AFRL-IF-RS-TR-2007-204
Final Technical Report
September 2007

ACCESSIBLE RESEARCH CYC

Cycorp, Inc.

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No.Q101/00

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

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

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-IF-RS-TR-2007-204 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/ /s/

PETER J. ROCCI, Jr.
Work Unit Manager

JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEP 2007	Final	Dec 03 – May 07

4. TITLE AND SUBTITLE

ACCESSIBLE RESEARCH CYC

5a. CONTRACT NUMBER
FA8750-04-C-0034

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62304E

6. AUTHOR(S)

Larry Lefkowitz, Jon Curtis and Michael Witbrock

5d. PROJECT NUMBER
COGV

5e. TASK NUMBER
00

5f. WORK UNIT NUMBER
03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cycorp, Inc.
3721 Executive Center Dr.
Austin TX 78731-1645

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFED
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2007-204

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# AFRL-07-0084

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The goal of this project was to produce a knowledge base (named ResearchCyc) and to modularize the Cyc knowledge base so that researchers could make use of just the ontology, or part of the ontology, or the knowledge base (KB), or part of the KB, or the inference engine, or just some heuristic level (HL) modules from the inference engine. Moreover, the goal included the capability for software power tools to provide machine-assisted (i.e., semi-automatic) mapping between Cyc's ontology and a non-Cyc ontology. The capability was aimed at dramatically increased usability (including stability) and modularity of Cyc, leading to widespread use of ResearchCyc by the R&D community. Another task involved a seedling effort to lay the groundwork for a Bootstrapped Learning initiative. This report also describes the results of the Bootstrapped Learning Seedling effort.

15. SUBJECT TERMS
Knowledge bases, ontologies, inference, Cyc knowledge base, bootstrapped learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Peter J. Rocci, Jr.
U	U	U	UL	22	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

1.0 Abstract

The goal of DARPA/IPTO's Accessible Research Cyc was to improve accessibility to the Cyc technology by the R&D community and to increase the potential for collaboration among researchers. Part I of this report describes the efforts towards these objectives.

In March 2006, the contract was modified to add a separate task involving a seedling effort to lay the groundwork for a Bootstrapped Learning initiative. This report also describes the results of the Bootstrapped Learning Seedling effort.

2.0 Introduction

This report describes the efforts and results of the Accessible Research Cyc DARPA/IPTO project, contract number FA8750-04-C-0034. This project comprised two major tasks, the original comprising an effort to increase accessibility to the Cyc technology by the R&D community and the second task, introduced in March 2006, being a seedling effort to lay the groundwork for a Bootstrapped Learning initiative.

The structure of the report reflects the dual nature of this overall project. Part I describes the goals, objectives, and results of the core Accessible Research Cyc project, including a quarter-by-quarter summary of the results. Part II describes the results of the Bootstrapped Learning Seedling effort.

3.0 Part I: The Research Cyc Platform

3.1 Objectives

A large effort has been conducted, over the past 19 years, to build up a large, broad ontology of over 100,000 terms (excluding proper nouns), to assert over a million general facts and rules about the terms of that ontology, and to construct an inference engine to operate on that knowledge base to answer queries deductively from it. As demonstrated in recent years in DARPA programs such as High Performance Knowledge Bases (HPKB), that technology - Cyc - has matured into something that can be successfully and cost-effectively applied to the task of building large knowledge-based systems of relevance to the Department of Defense. At least it can be applied successfully when its creators, Cycorp, are the ones applying it. It can almost - but not quite - be utilized effectively by the greater artificial intelligence (AI), Computational Linguistics, and Cognitive Science R&D communities today. Hence the problem. Leaders representing those three R&D communities came together for a DARPA IPTO-sponsored workshop, held at Cycorp on June 10-11, 2003. With a surprising degree of unanimity, they voiced a positive willingness, even eagerness, to obtain access to Cyc in a way in which they could make use of it. At the meeting, attendees articulated what the bottleneck was - what was limiting the transfer of the technology. They identified "The Problem". But much more - and more positively - than that, they also identified what incremental work would make it into something that they could and would use, to leverage in their own R&D. The Problem turned out to be essentially that Cyc is, while a potentially valuable resource, far too monolithic - not in the content of its KB but in its architecture. The R&D community wanted (1) the ability to mix and match which elements of Cyc (portions of the ontology, KB, inference engine, interfaces, etc.) they do and don't use, and (2) the ability to easily integrate their systems with Cyc, via various sorts of API's (which integration also should also facilitate inter-researcher collaboration and cross-use of each other's work).

3.2 Goals

The overall goal of this project was to produce a ResearchCyc knowledge base (KB) very much larger than OpenCyc, more on the order of Cyc itself, and to modularize Cyc so researchers can make use of just the ontology, or part of the ontology, or the KB, or part of the KB, or the inference engine, or just some heuristic level (HL) modules from the inference engine, etc. Moreover, the goal includes the capability for software power tools to provide machine-assisted (i.e., semi-automatic) mapping between Cyc's ontology and a non-Cyc ontology. The capability we were aiming at was dramatically increased usability (including stability) and modularity of Cyc, leading to widespread use of ResearchCyc by the R&D community.

3.3 Technical Approach

To increase the usage of Cyc by the R&D community, a number of efforts were necessary. These included simplifying ease of use, ensuring higher KB quality, easy licensing conditions, and improved performance. Some of the actions needed to address these needs include:

Knowledge Visualization: The new KB browsers and visualization tools were developed to provide access to Cyc knowledge and functionality for two classes of users, application-builders and researchers, whose going-in aims, backgrounds, needs, etc. form two relatively distinct clusters. This work leveraged the RKF (DARPA Rapid Knowledge Formation) clarification dialogue interfaces for subject-matter (naïve) users, as well as the best interfaces from Cyc, OpenCyc, and other modern knowledge based systems (such as SHAKEN).

Improved Quality Assurance: Historically, quality Assurance had been spotty for both OpenCyc and Cyc, driven more by particular project needs and user-reported bugs than anything else. The overall KB quality was improved during this project through a combination of:

- increased internal use of the tool for Cycorp operations;
- increased use of epistemological meta-knowledge within the system so that Cyc knew more about its own knowledge, the source of that knowledge, its history, etc.;
- internal ("red team") reviews of the ResearchCyc KB;
- feedback from external users;

Increased Researcher Collaboration: ResearchCyc users will want to rapidly integrate their KB's with ResearchCyc's, share their KB with others, and in turn easily – ideally automatically – make use of others' KB's. These KB's may be Cyc KB's or may be represented as OWL ontologies or other triple-store or even databases. To support this interchange of knowledge, it is possible to use ResearchCyc as an interlingua by adding Cyc assertions that "explain" the meaning of a term in X's ontology in ResearchCyc terms, and vice versa.

Reducing the learning curve: Cyc is complex, both in terms of its overall functionality as we well as the depth and breadth of its knowledge base. Cyc is also difficult to use, in part because of this complexity. However, there are several ways that the learning curve can be reduced and the ease of use improved. Partly this can be addressed by increased and improved documentation and training. The learning curve can also be reduced via simpler interfaces that better reflect the most common user functionality and are closer in form and function to interface with which users are already familiar. Complex, advanced behaviors can be hidden from the typical user. Finally, KB inconsistence can be reduced to make the system's behavior more predictable and, thus, less confusing.

Speeding up the performance of the system: As the KB grows, and as the complexity of the queries and tasks increases in depth and in breadth, system performance (in terms of responsiveness and even quality of results) may suffer. Providing the system with more information about its past inference results and enabling increased introspection, manual tuning, and automatic learning may provide was to allow the system performance to increase with experience and maintain, or even improve, its overall performance.

3.4 Results

This project was very aggressive in setting is objectives and much was learned during the course of the project that altered some of the initially anticipated research objectives. Some preliminary objectives proved more challenging than initially anticipated; other unforeseen requirements were discovered as the initial work proceeded; priorities were modified as new capabilities became apparent. Despite some mid-course corrections to address these modified needs and priorities, substantial progress was made on many of the core goals. A project review meeting was held at DARPA upon the completion of the Accessible Research Task and the results were encouraging enough for DARPA to request, and subsequently fund a follow-on ("Self-Sustaining Research Cyc" project.

To best appreciate not only the final project outcomes but also the path taken to achieve them, this section contains quarter-by-quarter summary of the key results.

Q1 2004

The initial objectives for this project included the development of a beta release of the ResearchCyc software. Achieving this required overcoming a number of barriers to usage of Cyc by the R&D community: quality control issues, overly-complex representation language features, content irrelevant to their particular research interests, restrictive licensing requirements, slowness and monolithic nature of the inference engine and related issues.

During this quarter, a number of improvements were made to the ResearchCyc interface for advanced users. In particular, we:

- Incorporated a new CycL editor applet into the HTML-based Cyc browser interface.
- Enabled Query Library application in ResearchCyc.
- Added support for collapsible rows in the index frame of the HTML-based Cyc Browser.
- Supported columnar display of search results, separating predicates, collections and individuals. Enabled display of inherited (rather than merely looked up) assertions in the content frame of the Cyc Browser.
- New options near the search box make it easier for the user to choose whether or not an English gloss should be displayed alongside terms returned as search

results. The CycL Editor Applet Every HTML-based text field in the ResearchCyc Browser interface which was used for entering CycL expressions has been replaced with the new CycL Editor Applet.

Q2 2004

During this quarter, several improvements were made to the Cyc knowledge base content, including the use of the "Predicate Populator" to achieve targeted areas of comprehensive coverage. An architecture plan was developed for the portioning of the knowledge base to support easier compartmentalization and sharing of knowledge. Work began on ease-of-use issues with the development of a main documentation page for ResearchCyc users. There were also significant inference speed-ups to Cyc (that were primarily funded by another project, but benefited ResearchCyc users as well).

Significantly, this quarter saw the first release of ResearchCyc to a set of external beta users. The list included users from the following organizations:

- Xerox PARC
- Daxtron Laboratories
- Lockheed Martin Advanced Technology Labs
- Houston VA Medical Center
- Institute for the Study of Accelerating Change
- University of Maryland
- Language Computer Corporation
- Northwestern University
- ANSER, Inc.
- MIT Media Lab
- Lyndon B. Johnson School of Public Policy
- Fraunhofer Institute
- University of Illinois at Urbana-Champaign
- NTT Communications Science Laboratories
- Stanford Natural Language Processing Department
- New Mexico Highlands University

Q3 2004

WordNet provides an alternate view for organizing semantic concepts (in the form of synsets) as well as a mapping of English words to these concepts. Mapping between WordNet synsets and Cyc terms can be useful for 1) expanding Cyc's ontology, 2) supporting NL capacities, and providing a bridge for those researchers interested in using both WordNet and Cyc. During this quarter, the first three prototype WordNet screens for connecting WordNet 2.0 to ResearchCyc were coded and deployed. Other work related to prototyping involved testing the Laszlo web application development environment as a way for ResearchCyc developers to more easily create user interfaces

and prototype screens. We found this environment to have a much easier learning curve than attempting to directly utilize the Cyc Java API.

Additional work was performed in supporting the flow of knowledge into and out of Cyc with the development of an XML output specification as well as initial work to support the importing and exporting of OWL files to and from Cyc. Work also continued on identifying and documenting gaps in KB completeness to improve users' abilities to predict whether queries were likely to succeed.

Q4 2004

This quarter saw the completion of the workflow screens for WordNet mapping and the import all WordNet terms minimally identifying each as a Cyc collection or individual. In addition, the set of ResearchCyc beta users was expanded and the feedback from ResearchCyc users was used to help prioritize upcoming work.

Work continued on the development of knowledge importing and exporting capabilities with significant progress being made on the OWL exporter. Additionally, exploratory work related to prototyping user interfaces (as a way for ResearchCyc users to eventually be able to more simply create their own user interfaces) was conducted, including evaluation of the Laszlo web application development environment. Preliminary results indicate this may provides an easier learning curve and greater productivity than attempting to directly utilize the Cyc Java API.

Q1 2005

While this quarter saw the completion of no major deliverables, significant progress was made on a number of fronts. The development of OpenCyc 1.0 neared completion, although the release was held up by issues with the Windows version. (The underlying issues were corrected in the ResearchCyc version, so migrating these results to the OpenCyc release should follow quickly.) Support was added to Cyc's built-in http server to enable the invocation of Cyc functions via HTML, but additional work is required in this arena. Support was added for the LINK parser and the SHOP-based planner functionality, along with examples if its user, were made available. Lastly, work continued in the support of external ResearchCyc users.

Q3 2005

This period saw the start of the ramp down of the ARC project and the preparation for the follow-on, Self-Sustaining ResearchCyc phase. In large part, the effort during this period was focused on using remaining funding to support existing ResearchCyc users, although some development work continued driven largely by the needs of the existing users. In particular:

- additional methods were provided to simplify and increase the flexibility of specifying and exporting selected portions of the knowledge base;

- the various potentially confusing (and possibly redundant) means for representing a "topic" within Cyc, useful in specifying a set of knowledge to export, were consolidated and simplified;
- lexifcation was improved on a significant subset of the KB content; *and*
- improvements were made to the system performance on Windows platforms, especially in the area of improved memory management.

Q4 2005

Several tasks during this final quarter were aimed at improving system robustness as well as increasing the KB content made available in ResearchCyc. Work began on a more formal review of methods for identifying KB and inference incompleteness as a precursor for addressing these gaps during the follow-on project. Planning began for vastly increasing ResearchCyc content by changing from an Opt-in to an Opt-out approach the specification of ResearchCyc KB content. In other words, by default, KB terms *would* now be included in ResearchCyc unless there were specific reasons to exclude them, such as being proprietary, internal, or experimental. [Note that during the summer of 2006, a version of ResearchCyc based on this approach was released that included a sharp increase in the amount of KB content made available; this increase was also reflected in a similarly larger OpenCyc ontology released at the same time.]

And Beyond…

As eluded to several times above, the ARC project results, while in some ways altered from the original project expectations based on experience gained during the course of the work and feedback from users, were sufficient to gain a loyal and growing cadre of ResearchCyc users. The feedback from these users was a significant factor in DARPA's decision to support a continuation of this work under the Self Sustaining ResearchCyc project. In the period since the completion of the ARC task of the initial project described herein, the ResearchCyc community has continued to grow steadily (to over 225 research organizations at the time of this writing) with the range of work being done by these researchers continuing to diversify. Lessons learned during this initial phase have played an important role in the development of subsequent work on the Cyc system and the capabilities offered to the R&D community.

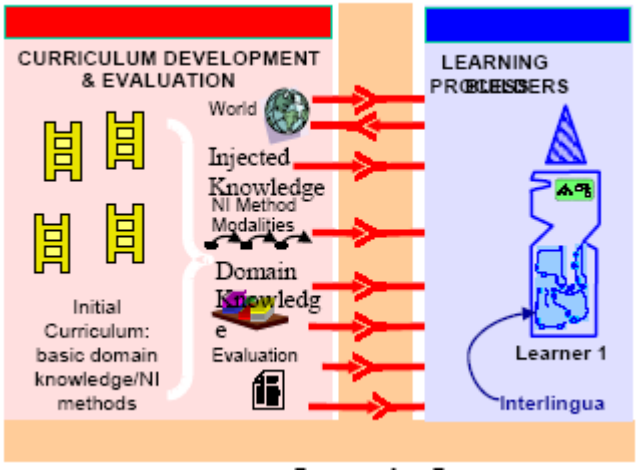
4.0 Part II: Bootstrapped Learning Seedling

4.1 Objective

Bootstrapped learning (BL) is a bold new area in machine learning (ML) that is distinguished from traditional ML in the following respects: While traditional ML is a process of discovery requiring exposure to thousands, if not millions, of annotated examples, BL is a process of directed, instruction-based learning, where the machine is effectively taught, using natural methods of instruction familiar from human learning, what the target knowledge and capabilities are. The benefits to the BL approach to ML include:

- A significant reduction in the number of examples needed to effect learning;
- The ability to build on what has been learned previously;
- A broader range of possible learning algorithms, reflective of the increased range of instructional methods;
- The opportunity for a broader population of instructors, who don't need to be experts in programming, but need only to be capable of natural instruction (with this comes the promise of field trainable systems).

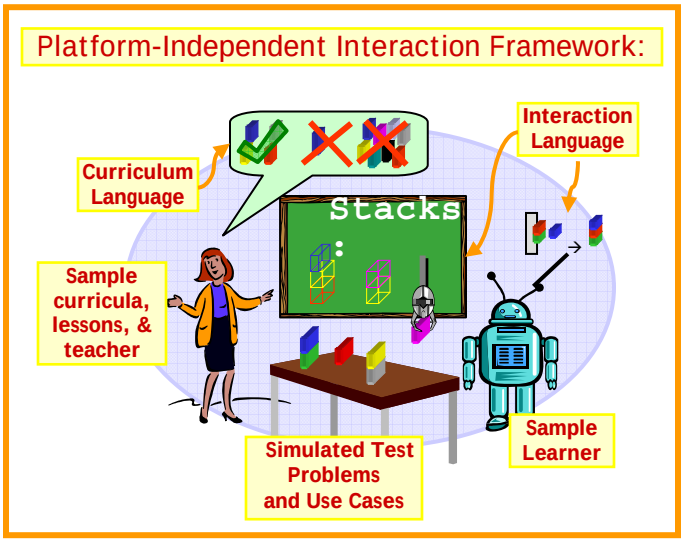
The purpose of the Bootstrapped Learning Seedling was to build the infrastructure behind which genuine bootstrapped learning research could proceed. Over the course of the Bootstrapped Learning seedling effort, we have created an end-to-end prototype of a system that demonstrates minimal versions all major capabilities required by the Bootstrapped Learning program. This prototype will allow rapid development during the next phase of the program, as individual components and capabilities can readily be separated and expanded upon by the appropriate performers; for example, the Basic Student can be elaborated with minimal effort spent coordinating with the communication framework.



This prototype includes:

- A platform-independent **Interaction Framework**. This framework includes parsers and interpreters for the necessary languages, as well as basic debugging support; architecture, components, and capabilities are fully documented.

- **Test problems and use cases,** including a simulator and set of curricula for the initial blocks-world effort, as well as designs for UAV and CAD domains. These initial curricula cover *all NI methods* at a basic level, as well as sample teachers.
- A **Basic Student,** which uses the languages defined to interact with the teaching framework, receives messages, takes tests, and receives "injected" knowledge when tests are not passed.
- A formal language specification (an **Interlingua**) for embedding the languages used to encode target and learned knowledge, upon which new knowledge can be learned.



- A number of **Core BL languages**, written in Interlingua, in which target and learned knowledge can be expressed.
- **Curriculum Language** (*Interlingua*), which allows easy crafting of lessons and learning tasks. This language was tested with blocks-world curriculum development.
- An **Interaction Language,** which allows separate learner technologies to interact with the teaching framework, as tested against the Basic Student.

This section describes the above architectural elements, some of the technical difficulties encountered and methodologies used. Full documentation of these deliverables, delivered to DARPA and distributed to Year 1 Bootstrapped Learning Program participants in May 2007, accompanies this report. The material covered here is organized topically into three areas: BL language development, the BL framework, and curriculum materials.

4.2 BL Language Development

Interlingua

The design of the BL interlingua, the "native" language of bootstrapped learning agents, was shaped by two major requirements: First, in order to support a variety of learning modalities and learning domains, the interlingua must be expressive enough to support any language optimized for representing knowledge. For example, procedural knowledge (knowledge how to do some task) is typically efficiently encoded in production rules or in the form of if…while…else routines. Similarly, theories of embedded languages of the interlingua are extensions of this upper ontology.

The second requirement was learnability: No matter what the target "core" language for a given learning task is, new concepts/knowledge encoded in that language must be

capable of being arrived at by applying a handful of simple algorithms for extending the language.

To achieve both, the Interlingua was designed on object-oriented, class-based paradigms, where objects are created by constructing instances of classes in a way that assigns values (other objects) to parameters that have been defined as valid for an instance of the class. For example, here is an object that instantiates the **Person** class:

```
Person(name=bob,age=39,spouse=Person(name=kate,age=37,spouse=bob))
```

Here the object has three parameters, **name**, **age**, and **spouse**, the first of which takes a symbol as a value, the second an integer, and the third another instance of **Person**. The **name** parameter can be used to assign a symbol to an object as a method of referring to the object (as in the value of **spouse** in the object designated **kate**). The validity of parameters and the range of their constraints is inherited from more general classes to more specific classes, so that, for example, the class **AmericanPerson**, as a sub-class of **Person**, would inherit **name**, **age**, and **spouse**. To support learnability, there are only two methods of extending this language. The first is to declare a new class, via the relation is. For example, the following declaration extends the class **Device** with a new subclass:

```
is Container Device;
```

The second is to declare the validity and value-range of a parameter for the members of a class. This is done using the relation **arg**. For example, the following declares the parameter **capacity** to be valid for any instance of **Container**, and constrains the range of values to instances of the **Volume** class:

```
arg Container capacity Volume;
```

One could then extend the **Volume** class by defining **Liter** with these two declarations:

```
is Liter Volume;
arg Liter number Number;
```

Jointly, the four declarations license the construction of the following IL object, which represents an instance of container with a 5 liter capacity:

```
Container(capacity=Liter(number=5))
```

This simple syntax[1] supports the learnability of new concepts (classes) by restricting the vocabulary with which these concepts can be described. The embedded languages for encoding target/learned knowledge are defined merely as slices of Interlingua; i.e., as

[1] The detailed specification and semantics are presented in the Interlingua specification document that accompanies this report.

extensions to a core number of classes that make up the Interlingua upper ontology. The concepts expressed in these languages are learnable in the sense described here: Each concept ultimately is captured in a series of **is** and **arg** declarations. The expressivity requirement is achieved by defining an evaluation environment that imposes its own semantics, combined with the ability to generate rich hierarchies by iteratively extending classes (via **is**) and adding and tightening parameter definitions (via **arg**). We discuss the essential details in the next section.

Core BL Languages

In computer science, different formal languages are optimized to support different tasks. SQL, for example, is a language optimized for efficient retrieval of data stored in relational databases. First order logic is a language that is optimized for the declarative representation of generalizations that support inference. Imperative programming languages, such as C, are optimized for encoding and applying procedural knowledge. The interlingua is capable of supporting a variety of languages, which allows it to support a broad range of types of learning. In this section, we first describe the IL infrastructure by which a language can be defined and embedded in the Interlingua. Full details are given in the accompanying Interlingua specification document.
A language can be embedded in IL in the following way:

First, the vocabulary of that language is defined by defining classes that extend the class **Executable**. **Executable** is defined with a **returnValue** parameter,

```
arg Executable returnValue Thing;
```

that is inherited by its sub-classes. This parameter serves to encode the return of any code executed in course of evaluating IL objects constructed from these sub-classes.

To define code for a sub-class of **Executable**, one instantiates the class **CodeBody**. Each instance of the **CodeBody** class will contain a formula that, when executed by the execution environment, will assign a **returnValue** value to the associated IL object.

Each extension of the class **ExecutionEngine** defines an execution engine to be used to execute the code associated with a class of executable invocations.
Finally, the classes that define the vocabulary of a language are associated with code and an execution engine by the meta-relation **defCode**:

```
defCode <Class> <Environment> <Code>;
```

For example, the class of **Factorial** function-invocations is defined (**ArithmeticFunction** is a sub-class of **Executable**):

```
defClass Factorial extends ArithmeticFunction (
   Number n
);
```

Then an instance of **CodeBody** (**ProcedureBody** is a sub-class of **CodeBody**) is associated with the **Factorial** class and the engine that will execute the code when a **Factorial**-instance is evaluated:

```
defCode Factorial ProcedureEngine ProcedureBody(
          If(Equals(n,0),
             Then(Return(1)),
             Else(Times(n,Factorial(n=Plus(n,Minus(1)))))
);
```

Thus the evaluation of this IL object,

```
Factorial(n=7)
```

results in the execution of the **ProcedureBody** instance above, and results in the calculation and assignment of the **returnValue**:

```
Factorial(n=7,returnValue=5040)
```

Using this method, the seedling effort produced specifications for languages optimized for learning in the following areas:

- Function composition
- Production Systems
- Declarative Knowledge Representation
- Logic
- Procedures
- Markov Logic Networks

Full details are in the accompanying Interlingua specification document.

Interaction Language

Not an embedded language *per se*, the Interaction Language consists of the IL classes that define what types of messages can be passed among instructors, students, and the world (simulators). It essentially defines a communication protocol and is not part of what the student learns. The **Message** class is defined with parameters that help provide context, such as timestamp information, who is authoring the message, who the intended recipient is, and which message, if any, a message is in response to:

```
defClass Message extends Object (
   Timestamp timestamp,
   Agent source,
   Agent addressee,
   Message responseTo
);
```

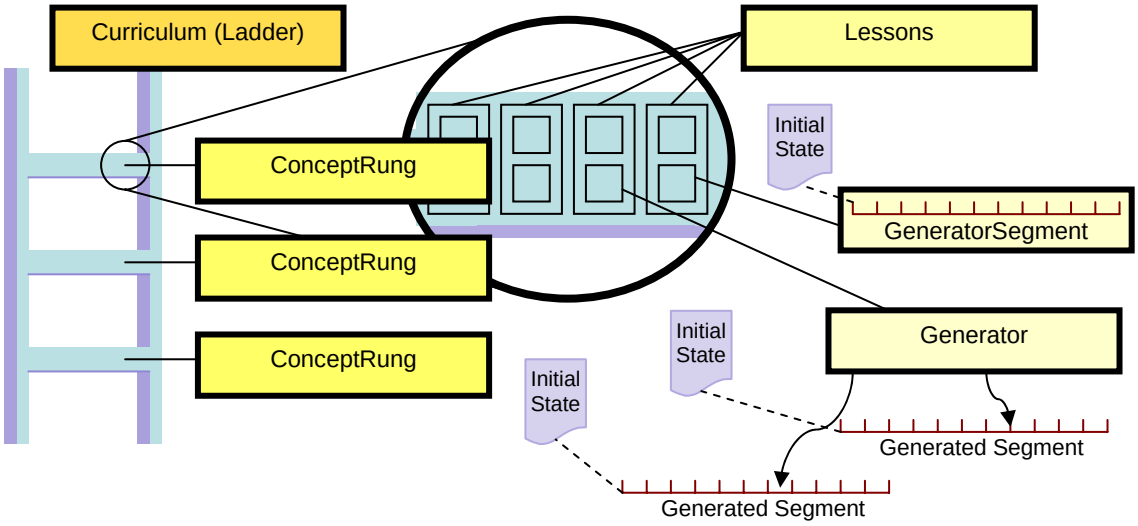
Sub-classes of **`Message`** reflect natural classes of interaction; for example, the **`Imperative`** class is used to form messages that contain requests; the **`Begins`** and **`Ends`** classes provide temporal information about the start and end of actions in the world; **`Perception`** messages allow the simulator to communicate how actions have affected world state; **`Utter`** messages support declarative interactions, including the ability to **`Gesture`** while describing an object or state of affairs.

Full details are given in the Interaction Language document.

Curriculum Language

The Curriculum Language is used to represent curricula and the objects used in their generation and execution. Like the Interaction Language, the Interlingua class hierarchy and definitions that comprise the Curriculum Language are neither targets of learning nor subject to revision or extension by learning algorithms.

It is helpful to describe a Bootstrapped Learning Curriculum as a *ladder*, where each rung on the ladder corresponds to a concept to be learned. Each rung contains a number of lessons designed to teach that concept. Individual lessons are executed by calls to a number of generators, functions that return *lesson segments*, defined as a series of Interaction Language messages to be posted to a timeline specific to a world. Concepts (rungs) are arranged hierarchically in a given curriculum, where the mastery of a 'lower' concept is a prerequisite for receiving instruction on each 'higher' concept.



A full specification of the IL classes that make up the Curriculum Language can be found in the accompanying Interaction Language document, which describes both the Interaction and Curriculum languages.

4.3 BL System Prototype

In November 2006, at the direction of the DARPA program manager, Seedling participants began work on the design and implementation of a prototype bootstrapped learning system, to be ready for Bootstrapped Learning Program participants by the time that program started. In this section, we describe the work on this system, an initial version of which was delivered on February 15, with updates delivered to DARPA on March 9, and finally distributed to program participants on May 2.

Conceptually, the prototype decomposes into a virtual machine and a framework for agent interaction. Each is described in turn.

Virtual Machine

The virtual machine (VM) is the component responsible for parsing, validating, and evaluating/executing IL objects (formulae). In so doing, the VM resolves symbolic references to the appropriate IL objects, which enables rapid type-checking. Validation and evaluation is done from the inside out, so that, for example, the validation of this form

```
Container(capacity=Liter(number=Plus(2,3)))
```

would begin with the validation of `Plus(2,3)` will be evaluated first, so that its `returnValue` can be checked to see if it meets the `arg` constraint for `number` in a `Liter` object.

In its current implementation, evaluation takes place at load time. This presented some challenges for the implementation of the Interaction Language, which required a quoting mechanism to prevent the validation of messages by the VM to cause premature calls to the simulator during the execution of curricula. The details of this workaround are part of the accompanying curriculum tutorial document. The full details of the evaluation of IL can be found in the Interlingua specification document.

Framework

The framework is code infrastructure that ensures that agents that interface with the framework will receive appropriate messages as they are posted to a *timeline*, which the framework implements as a temporal ordering of messages written in IL. The figure below illustrates how the framework handles message passing.

The framework produces a session trace which is useful for visualization and debugging. An example session trace for an early blocksworld lesson is included in the accompanying documentation. The framework was developed in Java, after potential BL researchers were canvassed for programming language preferences. Full details about the

framework are given in the accompanying Framework and Virtual Machine Overview document.

4.4 Curriculum Materials

In this section we describe the curriculum materials produced during the seedling.

Prior to developing the prototype system, the seedling effort produced curricula descriptions with sample code, to give researchers a sense of the kind of learning tasks that bootstrapped learning can support. These documents were:

Blocks World Curriculum Description: a simple example designed to convey basic NI methods and the look and feel of the Interaction Language.

UAV Curriculum Description: An example of using BL to train a UAV, which is considered a practical objective with a clear military transition. Such an application is not expected to require advanced forms of bootstrapping.

Representation Shift and Mapping: This example comes from the domain of computer assisted architectural design, involves shifting from one representation to another, as well as the integration of heterogeneous components. It is tantamount to bootstrapping with full generality, and a goal for the end of the Bootstrapped Learning Program.

The publication of these materials coincided with a document, produced by ISI, who subcontracted to Cycorp on this effort, which described the natural instruction methods used to teach a curriculum.

As part of the development of the prototype system, knowledge engineers at Cycorp developed an IL curriculum that could be loaded and tested in the framework. The lessons that comprised this curriculum touched all basic NI methods, and exhibited bootstrapping: Early lessons were essentially “physics” lessons that taught the student the how to apply actuators without error, and what their effects were on a simulated blocks world state. Later lessons used this knowledge to enable the student to learn how to make a three-block stack.

In order for the curriculum to be successfully executed, the Cycorp team also wrote a blocks world simulator, which was able to receive messages from the instructor and post appropriate `Perception` updates in response. The first version of the simulator was written in Lisp, to demonstrate the platform independence of the framework with respect to external components such as simulators. Because Java is more readily available than Lisp, the simulator was later rewritten in Java.

4.5 Enumeration of Bootstrapped Learning Deliverables

The following documents constitute all non-code deliverables for the Bootstrapped Learning Seedling:

Language Documents

Interlingua Specification: This document describes the Bootstrapped Learning *interlingua*, which serves as the knowledge interchange format for bootstrapped learning (BL). It is used by learning processes to encode knowledge learned from a curriculum (ladder). It also is used to represent assumed starting knowledge (*genetic/injected knowledge*) given to the learning system within a ladder.

Interaction Language and Curriculum Language Specification: This document describes the Interaction Language for the Bootstrapped Learning (BL) program. We begin with the central concept of a *timeline*, the mechanism by which all information is conveyed between the student and everything else in the world (including the instructor, the curriculum, and the physical world). We define the language for describing specific classes of *interaction modalities*, or methods of interaction. We then turn to the concept of a *world*, which, together with interactions and timelines, form the building blocks of *lessons* and *curricula*, treated in the final section.

Framework Documentation

Framework Description: This document provides an overview of the key classes and methods for invoking the Bootstrap Learning Framework in order to teach a curriculum to a student.

Javadocs: Boostrapped Learning framework interfaces.

Curriculum Materials

Blocksworld Document: Provides a simple curriculum for bootstrapped learning, which is intended to illustrate the structure of a curriculum and the use of the interaction language, worlds, states, timelines, etc.

UAV Document: Presents a sample curriculum for teaching an autonomous Unmanned Aerial Vehicle (UAV) to conduct progressively more complex surveillance missions.

Consider a unit deployed in hostile territory with insurgent behavior that has been presented with a UAV to assist with intelligence gathering. Initially, out of the box, it can fly a mission given various targets (e.g., GPS coordinates) and the orders to "survey" those locations, i.e., take various pictures of them. In addition, the UAV has basic object recognition capabilities and can label vehicles, people, buildings, etc. that appear in its visual field. The objective is to have the UAV acquire additional capabilities by learning from various types of instruction.

CAD Document: Illustrates the concepts *representation shift* and *representation mapping*. Representation shifts occur when the student decides to change its internal representation of a problem. Representation mappings occur when the student needs to take its internal representation of a problem and translate it to the representation needed to perform a different kind of problem solving. The ability to execute representation shifts and representation mappings are among the most difficult challenges that an automated learner might face.

Natural Instruction Methods Description: There are many alternative approaches for computers to acquire knowledge from human teachers. We can find diverse terminology and technologies in the literature of knowledge acquisition, programming by demonstration, machine learning, and natural language to name a few. The terminology used to denote these forms of learning does not make appropriate distinctions to characterize the research space in this general area. For example, "learning by observation" focuses on the teaching activity (having an indifferent teacher), "learning by demonstration" refers to the type of interaction, and "learning by refinement" refers to the student's learning process. In order to characterize alternative approaches to instruction, we define first some basic terminology and distinctions. We then describe a core set of instruction methods and characterize them in those terms.

Curriculum Tutorial: This document gives a description of lessons learned in the course of developing a fully fleshed-out blocks world curriculum, and attempts to provide an informal overview of the process of curriculum writing, along with examples and advice intended to shorten the time required to become skilled at curriculum development.

Loadable Curriculum Description: Presents an overview of the lessons and goals of the current blocks world curriculum.

5.0 List of Symbols, Abbreviations and Acronyms

API – Application Programming Interface

ARC – Accessible Research Cyc System

BL – Bootstrapped Learning

CAD – Computer Aided Design

Cyc – Cyc Knowledge Base System

CycL – Cyc Representation Language

HL – Heuristic Level

KB – Knowledge Base

ML – Machine Learning

NL – Natural Language

OWL – Web Ontology Language

RKF – Rapid Knowledge Formation

UAV – Unmanned Aerial Vehicle

XML – Extensible Markup Language