extensions of class1 and class2 are the same. The class comparison queries were dropped from OKBC, because at that time it was not clear to us how to specify portable definitions for these queries. In fact, it is straightforward to *structurally* define these operations by simply considering the class definitions: `equivalent-p` can examine the definitions of two classes and determine if they are equivalent in the sense that they have the same slots, slot values, facets, and facet values. The structural definition of `equivalent-p` gets around the need to make guarantees about the complete extension of a class, and returns a result based on the schema information explicitly asserted in the KB.

2.9.2 Enhancing a database query language to query schema

We believe that a database query language should provide natural support for the four classes of queries considered in the previous section. We first briefly discuss the schema queries supported in the current relational database management systems (DBMS) products and then propose a scheme to enhance the Object Query Language (OQL) to support schema queries. (OQL is commonly

Facet Name	Description
:VALUE-TYPE	A value C for facet :VALUE-TYPE of slot S of frame F means that every value of slot S of frame F must be an instance of the class C.
:INVERSE	A value S2 for facet :INVERSE of slot S1 of frame F means that if V is a value of S1 of F, then F is a value of S2 of V.
:CARDINALITY	A value N for facet :CARDINALITY on slot S on frame F means that slot S on frame F has N values.
:MAXIMUM-CARDINALITY	A value N for facet MAXIMUM-CARDINALITY of slot S of frame F means that slot S of frame F can have at most N values.
:MINIMUM-CARDINALITY	A value N for facet MINIMUM-CARDINALITY of slot S of frame F means that slot S of frame F has at least N values.
:SAME-VALUES	A value S2 for facet :SAME-VALUES of slot S1 of frame F, where S2 is a slot, means that the set of values of slot S1 of F is equal to the set of values of slot S2 of F.
:NOT-SAME-VALUES	A value S2 for facet :NOT-SAME-VALUES of slot S1 of frame F, where S2 is a slot, means that the set of values of slot S1 of F is not equal to the set of values of slot S2 of F.
:SUBSET-OF-VALUES	A value S2 for facet :SUBSET-OF-VALUES of slot S1 of frame F, where S2 is a slot, means that the set of values of slot S1 of F is a subset of the set of values of slot S2 of F.
:NUMERIC-MINIMUM	A value N for facet :NUMERIC-MINIMUM on slot S on frame F means that the minimum value of slot S on frame F is N.
:NUMERIC-MAXIMUM	A value N for facet :NUMERIC-MAXIMUM on slot S on frame F means that the minimum value of slot S on frame F is N.
:SOME-VALUES	A value V for own facet :SOME-VALUES of own slot S of frame F means that V is also a value of own slot S of F.
:COLLECTION-TYPE	The :COLLECTION-TYPE facet specifies whether multiple values of a slot are to be treated as a set, list, or bag.

Table 2: Facets Supported by OKBC

```
interface Person
(extent People)
{
        attribute String name;
        attribute Struct Address {Short number, String Street} address;
        relationship Person spouse inverse Person::spouse;
        attribute Integer age;
        relationship Set<Person> children inverse Person::parents;
        relationship List<Person> parents inverse children;
};
```

Table 3: A Sample Class Definition

Table 4: Example Constraint Queries. The `NUMERIC-MINIMUM` facet cannot be determined using the schema of Figure 2.9.2.

Query	Expected Result
`select facet(p.name, value-type) from person p`	string
`select facet(p.spouse, value-type) from person p`	`Person`
`select facet(p.children, value-type) from person p`	`Set<Person>`
`select facet(p.children, collection-type) from person p`	`List`
`select facet(p.children, inverse) from person p`	`parents`
`select facet(p.age, numeric-minimum) from person p`	0

adopted by object-oriented databases.)

The analog of a frame structure query such as `get-frame-slots` for a relational DBMS is to obtain a list of all the attributes of a relation. Traditionally, such queries have been answered by using the information in the data dictionary of a DBMS. The analog of a constraint query such as `get-facet-value` is to obtain the key of a relation or to obtain constraints attached to a relation. Even though constraint queries can be answered by querying the data dictionary, the current products do not offer the flexibility we propose. For example, in Oracle DBMS, it is possible to query the constraints associated with a table, but those constraints are returned as a string. Thus, if the numeric value of an attribute *Cost* was restricted to a positive integer, we will be returned the string "Cost $\geq$ 0". We then need to parse the result to determine that it represents a `:numeric-minimum` facet of OKBC. The class comparison queries are outside the scope of relational DBMSs. Thus, RDBMSs support only a subset of schema queries that are useful for querying multiple sources.

Release 1.2 of OQL did not provide any support for taxonomic queries [Cat95]. To some extent, the problem will be rectified in the upcoming Release 2.0 of ODMG [Cat97], as the new data model includes a class *MetaObject* that has a relation called *DefiningScope* that will allow OQL to query the class-subclass relationships.

A possible way to incorporate taxonomic queries in OQL is to view each class relationship as a

relation as proposed in XSQL [KKS92]. For example, the following XSQL query allows a variable to range over a class. XSQL syntax uses #X to distinguish the variables that range over classes.

```
SELECT #X WHERE Person subclassOf #X
```

The above query returns all the subclasses of the class `Person`. Each OKBC method corresponding to taxonomic queries can be represented as a relation in an OQL query to provide a comprehensive support for querying the schema information. For example, `get-class-superclasses` can be represented by the relation superclassOf. Then the following query returns all superclasses of a class.

```
SELECT #X WHERE #X superclassOf Person
```

A similar technique can be used for supporting frame structure queries if we allow #X to represent a variable that ranges over attributes. For example, if we assume that the relation attributeOf represents the association of an attribute with a class, then the query

```
SELECT #X WHERE  #X attributeOf Person
```

returns all the attributes of the class Person. Alternative syntax for querying slots is possible [LSS96], and investigation of the relative syntax merits is left open for future research.

Release 2.0 of the ODMG standard has limited facilities for querying the constraints on schema. It provides the method `getCardinality` to determine the cardinality of a relationship. We believe that more facilities should be provided to query constraints on an attribute or a relation. In OKBC, the constraints are represented using facets. The ODMG data model does not support cardinality and range constraints, which is unfortunate because they can be extremely useful in optimizing queries in a heterogeneous database environment.

A possible approach to support queries on constraints is to define a method called `facet` that can be invoked during the OQL queries. For example, consider the class definition shown in Figure 2.9.2. Given the definition of `Person`, we show some sample queries and their expected results in Table 2.9.2.

The method `facet` can be system generated when the schema is compiled. We believe that the ODMG data model should be extended to incorporate a larger set of constraints on the attribute and relationship values. The range and cardinality constraints supported by OKBC are good initial candidates for inclusion in the ODMG data model.

To incorporate the class comparison queries in OQL, an approach similar to the one for other queries can be used. We can introduce a relation corresponding to each type of class comparison inference. For example, if consistentWith represents a relation that holds between two classes that are consistent, the query

```
SELECT #X WHERE Person consistentWith #X
```

returns all the classes that are consistent with the class `Person`.

In summary, we believe that the schema queries can be extremely useful to support query formulation and provide information for query optimization. We identified four classes of schema queries that we have found useful while designing an API for KRSs: taxonomic, frame structure, constraint, and class comparison. We believe that if direct support for the four classes of schema queries identified here is provided in OQL, it will be a more powerful mediator language.

3 GKB-Editor

GKB-Editor was the driving application for the development of GFP. We upgraded GKB-Editor to use OKBC. Our initial goal of the project was to implement GFP bindings for several KRSs. With the adoption of OKBC by the HPKB project, several research groups developed OKBC bindings of their own (discussed later in the report), and therefore, we did not find it necessary to duplicate the work. In this report, we only describe our experience in reusing GKB-Editor with Ontolingua. We also discuss the spreadsheet and WWW interfaces to GKB-Editor.

3.1 Reusing GKB-Editor with Ontolingua

We tested the reusability of GKB-Editor with Ontolingua, the ontology server from KSL Stanford. Two main problems were encountered in the process: Ontolingua does not support unique symbolic frame names, and some of the browsing operations were found to be too slow over the network.

GKB-Editor relies heavily on the unique symbolic frame names for its operation. All the KRSs for which we have used GKB-Editor so far have supported unique symbolic frame names. Ontolingua, however, does not make this assumption. GKB-Editor can detect this difference by querying the value of the `:frame-name-required` behavior. Recall that the behaviors are a mechanism supported by OKBC. To address this problem, we extended GKB-Editor to use fictitious frame names whenever the frame name is **nil** and the `:frame-name-required` behavior is *false*. We expect that the use of fictitious names can be a general technique for porting OKBC-based software that was initially developed for a system with a value *true* for the `:frame-name-required` behavior to a system with a value *false* for this behavior.

While using GKB-Editor with Ontolingua, we found that some of the browsing operations were too slow. This occurs because while displaying a KB, GKB-Editor invokes numerous OKBC operations, each of which results in a network call. A possible solution to this problem is to use the procedure language of OKBC and combine several OKBC operations into one procedure that can be evaluated in just one network call. We used the following implementation strategy to incorporate procedures in GKB-Editor: Before starting a browsing task, predict the OKBC operations that will be invoked to perform that task, execute them in a procedure, and cache their results. When the same operations are invoked during the browsing task, no network calls are necessary because the results are already cached. This strategy required minimal rewriting of GKB-Editor, and was used to speed up some of the commonly used operations such as invoking the taxonomy editor or invoking the frame editor.

In spite of significant effort invested in speeding up the operation of GKB-Editor to work over the network with Ontolingua, the response time of many operations still remained quite slow. This suggests that it may not be always feasible to retrofit a legacy application to work using network OKBC. If an application is designed with the objective of network operation, most of its functionality can be embedded in the OKBC procedure language, ensuring a fast runtime response. This was not the case with GKB-Editor. This experience also suggests that a network API for the knowledge server should use operations that are more coarse grained than the OKBC operations. Otherwise, the network overhead of invoking OKBC operations can be excessive, reducing its practicality.

In summary, our ability to use GKB-Editor with Ontolingua is a testimonial to the ability of OKBC to enable the construction of reusable tools. The experience also suggests that an application designed to run across the network needs a coarser API than that offered by OKBC.

3.2 Spreadsheet Viewer for GKB-Editor

Spreadsheet programs can present large volumes of information very compactly. They also have analytical capabilities such as statistical functions, and histogram and pie-chart displays. Several users requested such functionality. Therefore, we undertook interfacing GKB-Editor to a spreadsheet program to allow the slot values from multiple frames to be viewed and analyzed within a spreadsheet.

Our initial plan was to employ a commercial spreadsheet program, for Unix, called eXclaim from Unipress, Inc. This product has a programmatic interface that allows dynamic importing of new data via interprocess communication. Our initial evaluation showed that eXclaim has several shortcomings for our work. For example, in eXclaim, it is not possible to protect certain spreadsheet cells to display the column headings. It is not possible to change the spreadsheet menus using the API. Import and export of data from GKB-Editor works using text files and is inefficient. While loading the data into the spreadsheet, it is not possible to execute any initialization commands. For example, if one wants the data values to be displayed in a certain way at start-up time, it cannot be done. It is not possible to get rid of unnecessary menu options. Because of these problems, we looked for alternative spreadsheet software and found that NExS, the Network Extensible Spreadsheet from X Engineering Software Systems, met most of our needs.

NExS was originally developed for Unix workstations and provides an easy-to-use graphical user interface that is fully compatible with X Windows and Motif. NeXS can deal with data sheets of 32,767x4,096 columns, has more than 237 built-in scientific programming functions, and has a worksheet that can be formatted on a cell-by-cell basis. It is available for a price ranging from $149 to $249, depending on individual needs.

During this project, we developed a functional NeXS/GKB-Editor interface. The user invokes the spreadsheet viewer on a specified class, causing all instances (or subclasses) of that class to be exported to a spreadsheet. The user can select whether all, or only a subset, of the slots within those frames are exported to the spreadsheet, select the order in which the instances should be displayed and define whether to display the superclasses of a frame. Slots form the columns of the spreadsheet, and frames form the rows. In a simpler case, the values of single-valued slots fill individual cells of the spreadsheet.

While interfacing NeXS to GKB-Editor, we faced several engineering difficulties. For example, Allegro Common Lisp, version 4.3, running under Solaris, supports only shared libraries. NeXS was not shipping shared libraries until we made a special request to do so. While exiting from a spreadsheet, NeXS writes out the data to a file. We changed its exit function so that the data is sent back to GKB-Editor and not to a file. We also revised the distribution procedure for GKB-Editor so that the NeXS libraries are transparently loaded at the user sites.

We did some preliminary experiments to estimate the speed of data transfer between NeXS and GKB-Editor. To transfer 600 frames from GKB-Editor to NeXS, while running with LOOM, takes about 4 seconds. We believe this is an adequate speed for the transfer of data between GKB-Editor and the spreadsheet.

In our future work with the spreadsheet viewer, we plan to deal with multivalued slots. Multivalued slots are complicated since it is not clear how to map multiple values of a single slot into the cells of the spreadsheet. We have experimented with several approaches — placing all values in a single cell in separate lines, allocating additional spreadsheet rows to show multiple values, and placing only the first value in a cell but allowing the user to inspect all values in a second spreadsheet window that is invoked through a mouse-click. At present, we are able to transfer the data from GKB-Editor to the spreadsheet, but are not able to save the updated data back to GKB-Editor. In future, we plan to support the import of updated values back into the KB.

3.3 WWW Interface for GKB-Editor

The WWW interface of GKB-Editor was undertaken to enable users to utilize the editor on virtually any platform without requiring any software installation. Making GKB-Editor available in this fashion would also make it easier for multiple groups to collaborate in developing a KB or ontology. Under this contract, we produced a prototype read-only implementation of the WWW interface of GKB-Editor.

The WWW implementation uses CWEST (CLIM-WWW Server Tool, pronounced "quest"), a toolkit developed at SRI for retrofitting existing CLIM applications to run over the WWW, and CL-HTTP, an HTTP server for Common Lisp, developed at Massachusetts Institute of Technology (MIT). CWEST forms a wrapper around GKB-Editor and causes it to generate appropriate graphical output (non-graphical requests can also be handled). That output is then captured and parsed: any pure textual portion is converted into HTML, while graphical regions are converted to GIF files. The CL-HTTP server receives the requests of the WWW clients, forwards them to CWEST for processing, and returns the result to the client.

Using CWEST, it was quite straightforward to reproduce many of the displays of the X-windows versions of GKB-Editor in its WWW version. We had feared that transmitting the displays of GKB-Editor as GIF images would be slow, but this did not turn out to be a serious problem in practice. Even though the response time is slower than the corresponding response time for the X-windows version, it is quite acceptable. In the X-windows version of GKB-Editor, a user can incrementally expand and contract portions of a KB. A similar facility is available in the WWW version except that the whole screen is redrawn every time a node is expanded or contracted.

In the future, we plan to incorporate two improvements in the WWW version of GKB-Editor: multiple users and user preferences. The current implementation of the WWW version of GKB-Editor allows access to only one user at a time. Since we now are supporting read-only access, an easy solution to this problem is to dedicate a GKB-Editor process to each user. This will require our server to distinguish between the requests coming from different users, which can be easily done by encoding this information within each request. At present, all the user preferences are not available in the WWW version. In the X-windows version, preferences are specified using CLIM menus. Since similar menus are not available in HTML, we will use a form-based interface for preferences.

GKB-Editor has already proven useful in the HPKB program and for Project Genoa. For our HPKB project, GKB-Editor helped us comprehend the HPKB upper ontology, which led to its effective reuse. In Project Genoa, it was used to develop an argument ontology. Several ontologies and KBs available at the end of the first of year of the HPKB program will be used as test cases. We also plan to make GKB-Editor available to Science Applications International Corporation, who is our team member in the HPKB project, for knowledge acquisition for HPKB challenge problems.

4 Reusing PERK with LOOM

The development of PERK started under a previous contract from the Air Force Research Laboratory. The task undertaken under the current contract was to test its reusability with LOOM.

Most existing KRSs, such as LOOM and Ontolingua, fully load a KB into memory before accessing any part of it. To provide persistence, KBs are written in their entirety to flat files on secondary storage. This approach is not scalable because the loading and saving times are proportional to the size of the KB rather than to the volume of information accessed in a given session, or to the number of frames modified in a given session. Our storage system, PERK (for

PERsistent **K**nowledge), submerges a DBMS in a KRS, and retrieves frames incrementally, on demand, from the DBMS. Because fetching of frames on demand from the DBMS, or *demand faulting*, is analogous to page faulting in operating systems, we call this process *frame faulting*. PERK tracks which frames have been modified and transmits those frames back to the DBMS when the KB is saved.

One of the challenges that we faced while reusing the storage system with LOOM was that the classifier code is not reentrant, that is, while we are classifying a concept A into the subsumption hierarchy, we cannot always start classifying another concept B until classification of A is complete. As a result, frame faults generated while classification is in progress cannot be serviced, because if we try to do so, the classifier may find some of the internal data structures in an inconsistent state. Therefore, before we start classifying a concept, we must make sure that all the concepts that will be referenced in the process are already faulted in so that no frame faults are triggered while the classification is in progress.

We found it difficult to predetermine all the frames that will be used while defining a given concept because there is no document describing the classification algorithm of LOOM. We identified the problem cases by testing our system with two real KBs, *naval theory* and *aircraft*, that were developed by the Information Sciences Institute (ISI) at the University of Southern California.

When a concept A is referenced in the definition of concept B, we can notice it while parsing the definition of A, and demand fault A before processing B. Another simple case arises while dealing with the superconcepts: before a concept is loaded, we make sure that its direct superconcepts have been loaded. It is easy to determine superconcepts because the superconcepts of a concept are stored in the database.

A somewhat more involved case arises while classification is in progress. Suppose concept A has two subconcepts B and C. When we load A, we will create stubs for B and C. When there is a demand fault for B, we fetch it from the DBMS. While B is being classified, the classifier considers classifying B under C, causing a demand fault for C. No matter whether we load B or C first, while classifying one of them, the other one will have to be faulted in. In this particular instance, we know that B cannot be classified under C, because otherwise this fact would have been stored in the database, and we would have faulted C before faulting B. Therefore, we modified the classifier to not consider C during the classification of concepts that are being loaded from the database.

Inverse relationships need special treatment because of the circular dependencies that they may generate. For example, consider a slot A that is the inverse of another slot B. Suppose we retrieve A when B is not loaded into LOOM. If, while A is being loaded, there is a frame fault for a concept C (perhaps C appears in the definition of A) that has B as one of its slots, we cannot define C because B is not yet defined. We could not go ahead and load B, because B is defined in terms of A, and any attempt to load B will trigger a cyclical frame fault for A. We addressed the problem by delaying the assertion of the slot values for C until both A and B have been faulted in.

Even though we have empirically tested frame faulting for several KBs, we have no way to guarantee that incremental loading has not changed the behavior of classification or that we have taken into account all possible situations under which a frame fault may be generated during classification. A principled study of classification in conjunction with demand faulting remains a problem open for future research.

Our experience in using PERK with LOOM showed that there can be many unexpected interactions between a storage system and the inference capabilities of a KRS, making it difficult to design a generic storage system. Therefore, if PERK were to be used with another KRS, different interactions with the inference capability of that KRS may arise and would have to be addressed.

A limitation of PERK is that once frames have been loaded into virtual memory, there is no

way to flush them out. Thus, PERK cannot deal with KBs that lead to a process whose size exceeds virtual memory. We have not encountered this problem with any of the KBs we have worked with so far.

The problem of flushing frames from memory is a problem that must be addressed in the long term for the scalability of PERK. Flushing frames from memory is not straightforward because frames can refer to other frames by pointers, and displacing a frame from memory can invalidate references to it. Special-purpose schemes have been developed to solve this problem [KK95]. We believe that some of the existing schemes can be incorporated into PERK.

5 Conflict Arbitration Interface for Collaboration System

Our collaboration system controls multiuser access to a KB, by first allowing the users to make independent changes to the KB in their private workspaces, and when they are done, checking for conflicting changes, and merging the changes into the public copy of the KB if the changes are conflict free. If conflicting updates are detected, they must be reported so that the concerned users can resolve the conflicts. Under the current contract, we initiated preliminary work on a conflict-arbitration interface that reports the conflicting changes.

The user changes to the public copy of the KB are recorded as a log called net-log. A log consists of a sequence of log records. Each log record is a list with two elements. The first element is a list containing the name of the OKBC operation and the arguments with which it was invoked. The second element is the list of any values that are being overwritten by the current operation.

For conflict detection with respect to a transaction T, we examine the portion of the net-log that contains operations executed after the begin time of T. To check the conflicts between a user transaction and the net-log, we must check for each operation in the transaction, if the net-log contains an operation that performs a conflicting update. Each update OKBC operation can, in general, involve multiple updates. For example, the `put-slot-values` operation deletes the old slot value(s) and inserts several new slot values. Furthermore, the number of possible OKBC operations is large (over 200), so that analyzing conflicts between all possible combinations of OKBC operations would be cumbersome. Therefore, before analyzing the conflicts, we translate the operations into a canonical set of three operations — `INSERT`, `DELETE`, and `REPLACE` — on the nodes and edges of the underlying KB graph. A `REPLACE` operation modifies an existing KB value. Since the number of operations in the canonical set is smaller than the number of OKBC operations, conflict analysis is considerably simplified.

In our initial design, the conflicts were displayed using an internal graph representation of OKBC operations that is used in conflict checking. Since the user should not have to know about the internal data structures, we needed a scheme to display the conflicts in a user-friendly manner. We investigated two alternatives. The first alternative is to associate, with each translated operation, the original OKBC operation that led to the conflict. Then conflicts can be reported in terms of the original OKBC operation. The disadvantage of this approach is that it requires additional bookkeeping to relate each translated operation with the original OKBC operation. Furthermore, some users may invoke OKBC operations by using a graphical tool such as GKB-Editor and may not be familiar with specific OKBC operations. For such users, reporting conflicts in terms of OKBC operations does not help much. The second alternative is to analyze the classes of conflicts between graph operations, design textual descriptions for them, and use the textual descriptions for conflict reporting. Since the number of cases of conflicts is small, it is a fairly tractable solution.

To consider the solution in detail, let us consider an example conflict between two `REPLACE`

operations. The first REPLACE operation changes X to Y, and the second operation changes X to Z. If X is a singleton, as in operations (REPLACE HEIGHT LENGTH) and (REPLACE HEIGHT HT), we know that these operations are translations for **rename-frame** operations. The conflict between them can be reported by

```
<User1> renamed HEIGHT to HT which was previously renamed
to LENGTH by user <User2>.
```

If X is a list of length 2, such as, (REPLACE ((John age) 30) ((John age) 35)), we know that the translated operation was generated from **replace-slot-value** operations. The conflict can then be reported by

```
User <User1> replaced age of John from 30 to 35 which was previously replaced
to 32 by user <User2>.
```

Using this approach, we were able to generate readable explanations for all the classes of conflicts. Extensive user testing of this feature remains to be done. We expect to undertake that task in our future projects.

6 Evaluation of OKBC

Defining a metric to measure the success of a generic API is difficult. We will argue that OKBC has been successful in its goal of enabling the construction of interoperable tools by presenting empirical evidence based on the definition of OKBC bindings for several KRSs.

6.1 OKBC bindings

Defining OKBC bindings for a KRS means implementing a subset of OKBC operations by using calls to the native API of that KRS [Ric98]. OKBC bindings for several systems have been defined by our research groups at KSL and SRI. At SRI, OKBC bindings were defined for LOOM [MB91], Theo [MAC+89], SIPE-2 [Wil88], and Ocelot [PLK97]. At KSL, OKBC bindings were defined for Ontolingua [FFR97], Abstract Theorem Prover (ATP) (a theorem prover developed at KSL), CML [FIFB96], Tuple-KB [Ric98], file system KB, and CLOS. ISI has now produced its own version of an OKBC binding for LOOM. An OKBC binding for Cyc [LG89] has been defined by Cycorp.

OKBC was recently licensed by Pangea Systems Inc. (see `http://www.panbio.com`) in support of its projects in the area of bioinformatics. It is used extensively in several ongoing projects at Stanford and SRI, and has been adopted by DARPA's HPKB program
(see `http://www.teknowledge.com/HPKB/`).

6.2 OKBC uses in DARPA's HPKB project

During the first year of the HPKB program, OKBC was used in the following contexts.

- SRI and KSL developed a translator to load the HPKB upper ontology into any OKBC server. (The HPKB upper ontology was released in the MELD format used by Cycorp.) As a result, the translator could be used by both SRI and KSL. The development of the translator itself was facilitated by OKBC, because we could use our GKB-Editor to compare the translations produced by KSL and SRI as they were being developed. The comparisons helped us identify semantic differences and led to a common translator.

- SRI used OKBC to interface the theorem prover SNARK to an OKBC server storing the World Fact Book KB (WFBKB). Since the WFBKB is quite large, we did not want to incorporate it into SNARK. The *procedural attachment* feature of SNARK was used to look up the facts that were available only in the WFBKB.

- MIT's START system used OKBC to connect to SRI's Ocelot/SNARK OKBC server. This connection will eventually give a user the ability to pose, in English, questions that are then transformed to a formal representation by START, and shipped to SNARK using OKBC; the result is returned using OKBC. During HPKB Year 1, START generated only prespecified formal representations of the challenge problem questions that were then answered by SNARK.

- ISI built an OKBC server for LOOM that was extensively used by Gheorghe Tecuci at George Mason University. SAIC built a front end to the OKBC server for LOOM, and it was extensively used by the members of the battlespace challenge problem team.

- SRI used a C client that was originally developed by KSL. The C client was tested with SRI's OKBC server without any difficulty and has been delivered to Pacific Sierra Research for use in Project Genoa.

We anticipate that the OKBC experiences during HPKB Year 1 will substantially scale up during Year 2. In particular, we expect greater use of OKBC in conjunction with theorem provers, stronger connection with MIT's START, and greater use within Project Genoa.

6.3 Limitations of OKBC

The goal of OKBC is to enable the construction of reusable KB tools — that is, the application programs that access a KRS to perform browsing, editing, or reasoning tasks. Empirical evidence has shown success in meeting this goal. Potential users of OKBC are usually concerned with whether they can successfully use OKBC in their projects. Here, we identify some of the commitments and sacrifices that they may need to make to use OKBC successfully.

To construct a new OKBC binding for a KRS, it is necessary to identify the knowledge model used by the KRS and define a mapping between it and the OKBC knowledge model. By providing both frame-oriented and assertional views of a KB, OKBC is capable of supporting a wide range of systems. Some systems do not easily admit to either of these views. While an OKBC binding can be defined for such systems, some users may not find it to be an intuitive or natural mapping. OKBC bindings work best when the knowledge model of the KRS closely matches that of OKBC.

OKBC bindings isolate a KB tool from many of the peculiarities of a KRS, but certainly cannot cover all of them. Therefore, porting a KB tool to a new KRS usually requires some additional effort. For example, Ocelot supports a slot type called `:unique`. The slots of this type are inherited by subclasses and instances, but their values are not inherited. For GKB-Editor to handle this peculiarity, a small amount of Ocelot-specific code had to be added. Similarly, ATP provides operations to return the proof that a value satisfied some query, but OKBC does not currently provide any operations for specifically extracting proofs.

OKBC is neither the lowest nor the highest common denominator protocol. We cannot hope that it will expose all of the functionality of every system, but it exposes what we believe most applications want. In addition, the protocol is specifically designed to be extensible by means of the behavior mechanism so that clients and servers can negotiate the use of a more powerful functionality than is provided by the protocol.

Some aspects of OKBC are easily extensible, for example, adding a new inference level or a new facet. Adding a new inference level requires giving its axiomatic definition and, if possible, providing a default implementation of affected methods. Extending OKBC to deal with arbitrary disjunctions and negations is not straightforward as they are not a part of the OKBC knowledge model. Support for assertions has also been identified as a requirement by the HPKB project and is planned as a topic for further work.

The guarantees given by OKBC are quite weak: if two OKBC servers have *different* behaviors, identical operations executed on them are likely to have *different* results. A part of the reason for such a weak guarantee is that OKBC is a generic API, and its specification has been purposefully kept flexible.

OKBC is a functional interface to a KB [BFL83] and does not specify the data structures that should be used to implement its knowledge model. Using OKBC, an application cannot manipulate internal KRS data structures that are used to implement frames.

OKBC does not solve the problem of semantic KB interoperation. For example, using the OKBC operation **get-slot-values**, an application may query the salary of a person from two different systems, but there is no guarantee that the returned values will be semantically identical — one system may return annual salary and the other system may return monthly salary. Semantic interoperation is beyond the scope of OKBC.

7 Directions for Future Work

The research and development effort undertaken under this project can be extended in many directions as suggested here.

7.1 OKBC applications

The object-oriented features of OKBC are well developed. Even though the OKBC knowledge model may be at the lower end of expressiveness of KR languages, it is more advanced than many commercial object-oriented languages, for example, the data description language for MPEG-7 [ISO98] or XML [TB98]. We believe that the commercial languages can significantly benefit from the design experience of OKBC. We would like to investigate that possibility in the future.

GFP was primarily geared toward object-oriented KRSs, and did not provide much support for general assertions in a KB. OKBC partially corrected this problem by supporting an assertional view of a KB. The current support for assertions is, however, quite limited and a lot more can be supported, suc as an operation to check the equivalence of formulas or an operation to produce the clausal form of a formula. Extending OKBC to provide support for assertions remains open for future work.

7.2 Speeding KB construction time

Developing a new KB is a time consuming and expensive process. At least two techniques exist for speeding up the KB construction process: creating a new KB by reuse and allowing multiple domain experts to simultaneously enter information into the KB. The knowledge reuse involves KB comprehension (understanding a KB), KB translation (putting it in the right syntax), KB selection (identifying portions of a KB that are of interest) and merging (one or more KBs to produce a new one). GKB-Editor is a valuable tool for the KB comprehension phase of knowledge reuse. OKBC enables KB translation, by providing an API to manipulate the KB, and by a well-specified knowledge model that is the basis for an easy interchange of taxonomic knowledge.

Under DARPA's HPKB project, we initiated some work on KB selection to extract portions of a KB that are of interest. These capabilities can be made available through GKB-Editor.

Our longer-term objective is to develop GKB-Editor into a comprehensive ontology development environment. In our future work, we would like to expand upon the ontology selection work that we initiated under the HPKB project. The graphical nature of GKB-Editor makes it easier for a user to identify the portions of a KB that are of interest. GKB-Editor does not yet have direct support for manipulating deductive rules. Since axiomatic knowledge expressed in form of deductive rules is an integral part of a KB, we would like to extend both GKB-Editor and OKBC to manipulate rules.

To speed up the KB construction time, a technique orthogonal to knowledge reuse is to have several subject matter experts directly enter knowledge into a KB. Both GKB-Editor and a collaboration system can play an important role in the process. GKB-Editor can be extended to provide a more intuitive interface for knowledge entry as compared to entering knowledge into a flat file by using logic. The current design of GKB-Editor is aimed at KR experts, therefore, in some cases, it exposes more representation details than a domain expert may need to know. Additional viewers would need to be incorporated to give domain-specific visualizations of a KB. A collaboration system can enable multiple domain experts to simultaneously enter knowledge into the KB. The conflict collaboration interface developed as part of this project will allow the resolution of any conflicting updates performed by domain experts.

7.3 Knowledge bases and knowledge discovery

Knowledge bases can enable knowledge discovery from databases (KDD) by providing a natural, object-oriented representation of an application domain, a powerful query language that can manipulate schema as well as the ground facts, and an easy-to-use graphical interface that can support interactive exploration [CK97, KCP99, BST+92]. KDD, in turn, can enable the construction of a KB by semiautomated derivation of rules of domain knowledge or by starting from a KB and refining it based on the data in a database. This two-way interaction presents a multitude of opportunities, and we address some of them here.

Many KDD engines use automatic statistical or machine-learning mechanisms to search for implicit patterns in data. The overall KDD task faced by an analyst, however, involves many activities in addition to those offered by the core KDD engine. The input necessary for a KDD engine is not usually available in the required format, and in most cases, has to be prepared by processing the data in an existing database. For example, in an analysis of commodities exported by a country, the export data may be available for each product (such as beef, chicken, etc.), but the input to the KDD engine needs to be represented in terms of abstract categories of products (such as animal products). In such a situation, an ontology categorizing commodities can significantly aid an analyst in preparing the data for input to the KDD engine. KDD tasks are usually iterative and involve experimenting with categories at different levels of abstraction. Frame Representation Systems, such as Ocelot, and graphical browsing and editing tools, such as GKB-Editor [KCP99], are natural tools for hierarchical representation, display, and selection of knowledge. Their utility is significantly enhanced with an interface to a commercial DBMSs supported by a system such as PERK [KCP99].

Large KBs such as the Cyc KB, the Sensus ontology, or the Ontolingua ontology library, are expensive to build [LG89, KL94, FFR97]. The output of a KDD task can contribute significantly to KB development. Many KDD tasks extract association rules from data, which can be integrated directly into a KB. If these newly learned rules are determined to be inconsistent with existing rules in the KB, this serves as an indicator of potential errors in the existing rules, or in the data

that was used to generate the new rules. In other cases, a KB may contain causal rules that do not have associated probabilities indicating the strength of causation. Probabilistic KDD tools can use empirical data to assign probabilities to these rules.

In summary, leveraging KB systems with KDD tools will permit more effective knowledge understanding by providing KB support to prepare data for the KDD process, and using the output of the KDD tools to refine the contents of the KB.

7.4 Object-relational knowledge servers

Our storage system PERK is aimed at extending the KRSs with "database-like" capabilities — for example, efficient storage and retrieval and multiuser access. In the current architecture of PERK, the KRS acts like an object-oriented cache to a DBMS server. The main advantage of this architecture is that we can provide efficient support for common KRS operations, such as taxonomic inference and queries on the KB schema within the cache. The disadvantage is that one is forced to reproduce, in the cache, some of the database functionality, for example, flushing out frames from memory and query optimization. The inability to flush the frames out of memory is in fact a known limitation of PERK and prevents it from handling KBs that are bigger than the virtual memory.

A fundamentally different architecture for PERK will be to exploit the object-relational capabilities of a DBMS. For example, Oracle 8 now supports limited forms of an object-oriented data model. It also supports a client-side cache to allow efficient retrieval for object navigation. Oracle 8 allows user-defined types and has provisions for adding data-type-specific access methods. Such extensibility features are known as *data cartridges* or data blades. Use of data cartridges is an attractive alternative to embedding database functionality in KRSs. For example, in an alternative design of PERK the inheritance capability can be added to the database as a cartridge. With the client-side cache facility, it is not necessary to cache objects in the KRS memory. Since Oracle supports buffer management of the cache, the problem of flushing out the frames from the KRS memory disappears. In this fundamentally different architecture of PERK, most of the database functionality is pushed into the database server, and the KRS capability is added to the database by using cartridges. If this architecture proves viable from an efficiency viewpoint, it can prove to be a general technique for integrating KB applications with a DBMS. We plan to investigate the use of an object-relational-based architecture in our future work.

8 Summary and Conclusions

The primary theme in this project was to investigate the techniques for constructing reusable knowledge base (KB) development tools. The technique employed for this purpose was Open Knowledge Base Connectivity (OKBC), which is an API for accessing KRSs.

OKBC evolved from the Generic Frame Protocol (GFP), its predecessor, and made several important contributions including the design of an assertional view of a knowledge representation system (KRS), constraint checking operations, and comparison with schema querying facilities of object-oriented databases. It was adopted as an API for DARPA's High Performance Knowledge Bases (HPKB) project and was used by several contractors in the HPKB program. The most powerful result of this work was the OKBC knowledge model which is reusable not just across KRSs, but also across a range of object-oriented applications.

We experimented with several KB development tools to test the ability of OKBC to enable the construction of reusable tools. These tools included GKB-Editor, a graphical tool for browsing

and editing KBs, and PERK, a system for storing KBs in Oracle and for controlling multiuser access to KBs. We were able to more easily enable the reusability of GKB-Editor than of PERK, because PERK needs to access many of the internal data structures of a KRS that are not exposed by OKBC.

For supporting reusability among KRSs, behaviors and additional return values proved to be central techniques for OKBC. Wherever it is not feasible to legislate certain requirements, KRSs can expose their differences from the OKBC knowledge model either globally by setting the value of a *behavior*, or locally by returning an additional value from an operation. For example, support for frame names can be advertised by using the `:frame-names-required` behavior, and the degree of conformity to the **:inference-level** argument is exposed by an extra return value.

Design experience with OKBC suggests the existence of an "expressiveness vs. generality" tradeoff that is similar to the "expressiveness vs. tractability" tradeoff [LB87]. Some of the KRSs with which we have used OKBC are highly expressive and would require OKBC to support an equally expressive knowledge model to expose their full functionality. A highly expressive knowledge model, however, makes defining OKBC bindings for KRSs with limited functionality difficult and time consuming. Throughout the design of OKBC, this tradeoff was a guiding principle for carefully controlling the expressiveness of the knowledge model. We believe that expressiveness vs. generality is a fundamental tradeoff in knowledge sharing.

The results of the project represent a substantial research and development activity. The tools developed during this project were heavily used in DARPA's HPKB project. GKB-Editor is being extensively employed to enable KB comprehension and reuse. The KB for project Genoa is being developed using GKB-Editor. OKBC is being used as an API by several participants in the HPKB program. We expect collaboration capabilities of PERK to be increasingly important in future projects.

References

[BBMR89] Alexender Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperine Resnick. CLASSIC: A Structural Data Model for Objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 58–67, Portland, OR, 1989.

[BCS98] Alex Borgida, Vinay K. Chaudhri, and Martin Staudt. Report on the 5th International Workshop on Knowledge Representation Meets Databases (KRDB'98). *SIGMOD Record*, 27(To appear), 1998.

[BDK97] Gerhard Brewka, Jürgen Dix, and Kurt Konolige. *Non Monotonic Reasoning*. Cambridge University Press, 1997.

[BFL83] R.J. Brachman, R.E. Fikes, and H.J. Levesque. KRYPTON: A Functional Approach to Knowledge Representation. *IEEE Computer*, 16(10):67–73, October 1983.

[BST+92] R. J. Brachman, P. G. Selfridge, L. G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. L. McGuiness, and L. A. Resnick. Knowledge Representation Support for Data Archaeology. In *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, 1992.

[Cat95] R. G. G. Cattell. *The Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann Publishers, Inc., 1995.

[Cat97] R. G. G. Cattell. *The Object Database Standard: ODMG-93, Release 2.0.* Morgan Kaufmann Publishers. Inc., 1997.

[CFF+97] Vinay K. Chaudhri. Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. The Generic Frame Protocol 2.0. Technical report, Artificial Intelligence Center. SRI International, Menlo Park, CA, 21 July 1997.

[CFF+98] Vinay K. Chaudhri. Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. OKBC: A Foundation for Knowledge Base Interoperability. In *Proceedings of the National Conference on Artificial Intelligence*, pages 600–607, July 1998.

[CK97] Vinay K. Chaudhri and Peter D. Karp. Querying Schema Information. In *Proceedings of the 4th International Workshop Knowledge Representation Meets Databases (KRDB'97)*, pages 4–1 to 4–6, Athens, Greece. 1997.

[FFR97] Adam Farquhar, Richard Fikes, and James P. Rice. A Collaborative Tool for Ontology Construction. *International Journal of Human Computer Studies*, 46:707–727, 1997.

[FIFB96] Adam Farquhar, Yumi Iwasaki. Richard Fikes, and Daniel G. Bobrow. A Compositional Modeling Language. In *Proceedings of the 1996 Qualitative Reasoning Workshop*. 1996.

[Gei95] Kyle Geiger. *Inside ODBC*. Microsoft Press. 1995.

[GF92] Michael R. Genesereth and Richard E. Fikes. Knowledge Interchange Format. Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department. Stanford University. Stanford, CA. 1992.

[ISO98] ISO. MPEG-7 Applications Document. Technical Report MPEG 98/N2462, International Organization for Standardization. October 1998.

[Kar92] P.D. Karp. The design space of frame knowledge representation systems. Technical Report 520. SRI International. Artificial Intelligence Center. 1992. URL `ftp://www.ai.sri.com/pub/papers/ karp-freview.ps.Z`.

[KCP99] Peter D. Karp, Vinay K. Chaudhri. and Suzanne M. Paley. A Collaborative Environment for Authoring Large Knowledge Bases. *Journal of Intelligent Information Systems*. 1999. To appear.

[KK95] Alfons Kemper and Donald Kossmann. Adaptable Pointer Swizzling Strategies in Object Bases: Design. Realization. and Quantitative Analysis. *VLDB Journal*, 4(3):519–566. 1995.

[KKS92] Michael Kifer, Won Kim. and Yehoshua Sagiv. Querying Object-Oriented Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 393–402, San Diego, May 1992.

[KL94] K. Knight and S. Luk. Building a Large-Scale Knowledge Base for Machine Translation. In *Proceedings of the National Conference on Artificial Intelligence*. Seattle. WA. August 1994.

[KMG95] P.D. Karp, K. Myers, and T. Gruber. The Generic Frame Protocol. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pages 768–774. 1995. See also WWW URL `ftp://ftp.ai.sri.com/pub/papers/karp-gfp95.ps.Z`.

[KRPPT96] P. Karp. M. Riley, S. Paley, and A. Pellegrini-Toole. EcoCyc: Electronic Encyclopedia of *E. coli* Genes and Metabolism. *Nucleic Acids Research*, 24(1):32–40, 1996.

[LB87] H.J. Levesque and R.J. Brachman. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence*, 3(2):78–93, 1987.

[LG89] Douglas B. Lenat and R.V. Guha. *Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project.* Reading, MA, Addison-Wesley Publishing Co., 1989.

[LSS96] Laks V. S. Lakshmanan. Fereidoon Sadri, and Iyer N. Subramanian. SchemaSQL — A Language for Interoperability in Relational Multi-database Systems. In *Proceedings of the 22nd International Conference on Very Large Databases*. Bombay, 1996.

[MAC+89] T.M. Mitchell, J. Allen, P. Chalasani. J. Cheng, E. Etzioni, M. Ringuette. and J.C. Schlimmer. Theo: A Framework for Self-Improving Systems. In *Architectures for Intelligence*, pages 323–355. Erlbaum. 1989.

[Mac91] R. MacGregor. The Evolving Technology of Classification-based Knowledge Representation Systems. In J. Sowa. editor. *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann Publishers, Los Altos. CA, 1991.

[MB91] R. MacGregor and M.H. Burstein. Using a Description Classifier to Enhance Knowledge Representation. *IEEE Expert*, 6(3):41–46. June 1991.

[PLK97] Suzanne M. Paley. John D. Lowrance, and Peter D. Karp. A Generic Knowledge Base Browser and Editor. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*. 1997.

[Ric98] James P. Rice. Writing an OKBC Application. Technical Report KSL-98-15, Knowledge System Laboratory. Stanford. CA. April 1998.

[TB98] C. M. Sperberg-McQueen Tim Bray, Jean Paoli. Extensible Markup Language (XML) 1.0. Technical Report rec-xml-19980210, World Wide Web Consortium, October 1998. See `http://www.w3.org/TR/REC-xml`.

[Wil88] D.E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann. August 1988.

*U.S. GOVERNMENT PRINTING OFFICE: 2000-510-079-81245