

SRI International

3 April 2012

MABLE FINAL REPORT

A005 - FINAL TECHNICAL REPORT

Contract No. HR0011-07-C-0060

SRI Project P19402

Covering the Period 05/02/2010-30/11/2011

Prepared by:

Dr. Michael Freed (PI)

Artificial Intelligence Center

THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.



333 Ravenswood Avenue • Menlo Park, California 94025-3493 • 650.859.2000 • www.sri.com

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 30-11-2011		2. REPORT TYPE Final Project Report		3. DATES COVERED (From - To) 05/02/2010-30/11/2011	
4. TITLE AND SUBTITLE MABLE Final Report				5a. CONTRACT NUMBER HR0011-07-C-0060	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Michael Freed				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Ave. Menlo Park, CA 94025				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA DARPA Contracts Management Office 3701 North Fairfax Dr. Arlington, VA 22203-1714				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER	
12. DISTRIBUTION AVAILABILITY STATEMENT unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This is the final report describing research objectives, approach and results for work by SRI International and its subcontractors under the DARPA Bootstrapped Learning Program.					
15. SUBJECT TERMS Programming by Instruction, Software Engineering, Machine Learning, AI					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 122	19a. NAME OF RESPONSIBLE PERSON Michael Freed
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			19b. TELEPHONE NUMBER (Include area code) 650-859-4864

TABLE OF CONTENTS

SECTION	PAGE
LIST OF FIGURES	iv
LIST OF TABLES	v
1. INTRODUCTION	1
2. INTEGRATED BOOTSTRAPPED LEARNING OVERVIEW (SRI INTERNATIONAL)	4
2.1 TASK OBJECTIVES	4
2.2 TECHNICAL PROBLEMS	5
2.3 GENERAL METHODOLOGY	6
2.4 TECHNICAL RESULTS	7
2.5 IMPORTANT FINDINGS AND CONCLUSIONS	8
2.6 SIGNIFICANT DEVELOPMENTS	9
2.7 IMPLICATIONS FOR FUTURE RESEARCH	9
3. LEARNING CONCEPTS FROM EXAMPLES AND FEEDBACK (UNIVERSITY OF WISCONSIN)	10
3.1 TASK OBJECTIVES	10
3.2 TECHNICAL PROBLEMS	11
3.3 GENERAL METHODOLOGY	11
3.4 TECHNICAL RESULTS	19
3.5 IMPORTANT FINDINGS AND CONCLUSIONS	20
3.6 SIGNIFICANT DEVELOPMENTS	20
3.7 IMPLICATIONS FOR FUTURE RESEARCH	21
4. LEARNING PROCEDURES FROM EXAMPLES AND FEEDBACK (OREGON STATE UNIVERSITY)	21
4.1 TASK OBJECTIVES	21
4.2 TECHNICAL PROBLEMS	22
4.3 GENERAL METHODOLOGY	24
4.4 TECHNICAL RESULTS	24
4.5 IMPORTANT FINDINGS AND CONCLUSIONS	32

4.6	SIGNIFICANT DEVELOPMENTS	33
4.7	IMPLICATIONS FOR FUTURE RESEARCH	34
5.	LEARNING CONCEPTS FROM TELLING (UNIVERSITY OF ILLINOIS CHAMPAIGN-URBANA)	35
5.1	TASK OBJECTIVES	35
5.2	TECHNICAL PROBLEMS	36
5.3	GENERAL METHODOLOGY	36
5.4	TECHNICAL RESULTS	39
5.5	IMPORTANT FINDINGS AND CONCLUSIONS	41
5.6	SIGNIFICANT DEVELOPMENTS	41
5.7	IMPLICATIONS FOR FUTURE RESEARCH	41
6.	LEARNING PROCEDURES FROM TELLING (ISI)	41
6.1	TASK OBJECTIVES	41
6.2	TECHNICAL PROBLEMS	42
6.3	GENERAL METHODOLOGY	45
6.4	TECHNICAL RESULTS	49
6.5	IMPORTANT FINDINGS AND CONCLUSIONS	63
6.6	SIGNIFICANT DEVELOPMENTS	68
6.7	IMPLICATIONS FOR FUTURE RESEARCH	74
7.	LEARNING BY NOTICING	75
7.1	TASK OBJECTIVES	75
7.2	TECHNICAL PROBLEMS	75
7.3	GENERAL METHODOLOGY	76
7.4	TECHNICAL RESULTS	78
7.5	IMPORTANT FINDINGS AND CONCLUSIONS	82
7.6	IMPLICATIONS FOR FUTURE RESEARCH	83
7.7	REFERENCES	85
8.	CONTROL OF LEARNING PROCESSES (UTAH STATE UNIVERSITY)	85
8.1	TASK OBJECTIVES	85
8.2	TECHNICAL PROBLEMS	86
8.3	GENERAL METHODOLOGY	86

8.4	TECHNICAL RESULTS	86
8.5	IMPORTANT FINDINGS AND CONCLUSIONS	87
8.6	SIGNIFICANT DEVELOPMENTS	87
8.7	IMPLICATIONS FOR FUTURE RESEARCH	87
9.	CONTROL OF LEARNING – PART 2 (UNIVERSITY OF MASSACHUSETTS)	87
9.1	TASK OBJECTIVES	87
9.2	TECHNICAL PROBLEMS	88
9.3	GENERAL METHODOLOGY	88
9.4	TECHNICAL RESULTS	89
9.5	LEVERAGING LEARNING FAILURES	91
9.6	IMPORTANT FINDINGS AND CONCLUSIONS	92
9.7	SIGNIFICANT DEVELOPMENTS	95
9.8	IMPLICATIONS FOR FUTURE RESEARCH	95
9.9	REFERENCES	96
10.	SERVICE FOR CONCEPT INTERPRETATION (UNIVERSITY OF TULSA)	96
10.1	TASK OBJECTIVES	96
10.2	TECHNICAL PROBLEMS	97
10.3	GENERAL METHODOLOGY	97
10.4	TECHNICAL RESULTS	99
10.5	IMPORTANT FINDINGS AND CONCLUSIONS	99
10.6	SIGNIFICANT DEVELOPMENTS	99
10.7	IMPLICATIONS FOR FUTURE RESEARCH	99
11.	A PROGRAMMING BY INSTRUCTION SYSTEM (SRI INTERNATIONAL)	100
11.1	OBJECTIVES	100
11.2	iBL OVERVIEW	102
11.3	APPLICATION LAYER	108
11.4	SIMULATION LAYER	109
11.5	TEACHING LAYER	112

LIST OF FIGURES

FIGURE	PAGE
1. General architecture for Programming by Instruction.	2
2. MABLE components and team roles.	2
3. MABLE domains.	3
4. Primary Task Objectives.	4
5. The Onion layers.	12
6. An illustration of a top-down ILP search for an inference rule to predict <i>target</i> , whose definition is the conjunction of literals $q \text{ AND } r \text{ AND } \dots \text{ AND } z$	14
7. Wisconsin's HCI paradigm	18
8. Prototype GUI for Wargus.	19
9. TellMe User Interface (screenshot).	50
10. Data Objects (screenshot)	56
11. Components (screenshot).	56
12. Procedure (screenshot).	57
13. System Learns (screenshot)	59
14. Parameter Setup (screenshot).	60
15. Execution Details (screenshot)	61
16. System Processes (screenshot).	62
17. Workflow Sample	63
18. Initial Interaction.	69
19. Articulated Instruction	69
20. Two examples of procedures	70
21. Examples of two sets of instructions entered by two different subjects	70
22. Subject chooses option.	72
23. Non-integrated steps (screenshot).	73
24. Additional Work Performed by CCM and CCMno Strategies (normalized to the baseline strategy).	95
25: FC-CBJ Algorithm.	98
26. FC-CBJ Labeling	98

27. FC-CBJ unlabeleding	98
28. iBL System Overview	107
29. UI Layout	108
30. Control Bar.....	108
31. Example RAP from Firby (1989) in the domain of robotic trucks	111
32. Teaching UI	113
33. (a) initial KAGs lead to (b) (partially) specified user intent.....	116
34. Dialog Episode Operators and sequence for iBL_demo_2	118

LIST OF TABLES

TABLE	PAGE
1: Subject Findings	71
2. Concept Learning Performance	94

1. INTRODUCTION

Current generation software is developed, modified, and maintained by time-consuming and expensive development processes that rely on computer programmers, not domain experts or users. This is especially problematic for complex software deployed in evolving operating environments where flawed initial understanding of requirements and gradual requirements drift are almost inevitable. The objective of the Bootstrapped Learning program was to make it possible for end users to bypass software engineers, modifying code by a process that resembles instructing a person rather than anything that requires direct interaction with computer code. In this vision, the capacity to be modified by instruction would be instantiated as a software service incorporated into new systems.

The overall problem of Programming by Instruction (PbI) can be thought of as having three parts: acquisition, distillation and modification. First, technology for interacting with users to acquire instructional content must bridge the gap between user and machine. That is, it must allow users to express instructional content in an acceptable way while cueing them to fill in needed information they wouldn't think to express. And it must be able to translate informally expressed content into formal representations that ultimately support specific changes to a pre-existing code base.

Second, precisely defined requirements must be distilled from this formalized instructional content. The key problem is that the amount of content users will be able to provide in any realistic setting will tend to fall far short of what would be needed to make this step easy. The central insight behind the Bootstrapped Learning program is that the problem of discovering the implied structure underlying a diffuse body of instructional content is formally equivalent to the problems that many machine learning (ML) algorithms were designed for. The research focus of the Bootstrapped Learning program was to test and elaborate this idea while adapting machine learning algorithms for this purpose.

Third, software requirements derived from user-supplied instruction must be translated into specific code changes to an otherwise working system. This requires understanding the requirements that the system currently meets and planning a set of specific point modifications to the code base that achieve the new requirement without undermining the old ones. Figure 1 shows a generalized system architecture involving these three steps and highlighting the three forms of human instruction investigated by the program:

- **Learning by telling:** teacher supplies fact and procedural knowledge directly
- **Learning by demonstration:** teacher shows system how to perform correctly
- **Learning by feedback:** teacher advises on how to improve on current behavior

In addition, we assume that all forms of instruction require that the learning system has some prior knowledge needed to interpret the instruction. This is either supplied as initial data (which the system uses to “bootstrap” acquisition of more knowledge) or is acquired incidentally during system operations through processes for **Learning by noticing**.

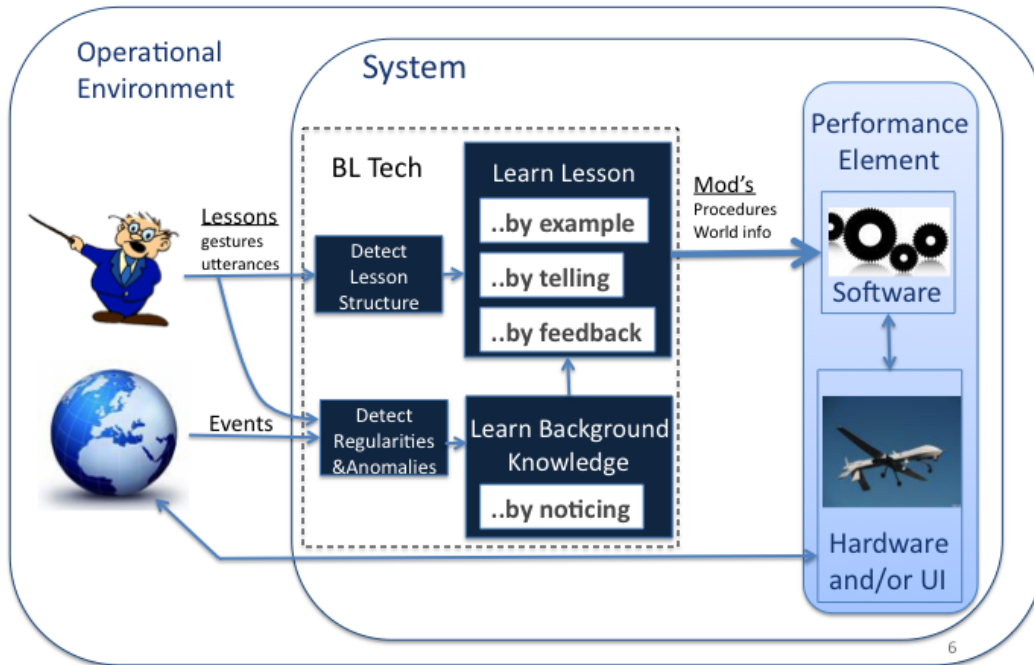


Figure 1. General architecture for Programming by Instruction.

The SRI-led research effort on Bootstrapped Learning was carried out in three, roughly year-long phases and included numerous subcontractor organizations focused on different parts of the overall problem. All phases included development of a core system to investigate called MABLE used to investigate an integrated approach to PbI – that is, an approach that allowed users to teach by any of the supported methods and to modify the behavior of several different systems. Subcontractor roles in developing aspects of MABLE are illustrated in Figure 2.

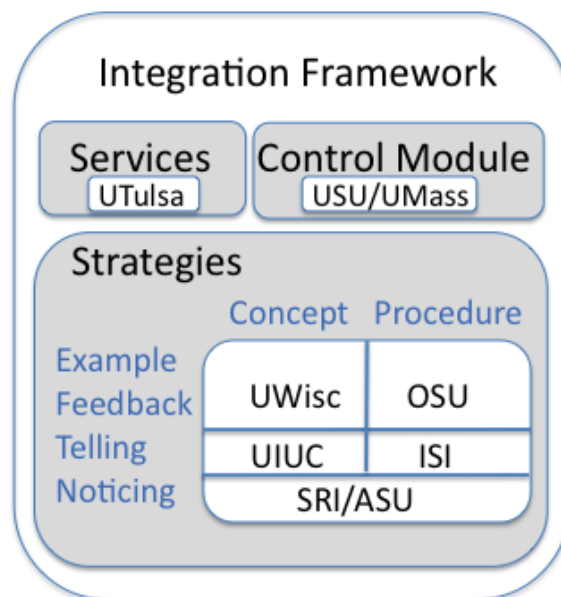


Figure 2. MABLE components and team roles.

BL demonstrated the first version of MABLE, including all four instructional interaction types, in August 2008. The two initial applications were chosen both for simplicity, i.e. the ability to drive platform development without excessive focus on the application itself, and for representativeness of the kinds of applications in which instructability is most needed. One of the chosen domains (“Blocksworld”) focused on simple mechanical design problems, driving capabilities related to problem-solving and transactional interaction modes. The second domain involved controlling a player in a simulated sports game (“Robocup Soccer”), representing BL applications centered on automation, control and interactivity. Subsequent phases developed MABLE using three additional domains, called “Diversity Domains,” all more complex and demanding than the initial two. In phases 2 and 3, MABLE was assessed yearly by an outside evaluation team led by BAE using another domain, referred to as the “Hidden Domain” because it was not known to the SRI-led research team. See figure 3 for a summary of these problem domains.

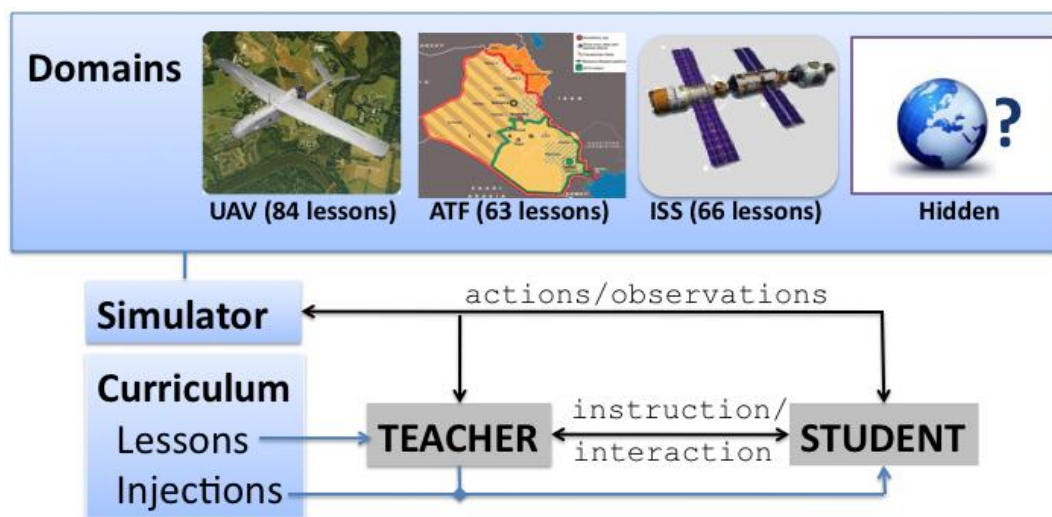


Figure 3. MABLE domains.

MABLE made it possible to investigate integrated PbI issues including how to combine different learning methods, how to accumulate knowledge and improve system performance over a sequence of lessons and how to decide which of several alternative ML-backed distillation techniques to use for a given body of instructional content. However, the requirements of developing component technology in an evolving, integrated system designed to meet the needs of many different research efforts inhibited investigation of some important PbI ideas. For that reason, most research teams carried out some research designed for inclusion in MABLE and some designed for separate evaluation.

In addition, MABLE was only designed to support research on the second of the three core PbI problems (distillation). After phase 2 evaluation showed very strong performance in that area, the SRI core team was able to expand its effort to the overall PbI problem including all three sub-problems (acquisition, distillation and modification). This resulted in a prototype PbI tool called iBL (interactive Bootstrapped Learning).

This report covers MABLE, component-level research efforts and iBL as follows. Section 2 describes the MABLE integration framework. Sections 3 through 9 describe research efforts of

specific sub-contractor teams including both MABLE components (as indicated in diagram 2) and component efforts outside of MABLE. Section 10 describes iBL and indicates future directions for Pbl research.

2. INTEGRATED BOOTSTRAPPED LEARNING OVERVIEW (SRI INTERNATIONAL)

2.1 Task Objectives

From the outset of the Bootstrapped Learning project, SRI had five primary task objectives related to the research and development of the Student (MABLE – Modular Architecture for Bootstrapped Learning Environments) portion of the Bootstrapped Learning Framework:

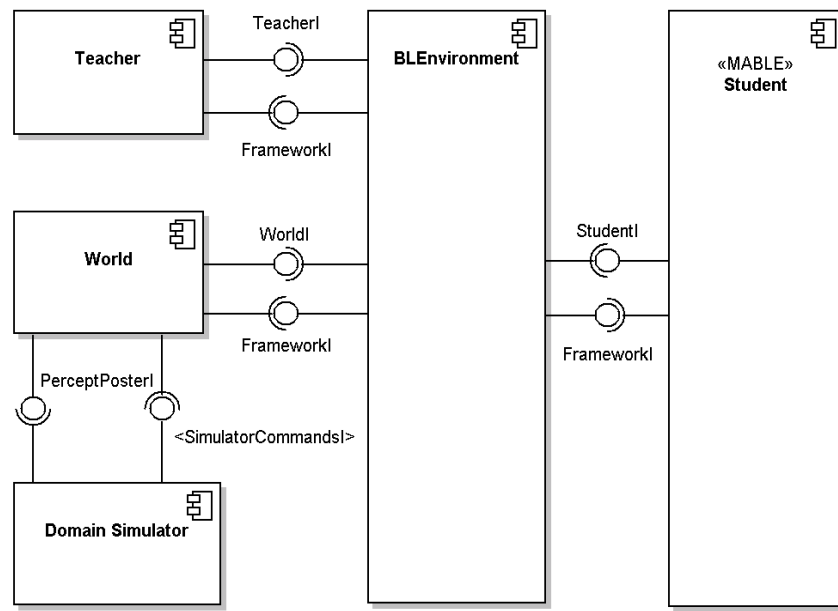


Figure 4. Primary Task Objectives.

- **Program Management** of the research, development, and testing of the student portion of the Bootstrapped Learning framework.
- **Integration** of contributions from SRI's subcontractors (research teams) to deliver a single, cohesive solution to the Bootstrapped Learning program. The provision of harnesses and experimental plans for testing the individual components of the system. In addition, SRI was responsible for integrating the Robocup Soccer simulator and providing a curriculum for use in evaluating the system at the end of Phase 1.
- **Control Module** design and implementation responsibility, with assistance from team members, along with supporting services – the Learning Control Strategy, Knowledge Management, and Execution Engine.
- **Learning Strategies, NIMs, and Interaction Modalities** design and development by subcontractors to be directed and coordinated by SRI. In addition, SRI was to provide

support to the curriculum development team by providing input into the design of the NIMs, the Interlingua, and the Interaction Language.

- **Learning Services** design and development were similarly to be directed and coordinated by SRI.

2.2 Technical Problems

The technical problems encountered changed as the project progressed through its three phases. In Phase 1, the focus was on putting in place the foundational software architecture from which to build out and test the research capabilities of the system. Issues of primary technical concern that were encountered, included:

- **Changing Interlingua and Interaction Language** definitions were frequent in Phase 1 as understanding of the needs of the project matured. As these languages acted as a common glue between the components in the system, changes in these often resulted in a significant ripple effect across the framework. This cycle was exacerbated by the fact that the framework was written entirely in Java and required a specialized interpreter be written for the Interlingua based languages (the ILVM). Initially, only a minimalistic API was available for working with these languages in Java. This API was verbose to code against and did not support static type checking of the language, which added to the difficulty in performing change impact analysis.
- **Cost of Integrating MABLE with new BL Framework** releases was significant in Phase 1. While releases of the framework were received promptly on a monthly cycle, it normally took up to a week to track down and fix problems that arose during the integration. This was due primarily to the effort involved in trying to diagnose the effects of changes to the Interlingua based languages and the behavior of their interpreter.
- **Runtime Performance** of the framework's messaging, Interlingua Interpreter, and MABLE's internal working memory was poor. It was possible to offset the effects of this in the end of year evaluation by extending the length of time available for the individual tests. However, it still impacted negatively on the daily code-test-debug cycle of those developing and testing code for the system.
- **Control Module to Learning Strategy Protocol** during preparations for the phase 1 evaluation, was found to be insufficient. It required the individual Learning Strategies to each perform a significant amount of preprocessing on the messages received on the timeline in order to be able to activate their learning capabilities. In particular, this caused significant repetition in code for handling basic tasks such as identifying the boundaries of lessons and determining the believed state of the world based on the perception information received.

Several initiatives were undertaken at the beginning of Phase 2 to address the issues encountered during Phase 1 of the program. SRI and BAE worked closely together to standardize on a common set of Java APIs (IL 2.0) to work with the Interlingua based languages. This required significant refactoring across the framework during the first three months of Phase 2 and the full framework did not stabilize for another 3 months. However, this work had significant benefit. Integrations of new releases of the framework and MABLE could be done in under a day, allowing weekly and sometimes daily integrations to occur when needed. This supported a more rapid turn around in meeting feature and defect fix requests.

During the refactoring stage of Phase 2, overall runtime performance across the main components in the system was improved. The protocol between the Control Module and Learning Strategies was redesigned. This reduced the preprocessing requirements of the individual Learning Strategies, permitting the developers of these components to focus more on their learning capabilities. Initially it had been intended to use Markov Logic Networks as the underpinning for sharing hypothesis and performing inference within MABLE. However, due to the immaturity and poor performance of available implementations at the time, an alternative mechanism was developed using planning based capabilities and a scoring mechanism for selecting between alternative hypotheses, which proved to be sufficient in meeting the required goals of the project. In addition, an extra service was added to the MABLE framework for use by all learning strategies to provide a common mechanism for handling the believed state of the world.

By the beginning of Phase 3 there were only 2 major outstanding technical problems. Firstly, the messaging between the major components within the system still required an ILVM to process the message at each point in a messages path, adding a costly performance overhead. This approach was replaced early in the phase, improving overall system performance. Secondly, the dialog between the teacher and the student was predominantly one way in that the Teacher would instruct and query the Student. The reverse was not possible, the Student could not make queries of the Teacher in order to support self directed progressive refinement of concepts. The difficulty with this arose primarily with how to automate the Teaching component sufficiently, in order to be able to handle ad-hoc queries from the Student. Automated testing of the interaction between Teacher and Student was critical during Phase 2 of the project in order to meet its evaluation criteria. SRI, in addition to meeting its Phase 3 evaluation criteria, developed the iBL (Interactive Bootstrapped Learning) capability, which investigated an alternative approach to actively learning, via dialog, refinements to concepts to instead use the dialog with a user to guide code transforms to an existing system in order to obtain desired behavior from it.

2.3 General Methodology

SRI's primary role was to integrate contributions from its subcontractors to deliver a single, cohesive solution to the Bootstrapped Learning program. To achieve this a tailored set of Software Engineering best practices were used:

- **Capability Based Milestone** tracking was used to focus efforts early on in order to meet end of phase evaluation criteria. These milestones were coarse grain, based on achieving certain levels of functional capability within the MABLE Student. This approach reduced project administration overhead across each of the research teams. Progress was monitored qualitatively through the assignment of individual liaisons from SRI to each of its subcontractors coupled with weekly status review meetings. These allowed SRI to assess individual subcontractor progress and to provide assistance where needed in a proactive manner. Quantitative progress was tracked using a suite of functional tests that measured the capabilities of the system.
- **Branch and Merge Source Control** was the predominant coordination mechanism used between SRI and its subcontracts for integrating individual components into the system. Each subcontractor was provided with a private branch on which they could develop their components. SRI was responsible for maintaining the main trunk of development and

pushing stable baselines out to the individual branches. In addition, SRI took responsibility at predetermined milestones to merge work from individual branches back into the main trunk. At the latter end of each phase, each subcontractor would normally have migrated from working on their own private branches to the mainline, as the focus of efforts was usually on fixing integration issues and finalizing functionality to meet testing criteria. This approach worked due to the partitioning of functional responsibility in a manner that allowed individual subcontractors to work fairly autonomously of each other, which helped reduce required communication overheads significantly.

- **Hackathons** were introduced in Phase 2 of the project as an effective means of converging divergent work. As each of the subcontractors worked to a large degree independently on their own private branches, focusing on their specific areas of research, an effective, low cost, and timely mechanism was required to synchronize efforts across the project. Hackathons, which consisted of week long collocated collaborative coding sessions including SRI, its subcontractors, the Bootstrap Framework development team (BAE), and the curriculum development team (Cycorp), were used to achieve this. Three such events were held in Phase 2 of the project in order to meet the end of phase evaluation criteria. While intensive to participate in, these sessions proved extremely effective in meeting their goals.
- **A Tiered Testing Harness** was evolved over the first 2 phases of the project. Initially, this comprised three tiers, consisting of a suite of unit tests, testing of individual Natural Instruction Methods (NIMs) across provided Diversity Domains (DDs) from the curriculum team, and full end-to-end DD curriculum tests. By midway through Phase 2, an additional five tiers of testing were added. The most critical of these was the addition of the message replay tier. This allowed a learning strategy developer (i.e. most of SRI's subcontractors) to test their strategy standalone without the need to run the whole bootstrapped learning framework and supporting simulators in order to quickly evaluate incremental developments on their part. The other additional tiers consisted of refinements between the NIM and end-to-end curriculum tier, which included tiers for evaluating learning of individual Concepts, Multiple-Concepts and sub-trees of a DD's curriculum. The final top level tier, performed stress testing across multiple runs of the same end-to-end DD tests in order to ensure stability.
- **Automated Continuous Integration and Testing** was put in place at the early stages of the project. On each source code commit to the project the full suite of unit and message replay tests were run in order to ensure stability. If defects were identified the parties concerned were notified in a timely manner. Higher levels of the testing tiers were run automatically on a nightly and weekly basis (top level stress tests) due to the time and hardware needed to run these end to end. In addition to receiving monthly framework releases from BAE, SRI in response would provide a stable version of the MABLE student to the framework and curriculum development teams so that they could run their own intermediary evaluations of the Student.

2.4 Technical Results

SRI along with its subcontractors was successful in delivering an electronic Student whose capabilities met each of the project's end of phase evaluation criteria. For Phase 1 this consisted of developing a curriculum using the RoboCup simulator and evaluating a select subset of the

learning strategies to be developed. Phase 2 saw the development of the MABLE student against 3 new diversity domains – Armored Task Force planning (ATF), International Space Station fault handling (ISS) and Unmanned Aerial Vehicle operations (UAV) – where it was capable of passing 97% (208/213) of the tiered tests across these domains before it was submitted for independent evaluation. The passing grade in Phase 2 for the MABLE Student was to be 75% of human performance on a hidden diversity domain. Human performance on this domain was scored at 91% on six final exam problems requiring MABLE to score 68% or higher in order to pass. MABLE surpassed requirements and human performance, scoring a perfect 100% in this evaluation. Phase 3 was similarly completed successfully, and included additional relaxations in the natural instruction constraints between the Teacher and Student components along with the addition of a new development diversity domain.

2.5 Important Findings and Conclusions

The findings and conclusions on the project from SRI's perspective, relate primarily to the overall system architecture and supporting processes for integrating work from distributed research teams:

- **Project Management and Organization** was effective between SRI its subcontractors and the other primary contractors on the project responsible for delivering major portions of the BL Framework (specifically BAE and Cycorp). The main element considered to have contributed to this was the willingness of each group to work collaboratively together on tasks that were not solely within their contractual obligations in order to meet project wide objectives. Shared continuous integration and test practices, regular cyclical releases, and colocated work sessions (e.g. Hackathons) were all seen as useful contributing factors to these collaborative efforts. Between SRI and its subcontractors, responsibilities and the Software Architecture of the MABLE Student was partitioned in a way to minimize dependencies between teams and related coordination costs. However, in cases where close collaboration was required between distributed teams, coordination of efforts was found to be costly. Collocating teams for weekly periods during Hackathons was found to be the most effective way to come to closure on closely dependent work. It is considered an ongoing objective to develop methods and approaches that are more effective with highly dependent distributed teams.
- **Proprietary Languages** if not the primary focus of the project should be avoided when possible. This applies specifically to the Interlingua and its sub-languages that were used on BL, which acted as the common glue between all of the major components in the system. The Interlingua languages had the dual role of providing representation and control support. In particular, a shared representation mechanism was critical to aligning interfaces, in a decoupled manner, between disparate components. However, the control aspects of these languages were less successful due to the expensive and time consuming effort required to develop, debug, and train individuals on their proprietary usage. An alternative to developing these languages from scratch would have been to use existing openly available representation and control languages. This would have leveraged existing skill sets, community support, and avoided significant development costs.
- **COTs Simulators** when available should be used in preference to custom developed ones. The Bootstrapped Learning program developed simulators for each of its evaluation diversity

domains. Developing high fidelity simulators with good software engineering “ilities” is a major undertaking. In BL, reliance on immature or weakly supported simulation code required diverting very substantial resources to work around problems. The use of the comparatively high quality X-Plane flight simulator package later in the program provided a useful counter-example in which the simulator was seen as spurring innovation rather than absorbing resources.

- **Trade Offs Between Evaluation and Research** effort is difficult to balance. End of year evaluations, if properly constructed, provide a mechanism for establishing shared goals, focusing efforts, and measuring progress. However, the effort involved can often detract from more research oriented goals. An example, of where this may have been unbalanced was when evaluating the domain independence criteria of the MABLE student. In Phase 1, two DDs were used, in Phase 2 this increased to 5 plus the Hidden Evaluation DD, and in Phase 3 an additional DD was added. While effective in demonstrating domain independence, each of these took significant effort to develop and test against. One or possibly two less diversity domains could have been used to ensure the same capability and would have allowed resources to be applied to investigating deeper levels of learning capability as opposed to the breadth that was achieved on the project.
- **Standardized Build and Release Management** was introduced at the beginning of Phase 2 of the project by SRI for use with its subcontractors. This required a significant amount of retooling and training across the whole team, which initially was considered to be distracting from the main goals of the project. However, once in place, this infrastructure proved extremely effective in helping to manage and support the projects development. However, establishing this kind of infrastructure as early as possible on a project is preferable.

2.6 Significant Developments

At the beginning of Phase 3, SRI undertook an investigation to explore and develop an approach to interactive communication between the electronic Student and the Teacher/User. This had been highlighted as an important capability that was lacking in the system at the end of Phase 2. The approach taken was different than the standardized active learning methods usually employed by machine learning algorithms to help refine a hypothesis space in conjunction with user input. Instead, a user is facilitated in modifying a target performance system through instructional dialog used to generate a code transformation, a set of point changes to the performance system that together realize user intent. User intent is modeled using a temporal plan network notation based on Constraint-based Attribute Interval Planning (CAIP). Overall, the approach integrates several AI technologies including machine learning, dialog management, AI planning and automated model construction. In addition, it works in unison with MABLE’s learning capabilities, which are best employed to constructively add new concepts to a system while iBL targets the modification of an existing code base.

2.7 Implications for Future Research

The MABLE framework over the course of the project demonstrated what is possible with current state of the art machine learning techniques to communicate knowledge from a human to a computer. The Interactive Bootstrapped Learning (iBL) approach showed how this dialog could be used to modify existing knowledge already contained within a computer. Both

approaches work in unison with each other in a domain independent manner. However, there is still significant progress to be made before the overarching goals set out at the beginning of the program can be fully realized. In particular the learning capabilities of the MABLE system are to a large degree independent of each other. Further work on interleaving instruction methods and more fine grained sharing of hypothesis information between learning strategies could significantly improve the capabilities and usability of the approaches developed. The Interactive centered portion of the research (iBL) provided a good, demonstrable proof of concept of this approach to leveraging user/computer dialog to update a performance system. However, further investigation and development are needed to extend the notation used for representing user intent so it can better express repetition and conditional execution so that a broader range of code transformations can be supported by the system. In addition, creating an extensive library of dialog operators (used to communicate with the user) and possible code transformation would assist in exploring the large application space that these techniques could be applied to.

3. LEARNING CONCEPTS FROM EXAMPLES AND FEEDBACK (UNIVERSITY OF WISCONSIN)

3.1 Task Objectives

A. MABLE

One of Wisconsin's primary objectives was the development of learning services within the MABLE framework. Broadly, at the front end, the goal was to develop and deploy modules that could handle different modes of natural teacher instruction: teacher-provided examples, advice and feedback, as well as use previously learned bootstrapped concepts. At the back end, the goal was to implement existing approaches and develop novel techniques to perform robust and efficient learning of discriminative, functional and procedural concepts arising from a diverse range of domains. Wisconsin aimed to implement learning services based on the existing machine learning methodologies of inductive logic programming (ILP) and knowledge-based support vector machines (KBSVMs). Specific task objectives were:

- Develop learning services in MABLE for the "Learning from Examples" and "Learning from Feedback" natural instruction method (NIM), including learning discriminative predicates, functional calculations and procedures.
- Develop methodologies to effectively and efficiently interpret and represent teacher advice in order to learn concepts from a small number of examples.
- Integrate learning services into MABLE at the system level by handling teacher and intra-student messages in the Interaction Language (ITL) and Interlingua (IL) and background information, as well as providing learned concepts.
- Evaluate successfully on the lessons in the domains during each project phase, as well as the hidden domain benchmarks at the end of each project phase.

B. Research

Another of Wisconsin's main objectives was the development of technically innovative approaches to challenges arising from the implementation of the learning methodologies above when confronted with increasingly complex domains. In addition, the focus was also on developing novel techniques for many open research issues. Significant target challenges for the use of ILP within BL were: the incorporation of teacher advice, automatic generation of background knowledge, and robust parameter selection over diverse domains without the intervention of an expert user. Significant target challenges for KBSVMs were the development of efficient algorithms, and the ability to refine inaccurate teacher advice. Other notable research targets were learning from demonstration via inverse reinforcement learning (IRL), efficient inference in Markov Logic Networks and knowledge capture from non-experts via a human-computer interface.

3.2 Technical Problems

None.

3.3 General Methodology

A. MABLE

A.1. Design and specification. Wisconsin participated in many design discussions resulting in direct contributions to the eventual specification of representations, natural instruction methods such as "learning from feedback", teacher gesture specifications and the teacher's advice language syntax and semantics. In these activities, Wisconsin collaborated extensively with members from the other teams including OSU, SRI International, BAE and Cycorp Inc. Wisconsin assisted in formalizing the eventual specification of syntax for the interaction language aspects of teacher utterances, gestures and relevance types.

A.2. Inductive logic programming and learning from examples. Wisconsin developed learning services in MABLE based on the artificial intelligence paradigm of inductive logic programming (ILP). ILP is especially well-suited as the "learning from examples" component of MABLE for two reasons. First, it can use a rich knowledge base constructed from the initial background, and can be easily augmented with additional background learned during earlier lessons. Second, the declarative representation of both examples and learned rules makes it easier for teacher and student to communicate about what has been learned so far; for example, a teacher can identify and correct student mistakes from earlier lessons.

The Wisconsin Inductive Logic Learner or WILL was the primary learning module implemented by Wisconsin as part of their learning services within MABLE. The WILL system was designed based upon the Wisconsin group's extensive experience with logic programming (especially YAP Prolog) and ILP (especially the Aleph system). At the core of the object-oriented WILL is a general-purpose "search engine" (in the AI sense, e.g., OPEN lists, heuristic functions, etc., as opposed to the Google sense). However, the WILL system also features several significant technical improvements over existing ILP systems so that it can learn concepts effectively within the unique framework of Bootstrapped Learning. The following, which are also general open questions in ILP research, were the three main challenges faced by Wisconsin:

- **Automating the ILP setup task.** One limitation of ILP is its reliance on a large number of user-defined parameters. These parameters control a number of important aspects of the ILP algorithm, the most important of which involves determining the correct search space. A balance must be found such that the search space is large enough, but not too large. If the user specifies parameter settings leading to too small a search space, no hypotheses will exist that perform well on the training data. Conversely, if the user specifies too large a search space, the ILP search may never find an acceptable hypothesis, and may waste too much time exploring areas of the search space without acceptable hypotheses. An added complication in the BL framework is that MABLE is confronted with different types of lessons from diversely different domains, and no intervention from an ILP expert is possible for parameter selection. Thus, the need for automating the ILP setup task. Many existing approaches suffer from at least one if not both of the following drawbacks: they are computationally inefficient, or they require the intervention of an ILP expert; this required Wisconsin to develop new methodologies.
- **Incorporating teacher-provided advice into ILP.** In BL, the teacher provides hints, relevance information about features, labeling of examples and partial concepts. Given that WILL has to learn from a small number of examples, the incorporation of teacher information becomes crucial during learning. It thus becomes imperative that teacher advice is converted to background knowledge that can be utilized by WILL while learning. A second issue is that the advice is usually provided about specific examples and there is the need to generalize it to accurately apply to future examples.
- **Robustness to teacher errors.** One of the key assumptions in BL is the presence of a helpful teacher. However, for general applications, this is an unrealistic assumption. An important challenge is to ensure the learning methodologies used are robust to example-label noise that can arise from teacher's errors of commission, and are robust to advice noise that is likely to arise from teacher's errors of omission.

A.2.1. The ONION. Wisconsin developed a multi-layered approach to automated parameter selection called the ONION, which was implemented within WILL. Much like a grid search (i.e., a systematic search of possible settings for all of an algorithm's parameters), the ONION performs multiple ILP searches with each search using different parameter settings (akin to the layers of an onion). Unlike grid search, the ONION orders the layers, prioritizing layers corresponding to smaller search spaces.

Additionally, if ONION finds a hypothesis satisfying some stopping criteria, it stops searching early. This early stopping both reduces the runtime and provides some resiliency to over-fitting. Initially, the ONION uses stringent early-stopping criteria, slowly relaxing them as the search proceeds. Figure 5 illustrates a broad overview of the Onion algorithm. Further details may be found in Natarajan et al. (S. Natarajan, G. Kunapuli, C. O' Reilly, R. Maclin, T. Walker, D. Page and J. W.

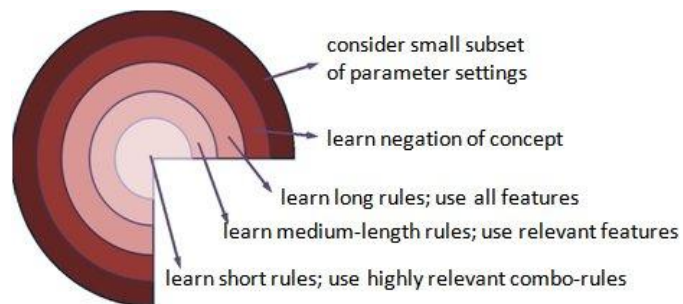


Figure 5. The Onion layers. An iterative search through parameters, starting with small constrained search spaces and iteratively expanding the search space in layers until an acceptable answer is found.

Shavlik. *ILP for Bootstrapped Learning: A Layered Approach to Automating the ILP Setup Problem*. 9th Intl. Conf. Inductive Logic Prog. [ILP], 2009)

In order to select parameters, the ONION iterates over a list of possible parameter-setting combinations, executing WILL with each combination in a predetermined order. As the ONION executes each layer, it evaluates the learned theory (if any) against a tuning set or via cross-validation, stopping when the tuning set performance meets a number of criteria. In this aspect, the ONION resembles an iterative-deepening style search, although unlike the latter, the ONION dramatically changes the search space over time by varying parameters controlling more than just the depth of the search. The ONION orders layers such that it searches small hypothesis spaces initially and gradually expands the hypothesis spaces until it learns an acceptable theory. This ordering of hypothesis spaces attempts to favor simpler hypothesis and provides some resistance to over-fitting. Additionally, the preference for smaller hypothesis spaces improves efficiency of the search over other possible search orders.

Finally, while the ONION is a parameter-tuning approach, it can also exploit *relevance* information about the candidate predicates that WILL uses to construct hypothesis clauses. Wisconsin's handling of teacher relevance, a part of which is exploited by the ONION, is explained in more detail later. In brief, if the teacher is able to provide relevance information or hints to MABLE, this relevance is interpreted by the ONION as the teacher's belief that the relevant predicate will appear in the learned concept. This form of domain knowledge does not require detailed understanding of the underlying ILP algorithm and provides an excellent way for the teacher to provide additional hints to WILL, which the ONION can immediately incorporate as a means to prune the search space.

A.1.2 Handling Teacher-advice in WILL. In BL, teacher advice plays an important role, especially in the NIMs that Wisconsin handles. As the teacher provides only a very small number of training examples, effectively incorporating advice into WILL becomes a crucial task. The BL teacher provides relevance information about *specific examples* including relevant features and predicates, type information, partial explanations of learned theories and partial procedures. Wisconsin incorporate teacher advice into WILL by converting it into background knowledge and adding this into the search space. This raises two important issues: the expansion of the search space and generalization of advice from specific examples to all examples. The methodology adopted by Wisconsin is briefly described below. Additional details can be found in Walker et al., (T. Walker, C. Reilly, G. Kunapuli, S. Natarajan, R. Maclin, D. Page and J. W. Shavlik. *Automating the ILP Setup Task: Converting User Advice about Specific Examples into General Background Knowledge*. 20th Intl. Conf. Inductive Logic Programming [ILP], 2010.). Teacher advice takes the form of logical statements. From this information, Wisconsin construct new background knowledge representing sub-concepts. In addition, Wisconsin attaches priorities based on the relevance information to all of the generated background knowledge for use by their ONION algorithm (described above).

The process proceeds in several phases. The first phase variablizes (i.e., converts constants to variables) the ground advice statements (as they are declared for specific examples). This may be either a direct-mapping that maps all occurrences of the same constant to the same variable and occurrences of distinct constants to distinct variables, or an indirect-mapping, where occurrences of the same constant can be mapped to different variables. Indirect mappings perform what is sometimes called "variable splitting" in ILP, where two occurrences of the same term are

generalized to different variables. It is well-known that variable splitting can lead to an increase in run-time that is exponential in the number of occurrences of the same term within a formula. To prevent this exponential worst-case increase, in practice, Wisconsin limits the maximum number of variable splittings to some small constant k . In the third phase, Wisconsin generate compound logical formulas by connecting the generalizations for different pieces of advice with the AND and OR logical connectives. This includes the generation of *mega rules*, which attempt to capture all of the advice into one logical statement, by conjoining direct-mapping generalizations of all advice from all examples. In the final phase, the generated formulas are converted to implications (as a precursor to creating ILP background knowledge). It is during this phase that a search priority is assigned, with a preference for longer formulas, i.e., those that use as much of the user-provided advice as possible. Mega rules receive the highest priority, followed by per-example formulas, and finally per-piece formulas. The ONION uses these priorities to order the ILP search.

A.2. Knowledge-based support vector machines and learning from examples. KBSVMs extend the classical discriminative function learning approach of support vector machines by incorporating expert advice, thus enabling them to learn effectively from a small number of examples. Advice, or prior knowledge, is provided to the learner in terms of simple, conjunctive, implicative rules about regions in the feature space. The advice specifies labels of examples that lie within the specified advice regions. KBSVMs are ideally suited for function learning from examples due to the compact (sparse) representation of learned rules and their powerful ability to learn non-linear rules through the use of kernels. Some important challenges that were addressed were:

- **Representation of examples and advice.** A significant challenge in using KBSVMs within MABLE is that they use a fixed-length non-relational feature representation, while the BL tasks and domains are all inherently relational in nature. Wisconsin used a propositionalization approach, in which the relational data is flattened into a non-relational feature vector with logical relations being expressed through Boolean features.

Specifically, Wisconsin implemented two kinds of feature-construction methods: in the first method, the relational data is flattened by considering the possible legal combinations of the relational feature values along with an aggregator (such as count, average etc); in the second method, the aggregation takes place directly on the values at the relational level and the propositionalized data is created using these aggregated values.

- **Efficient solution of KBSVMs.** The classical KBSVM formulation results

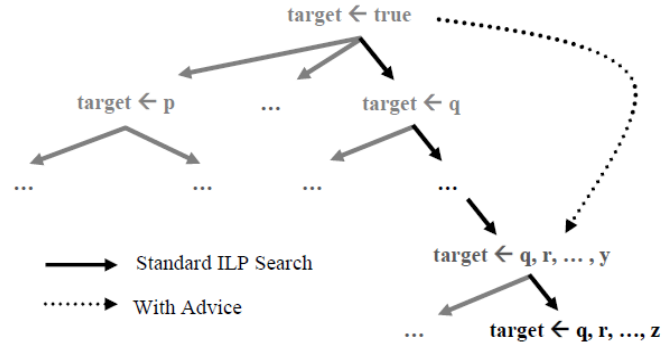


Figure 6. An illustration of a top-down ILP search for an inference rule to predict *target*, whose definition is the conjunction of literals *q* AND *r* AND ... AND *z*. Finding a long clause such as this can be quite hard, but if a teacher gives advice (possibly across multiple examples) that the conjunction of literals *q* through *y* is relevant, then finding the correct definition is much easier; that is, via the dotted line, where only one literal needs to be added to the user advice. It is also more efficient, as a large portion of the search space is directly circumvented.

in a linear program or a quadratic program whose size depends on the number of examples, the number of features and the expert advice. The nature of the propositionalization can drastically affect the size of the optimization problem which can be solved using existing solvers (such as CPLEX). However, such solvers may not be efficient for KBSVMs since they ignore the underlying structure of the problem; this necessitates the development of algorithms that can exploit problem structure to provide gains in efficiency. Also, many such solvers are not freely available, which is a significant bottleneck to their use within this project.

Wisconsin developed and deployed an implementation of their KBSVM formulation within MABLE for use in learning function calculations through examples. The method implements a fast Newton method for solving the linear programs due to Mangasarian and Wild. This approach is highly efficient and provides solutions that are sparse linear combinations of the training data.

A.3. Integrating learning services into MABLE. Several different types of tasks were to be learned from examples, and by feedback: discriminative (a classification task, where the goal is to learn predicates that can answer either yes/no questions, or multi-class classification), functional (a calculation task, where the goal is to learn functions that can compute/estimate target variables) and procedural (learning processes as multi-step action sequences). Identifying the exact nature of the task based on the NIM and concept dimension is critical to learning the correct target concept with the correct underlying learning service. To this end, Wisconsin's unified interface in MABLE, the *WisconsinStrategyHandler* was the front-end in MABLE that processed all the teacher messages and identified the correct learning services.

- **Learning services.** Wisconsin provided two key learning services in MABLE: the *PredicateLearningService* and the *FunctionLearningService*. These services integrate the underlying learning modules: WILL and KBSVMs into MABLE.
- **Representation of examples and advice.** WILL represents information as examples, facts and background knowledge. As the teacher instructs the student naturally through IL, the messages must be represented to the learner accurately. This representation includes type hierarchies and mode specifications, which are used by an ILP engine to control the search for good concepts. While the fact that IL is also declarative helps, the translation from IL to ILP representation is non-trivial because of the need for automatic background generation across different domains and tasks. In addition, the teacher also interacts with the student in many other complex ways such as gestures which are followed by relevance information and advice pertaining to the object being gestured to.

Wisconsin collaborated extensively with Ciaran O'Reilly at SRI International to develop two robust translation services: the *WILLILFOLTranslator*, to translate non-advice teacher messages to background (including modes required for learning, type hierarchies and facts) and the *WisconsinRelevanceInterpreter*, to carefully translate teacher advice, relevance information and hints (given in conjunction with gestures to specific objects) to background for use by the ONION described above. One notable technical accomplishment here was the development of an approach to automatically construct modes from type hierarchies.

- **Providing learned WILL theories to MABLE.** After learning, WILL returns a learned theory which can be evaluated on new, unseen examples. However, this theory is provided the Prolog-like notation used by WILL. Two steps are necessary to ensure that the learned

concepts are available to MABLE, either as background knowledge or as a bootstrapped concept for learning higher-run lessons. Wisconsin provided the *WisconsinTheoryHandler*, an Interlingua wrapper around the concepts produced by WILL and KBSVMs. In addition, Wisconsin also collaborated extensively with Ciaran O'Reilly at SRI International to help develop the *WILL-ILVM*, which is an Interlingua execution mode within MABLE. This execution mode interfaces with the WILL module, creating a small, light-weight execution engine that is capable of answering queries concerning the concepts provided by Wisconsin's learning services.

B. Research

The novel solutions to the various technical challenges arising from the Bootstrapped Learning project also represents research progress made by Wisconsin. In addition, Wisconsin conducted research on several fundamental problems on areas they identified as critically related to the Bootstrapped Learning project and the issues arising from it. These are outlined below.

B.1. Inductive logic programming. Two important aspects of the successfully implemented WILL system, the multi-layered automatic parameter selection approach (the ONION) and incorporating teacher advice as background knowledge, address two longstanding open issues in ILP. The novel approaches described above not only represent significant progress in addressing issues of automating the ILP setup problem, but also enabling the easy use of ILP by non-AI domain experts (such as teachers in BL). In addition, Wisconsin also undertook comprehensively testing their ONION on Phase II and Phase III lessons, particularly the UAV and the ATF domains to demonstrate that this generalized approach allows ILP learning to be domain- and task-agnostic. Many other advances and details may be found in thesis of one of Wisconsin's personnel, Trevor Walker (T. Walker. *Broadening the Applicability of Relational Learning*. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, 2011. Also appears as UW Technical Report CS-TR-11-1698).

B.2. Knowledge-based support vector machines. Two important outstanding issues in KBSVMs are the continual quest for more efficient algorithms and the need for refinement of inaccurate teacher advice. These were an important focus for Wisconsin's research during the tenure of the BL project.

- **Online KBSVMs.** Wisconsin formulated and implemented a novel approach for incorporating prior knowledge into the SVM classification problem in an online setting. In an online setting, the learner receives examples one at time incrementally, rather than all at once (batch learning). Advice helps speed up and bias learning so that generalization can be obtained with less data. Wisconsin's online KBSVMs are based on the passive-aggressive algorithm (PAA) framework. In this framework, updates are made to the current hypothesis only when it suffers a loss. This loss is measured using a hybrid loss that takes into account the margins of both the hypothesis and the advice on the current point. In addition to tight theoretical loss bounds, Wisconsin validated this approach with computational results on different data sets, notably on a Tuberculosis data set. Further details can be found in Kunapuli et al., (G. Kunapuli, K. P. Bennett, A. Shabbeer, R. Maclin and J. W. Shavlik. *Online Knowledge-Based Support Vector Machines*. 23rd European Conf. Mach. Learn. [ECML], 2010).

- **Advice Refinement in KBSVMs.** An essential component of robust learning with KBSVMs is their ability to learn well even in domains where the expert advice is inaccurate. In their original formulation, the success of KBSVMs is strongly dependent on "good" expert advice or prior knowledge. In several real-world applications, this is a highly idealized and unreasonable expectation. Wisconsin's approach to advice refinement involves introducing "error terms" into the polyhedral constraints that measure how well the advice fits the data. This is in addition to existing "slack variables" that measure how well the data fits the advice. This results in a bilinear program, which is non-convex. Wisconsin solve their proposed model using two iterative approaches: successive linear programming and a constrained concave-convex approach. Experimental results demonstrate that these algorithms yield useful refinements to expert advice, as well as improve the performance of the learning algorithm overall. Wisconsin calls this novel approach Advice Refining Knowledge-based SVMs or ARK-SVMs (see G. Kunapuli, R. Maclin and J. W. Shavlik. *Advice Refinement in Knowledge-Based Support Vector Machines*. 25th Conf. Neural Inform. Proc. Sys. [NIPS], 2011).

B.3. Transfer learning. When many machine learning domains contain several related tasks, instead of learning each one from scratch, agents in such domains should be able to use knowledge learned in previous tasks to speed up learning in later ones; this is the goal of transfer learning. Wisconsin's approach to transfer learning involves the use of *rule extraction*, which is done for the purposes of human interpretation of machine learning models. However, when used for transfer learning, the extracted rules can be considered advice on how to approach the second task. Wisconsin developed two approaches: a policy-transfer-based approach and a skill-transfer-based approach to transfer rules between SVM-based reinforcement learning tasks. With an SVM-based reinforcement learner, this advice is then incorporated as a soft constraint, allowing the rules to improve learning in the target task to the extent that they are relevant, but also providing protection against negative transfer effects. For details, see Torrey et al., (L. Torrey, J. Shavlik, T. Walker and R. Maclin. *Transfer Learning via Advice Taking*. In J. Koronacki, S. Wierzchon, Z. Ras & J. Kacprzyk, eds, *Recent Advances in Machine Learning*, Springer, 2009).

B.4. Inverse reinforcement learning. Learning the reward function of an agent by observing its behavior is termed inverse reinforcement learning and has applications in learning from demonstration or apprenticeship learning. Wisconsin studied various extensions of inverse reinforcement learning problem. This included the multi-agent IRL, where reward functions of multiple agents are learned by observing their uncoordinated behavior. A centralized controller then learns to coordinate their behavior by optimizing a weighted sum of reward functions of all the agents. They evaluate their approach on a traffic-routing domain, in which a controller coordinates actions of multiple traffic signals to regulate traffic density. For details, see Natarajan et al., (S. Natarajan, G. Kunapuli, K. Judah, P. Tadepalli, K. Kersting and J. W. Shavlik. *Multi-Agent Inverse Reinforcement Learning*. 9th Intl. Conf. Mach. Learn. & Apps. [ICMLA], 2010).

B.5. Inference in Markov logic networks. Statistical Relational Learning is a relatively recent development within the field of artificial intelligence which combines the first-order logic with the knowledge representation formalisms of probabilistic and graphical models to model relational domains. Markov Logic Networks (MLNs) are an SRL approach which applies the undirected graphical models arising from Markov networks to first-order logic. This enables the learned model to perform inference in the presence of uncertainty. Wisconsin studied the

important issue of inference in MLNs, which requires the counting of the number of ground logical statements in the model, which can become impractical for large models. Wisconsin proposed a novel pre-processing scheme to substantially reduce the effective size of MLNs by counting how often the evidence satisfies each formula, irrespective of the truth values of the queries. Wisconsin showed empirically that the gains provided by such a preprocessing step can be significant, sometimes even allowing for exact inference in some domains. For more details, see Shavlik and Natarajan (J. Shavlik and S. Natarajan. *Speeding Up Inference in Markov Logic Networks by Preprocessing to Reduce the Size of the Resulting Grounded Network*. 21st Intl. Joint Conf. AI [IJCAI], 2009).

B.6. Integrating knowledge capture and supervised learning. A fundamental issue that is coupled with the automating many aspects of an advice-taking ILP is the design of a system that facilitates the use of such a system by non-AI domain experts. Wisconsin developed a paradigm based on an interactive human computer interface (HCI) that aids in automatically obtaining relational knowledge, and then integrating it with an underlying ILP system.

Central to this approach is Wisconsin's view that advice acquisition is an iterative process (see Figure 7) of a user specifying advice, an algorithm learning a model, a user reviewing results, and a user refining or augmenting the advice. The iterative learning process proceeds in four stages. In the first stage, the user is presented with an HCI through which advice can be specified. Typically, the HCI displays information about a single training example and asking the user why that example was positive (or negative). Through the HCI, the user can express this knowledge in ground logical format without the need to know the knowledge representation. In the second phase, the advice is translated into a form usable by the learning algorithm. This usually entails generalizing the advice and possibly changing the representation of the advice. Phase three is the actual learning phase, at which point, the learning algorithm is provided with both the training examples and the generalized advice. The learning algorithm then produces a model. Finally, in phase four, the model can be evaluated against additional examples. The crucial iterative aspect of Wisconsin's approach involves presenting the user with a second HCI allowing the user to review the effectiveness of the model. The reviewing HCI presents the user information about which 'testset' examples were correctly or incorrectly predicted by the model. Based on this, the user may elect to return to phase one in order to adjust previously presented advice, provide new advice, or label new examples.

Wisconsin developed and deployed a graphical user interface for the HCI for tasks in the Wargus domain. Wargus is a real-time strategy game in which two or more players direct groups of different types of units such as peasants, swordsmen, archers, etc., in an attempt to conquer the

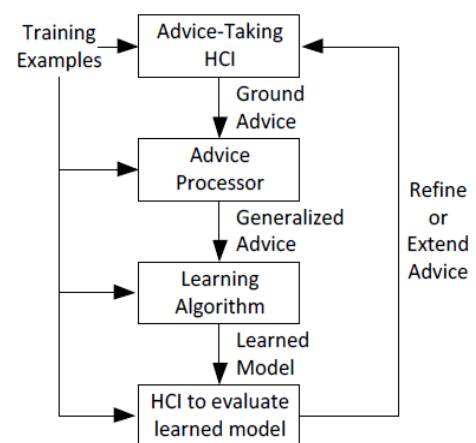


Figure 7. Wisconsin's HCI paradigm. Initially the user specifies advice through an HCI. Then the advice is processed and learning occurs. Afterward the results are presented to the user via an evaluation HCI. The process iterates until the user is satisfied with the results.

opposing players. Design of the HCI depends greatly upon the type of the knowledge being gathered. Figure 8 depicts a simple prototypical GUI designed for the Wargus Tower-Defense task in which an attacking team assaults a single tower belonging to the defenders. The learning task consists of predicting whether the tower will survive the attack given the size and composition of the attackers. The user provides advice through the HCI about specific examples. The HCI itself allows for the selection and naming of entities or groups of entities, indicating relations among selected entities and review, and possibly edit, previously specified advice.

The system uses WILL for executing the second phase in which it generalizes advice over specific examples. The third phase, i.e., learning with advice is done through boosted relational dependency network (bRDNs, developed with the support of DARPA’s *Machine Reading* project). The bRDN algorithm combines relational dependency networks (which approximate joint distributions as products of conditional distributions) with a form of gradient-tree boosting. After learning, the system provides the learned models to the user for review. See Walker et al. (T. Walker, G. Kunapuli, N. Larsen, D. Page & J. Shavlik, *Integrating Knowledge Capture and Supervised Learning through a Human-Computer Interface*, 6th Intl. Conf. on Knowledge Capture [KCAP], 2011) and Walker’s PhD thesis (cited above) for additional details.

3.4 Technical Results

Wisconsin successfully passed the internal benchmarks at all three phases of the BL project, as well as the hidden domains at the end of each phase.

- In Phase I, Wisconsin successfully passed the lessons in the BlocksWorld and the Robocup domains, as well as on a Phase I hidden domain .
- In Phase II, Wisconsin successfully passed all the lessons in the Armored Task Force (ATF), International Space Station (ISS) and Unmanned Aerial Vehicle (UAV) domains, as well as a Phase II hidden domain/
- In addition, the ATF, ISS and UAV lessons were used in extensive testing of the WILL technologies developed to handle the challenges of complex domains without user intervention. This comprehensive evaluation also involved testing with noisy data and advice (deliberately introduced into the lessons to simulate teacher errors of omission and commission). Analysis of the results showed that WILL, and its implemented technologies are robust to a high degree to example and advice errors. When advice literals were dropped randomly (simulating teacher errors of omission), the test set accuracy over all the lessons

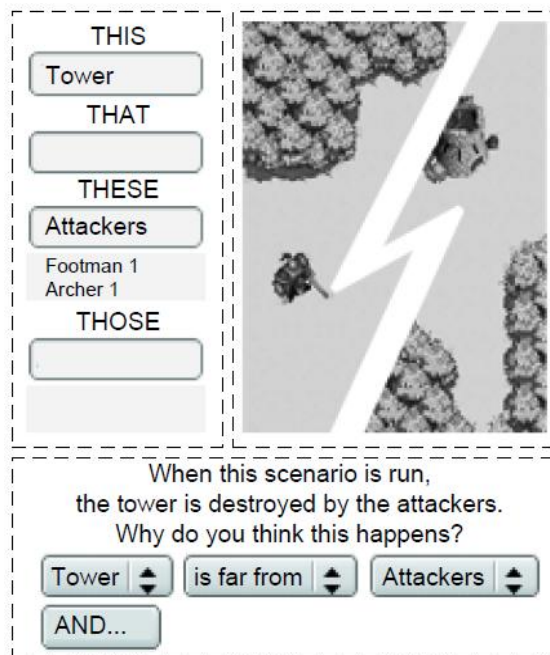


Figure 8. Prototype GUI for Wargus. It consists of 4 sections: (upper-left) entity selection and naming, (upper-right) current game board, (lower) specifying relations between selected entities, and (not shown) a list of previously specified advice.

remained around 80% on average (with a 75% chance of dropping the literal i.e., probability that the teacher drops a literal is 0.75). When learning with noisy examples, with a noise rate of 20% (i.e., 20% of the training examples were mislabeled), the average test set accuracy over all the lessons was around 80% without advice and 98% with advice. For more details, see the technical report of Walker's PhD thesis cited above.

- In Phase III, Wisconsin successfully passed all the relaxation trajectories introduced into the ATF, ISS and UAV domains.
- At the end of Phase III, Wisconsin also collaborated closely with BAE Systems in successfully learning theories for the Intelligent Reconnaissance and Surveillance (ISR) domain to demonstrate that instructable computing and learning technologies can benefit ISR analysis problems. Wisconsin also collaborated with SRI's investigation of an ambitious case-study of the application of BL to UAVs.

3.5 Important Findings and Conclusions

Teacher advice is beneficial when learning from examples. Learning from examples is a standard and well-studied paradigm in machine learning and artificial intelligence. Most existing systems are based on statistical assumptions on the data and require large numbers of both positive and negative examples. In many practical applications, it is more common to have a teacher who is capable of providing a small amount of advice regarding the domain. The incorporation of such advice is crucial, and can help learn concepts and crucially so in domains where examples are hard or expensive to come by. Advice can be incorporated into a diverse variety of learning algorithms, and is almost always beneficial as it speeds up learning, requires fewer examples and provides concepts that generalize as well, if not better than learning with no advice.

Non-AI domain experts can make effective teachers. The automation of the ILP setup task and incorporation of advice as background knowledge into ILP are two significant advances in the field. This, combined with Wisconsin's approach to integrating knowledge capture through a human-computer interface, effectively encapsulates the AI aspects of ILP from the user. Also encapsulated are the underlying representation of the learning problem, the learning algorithm itself, and its various parameter settings. Extensive success on diverse domains in which WILL was instructed by teachers through natural instruction methods, and via humans via the HCI suggests that it is possible to deploy effective human-instructable machine-students, where they are instructed by domain experts who do not have to be computer-science experts.

3.6 Significant Developments

The development of the Wisconsin Inductive Logic Learner (WILL) represents a significant advance as it addresses several open questions in logical reasoning systems, specifically inductive logic programming (ILP). Among its most significant features are

- ease of use by non-AI experts, owing to the automation of many ILP setup tasks that previously required intervention by an ILP expert;
- ability to incorporate different kinds of teacher advice, relevance information and hints as background knowledge, and consequently learn target concepts with significantly smaller number of examples than needed by a general example-based learning system; and

- robustness to teacher errors of omission and commission, which ensure that the system will attempt to learn something useful even when confronted with noisy data or inaccurate advice.

3.7 Implications for Future Research

Expert advice from domain experts can help machine learning algorithms immensely; the Wisconsin work in the BL project has provided further evidence and additional case studies in support of this. A significant bottleneck for such experts in providing effective advice is the necessity for such people to also have a solid understanding of the computer's representation of its background knowledge and the operation of the learning algorithms. A major implication of Wisconsin's research on this front is that human-computer interfaces (HCI) and automatic background-knowledge generation systems can aid non-AI domain experts in providing advice and knowledge to machine learners. The needs are two-fold: first, the understanding and development of effective HCI (or natural-language understanding) systems that can capture advice from domain experts, and second, advanced computer systems capable of providing expert advice to the underlying learner in its native representation without loss of (much) information. The biggest short-term challenge here is designing an intuitive, easy-to-use graphical-user-interface-based HCI that is versatile enough to allow experts to provide advice without many restrictions while abstracting the details of the knowledge representation. The longer challenge is to develop computer algorithms that can translate from ordinary English (or any naturally occurring language) into the formal language(s) used by advice-taking machine learning algorithms; it is particularly appealing to imagine using BL to teach such an English-to-advice translation system.

4. LEARNING PROCEDURES FROM EXAMPLES AND FEEDBACK (OREGON STATE UNIVERSITY)

4.1 Task Objectives

The objectives pursued by the Oregon State University (OSU) team were unified by the theme of learning sequential-decision-making tasks via interaction with a teacher. The OSU team focused on the following five objectives during their two year effort. The objectives will be referred to by their bold headings below throughout the report.

- **RL+Critiques.** Learning closed-loop policies via reinforcement learning (RL) and situated critiques from end-users. This objective was intended to capture the type of student-teacher relationship that occurs when a student is allowed periods of autonomous practice, followed by periods of critiques by the teacher.
- **Active Learning.** Learning closed-loop policies via actively querying a teacher. This objective was intended to study the utility of a student actively querying a teacher during learning of complex sequential tasks.
- **Object-Based Queries.** Theoretical analysis of the benefits of learning first-order structures via natural object-based queries to a teacher. This objective was intended to leverage the fact that in complex first-order worlds, students and teachers can directly “talk about” objects.

- **MABLE Component.** Learning procedures for open-loop policies via examples and feedback for the common MABLE system. This objective was the main contribution of OSU to the MABLE system and handled all learning of sequential-decision procedures.
- **Stratagus HIL.** Developing a human-in-the-loop (HIL) interface for teacher-student interaction in the real-time strategy game engine Stratagus. This objective was intended to drive further research into bootstrapped learning by providing a natural HIL interface in a complex reconfigurable environment.

As discussed further in this report, significant progress was made on all of these objectives and in each case important research directions have been identified.

4.2 Technical Problems

Each of the bootstrapped learning objectives presented a number of challenging and unique technical problems. Below the most significant problems faced for each objective are described.

RL+Critiques.

This objective was formalized in the framework of Markov Decision Processes (MDPs), where MDPs were used to model specific sequential decision tasks. Specifically, it was assumed that the student had the goal of learning a closed-loop policy for maximizing reward in the MDP, which modeled the task of interest to the teacher. In traditional RL, the learning agent, or student, must learn exclusively from interaction with the MDP which provides information about changes in the environmental state and rewards. In complex environments, pure RL of this form can degrade to essentially random exploration and learning is impractically slow.

The RL+Critiques framework allows a teacher to periodically observe the current learned behavior of the student and to provide critiques about the behavior. In the current work, the critiques took a simple form of identifying states on the student's behavior trajectory and indicating whether certain actions in those states were good or bad. For example, if the student makes a clear mistake in a state, the teacher could indicate that the action taken there was incorrect and if desired could also suggest a better action.

After interacting with both the environment and teacher, the student is left with trajectory data, as used by traditional RL, and also a set of critique data. *The fundamental technical problem is then to combine these different modalities of data in order to select a good policy.* Ideally, the selected policy should be better than policies selected based on only one modality, particularly when the data sets are small. One aspect of this problem that was particularly challenging was that no assumption was made about the completeness or quality of the critique data other than that the teacher is generally helpful (but may make mistakes). This meant that the student must be robust to incomplete critiques and noise in the critique data.

Active Learning.

Again, this objective was modeled in the framework of MDPs. In this setting it is assumed that the teacher has a target policy that they would like to teach the student. A traditional way to teach such a policy is via imitation learning, where the teacher demonstrates entire trajectories of the target policy to the student. The student then passively consumes those trajectories and attempts to learn a policy that matches them, while trying to ensure good generalization.

A major problem with traditional imitation learning is that generating a set of complete trajectories of a policy can be extremely time consuming for a teacher and in some cases even impractical (e.g. in a real-time, multi-agent decision making scenario). This problem is the motivation behind OSU's active learning objective. The goal is to reduce the teaching effort by allowing the student to ask queries about what to do in particular states of the environment. Ideally, the effort required to answer a sufficient number of the actively selected queries will be much less than the effort required to generate a sufficient number of complete trajectories.

The key technical problem in this active imitation learning framework is how to select the state query that will most aid the student in learning the target policy. A major challenge compared to i.i.d. active learning is that the agent does not know the target distribution of states, or in other words, does not know which states are potentially relevant to the target policy. Asking queries in states that the target policy never, or rarely, visits will not result in useful learning, even if there is high uncertainty about what to do in those states. Thus, the challenge is to select query states where there is both uncertainty about what to do and that are likely to be relevant to the target policy. Identifying and balancing these two factors was a central contribution of OSU's work.

Object-Based Queries.

This objective was motivated by work on the MABLE system where several components involved learning procedures expressed in first-order representations. The use of first-order representations allows for student and teacher dialogs to reference specific objects in training examples. In particular, the student can ask about the relevance of certain objects in training examples and the correspondence between objects between examples. *A fundamental technical problem raised by this possibility is to develop a theoretical model of learning with object-based queries and understanding any fundamental theoretical advantages of using such queries.* Before OSU's work no such model existed and no learning algorithms existed that could take advantage of such object-based queries.

MABLE Component.

This objective involved creating a component for the MABLE system that handled all NIMs aimed at teaching procedural knowledge to the student. In particular, procedural knowledge was encoded in a representation called IL-lists, which are conditional lists of parameterized Interlingua code fragments. *The fundamental technical challenge was to deal with the inherent complexity of learning procedures in such a rich language from a small number of examples and/or feedback from the teacher.* In the most general case, this challenge is insurmountable as guaranteed by negative results from theoretical machine learning. The key idea behind OSU's approach was to recognize that teachers will generally be helpful and have certain teaching biases that can and must be exploited for efficient learning.

Stratagus HIL

This objective was motivated by the need to have a rich interface for student-teacher interaction in a complex dynamic environment. The open-source Stratagus engine is an engine for real-time strategy games, which, for example, can be used to play Wargus, a clone of the popular game Warcraft II. Such games are extremely complex and there is significant opportunity for rich student-teacher communications. *The key technical problems raised by this objective were to*

design the different types of interaction that would likely be useful and to overcome the difficult engineering challenges in implementing the interface options.

4.3 General Methodology

OSU's general methodology was to formulate each objective in a mathematically rigorous way and to then derive principled algorithms. At the highest level the objectives fell into one of two areas: 1) Learning in the context of different modalities of information acquired from a teacher and possibly autonomous exploration, and 2) Actively acquiring instruction from a teacher. The first area covered the RL+Critique and MABLE-Component objectives, while the second area covered the Active Learning and Object-Based Queries objectives. The Stratagus HIL objective spanned both areas.

The general methodology for the first area was to combine the information available to the learning algorithm into a single optimization problem. The solution to the optimization problem was used as the output of learning. In the RL+Critique case, the target output was the parameters of a policy, which led to a numeric optimization problem. In the MABLE Component case, the desired output corresponded to a combinatorial structure which did not fit nicely into a numeric optimization framework. Rather, we formulated a nested sequence of discrete optimization problems, where the earlier problems in the sequence were more constrained than later ones. The constraints corresponded to assumptions about good teaching, which meant that earlier problems made stronger assumptions and those assumptions were dropped in later spaces. The optimization objective was then to return the solution for the earliest problem whose constraints could be satisfied. The idea was that in cases where many assumptions about good teaching were satisfied, learning would be efficient in terms of the number of examples since the effective hypothesis space would be smaller.

The general methodology for the second area of active instruction acquisition was to consider the information content of the possible teacher queries. In the case of Active Learning, a Bayesian framework over a parametric solution space was considered. An entropy-based measure of information was then used to select a query for maximally helping to reduce uncertainty about the target policy. In the case of Object-Based Queries, the framework was exact learnability of a first-order structure. Queries were selected by maintaining a representation of the uncertainty over these objects and asking queries that helped resolve the ambiguity. Details regarding each of these techniques are provided in the next section.

4.4 Technical Results

Below the technical approaches and results developed by OSU for the 5 objectives are reviewed.

RL+Critiques

The details of OSU's approach and methodology are in the following paper:

Kshitij Judah, Saikat Roy, Alan Fern, and Thomas Dietterich. (2010). Reinforcement Learning via Practice and Critique Advice. *AAAI Conference on Artificial Intelligence*.

OSU focused on learning parametric closed-loop policies. In particular, their work used log-linear policy representations, which given a set of parameters and a state produced a distribution

over actions. The goal was to adjust the policy parameters in order to maximize expected total future reward. The learning loop alternated between two stages: 1) A practice stage, where the learner conducted autonomous practice in the environment for some period of time, and 2) The critique stage, where the teacher was allowed to observe trajectories of the student’s behavior and then provide critiques on those trajectories. The first stage produced a set of trajectory data T and the second state produced a set of critique data C . The key problem was to select policy parameters based on the information in T and C that resulted in maximizing the expected reward.

The high-level approach to this problem was to define the following objective function that measures the goodness of the parameters θ with respect to the sets T and C :

$$J_{\beta}(\theta, C, T) = \beta \cdot U(\theta, T) - L(\theta, C)$$

Here, U is an estimate of the average reward of the policy with parameters θ based on the trajectories. In particular, a standard importance weighting estimate was used for this purpose. The component L is a loss function that measures the amount of disagreement of the policy with the critique data. One of the key contributions of the work was to define this loss function so that it takes an error model of the teacher into account in a principled way. The weighting factor β trades off the relative contribution of the trajectory and critique data.

Given the above objective function, the parameters were optimized using standard quasi-Newton gradient-based search with random restarts. The value of β was selected based on validation, where a search over β values was conducted to find the value that produced the policy that achieved the largest average reward in the actual MDP. This approach to setting β allowed for an automatic way to trade-off between the relative influence of the practice and critique data depending on their quantity and quality.

We conducted experiments using human subjects on a complex learning task involving tactical battles in a real-time strategy game. In particular, the goal was for the system to learn a policy for controlling a set of 5 friendly units in a battle against 5 enemy units. A user interface was constructed for the human teachers the human teachers each trained the systems two times. In one trial, the system was trained using only critiques from the human with no practice. In the other trial, the system was trained using both critiques and practice. Note that the system was unsuccessful at learning from pure autonomous practice, which indicates the complexity of the problem being considered.

Using all of the data collected from the user study we first ran synthetic experiments in order to produce averaged learning curves (averaging across multiple runs of the learner) that compare the impact of different amounts of practice and critique data. Such averaging was not possible using actual human subjects since repeating learning trials both biases the human and would require far too much time from the subjects. The results, which are detailed in the paper, show two clear trends: 1) Given a fixed amount of critique data, performance improves as the number of autonomous practice episodes grows, 2) Given a fixed amount of practice, performance improves as the amount of critique data grows. Overall there is a synergistic effect between the critique data and autonomous practice so that the sum is better than the parts.

We also conducted a more qualitative analysis of the actual learning results from the user study. Overall, the subjects were able to train both systems (one with only critiques and with critiques+practice) to perform much better than pure RL applied for the same amount of times. In

fact, pure RL was allowed a much longer amount of time than the user study time limit and still was far worse. This clearly demonstrated the utility of allowing an end-user in the learning loop. However, unexpectedly, there did not seem to be a significant difference between the learning results of the two systems, though there was a small edge for the system that used only critiques. That is, most users were just as effective at training the system with only critiques, compared to using critiques and autonomous practice. Further, some users were slightly more successful using only critiques.

In order to help understand this unexpected result, the OSU considered the questionnaire feedback provided by each participant. There appeared to be a combination of reasons for the result. Some users found the pure supervised system to be more user friendly, since there was no delay experienced while waiting for the practice stages to end. Because of this, the users were able to see the effects of their critiques almost immediately, and as a result they gave more and possibly better advice. For the majority of the users, the amount of advice given to the pure supervised system was nearly twice that given to the combined system. Note that in a real scenario the end-user would not need to wait around for the practice period to end, but could periodically check in on the system.

One could argue that the practice episodes of the combined system should compensate for the reduced amount of critique data. However, it was observed that, for some users, the policies returned after certain practice sessions were quite poor and even appeared to ignore the user's previous critiques. The users found this quite frustrating, and it likely impacted the quality of their critiques. We believe that such occurrences can be partly attributed to the fact that the amount of practice used in the user study between critique stages was not sufficient to overcome the reduction in the amount and perhaps quality of critique data. Indeed our simulated experiments showed that, with a reasonable amount of practice, there were significant performance gains over no practice at all. An important future investigation is to design a user study where the user is not captive during the practice sessions, but rather checks in periodically, which will better reflect how the combined system might be used in reality. Nevertheless, in general, it is likely that with our current combined system, users will experience occasional decreases in performance after practice and perceive that the system has not taken their critiques into account. A lesson from this study is that such behavior can be detrimental to the user experience and the overall performance achieved.

Active Learning

The details of OSU's approach and methodology for active learning of closed-loop policies are in the following paper:

Kshitij Judah, Alan Fern, and Tom Dietterich. (2011). Active Imitation Learning with State Queries. *ICML 2011 Workshop on Combining Learning Strategies to Reduce Label Cost*.

An extended version of the paper is also under review for the AAMAS-2012 conference.

The focus of the work was on learning a closed-loop policy represented as a classifier given the ability to interact with a dynamic simulator of the MDP and the ability to ask queries to a teacher. Each query presented a state of the environment to the teacher. In response, the teacher could either provide the learner with the correct action for that state according to the target policy, or return "bad state" to the learner. The later, "bad state" response, indicates that the

queried state is not one that the target policy would ever visit. For example, a “bad state” in a helicopter flying task would be one where the helicopter enters an unrecoverable fall.

It is important for the learner to attempt to minimize the number of queries in such bad states. First, a teacher is likely to get quite frustrated with a student that frequently asks such queries. Second, even if the teacher does provide an action in such states, learning in those states is not useful. A key observation of the OSU study is that a naïve application of traditional i.i.d. active learning algorithms is prone to asking many bad queries. This is due to the fact that most such active learning techniques are based on the principle of asking queries about examples (in our case states) where the uncertainty of the current classifier (in our case policy) is maximized. Unfortunately, in the sequential decision making setting, bad states are often those where uncertainty about the best action is high. This causes pure uncertainty based active learning algorithms to perform poorly in our setting, which is demonstrated in the experimental evaluation.

A novel approach was developed by the OSU team that accounts for both action uncertainty in a state and likelihood of the state being relevant to the target policy. The key idea was to formulate the problem in the framework of Bayesian active learning, which has well established techniques, some with performance guarantees. In particular, we used the generalized binary search approach which results in querying a state that has the maximum entropy outcome distribution, where the possible outcomes are either an action or the “bad state” response. Interestingly, it was possible to derive an expression for this selection rule that has an intuitively meaningful interpretation. In particular, the resulting algorithm queries the state s that maximizes the following expression $Q(s)$:

$$Q(s) = \alpha(s)H(s) + \alpha(s)$$

where $\alpha(s)$ is the posterior probability that the target policy can possibly reach state s (i.e. probability that it is not a bad state) and $H(s)$ is the entropy of the action selection distribution across policies that can possibly reach state s . The heuristic is dominated by the leading product term indicating that $Q(s)$ will be small if either a state is unlikely to be visited by the target policy or the action uncertainty at that state is small (i.e. a small entropy). This agrees with intuition and shows that a heuristic based on a product of these two components is formally justified within the Bayesian active learning framework.

The above approach to active learning was formulated for an idealized setting where it was assumed that the number of states and possible policies were relatively small. This led the OSU team to design a sample-based approximation of the exact algorithm called Imitation Query-by-Committee (IQCB), which approximates the posterior over policies via a sample, or committee, of policies. Further, the algorithm only considers querying states that are on trajectories generated by those policies. The resulting algorithm is scalable to large problems and was shown to be effective in experiments. Interestingly IQCB is a strict generalization of the traditional Query-by-Committee algorithm for i.i.d. learning. In particular, the i.i.d. active learning setting corresponds to the case of a degenerate MDP, where actions do not influence the next state (next states are drawn i.i.d.) and rewards indicate whether the target action was selected in a state. For such an MDP, IQCB can easily be shown to be equivalent to traditional uncertainty-based QBC.

The OSU team conducted experiments in five domains: 1) A grid-world with traps, where the agent is attempting to reach a goal location but can fall into traps if errors are made, 2) The

standard cart-pole RL benchmark, where the agent must learn to balance a pole by moving a cart that it is attached to, 3) The standard bicycle, RL benchmark, where the agent must learn to balance a bicycle, 4) A tactical combat problem in the RTS game Wargus, and 5) The Nettealk phoneme labeling task. In all but the last domain, it is possible for the learner to enter states that are extremely bad in the sense that they correspond to catastrophic failures or are exceedingly costly to recover from. For example, it is not possible to recover from a fall on the bike. These are exactly the types of states that the learner should avoid querying about.

The OSU team implemented a number of competing techniques, including standard query-by-committee, which queries states based on only uncertainty about the action choice. Also included was the confidence-based autonomy (CBA) approach, which is the only prior active policy learning approach that can scale to the problems considered in the experiments.

The experiments, which are detailed in the paper, show conclusively that IQBC is the most robust active learner across the domains. The results show that for the majority of the domains the competing approaches learn extremely slowly or fail to learn by either asking too many bad queries, or becoming overly confident in the case of CBA and deciding not to ask further queries.

Object-Based Queries

The details of OSU’s approach and methodology for learning first-order representations via object-based queries are in the following paper:

Joseph Selman and Alan Fern. (2011). Learning First-Order Definite Theories via Object-Based Queries. *European Conference on Machine Learning (ECML-2011)*.

The first challenge was to formulate a theoretical framework that allowed for a formal analysis of the potential utility of object-based queries. For this purpose, the OSU team chose to study learning first-order definite Horn theories in an exact learning framework. In particular, the exact learning framework is concerned with bounding the number of queries required by a student in order to learn a target theory. In prior work, two types of queries had been considered for learning first-order definite theories. First, equivalence queries ask the teacher if the current hypothesis is correct and if not the teacher provides a counter example. Second, membership queries present the teacher with a training example (in our case ground Horn clauses) and ask whether it is a positive or negative example (in our cases, whether the example is subsumed by the target theory or not). Prior work has demonstrated that only a polynomial number of such queries are required for learning first-order definite theories given a bound on the number of variables involved in clauses of the target theory and a bound on the predicate arity.

One practical issue with the prior results is that membership queries are well known to be problematic for teachers. In particular, the examples presented to the teacher are often meaningless at a semantic level since they are not generated based on a model of legal or reasonable inputs. This motivated the goal of attempting to reduce the number of required membership queries via the use of object-based queries about examples provided by the teacher. Intuitively, the teacher already has a good understanding of the training examples that they construct for the system, including the roles of objects in those examples. Thus, asking queries about the objects in those examples will arguably be an easier task for the teacher than answering an arbitrary membership query.

The first type of query considered in OSU's work was the object relevance query. Given an existing example, an object relevance query simply presents an object in the example to the teacher and asks whether the object is relevant to the classification of the example or not. The notion of relevance is formalized in the paper, but intuitively an object is relevant if removing it or changing some of its properties or relations would impact the classification of the example. The first theoretical result concerning object relevance queries showed that the number of membership queries could be dramatically reduced, but not eliminated, via a comparatively small number of object based queries. The proof was constructive given a learning algorithm that achieved the bounds. The second theoretical results showed that if answers to equivalence queries were negatively biased (preferring to provide a negative counter-example if one exists) then membership queries could be eliminated completely given the additional object relevance queries. This result is of practical relevance since in practice it is often possible to provide such a negative bias. In particular, given a large set of unlabeled training data that is known to be highly biased toward negative examples, we can use those examples to approximately answer equivalence queries in a negatively-based way by finding an example that is covered by the current hypothesis.

The analysis was also extended to the case where queries are answered inaccurately. In particular, the cases of conservative and verbose teachers were considered. A conservative teacher is always correct when they say that an object is relevant, but may say that some relevant objects are irrelevant. A verbose teacher is one that is correct when they say an object is irrelevant, but may indicate that some irrelevant objects are relevant. Theorems were provided for both types of teachers, which bounded the number of queries required in terms of the degree to which the teacher is either verbose or conservative. The results showed that clearly, when membership queries are available, a conservative oracle is superior to a verbose oracle with respect to the number of membership queries. This was because it is much easier to identify an example missing relevant objects than an example with extra objects. This result has implications for domains where heuristics are used to judge the relevance of objects that are not available via an interface for a human teacher to judge relevance. In particular, it suggests that when membership queries are available using a conservative heuristic is desirable.

Finally, an additional type of object-based query was considered, the pairing query. A pairing query presents two positive examples to the teacher and asks the teacher to indicate which objects in those examples correspond to one another. The notion of correspondence was made precise, but intuitively two objects correspond if they play the same role in identifying the examples as positive. The main theoretical result was that with the availability of pairing queries, exact learning is possible without membership or relevant object queries. Thus, first-order definite Horn theories are exactly learnable with only pairing and equivalence queries. The question of learning in the presence of noisy pairing queries, either verbose or conservative, is still an open question.

MABLE Component

OSU developed learning components for NIMs involving learning sequential procedures from examples and feedback.

In the case of learning from examples, the input came in the form of a sequence of Interlingua messages from the teacher, which formed a lesson for a particular procedure. The output of the

learner was an Interlingua procedure representing the learner's best guess about the target procedure given the current set of lessons.

In the case of learning by feedback, the learner was first asked to execute its current procedure in a particular situation. The teacher then indicated whether the execution was correct or incorrect. In the case of an incorrect execution, the teacher would provide a demonstration of the correct procedure and possibly provide additional annotation or advice, which was similar to the type of lessons in the learning-by-examples case.

Rather than directly learn procedures in the full language of Interlingua, the OSU learner used a hypothesis space more closely related to first-order Horn expressions in order to exploit their expertise in that area. In particular, the language of IL-Lists was defined for representing and learning procedures. An IL-List is simply a decision list of IL-rules, where an IL rule is a first-order Horn clause, whose body consists of a conjunction of predicates that represent the rule's precondition, and the head is a first-order term representing an Interlingua expression. An IL-List is executed on an input by finding the first rule whose preconditions can be satisfied, then instantiating the head of the rule to produce a grounded Interlingua expression, which is then sent to the Interlingua interpreter for execution. Below are two examples of IL-Lists learned by the OSU system. The first is a multiple rule list for a procedure called `SetSegmentFormation`, and the second is a single-rule list for a procedure called `LandAtLocationFromAerialLocation`. Prolog syntax is used where each rule is of the form *Head :- Body*.

IL-List for SetSegmentFormation:

```
Exec(SetFormation(X,columnFormation)) :- arg1(X),CallsForColumnFormation(X)) .
....
Exec(SetFormation(X,wedgeFormation)) :- arg1(X),CallsForWedgeWormation(X)) .
```

IL-List for LandAtLocationFromAerialLocation:

```
Exec(SEQ(SetGear(State="DOWN", UAVId=V3),
        Move(LoiterPattern="CIRCLE", Speed=50,
              WayPoints=LocationList(x=V1, V2, V4),
              Type="LAND", LoiterDir="CLOCKWISE",
              LoiterParam1=30.0, UAVId=V3)))
:- arg1(V3), arg2(Location(lat=V1, long=V1, z=V4)).
```

The primary difference between IL-Lists and traditional first-order Horn theories is that the heads of the IL-rules may include structured terms, whereas typical learners for Horn theories consider only function free heads (i.e. a predicate applied to variables or possibly constants).

The above examples illustrate that that space of possible IL-Lists is enormous, which suggests that without further assumptions the number of training examples needed for good generalization will be correspondingly large. This conflicts with the BL goals of learning from a much smaller amount of interaction with a teacher. To overcome this issue the key idea was to use a *hypothesis bias backoff (HBB)* approach. The main idea behind HBB is to attempt to learn a new concept in as highly a constrained hypothesis space as possible. The learner begins with a very constrained space and if learning fails, then the learner backs off on some of the constraints and attempts to learn in a larger space. The sequence of backoff steps continues until learning is successful (e.g. as measured by cross-validation accuracy or degree of consistency with training data). If the

sequence of possible hypothesis spaces agrees with the general biases of a good teacher, then learning will be fast since early spaces will be successful, which require few examples to learn in.

The key question with this approach is how to select the sequence of hypothesis spaces. The OSU system implemented a fixed set of spaces based on the principle that early spaces should be more biased but in ways that are consistent with reasonable assumptions about human teaching. For example, early spaces assume that equality is not coincidental and that procedure arguments have special status. The specific operations of the learning algorithm and sequence of spaces considered are as follows.

First, the learner parses the demonstrated procedure from an example lesson in order to form a ground expression representing the head of an IL-Rule. This generally consisted of simply composing the sequence of procedure calls in “Watchme” utterances. In other cases, special utterances about iteration allowed the learner to create head expressions containing loops. The set of all such structurally different heads across examples is then used to constrain the possible heads in the hypothesis space. In particular, the heads used in the hypothesis space must follow the structure of one of the example heads, but may introduce variables into the structure.

Given the constraints on the head, it remains to specify the constraints on the body. It is these body constraints that are relaxed to produce the backoff hypothesis sequence. The first hypothesis space allows type and argument constraints in rule bodies while treating procedure arguments as atomic objects. Also any predicates or facts indicated as relevant by the teacher are valid to include in the body. As an example, the first hypothesis space contains the following rule.

```
Exec(SEQ(SetSegmentMovementTechnique(X),SetSegmentFormation(X)))
:- arg1(X),SegmentType(X).
```

If the first hypothesis space is not expressive enough, then the space is extended to allow for procedure arguments to be treated as composite objects. For example, this space contains the following rule, which is not part of the first space.

```
Exec(SEQ(SetGear(State=, ..., LoiterParam1=30.0, UAVId=V3)))
:- arg1(V3), arg2(Location(lat=V1, long=V1, z=V4)).
```

Note that in both of the first two spaces a key assumption is that a good teacher will indicate that a fact or object is relevant if it is not part of a procedure argument. One can imagine that this is a reasonable assumption when a human teaches another human. That is, if there is one crucial piece of information out of a sea of other possibilities, a good teacher will give that information realizing that the student has little chance of guessing correctly on their own.

Finally, the third space extends the second space so that the body can test arbitrary facts that are “one relation away” from an object in a procedure argument. This idea could be extended to allow for additional backoff spaces by allowing for facts that are “n relations away”. However, further backoff spaces were not necessary for the concepts learned in the BL program. Further, such spaces become extremely large and would require a large number of examples for reliable learning. In such cases, it is reasonable to instead expect the teacher to provide more useful guidance, such as providing more relevance information.

It remains to specify how learning is done for each hypothesis space. We use a straightforward approach based on the idea of specific-to-general learning via least-general generalization (LGG) operators. In particular, given a set of examples and a hypothesis space, the first step is to compute the most specific ground IL-rule in the space for each example. Next, we follow a standard specific-to-general learning approach, where pairwise LGGs are computed until we observe an over-generalization (a negative example is covered, or too many implicit negatives are covered). The resulting rule is then added to the IL-List and the next rule is learned in the same way using the remaining examples. This process is repeated until all examples are covered by some learned rule.

The approach was implemented in JAVA using the standard protocols of the MABLE system and successfully passed all of the lessons assigned to it from the development lesson set.

Stratagus HIL

The work on the Stratagus HIL objective primarily involved specifying interface options and more substantially the engineering effort required to implement those options.

Some of the main interface functionalities that were implemented include:

- The ability for a human teacher to pause the game at any moment and observe the current status and action history of each unit in the game by clicking on the units.
- The ability for the human to navigate to a certain previous clock cycle of the game in two ways: 1) Clicking on an action in the history of a unit to go to the frame directly before the action was taken, and 2) Using a clickable bar at the bottom of the game panel.
- The human teacher can enter demonstration mode, where from any game state the actions taken by the human in the game are recorded for the learner to see.
- The human can provide critiques by clicking on a game agent and selecting a preferred action from the standard action menu.

We are currently continuing to test and expand this interface for use in an ONR funded project on active transfer learning, where a two-way communication channel between the teacher and student is important.

4.5 Important Findings and Conclusions

RL+Critiques

The main contribution of the work was to develop a principled and coherent approach for combining critique and practice data for the purpose of policy optimization. The few prior related efforts combined the different types of information in various ad-hoc ways with unclear interpretations. A key shift in our work was to follow a policy search approach, rather than value-function approach, which facilitated the formulation of a clear optimization objective that was solved via gradient-based methods.

Empirically, we found that on average across many repeated learning trials (using simulated teachers) the approach is able to effectively combine critique and practice data, producing a synergistic impact on policy quality.

However, we also found that in the individual trials with human subjects, the impact of including practice in addition to critiques was generally small and even hurt performance in some cases. Based on a qualitative analysis of user questions the reasons for this are likely due to the sometimes unpredictable behavior of autonomous learning and the delays caused by waiting for autonomous learning.

Active Learning

One of the main contributions of the work was to identify a key distinction between standard active learning in the i.i.d. supervised setting and active learning for policies in sequential domains. In particular, in the non-i.i.d. setting, the active learner does not have direct access to the target distribution over states and therefore must take care to not ask too many queries in bad or irrelevant states.

A second contribution was to derive a simple and intuitive active learning rule for sequential domains based on a Bayesian active learning formulation. The rule provides a principled way for trading off between action uncertainty in states and the likelihood that a state is relevant to learning the target policy.

Our experiments in 5 domains show that our active learning approach results in learning that is as fast and often much faster than a number of competitors. Most importantly it was much more robust than the competitors, which all failed very badly in at least some of the domains.

Object-Based Queries

One of the main contributions of the work was to introduce the problem of learning first-order structures in the context of object-based queries. Until this work, prior methods only considered more traditional membership and equivalence queries, despite the fact that human teachers are able to answer more expressive queries.

The theoretical results in the paper demonstrate that in the setting of exact theory identification, the inclusion of object-based queries allows for a large reduction in the number of membership queries. Further, conditions were shown where membership queries could be eliminated altogether.

MABLE Component

The results showed that complex procedures can be learned in the context of MABLE NIMs via the hypothesis-space back-off approach.

4.6 Significant Developments

MABLE Component

The hypothesis space back-off approach was full implemented and integrated into the MABLE environment. The learning component can learn procedures via arbitrary instruction sequences that follow the contract of learning procedures by example or by feedback.

Stratagus HIL

A prototype system was completed that provides a rich interface for human instruction in the Stratagus real-time strategy game engine. The instruction sequences produced by a teacher can be easily accessed by a learning agent via a simple socket interface. This interface is currently being tested, used, and extended in the context of an ONR project on active transfer learning, where learning from rich human interaction is a key element.

4.7 Implications for Future Research

RL+Critiques

Our results are promising. However, the user study suggests significant usability challenges when end-users enter the learning loop. This is particularly challenging when autonomous learning is in the loop, which sometimes results in the appearance that the performance of the learning agent is degrading due to the exploratory process of the learner. This suggests rethinking the way that autonomous learning is conducted with a human user in the loop. In particular, average case performance results, which tend to indicate smoothly improving performance of an RL system, are not sufficient. Rather, variance in the learning results must also be controlled and in particular the user should be shielded from sudden momentary crashes in the learner’s performance, which can be typical of RL on individual runs.

There are many directions for potential future work including: enriching the forms of advice that can be provided, studying user models that better approximate end-users, and incorporating end-users into model-based RL systems (as opposed to our current model-free setup).

Active Learning

There are many directions for future work on active learning of policies in sequential domains. First, it is important to consider policy optimization techniques that directly take “bad state” responses into account. Currently, states with those responses are ignored since we are using standard classifier learners to induce policies from the training data. It is also important to consider extending the notion of query to short action sequences that can be critiqued by a teacher. Providing such critiques will likely not be much more difficult than providing feedback on individual states and provides the user with more flexibility.

Importantly, these ideas need to be evaluated in the context of realistic human and compute-intensive automated teachers. In the former case, user studies are needed in order to fully understand the usability issues posed by active learning. In the later case, we plan to evaluate our approach for learning reactive policies to imitate the action choices of computationally expensive online planning algorithms. In such cases, it is important to minimize the number of queries to the planner/teacher since each call requires significant computation time.

Object-Based Queries

This work shows the promise of considering new types of queries in the context of learning first-order structures. This motivates further work on examining additional types of queries and understanding their theoretical and practical benefits.

While this work hypothesized that object-based queries might be easier for a human to use than traditional membership queries, no user study has been performed to evaluate this claim. This motivates a user study where users act as various oracle types to try to teach a concept. Users could also be asked what sort of information they would like to be able to express, possibly giving motivation for new query types or annotations.

Finally, the current study has provided only a first step toward understanding learning in the context of teacher errors. The types of teacher errors considered were both deterministic, but such a restriction may not be practical. A probabilistic model of oracle responses may be more likely in practice. Other error models can also be considered, such as a mixing of the two error types, or providing a large set of negative examples that are correct with some probability.

MABLE Component

One of the main open questions regarding the approach used for the OSU learning component is the generality of the hypothesis bias backoff idea. This idea is similar in nature to the concept of structural risk minimization in learning theory, but attempts to explicitly align the hypothesis space biases with expectations about a good teacher. There is room for future work in making this idea more formal and to develop a corresponding theory. Another important direction of future work is to test the generality of the idea using human subjects with a larger variety of target learning tasks.

Finally, in cases where the teacher does not perform up to expectations, either due to an error or neglect, it would be useful for the system to gracefully recover. Currently, once the system backs off too much, the hypothesis space becomes enormous and learning becomes impractical via just examples. In such cases, it would be better for the learner to indicate its confusion and to conduct a dialog with the teacher to elicit useful information other than examples. It is conceivable that the dialog could be structured around the hypothesis bias backoff idea, where the learner is guided by trying to reinstate previously removed constraints.

Stratagus HIL

Given the availability of the Stratagus HIL there is much room for future work exploring BL and more generally machine learning with end-user teachers. A particularly compelling challenge problem is to create a learning component that could be taught interesting strategies for sub-problems of the game via the interface. It is very likely that future work in this direction would highlight additional interface options that are currently missing and also identify current options that are not particularly useful or natural.

5. LEARNING CONCEPTS FROM TELLING (UNIVERSITY OF ILLINOIS CHAMPAIGN-URBANA)

5.1 Task Objectives

The University of Illinois team was responsible for researching and implementing the “Telling of Conditions” strategy.

5.2 Technical Problems

The Telling Of Conditions (TOC) learning strategy is concerned with learning named concepts that correspond to conditions. This broad definition can be broken down into three sub-tasks:

- *Truth Conditions* - the target concept is a parameterized predicate representing a truth condition that can be evaluated given a state and parameters.
- *Execution Conditions* - the target concept is a precondition to performing a procedure (“when to” statement). This precondition is checked within a specified context, another procedure.
- *Effects* - the target concept describes the state after executing a given procedure.

We study the research aspects of the issues raised while working on the implementation of the learning of constraints strategy. Our key concern was retaining the “natural” aspect in the teacher-student interaction while increasing the complexity of the concepts described in this interaction.

This can be done by allowing some degree of freedom in the teacher-student interaction, resulting in some mismatch between the expressivity of the teacher’s lesson (i.e., ITL) and the student’s knowledge representation language (i.e., IL). Such mismatches are due to difference in ITL and IL vocabularies and underspecified lessons, containing references to entities and concepts not explicitly defined in the lesson. Our research, described in the following paragraph, is concerned with accommodating such extended expressivity.

5.3 General Methodology

We study the research aspects of the issues raised while working on the implementation of the learning of constraints strategy. We looked into two key research directions:

(1) Reading in Context

We abstract the problem of interpreting the lesson to a natural language interaction between a teacher and student. In this framework we study language comprehension in the context of acting in a world model.

A straightforward approach to this problem is to convert the lesson directly into complete logical meaning representation that can be used as a rule based system in the relevant world context. However taking this approach would limit the teacher to a very restricted type of lesson and would considerably compromise the teacher’s ability to naturally interact with the student. To alleviate this difficulty we study the question of whether the student can learn how to interpret the teacher’s instructions. In the settings we study the teacher is allowed a high degree of freedom when formulating the lesson, which, while allowing a natural form of teacher-student interaction, results in a difficult learning problem.

In our research we exploit the dependency between the problem of understanding the lesson, and acting accordingly, formulated as a different learning problem, which can be learned using other strategies.

We study how acting in the world can help improve the student’s interpretation of the lesson, and how the student can benefit from instructions by integrating its interpretation into a learning

process for the actionable world task. To accommodate it, we break the Reading in Context framework into two learning tasks:

- a. **Learning to Read:** A lesson-understanding learning task that relies on interaction with the external actionable context to provide supervision. Unlike existing approaches to semantic interpretation that use annotated data mapping text to its logical interpretation, we rely on the environment to provide us with an indication about the correctness of our hypothesized interpretation and use it as supervision. In our experiments we teach an agent to use natural language to access a formal database using the retrieved response as supervision. We show how we can effectively train a semantic parser, converting ambiguous natural language questions into a formal language representation using only the supervision signal obtained from world interaction. A natural extension to this work is to move to other domains containing more challenging tasks and more complex natural language instructions. We are currently looking into computer planning domains. A second aspect of Learning to Read we are currently looking into is a new meaning representation paradigm, which focuses on actionable representations of natural language. This is a Task-oriented relaxation to the task of semantic parsing, which does not aim to transform the text into an equivalent meaning representation, but rather extract knowledge from the text, useful for the task at hand. This approach allows us to consider larger spans of text than the ones considered by current semantic parsing approaches.
- b. **Reading to Learn:** a domain task learning process that relies on the interpretation of instructional text to guide the learning process. Constructing a rule-based system from the interpretation of instructional text is a risky move. The agent's ability to interpret the instructions correctly is very likely to be limited. Moreover, the instructions may be incomplete or ambiguous, resulting in an incorrect set of rules regardless of the agent's reading abilities. To avoid these difficulties we integrate the interpreted instructions into a task learning process thus allowing the learner to use task-related supervision to improve the interpretation. In our experiments we teach an agent the rules to the game of Solitaire by combining both textual instructions and labeled card game data and show that using text facilitates the learning task.

Our current research looks into new ways in which the domain task can be exploited. One direction we are currently studying is extracting knowledge via domain interaction. The question we pose is whether domain knowledge obtained via domain interaction can be integrated into the representation obtained from the text.

The contribution of this research to the Bootstrapped Learning program is twofold. First, our learning methods are designed to handle noisy, ambiguous data, and learn from it even when very little supervision is available. The success of these methods depends on their ability to draw additional supervision from prior domain knowledge, not directly related to the specific learning task. We envision applying these methods when interpreting highly ambiguous Interlingua utterances using the properties of the domain, learned using other means. Second, we believe that this research direction can lead to deeper interaction between learning strategies since our method provides a protocol between a "telling of" strategy and a strategy for learning from examples.

To summarize, we suggest viewing learning a decision function as a natural language lesson interpretation problem instead of learning from labeled examples as done traditionally. This interpretation of machine learning is motivated by human learning processes, in which the learner is given a lesson describing the target concept directly and a few instances exemplifying it. We introduce a learning algorithm for the lesson interpretation problem that relies on lesson performance as feedback, and learns both tasks jointly. This approach alleviates the supervision burden of traditional machine learning by focusing on human-level task expertise for learning. We evaluate our approach by applying it to the rules of the Freecell solitaire card game. We show that our learning approach can learn the target concept and play the game. Furthermore, we show that the learned semantic interpreter also generalizes to previously unseen instructions.

In the course of this research we have made contributions to the study of new learning protocols and have developed algorithms to support it. In particular, we developed algorithms for **Learning with Indirect Supervision**.

(2) Student-teacher interactive learning protocols:

We are focusing in this part of the work on extending existing active learning protocols to interactive learning protocols. On the way, we have made significant progress in the vein of extending existing active learning protocols for structured and pipelined output spaces.

This work studies learning protocols where facilities for interaction between the learning algorithm and a domain expert exist throughout the learning process to improve system performance while reducing the amount of required labeled data and facilitating adaptation to new domains. The progress made focuses on increasing the bandwidth of the interaction with the expert, beyond the one typically studied in the active learning protocols.

One drawback of the active learning protocol is that the learner only incorporates this information at the level which it can induce from the labeling. Therefore, we have been pursuing more sophisticated interaction protocols, which allow the domain expert to interactively impart much of this information to the learner directly. Recently, we have made significant progress in this direction by publishing results introducing the interactive feature space construction (IFSC) protocol. In this protocol, the learner presents examples for which it believes its current feature space to be deficient for accurate predictions and for which external semantic resources are known to exist. In our particular example, it would be beneficial to know that west is a member of the semantic class direction and that arm means “subsidiary” in this context. This semantic feature abstraction would account for cases where only north is seen during training, but other directions are seen during testing. While existing semantic resources contain this information, a naïve incorporation into the feature space may result in higher sample complexity due to increased ambiguity introduced through these sources. By analyzing examples selected by the learner, the appropriate level of semantic abstraction is targeted for the specific task, thereby inducing a selectively more informative representation. Once learner-recommended abstractions are verified by the domain expert, these existing resources are incorporated into the feature representation by replacing features with their semantics classes, resulting in a simpler classification task. We have shown improved accuracy with reduced annotation requirements with this protocol on the entity and relation task described above. We conducted a direct comparison between IFSC and previous active learning methods, demonstrating the additional utility of a more sophisticated interaction protocol both in terms of performance and annotation

requirements. Recently, we have been exploring variance in the skill level of our domain expert, where we have seen that for our relation extraction task, the system is relatively robust to the domain expert performing the interaction. We are presently investigating this aspect more rigorously.

5.4 Technical Results

The details of our technical contributions in (1) Reading in Context and (2) Student-Teacher Interactive learning protocols are summarized in the publications below.

The results are in:

- (i) The development of the Reading to Learn and Learning to Read protocols. The underlying learning paradigm, which we call “Response Based Learning” supports Learning from Natural Instruction and is best described in Chang et. al, CoNLL’10 and in Goldwasser and Roth, IJCAI’11.
- (ii) The development of learning algorithms that are driven by indirect supervision. These are best described in Chang et. al, ICML’10 and Chang et. al NAACL’10.
- (iii) The development of active learning protocols for structured problems and for pipeline models, Roth and Small 2006, 2008.
- (iv) The development of Interactive Learning protocols, Roth and Small 2009.

Relevant Publications:

D. Goldwasser and D. Roth, “Learning from Natural Instructions”, International Joint Conference on Artificial Intelligence (IJCAI), 2011.

G. Kundu, D. Roth and R. Samdani, “Constrained Conditional Models for Information Fusion”, International Conference on Information Fusion, 2011.

G. Kundu, M. Chang and D. Roth, “Prior Knowledge Driven Domain Adaptation”, International Conference on Machine Learning (ICML), Workshop on Combining Learning Strategies to Reduce Label Cost, 2011.

G. Kundu and D. Roth, “Adapting Text Instead of the Model: An Open Domain Approach”, The Conference on Computational Natural Language Learning (CoNLL), 2011.

D. Goldwasser, R. Reichart, J. Clarke and D. Roth, “Confidence Driven Unsupervised Semantic Parsing”, Annual Meeting of the Association for Computational Linguistics (ACL), 2011.

J. Clarke, M. Chang, D. Goldwasser and D. Roth, “Driving Semantic Parsing from the World’s Response”, Conference on Computational Natural Language Learning (CoNLL), 2010.

M. Chang, D. Goldwasser, D. Roth and V. Srikumar, “Structured Output Learning with Indirect Supervision”, International Machine Learning Conference, (ICML) 2010.

M. Chang, D. Goldwasser, D. Roth and V. Srikumar, “Discriminative Learning over Constrained Latent Representations”, North American Conference on Computational Linguistics (NAACL), 2010.

- K. Pham, N. Rizzolo, K. Small, K. Chang and D. Roth, “Object Search: Supporting Structured Queries in Web Search Engines”, North American Conference on Computational Linguistics (NAACL), Workshop on Semantic Search, 2010.
- I. Titov, A. Klementiev, K. Small and D. Roth, “Unsupervised Aggregation for Classification Problems with Large Numbers of Categories”, International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.
- K. Small and D. Roth, “Margin-based Active Learning for Structured Predictions”, Accepted for publication in the International Journal of Machine Learning and Cybernetics (IJMLC).
- D. Goldwasser, M. Chang, Y. Tu and D. Roth, “Constraint Driven Transliteration Discovery”, Recent Advances in Natural Language Processing. Editor: Nicolas Nicolov, Publisher: John Benjamins, September 2009.
- A. Klementiev, D. Roth, K. Small and I. Titov, “Unsupervised Rank Aggregation with Domain-Specific Expertise”, International Joint Conference on Artificial Intelligence (IJCAI), 2009
- J. Eisenstein, J. Clarke, D. Goldwasser and D. Roth, “Reading to Learn: Constructing Features from Semantic Abstracts”, Conference on Empirical Methods on Natural Language Processing (EMNLP), 2009.
- D. Roth, K. Small, and I. Titov, “Sequential Learning of Classifiers for Structured Prediction Problems”, International Conference on Artificial Intelligence and Statistics (AISTATS), 2009.
- D. Roth and K. Small, “Interactive Feature Space Construction Using Semantic Information”, Conference on Computational Natural Language Learning (CoNLL), 2009.
- D. Goldwasser, M. Chang, Y. Tu and D. Roth, “Unsupervised Constraint Driven Learning for Transliteration Discovery”, North American Conference on Computational Linguistics (NAACL), 2009.
- S. Riedel and J. Clarke, “Revisiting Optimal Decoding for Machine Translation IBM Model-4”, North American Conference on Computational Linguistics (NAACL), 2009.
- A. Klementiev, D. Roth and K. Small, “A Framework for Unsupervised Rank Aggregation”, SIGIR Workshop on Learning to Rank for Information Retrieval, 2008.
- D. Goldwasser and D. Roth, “Active Sample Selection for Named Entity Transliteration”, Annual Meeting of the Association for Computational Linguistics (ACL), 2008.
- D. Roth and K. Small, “Active Learning for Pipeline Models”, National Conference on Artificial Intelligence (AAAI), 2008.
- A. Klementiev, D. Roth and K. Small, “Unsupervised Rank Aggregation with Distance-Based Models”, International Conference on Machine Learning (ICML), 2008.
- A. Klementiev, D. Roth and K. Small, “An Unsupervised Learning Algorithm for Rank Aggregation”, European Conference on Machine Learning (ECML), 2007.
- D. Roth and K. Small, “Margin-based Active Learning for Structured Output Spaces”, European Conference on Machine Learning (ECML), 2006.

D. Roth and K. Small, “Active Learning with Perceptron for Structured Output”, International Conference on Machine Learning (ICML), Workshop on Learning in Structured Output Spaces, 2006.

5.5 Important Findings and Conclusions

Our key findings are summarized below:

1. The development of the Reading to Learn and Learning to Read protocols. The underlying learning paradigm, which we call “Response Based Learning” supports **Learning from Natural Instruction** and is best described in Chang et. al, CoNLL’10 and in Goldwasser and Roth, IJCAI’11.
2. The development of learning algorithms that are driven by indirect supervision. These are best described in Chang et. al, ICML’10 and Chang et. al NAACL’10.
3. The development of active learning protocols for structured problems and for pipeline models, Roth and Small 2006, 2008.
4. The development of Interactive Learning protocols, Roth and Small 2009.

5.6 Significant Developments

Beyond algorithmic developments and our contribution to the SRI system we have developed two real time demonstrations of the Response based Learning protocol.

1. We developed a real time demonstration supporting a teacher giving real time, natural language instructions in a UAV environment. The demonstration was built in collaboration with BAE and presented to DARPA.
2. We developed a real time demonstration in the context of reading instructions for card games, with the goal of understanding the instructions so that we can play the game legally.

5.7 Implications for Future Research

We continue to work on Learning from Natural Instructions (Goldwasser and Roth IJCAI’11), focusing on a level of interaction that does not require the teacher to be a machine learning expert but rather only a domain expert.

6. LEARNING PROCEDURES FROM TELLING (ISI)

6.1 Task Objectives

End users today are able to create numerous applications such as spreadsheets, web sites, and games. How can they do this with no programming background? They are empowered by interfaces and languages that are designed for a given type of task and are natural to use. These interfaces and languages are not necessarily simple, they can be quite complex and although they may require some effort to learn they are learnable within reason. End user programs have a very different flavor, as they are more focused on core programming concepts and design rather

than on editing raw code. Some argue that programming applications have moved from having dozens of markets of millions of users (focusing on large software applications of universal appeal) to having millions of markets of dozens (focusing on reusing generic services and components). The future may be one where there will be millions of markets of one user, each representing a personal need that cannot be fulfilled by off-the-shelf code developed by someone else. This would be a manifestation of what is known as the “long tail of programming”.

The successful examples of end-user applications today typically focus on data manipulation through spreadsheets and web forms. However, there are no practical approaches that allow end users to specify procedures to process data or to control a physical environment. Some approaches have been designed to learn procedures from examples provided by end users through demonstrations or through observation, but learning from examples alone is difficult when the tasks are even moderately complex. A useful alternative would be for users to give the system a description of the procedure. This would make it easier for the user to tell the system how to generalize from the examples, but it would be necessary for users teaching tasks of reasonable complexity. In addition, human teachers find it natural to tell the procedures, in lieu of or in addition to providing examples.

Designing an electronic student that can learn procedures from human tutorial instruction raises many research questions that were addressed in this project. What form can tutorial instruction take? What kinds of information will need to be conveyed to the student during instruction? In what modality would this information be conveyed? Will instruction provided in a natural way by a human contain adequate information for the electronic student? What capabilities would such a student be expected to have?

Over the last few years, successful approaches have been proposed to allow end users to teach procedures through demonstrations. From a few demonstrations, a system induces a general procedure that generalizes from the particulars of the examples shown by the user. However, when procedures are complex it is hard to create demonstrations that cover the space of possible generalizations particularly if the user is to provide only a few examples. A complementary approach would be to teach procedures through tutorial instruction, a method commonly used by people to teach procedures to other people. In tutorial instruction, the teacher provides a natural language description of procedures using general situations and abstract objects. This is in contrast with situated instruction or demonstrations where a particular state is used to illustrate the procedure.

Tutorial instruction is an important approach to teaching computers since it offers a concise way to communicate complex procedures. It can be supplemented with demonstrations or practice to improve the learning process.

6.2 Technical Problems

There are many technical challenges in learning procedures from tutorial instruction. Human instruction by telling is naturally plagued with omissions, oversights, unintentional inconsistencies and errors. Teachers also often make incorrect assumptions about the student’s background knowledge and learning abilities. Humans can learn from such imperfect instructions because 1) they assume that such phenomena occur in the instruction and 2) they have strategies to work around such imperfections. The electronic students that we developed in

the BL program for learning from being told have similar assumptions and strategies. However, the student's task is easier or harder depending on the degree and nature of the imperfections in the instruction. Five key factors that influence the difficulty of learning are:

1. Situating the instruction by introducing objects
2. Describing steps in adequate detail
3. Specifying relations between steps
4. Complexity of conditionals and iterations
5. Background knowledge available to supplement the instruction

The first factor is the situation of instruction through introduction of objects. That is, instruction of procedures is often framed in the context of a general situation, specified by introducing a set of generic objects. For example, a lesson may start off by saying "suppose you have a route to follow, and a vehicle that cannot negotiate slopes of more than 30 degrees", which introduces two objects and their types plus a property constraint. If the instruction introduces all the objects and constraint that will be used in the lesson, it is easier for the student to understand how the objects are used throughout the procedure. If the instruction does not introduce some of the objects, then the student has to hypothesize what the objects are and what their types and constraints might be in the situations where the procedure must be applied. The more objects that are not properly introduced by the teacher, the harder the learning task because there are more hypotheses to explore. A different dimension that makes learning harder is the resolution of references when objects are similar. If there are several objects of similar types that are not specifically introduced by the teacher, it will be harder for the student to figure out which object references might be the same and which might refer to a distinct object and therefore figure out how many distinct objects need to be present. In those cases, the student has to hypothesize and explore more alternative assignments or permutations of the data objects.

In introducing new objects, another source of difficulty for learning is mixing the introduction of the objects within other expressions. A case of this is the need to introduce existential or universal quantification. For example, in the instruction "Suppose there is a company unit c1. If c1 has a mine plow, and then do X.", c1 is an input argument for the function being learned. However, in general, there can be multiple different interpretations of the same expression, depending on the nature of the introduced objects and how they are related. For example, with this instruction, "Suppose there are a Company unit c1 and a Platoon p1. If c1 has p1 and p1 has a mine plow, then do X.", if the student knows that multiple platoons are organized into a company, the instruction can be also interpreted as "Given a company c1, if all the platoons p1 in c1 have a mine plow, then do X", or "Given a company c1, if there exists a platoon p1 in c1 that has a mine plow, then do X. If such quantification is not explicitly stated, the student needs to form multiple different hypotheses. The number of hypotheses will grow when the student has to consider alternative interpretations of multiple objects.

The second key factor that makes learning more difficult is the lack of necessary details in describing a procedure's substeps. For example, the instruction may call a subprocedure as one of the steps but omit arguments that are required by the definition of the subprocedure. In that case, the student must hypothesize what objects may be used to invoke the subprocedure, perhaps also hypothesizing objects that were not introduced in the instruction. The more

arguments missing and the more candidate objects, the larger the hypothesis space to explore. Steps can also be specified indirectly by mentioning the effect of an action but not the action itself. For example, the instruction may say “Before starting the engine make sure there is gas in the tank.” In those cases, the student must rely on prior knowledge about actions that may have the effect mentioned, then insert the action where it may be appropriate in the procedure. The more candidate actions and insertion locations, the harder learning is. Instruction may also lack information about what to do when exceptions arise. Similarly, else statements may not be specified. In these cases, the student is unlikely to infer what the missing information is and would have to wait for other lessons to learn that (e.g. follow on practice).

The third key factor is how the relations among steps are specified. This includes ordering relations, causal relations, dataflow relations, resource relations, and temporal relations. In particular, instruction is always provided in a sequential order but a procedure’s step structure may be quite complex and non-linear. Of immediate concern are step ordering relations. The instruction may specify steps in a convenient but not necessarily correct order. In addition, steps are given in a sequence when the actual dependencies among steps are better represented as a partial order. If the step ordering is not fully specified or incorrect, the student needs to reason about possible steps and step orderings, as well as other relationships among them such as causal or dataflow relations. The space to explore can grow quickly as the number of hypotheses or the degree of ambiguity is higher.

An important relationship among steps is based on how the results and effects of each step are used by others. Some actions have side-effects and do not return any specific result, while other actions return an object that can be used as an argument of subsequent steps. The former typically correspond to representations of physical procedures while the latter are functions that are meant to be evaluated to find a value. If the student knows that it is learning a function, then it can reason about what each step is returning and the compatibility between the objects produced and returned by each substep as well as the entire function. If the student does not know whether it is learning a function or a procedure, then it must reason about the effects of each action (including side-effects and conditional effects) and the prerequisites of other actions in order to figure out how the substeps are related.

The fourth key factor is the complexity of conditionals and iterations. These control constructs are notoriously challenging for human teachers to specify in all the detail necessary in order for a procedure to be executable. That is, natural human instruction typically leaves out important things such as initialization and termination conditions for iterations, clauses in conditions, and the need to create temporary variables (e.g., counters) for iterations. When conditions and iterations are present, the student’s analysis of the procedure becomes more complex. For conditional branches, the student needs to consider the alternative data flows and control flows possible. For iterations, the student may need to analyze several folds in order to understand how a loop needs to work. Iterations that are easier to learn involve processing sets of objects one at a time. Harder iterations to learn are those that require the student to set up loops with new variables and infer exceptional initial and termination conditions. Conditionals that are easier to learn involve checking the state for a new object or property value to manifest. More complex conditionals can mention disjuncts and negations whose scope may be hard to determine from the instruction.

Combinations and nestings of conditionals and iterations within a procedure make learning harder. When conditional statements are nested, instructions typically do not specify the scoping of each statement. When scoping is ambiguous, the student must create and explore alternative hypotheses. The more combinations, the larger the search space that the student must explore. Iterations can also be nested, raising the complexity of learning because of potential interactions between the objects in the inner and outer loops.

The fifth key factor that influences the difficulty of learning is the availability of adequate background knowledge to help the student supplement what is stated in the instruction. The student must rely on background knowledge to detect and find workarounds to omissions and errors in the instruction. For example, if the knowledge base has overgeneral argument types for a function, it can lead to overly general hypotheses for an object type when the object is used for invoking the function. This makes the space of hypotheses larger and harder to explore. Also, the system may not be able to detect wrong use of an object as an argument of a procedure or function. For example, when the argument type is defined as a Number instead of a Speed, when a Distance object is assigned to the argument the student will not be able to detect the inconsistency. Given a set of arguments with overgeneral types (e.g. several Number arguments), and a set of objects to be assigned to the arguments, alternative assignment combinations would need be considered.

It is easier to learn when all concepts, vocabulary, and substeps have been previously defined for the student (in a prior lesson or as background knowledge). It is harder to learn if new vocabulary such as a new type of object is introduced in the middle of describing how a step is performed, since the student may have to backtrack to reason about earlier steps differently given the new assumptions introduced by the new object. It is also easier to learn a procedure if the substeps have been taught previously, since the student can reason about their nature and their effects to rule out implausible hypotheses rather than having to treat them as a black box of unknown nature that does not help the student constrain the space of possible models of the procedure being learned.

Finally, the difficulty of learning increases in lessons where larger numbers of omissions and errors are present. To learn from the lesson alone, the student must consider many combinations of hypotheses that lead to alternative models of the procedure. The student must also be equipped to reason about the plausibility and likelihood of models by reasoning over larger amounts of background knowledge. In practice, telling is combined with other learning methods. A lesson by telling may be followed by a demonstration or examples. Alternatively, the student could test the alternative models learned from telling, by exercising them in a simulation and measuring failures or by getting feedback from the teacher on practice problems.

6.3 General Methodology

The goal of this project was to create an intelligent system that can learn from tutorial instruction of procedures expressed in natural language and in a way that a human would find natural to provide. We developed a novel approach that combines several key features: 1) the use of a command line interface to guide the user to express instruction based on what the system already knows, 2) the use of paraphrase patterns to map the user's instruction into commands that the system understands, 3) sharing with the user what the system assumes of the instruction as well as alternative assumptions that could be possible, 4) deductive and heuristic reasoning to

complete and correct the instruction, and 5) facilitating disambiguation through option presentation to the user.

We describe the main features of our approach using an example of the interaction experienced by one of our users who is a non-programmer and used our TellMe system to create a procedure.

We use scenarios and examples where the system learns procedures executed by airplane pilots to patrol an area looking for oil pollution from ships. The user is teaching the kinds of reconnaissance tasks that pilots do in the Belgian Navy, which are scenarios that we chose for our evaluation as we will describe later on. The system starts off having a number of primitive actions for recording the situation with a variety of instruments, including infrared and ultraviolet cameras, SLAR cameras, and digital picture and video cameras. There are also primitive actions to send alerts and reports back to the base, and to generate initial estimates of the volume of the spill.

Below is an example of the utterances and the interaction of a user with the system to teach a procedure where a plane is to descend closer once a spill is found, then take videos and send them to the headquarters, and to record the GPS readings and send them along as well. The utterances are verbatim what the user would type. We will show later a screenshot of the current user interface, but for now we want to abstract from the interface details and focus on what the user and the system are communicating. The utterances are verbatim the instructions users would type to teach TellMe. Here is the interaction:

User: "**find oil spill, descend to a position of height 200**"

<The system shows the user that it assumes that meant to descend after finding the spill. It also shows the user an alternative interpretation where the descent was meant to happen before finding the spill. It asks the user to either accept the assumed interpretation and if not to choose the alternative.>

User: "**film the spill**"

<The system indicates it did not understand that>

User: "**record videos and send them**"

<The system shows the user that it assumes that meant to iterate over each of the videos and send each in turn, since the send action is defined for sending one document at a time.>

<The system shows the user that the result of sending the videos is a series of message receipts.>

<The system shows that it assumes the position to be after the descent, it also shows the alternative interpretation that it is the position before descending. >

User: "**record GPS reading**"

<The system shows the user that it assumes that the instruction meant to record the GPS reading over the position after descending. It also shows the user an alternative interpretation which is to record the GPS reading at the position when the spill was first found. It asks the user to either accept the assumed interpretation and if not to choose the alternative.>

User: <selects the latter option>

User: "**record image**"

<The system shows three interpretations, one for the action to record IR image, another for record UV image, and another to record SLR image>

User: “**send thickness image**”

<The system shows the user that it is not familiar with the term high level alert, but that it assumes it is a kind of alert given the context in which the term is used. The user does not have to interrupt the instruction and define it now.>

<The system shows the user that the send action requires some evidence as input, and that it assumes that to be the output of the record GPS reading action.>

Key Feature: Exposing Prior Knowledge

A challenge that users face when teaching a system is to figure out what the system already knows or what capabilities it has. Lessons always build on prior knowledge, using it as building blocks to the instruction.

Our approach is to constrain the user’s input with a command line interface that completes the user’s utterance based on the objects and actions that are already known to the system. In this way the system exposes known actions and object types, which serve as building blocks to the user’s expression of each instruction command. Here is an example of how the system exposes what it knows about properties of positions: when the user types “descend to a position with” then the system shows a pull-down menu that includes “descend to a position with height”, “descend to a position with longitude”, and “descend to a position with latitude”.

Key Feature: User Input as Controlled Natural Language

One important challenge that we need to address is that while natural language is a very natural way to provide tutorial instruction, interpreting unconstrained natural language is far beyond the state of the art.

Our approach is to use a paraphrase-based interpretation system that matches the user’s utterance against a set of pre-defined paraphrase patterns, following the approach in our prior work on the DARPA PAL program as part of the CALO project. Each paraphrase pattern is associated with a set of primitive commands that the user would have to use in order to have the intended effect that is described with the paraphrase pattern. The paraphrase patterns are exposed to the user through the command line interface described above.

For example, the utterance “descend to a position with altitude 200” is mapped to a paraphrase pattern component-as-verb +output-object +output-property + output-property-value. This paraphrase pattern is tied to a command that finds a component whose name matches the verb and adds it to the procedure. It further determines which of its defined outputs matches the uttered output object and asserts the output property value for the output property of the object corresponding to that output.

When an utterance cannot be mapped to any paraphrase pattern, the system indicates so to the user and then the user has to reformulate that instruction. This is the case with the utterance “film the spill” shown in the dialogue.

Several studies have found that users bring up new terms in any domain following a Zipf’s law and there are always new terms that come up. When a new term appears in an utterance, TellMe

will make assumptions about what it might mean. For example, when the user utters “send thickness image” and the system is not familiar with that term, it will assume that the term refers to an object (as opposed to an action or a property), and that it is a way to refer to an IR image since that is a type of image output by a step that is already in the procedure.

The combination of the command line interface and the paraphrase-based interpretation system gives the user the illusion of entering free text while the system actually is controlling what the user can input in ways that are amenable to understanding and interpretation.

Key Feature: Shared Learning State to Establish Trust

An important principle in user interface design is establishing user trust. A user needs to understand what the system is doing about the input she provided, and trust that the system is taking appropriate action. In our case, the system should give feedback to the user about what it is learning from the instruction. It must do so unintrusively, more as a nod than a detailed report, so that the user can focus on continuing with the lesson. Users need to know what the system has understood and learned so far as the lesson progresses.

Our approach is that the system always shares its internal learning state. For example, in many cases the user’s instruction is ambiguous and the system creates alternative interpretations, each resulting in a different procedure hypotheses. To show that it is considering these hypotheses, it shows them to the user. She is always asked to select one of them.

For example, in the third utterance the user specifies “record GPS reading”. The user did not say from what position to take the reading. In this case, the system generates three interpretations and shows them as options in the history window. The first interpretation is that the image should be taken at the position after the descent. But it is possible that the user meant the position before the descent, and that is presented as a second option. The third interpretation considers taking the reading from yet another position that the user may want to describe later. The user has to select one before continuing, and the top option is selected by default. As we will see next, TellMe uses heuristics to rank these options, and because it considers the first interpretation of the three to be more likely it will rank it first.

Key Feature: Deductive and Heuristic Reasoning

One important challenge is that natural instruction is often incomplete. Therefore, the system has to address those shortcomings if it is to learn the complete procedure. Our approach is to use deductive and heuristic reasoning.

Deductive reasoning is used to make assumptions about the objects and steps in the procedure in order to create constraints on objects that are underspecified. The system is effectively performing deductive reasoning to infer what is not mentioned in the instruction. All the constraints shown in the top right panel were deduced by the system, and most refer to objects that were not mentioned in the instruction. For example, the user does not mention that the input to the procedure is an area to survey to find the spill, but the system deduces that from what the instruction says. Also, the fact that taking a picture results in a new image being created is not mentioned in the instruction, but the system adds that to its procedure hypothesis.

Deductive reasoning is also used to interpret new terms that the system has never seen before. Recall that based on their role in the paraphrase patterns the system assigns a syntactic category. Through deduction, the system infers what is the type of new terms and possibly other constraints based on their role in the procedure. In our example, “thickness image” will be classified as a type of image.

The second kind of reasoning used in TellMe is heuristic. Heuristic reasoning is used to figure out what information about the procedure is still missing given the instruction so far, and what are possible ways to complete it. These heuristics essentially create possible completions or corrections of the procedure hypothesis that the system created from the user instruction. TellMe shows the user options that are ranked heuristically.

Heuristic reasoning makes the instruction more natural in that the system not only has identified what issues to resolve, which would result in questions to the user. The system has gone further in taking the initiative to formulate possible answers to those questions. This makes the instruction more natural because this is something that teachers expect from human students.

Key Feature: Selective Questions

An important principle in designing effective user interfaces is to take into account the cost of requesting user interventions. Because instruction is incomplete, the system may have many possible interpretations and therefore it could ask many questions to the user to determine which is the one that the user intended. Yet, later instruction may address those questions and so the user intervention was unnecessary and would not be considered natural. Although a user in teaching mode can be expected to be more willing to cooperate than in other circumstances, the system should not insist on asking questions constantly just to satisfy its learning goals to disambiguate and to complete the instruction.

Addressing this challenge is difficult, because if the system postpones all its questions then there may be a large space of possible candidate interpretations that would make learning very unmanageable.

We use eager questioning to ensure that a single procedure hypothesis is chosen by the user. The system asks the user to select among procedure hypotheses when several are possible. We use lazy questioning for other matters. For example when an unknown term is used in the instructions, the system makes assumptions about it and proceeds without interrupting the user with questions. This is the case in the interaction example above when the user refers to a “thickness image” which is an unknown term.

Based on this approach we implemented TellMe, an intelligent system that can learn procedures from natural tutorial instruction.

6.4 Technical Results

The TellMe System for Learning Procedures from Tutorial Instruction

TellMe reasons about 1) the current user’s utterance, 2) the results of the interpretation of prior utterances, and 3) its prior knowledge about the objects and actions in the domain of discourse. During this process, the system will detect omissions in the instruction and generate possible

alternative interpretations to present to the user as options. Any interpretations that are found to be inconsistent with the system's prior knowledge are ruled out. As the system processes the instruction, it may detect incorrect aspects and give it back to the user for reconsideration. The options are ranked heuristically and presented to the user for selection. At any given point, the user and the system are pursuing a single option as the active procedure hypothesis.

We first describe the user interface, and then describe four major modules in the system to process user instruction.

To be able to test the procedures learned from the teacher through the TellMe interface, we integrated TellMe with a powerful computational workflow system that includes Wings (<http://wings.isi.edu>) and Pegasus/Condor (<http://pegasus.isi.edu>). Wings includes semantic reasoners that are used to assist the user in setting up workflows correctly. Pegasus allows for large-scale distributed processing of data. The Wings workflow system has a web portal that allows users to upload data, run workflows, and upload new procedure steps. It is being used for a variety of scientific data analysis applications, including biomedical image analysis, population genomics, and text analytics.

TellMe User Interface

This is a screenshot of the TellMe user interface that corresponds to the interaction shown above:

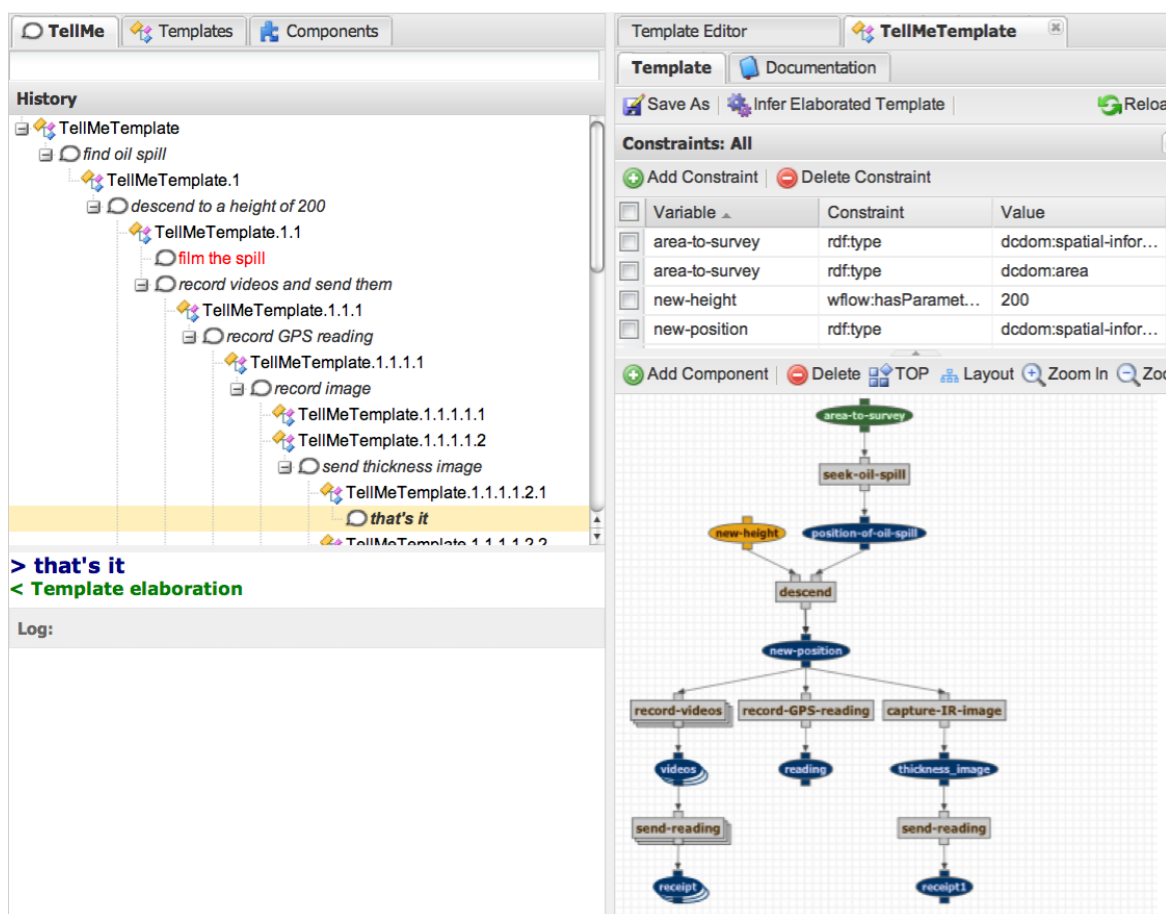


Figure 9. TellMe User Interface (screenshot)

The user enters her utterances (e.g., “Find oil spill”) in the command line on the top left. TellMe tries to provide the user with a menu of options to complete the current utterance.

When the utterance is completed, TellMe will show on the right hand side the leading hypothesis for the procedure being taught. At the bottom a dataflow diagram is shown. At the top, a set of constraints is shown, most of them inferred by the system. Green ovals show the inputs to the procedure, in this case the area the user wants to survey.

The previous utterances of the user, registered by the system are shown in the History area on the top left.

When TellMe is unable to parse or understand an utterance, it is shown in red in the History. This gives the user the option to rephrase her input. This kind of interaction allows users to quickly learn how to communicate effectively with the system – as we found in our formative user studies (see below).

When TellMe does not understand a particular word in the user’s instruction, it will create a new object for it in the procedure but will mark it in red to point out that it did not understand it. It will further infer as much as possible by context or from background knowledge. Entering “Find oil spill in the bay”, e.g., when the system does not know what “bay “ is, TellMe will mark it red, but also infer that “bay” is an area, because it knows how to find spills in an area. Users are not required to immediately answer questions about the new object.

When TellMe finds several possible ways of understanding a user’s utterance, it will create several options, listed in the History window. Before continuing with the instructions, the user can see the respective graph of each of these by clicking on them. This way she can choose the one she wants to continue working on. For example, the utterance “record image” has three interpretations which are shown as three possible branches of the option tree shown in the History.

A significant benefit of the History window is that the user can always go back if she later changes her mind about a choice she made or if she types something she would like to undo. She can then continue from that previous version from the History.

The user finishes a teaching session by typing “the end” or “that’s it”. TellMe will then further elaborate the learned procedure, as we describe in more detail below.

TellMe Matcher: From User Utterances to Commands

The TellMe Matcher builds on our prior work on mapping to-do lists into declarative agent capabilities through paraphrase pattern matching. In that approach, each formal action corresponds to a set of alternative paraphrase patterns that a user could use to state it. In TellMe, each command that would modify a procedure is associated with a set of paraphrases. For example, the utterances “descend to a new position” (a verb), or “descending to a new position” (a gerund, as in “after descending to a new position, record gps readings”) both map to a single command `AddNodeWithoutOutput +c +o` to add a component `+c=“descend”` and an output `+o=“new position”` to the procedure being created. In addition to paraphrase patterns for commands, TellMe also has a collection of paraphrase patterns to express step orderings (e.g., “first find spill then descend”), steps, data, and data properties. For example, “descend to a position with altitude 200” and “descend to a position with 200 altitude” are alternative ways to

express properties of the position after descending. Step ordering are enforced via dataflow links in the procedure graphs. All the paraphrase patterns are domain independent. TellMe also has a collection of domain-specific paraphrase patterns that represent equivalent terms to refer to each component, data, and property. For example, the “capture-IR-image” component correspond to the terms “capture IR image”, “take IR picture”, “capture infrared image”, “record IR image”, and so on. There are paraphrases that represent morphological variations, for example plurals of nouns and the gerunds above. We have two dozen paraphrase patterns for the oil spill domain.

The TellMe Matcher uses all these paraphrase patterns to generate the possible utterance completion options to the user. The result is a specific set of formal commands to modify the procedure whose rendering is the user utterance. For example, for “after descending to a new position, record gps reading”, TellMe generates a set of commands:

```
AddNodeWithOutput n1 +c=descend +o=new-position
```

```
AddNode n2 +c=record-gps-reading
```

```
AddLink n1 n2 +v1=+o
```

All these paraphrase patterns used by the TellMe Matcher are created manually beforehand by analyzing a corpus of typical expressions of procedures in the domain. This is typical practice for speech recognition and natural language input systems. This component could in principle be extended with WordNet synonyms (synsets) and with more extensive grammars than our current paraphrase-based approach. However, our current simple approach fulfills the important function of guiding user to express their utterances in a way that has the structure needed by the system to turn them into formal commands.

The TellMe Matcher also handles new terms in the instruction. For example, an utterance such as “descend to a safe position” mentions a “safe position” which is not defined anywhere in the system. This is a very common occurrence in instruction, as studies have shown that the number of terms used in subsequent commands increase and continue to grow following Zipf’s law. The TellMe Matcher hypothesizes the grammatical category of the new term based on its function in the paraphrase. In the example above, it will assume that “safe position” is a type of object since it is mentioned where an object type would be mentioned. The TellMe Matcher also assigns it a type based on the other known terms in the utterance. In the example, since “descend” outputs a position then “safe position” is assumed to be of type position. The new term is nevertheless marked as never seen before, so that the user can either corroborate these assumptions or retract the instruction and retry expressing what she intended. Additional information about the new term can be learned later on, but sometimes just going along with the assumptions is enough information to be able to learn a procedure that will execute just fine even with limited knowledge about the new type of object.

TellMe Creator: Extending the Procedure

The TellMe Creator adds to the existing procedure sketch according to the commands that result from the TellMe Matcher. The underlying workflow framework has a graphical editor for procedures, where users drag and drop components and connect them through links. The commands that are input to the TellMe Creator correspond to editing commands in that editor.

The instruction, however, may result in underspecified editing commands. In the graphical editor, one has to select the argument (or port) where an input is linked. In contrast, instruction tends to be incomplete because a user would rarely mention in an instruction what argument identifier the input corresponds to. Users would expect the system to figure this out based on the definition of each action. The TellMe Creator does this by checking the types of the inputs and assigning the link to the input that has a compatible type. For example, if the instruction stated “find spill and descend at that position”, a component to descend is added and an object of type position is added. In this case the input to descend is of type position, so the link is compatible and the argument identifier is clear. If more than one link is possible, then the TellMe Creator generates an alternative procedure hypothesis for each link. The set of hypotheses is carried to the next module.

The TellMe Creator handles some forms of incorrect instruction. An interesting case that arises is when the types assigned to objects are not compatible. An example is the instruction “estimate volume of oil spill with a UV image”, which results in a component to do the estimate being added and an input to it also being added with type UV-image. In this case, the definition of the estimate component specifies that its input must be an IR image, not a UV image. The instruction is considered incorrect, and TellMe will present the utterance marked in red back to the user so she can reconsider what she stated.

In extending the procedure, TellMe handles incompleteness in the specification in the instruction of an individual step. For example, for the utterance “descend,” TellMe will assume that the user meant to specify “descend from a position to another position with a specified altitude.” Even though those objects are not mentioned as part of the step, the system will add them as entities relevant to the procedure. These objects will be used in the module that we describe next.

The TellMe Creator also handles collections of objects by introducing iterations when an action input is limited to one object but the instruction refers to a set. This is how the utterance “record videos and send them” is handled.

TellMe Unifier: Tightening the Procedure

Instructions also tend to be incomplete in that they do not express that the same object is relevant to several steps. Consider the following utterance: “Find oil spill, descend, and take a picture”. This utterance does not say that the position where the oil spill is found is the same position where the descent starts. It also does not say that the position where the descent ends is the same position where the picture should be taken. So this instruction may be interpreted as each of those steps referring to different positions. TellMe uses a tightening heuristic that suggests that objects of the same type tend to be the same across steps unless otherwise indicated in the instruction. As any heuristic, this is not always the correct assumption and we need to give the user a way to indicate what the correct correspondence between objects is. Therefore, the TellMe Unifier makes as many mappings between objects as are compatible given the current procedure hypothesis, and will make them all possible interpretation options to the user. It will use the tightening heuristic to rank these options, presenting to the user the option where all objects are the same across steps as the most likely one. In cases when the heuristic is not correct, the user simply selects another option among those presented. This way, the system exploits an effective heuristic to make assumptions about how to complete the instruction while making it easy for the user to override those assumptions.

TellMe Elaborator: Elaborating the Procedure

The TellMe Elaborator is applied for efficiency considerations only when the user concludes the instruction, indicated with utterances such as “that’s it” or “the end”. It uses additional knowledge about the steps in order to check the validity of a procedure hypothesis. Recall that the TellMe Creator already does some validation by using the types of the inputs and outputs of steps to validate the links. The TellMe Elaborator applies other constraints in the steps and propagates them through the procedure as it is defined. For example, suppose the instruction states “descend to a position of altitude 200.” The previous modules will create a step where the output is a position whose has-altitude property has the value 200. A rule about the descend step will be used by the TellMe Elaborator to set the input parameter to 200 in order to obtain that desired output or effect. The TellMe Elaborator applies constraint propagation algorithms. Those additional findings were not specified in the original instruction, but are important to make the procedure executable. The TellMe Elaborator will add them as constraints to the procedure.

Integration of the TellMe System within the MABLE framework and Participation in the BL Program Annual Evaluations

In preparation for the BL Program evaluations and the integration with the SRI MABLE system, we started out the project carrying out several activities that would enable integration of our system for Telling of Procedure (TOP) within the SRI MABLE system, interacting with other learning (Natural Instruction Methods (NIMs) while ingesting appropriately the lessons received from the BAE curricula. We described the requirements for a Bootstrapped Learning process language and enumerated desired language features. In both cases, we provided examples from the UAV domain for illustration. This resulted in the document “Bootstrapped Learning Process Language Features v.0.1”, released to BL program participants in November 2007. We provided extensive feedback to BAE on the Procedures by Telling NIM contract to ensure that it not only prescribes learnable curriculum but also is as natural as currently possible. We also conducted systematic reviews of the BL diversity domain curricula and provided feedback to BAE. We also worked with the Learning by Noticing team in order to use their learning strategy during Learning by Telling’s process. This integration was demonstrated using the BlocksWorld curriculum in January 2008.

For the BL Phase I evaluations, we developed several extensions of our system to integrate it with the SRI MABLE Phase I system. We implemented a mapping between teacher utterances and our procedure representation. We also implemented mappings of procedure hypotheses to constructs in IL defcode. For interaction with other NIMs, we carried out thorough analyses of lessons that contain some mixture of procedural and conditional utterances.

For the BL Phase II evaluations, we improved our system over the phase I system by incorporating various control constructs that appeared in Phase II lessons and updating the integration with the SRI MABLE system. We continued to work on the integration with the new revisions of the SRI MABLE system and the Interaction Language (IL). For integration with MABLE, we have implemented a mapping between teacher utterances and our procedure representation. We also have developed mappings of procedure hypotheses to constructs in IL defcode. For interaction with other NIMs, we have developed an approach for handling lessons that contain some mixture of procedural and conditional utterances. In addition, we worked to

ensure that all constructs with the finalized Telling of Procedure NIM contract were supported. We developed and implemented a formalism and representation for dealing with the following constructs in the instruction: 1) control structure: conditions with When-ThenOtherwise, step ordering, return; 2) nestings of the control structure; 3) logical expressions; and 4) object descriptions: Suppose, Let, ANew. Finally, we continued testing the Diversity Domain (DD) lessons and providing feedback to BAE. We continued providing extensive feedback to BAE on diversity domain curriculum, as lessons have been updated and added over time. We also contributed to the team's curriculum guideline document that describes learnable curriculum. Our system was able to handle all necessary Phase 2 constructs, and was passing all DD lessons. We continued providing extensive feedback to BAE on diversity domain curriculum, as lessons have been updated and added over time. In collaboration with the Control Module and Learning by Noticing teams, we implemented the TOP and Telling of Concepts (TOC) interaction, which enables the use of TOC techniques for dealing with conditions appearing in TOP lessons.

In 2009, we implemented the TOP NIM Verifier, a tool for checking TOP lessons for their compliance with a crisp formulation of the NIM contract. This tool was actively used by our group as well as BAE/Cycorp to verify lessons and to identify bugs otherwise.

For the BL Phase III program evaluations, we extended our system to accommodate the new lessons and software changes in the evaluation harness.

Integration of TellMe with SRI's iBL System

ISI and SRI designed an integration of the TellMe system developed at ISI with the iBL environment developed by SRI. As a running example, we used a simple procedure to control a camera and take pictures of a desired target. This procedure involves the following:

- turn on a camera
- warm up the camera for a few minutes
- point the camera to a target
- capture an image with the camera
- process the image data captured
- transfer the processed image to secondary storage
- match the image captured against a template image (the template image represents what we are looking for, such as a silo or a runway) and annotate it with a border if the template matches
- transfer the annotated image to secondary storage

Before a user can teach procedures, the system would have pre-defined all the primitive actions that can be executed by the system. The basic primitives that need to be specified are *data objects* and *components*.

The components are the primitive actions that can be used as steps in the procedures that TellMe will learn. Data objects refer to the types of objects that are used or produced by the components.

The data objects defined for the initial example above are shown below. They are organized in a type hierarchy and can have properties that are inherited by the subtypes. The type hierarchy is shown on the left, when the type camera is selected its properties are shown on the right side.

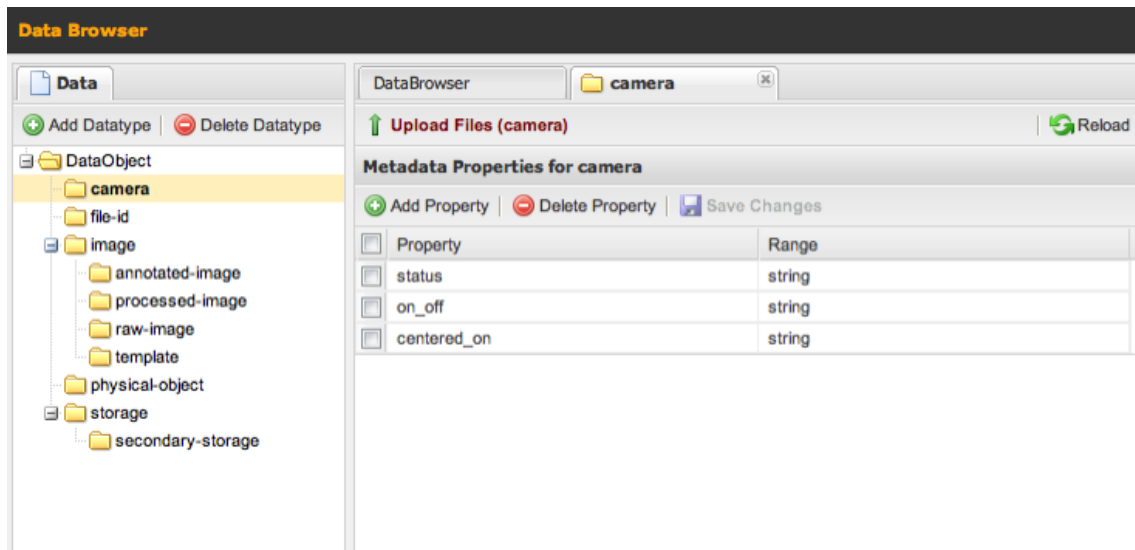


Figure 10. Data Objects (screenshot)

The components for our initial example are shown below. Each component has input data, input parameters, and output data. The input parameters control the action, for example for matching the image we could have a correlation threshold as a parameter.

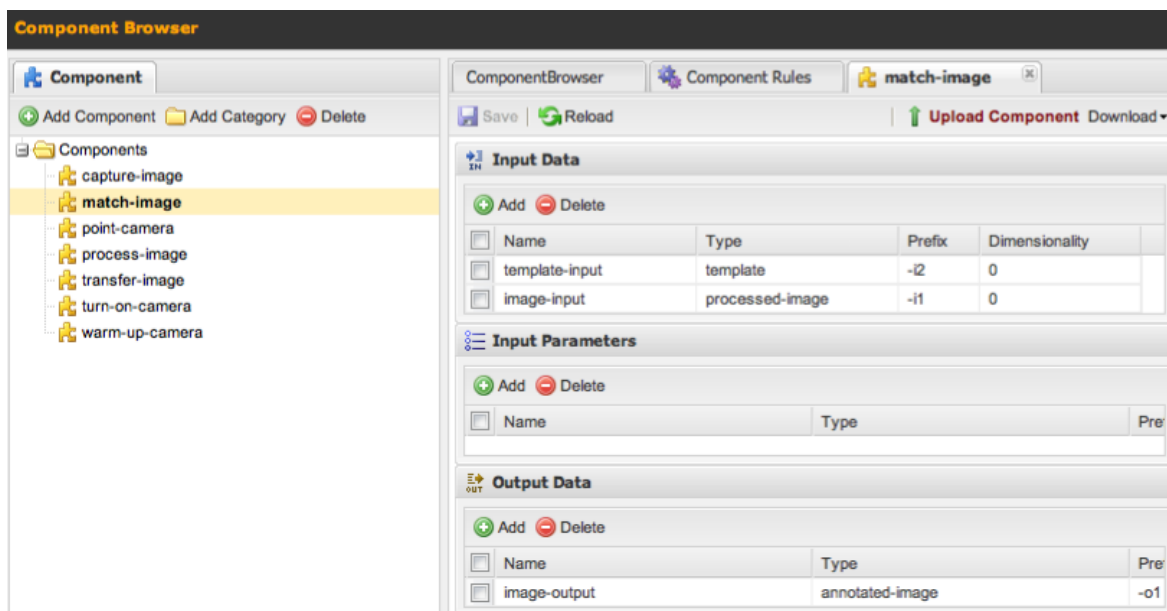


Figure 11. Components (screenshot)

Below is the procedure that the TellMe system would learn, and then compile it and export it for use in the iBL environment to control the iBL simulator.

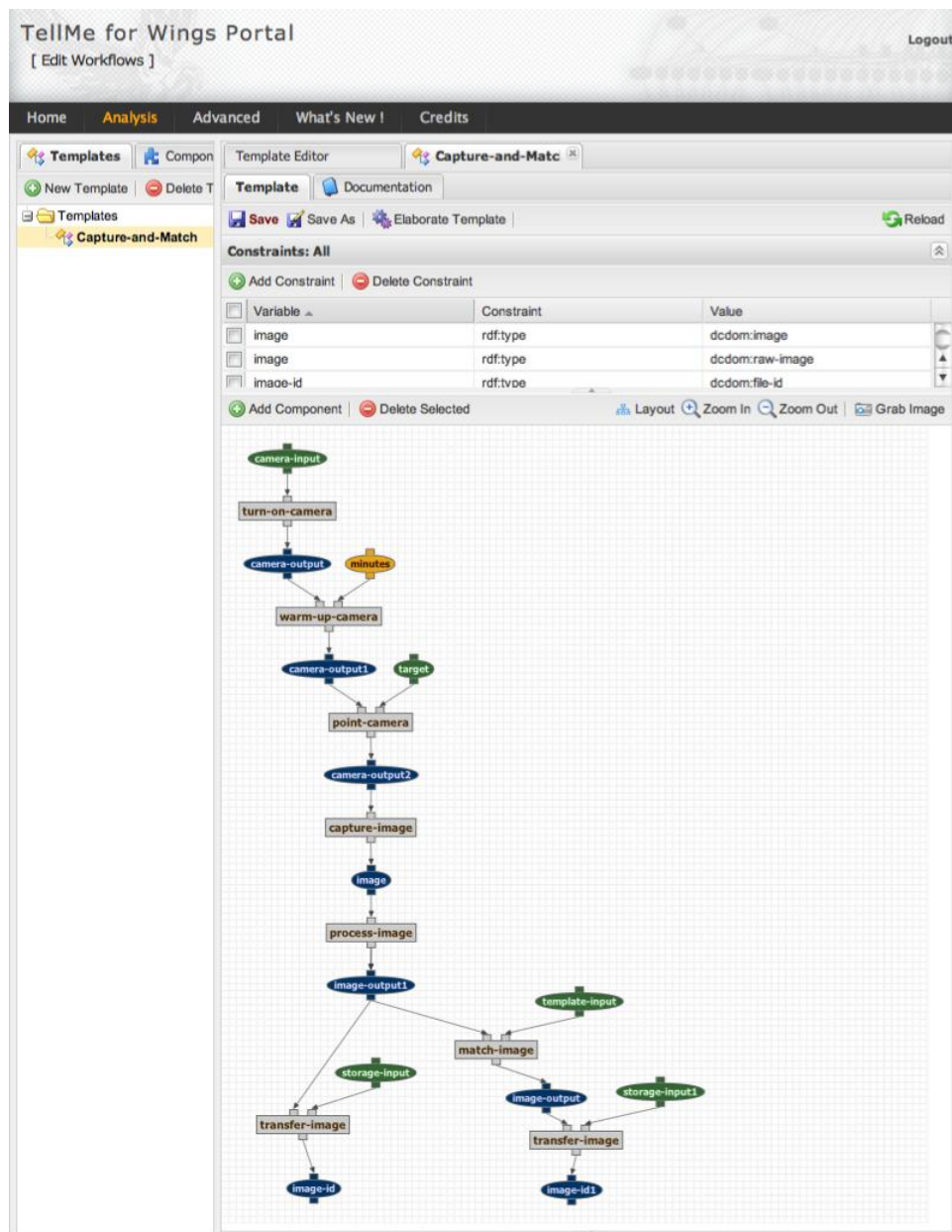


Figure 12. Procedure (screenshot)

The interface shows two aspects of the procedure. The graph illustrates the data flow among components, that is, what data is output by a component and used by another one. The second aspect of the procedure shown in the interface is the semantic constraints on the components and datasets. For example, the result of capturing an image is of type `raw-image`, which is inferred by the system automatically based on the definition of `capture-image`.

Demonstration of TellMe for ISR Scenarios Created by BAE

ISI and BAE developed a demonstration scenario for TellMe that is part of an ISR scenario developed by BAE. This resulted in a demonstration of the TellMe system in a DARPA-relevant domain, where TellMe could learn imagery processing procedures for analyzing satellite images. The procedures are novel compositions of basic imagery processing steps that are already pre-defined in the system.

We worked with BAE to use this work to support a demonstration for the National Air and Space Intelligence Center (NASIC), focused on ISR and in particular pattern of life analysis. The TellMe system was extended to accommodate the ISR domain to learn imagery processing procedures for analyzing satellite images. We created workflow components using the OpenCV library, and open source satellite imagery to illustrate the execution of the workflows.

The demonstration scenario was a clandestine theft scenario. The goal of the system is the detection of a clandestine theft at a factory based on GMTI and infrared data. The demonstration used actual imagery of a factory in Kirkuk, Iraq with GMTI and infrared tracks added manually. After detecting the clandestine theft, the analyst wants to do some further analysis involving the factory. The analyst wants to search for images of factories and see if any suspicious activity has been reported around them. The analyst might also want to compare images of the same factory over time to see if there has been any change that might explain why the theft is occurring no, for example because the factory has been adapted to produce some hazardous material.

TellMe is able to support the above scenario by allowing the analyst to specify procedures for analyzing satellite imagery that are composed of image analysis software components. Suppose a user wants to find a feature in an image that is described in a “template” file. The user can type:

```
sharpen image
match with template
annotate image
```

From these simple commands, the system infers a procedure that uses image processing software to do those tasks.

Below is a screenshot of the TellMe interface. The user types in the command line of the top left, a history of the commands issued by the user is shown on the left. The system shows on the right side the procedure that the system learned:

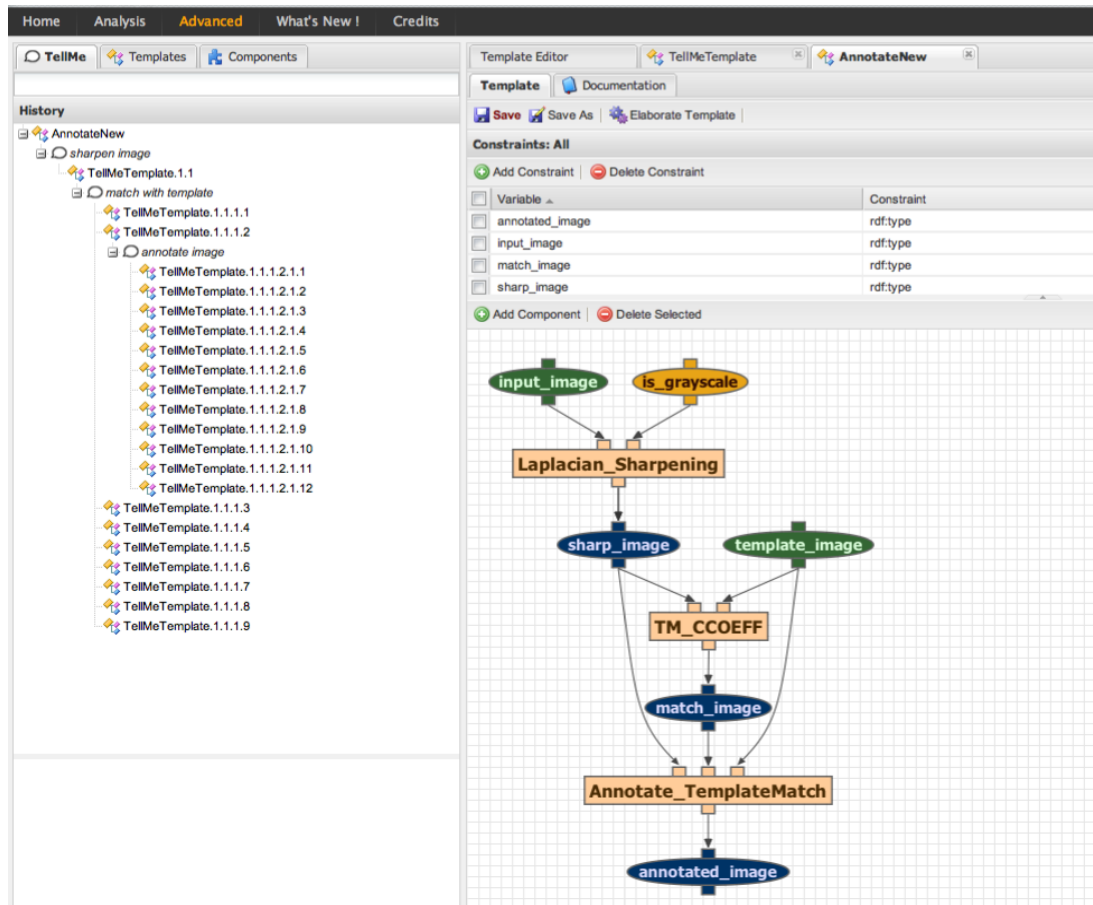


Figure 13. System Learns (screenshot)

Notice that TellMe has learned constraints about the types and other properties of the data that are used in the analysis. Based on this information, it can help a user to set up correctly the parameters for the analysis as we will see next.

The new procedure can be immediately executed. To do so, a user specifies as input a satellite image (“input_image”) and a template of something they are looking for (“template_image”):

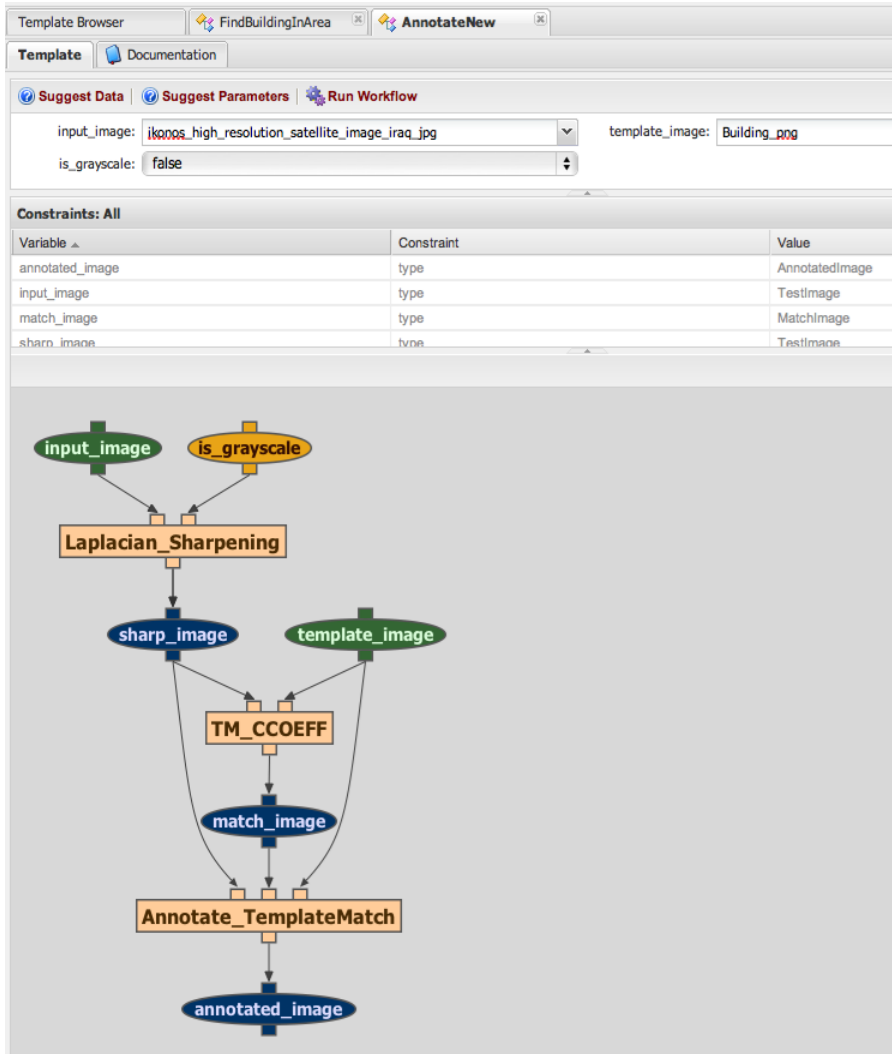


Figure 14. Parameter Setup (screenshot)

One the input image and the template that we are searching (in this case a silo-like structure), the system can automatically set (or suggest, whichever the user prefers) values for parameters.

Here, the system knows that the `input_image` is a color image and sets the `is_greyscale` parameter to false for the sharpening step.

When the execution is completed, the system can show the execution details of how the new information was derived. This screenshot shows the results of executing the workflow, with the inputs shown in the two pop up windows on the right and the result shown at the bottom right:

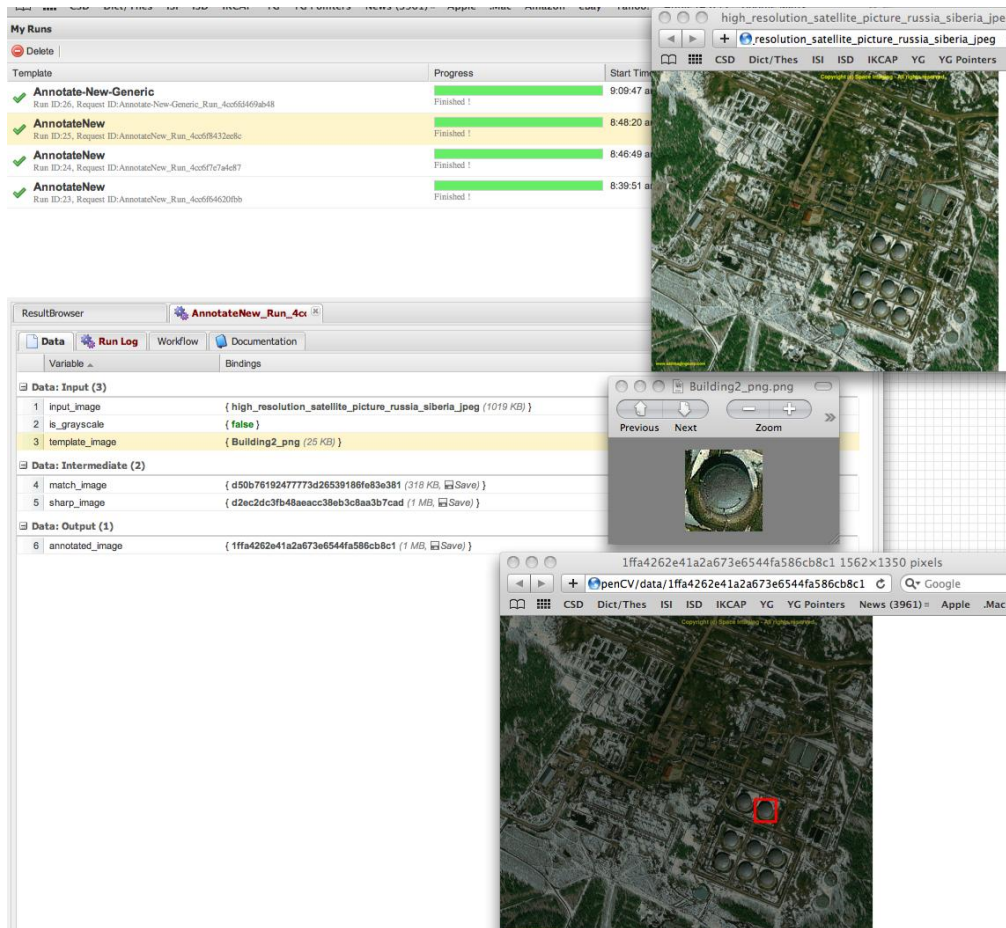


Figure 15. Execution Details (screenshot)

A user can specify more complex procedures with equally simple instruction. The user can type:

- sharpen a set of images
- match images with a template
- annotate match images

From these simple commands, the system infers that it needs to handle collections of images and collections of results for each processing step.

Below is a screenshot of the TellMe interface. The user types in the command line of the top left, a history of the commands issued by the user is shown on the left. The system shows on the right side the procedure that the system learned:

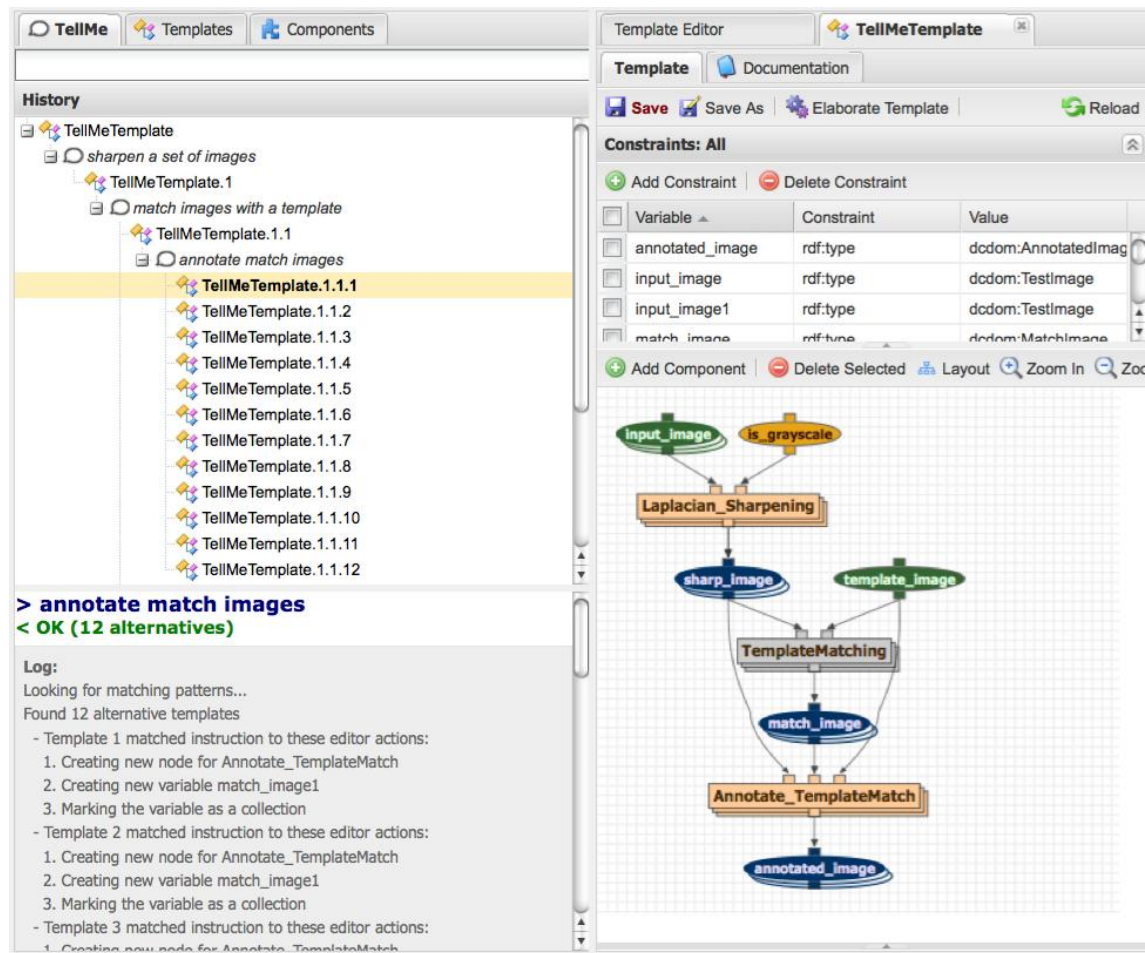


Figure 16. System Processes (screenshot)

Notice that the procedure learned shows “stacked” items that represent data collections. For example, `input_image` is a collection, each of its items will be processed with a separate `Laplacian_Sharpening` step, and so on.

The system will process all the images in parallel, submit the jobs in order based on the data dependencies in the workflow, and allow the user to browse the results similarly to what was shown earlier.

TellMe could be used to learn procedures that go beyond image analysis and into multi-intelligence data analysis. TellMe could be used to set up an ISR monitoring and warning system that tracks an input stream. This system could be used to define new imagery processing procedures to detect threats of interest, and have them running continuously against newly collected imagery. The system can seamlessly handle very large datasets concurrently, and would generate alerts when a result of interest is found.

TellMe could also be used to combine image processing procedures with threat detection procedures that operate on relational network data. Because the underlying workflow system is very general, other kinds of processing steps could be included for a more comprehensive framework for warnings. Additional steps could be added to analyze relational and social network data. The user could use these steps to define procedures for network data analysis. For

example, given the type of facility found in an image using the workflows above, a procedure could be created to determine whether anyone in a watchlist or their associates has traveled to the vicinity of that location based on travel data and on group finding algorithms.

As an example, the workflow below was used in a previous project funded by AFRL:

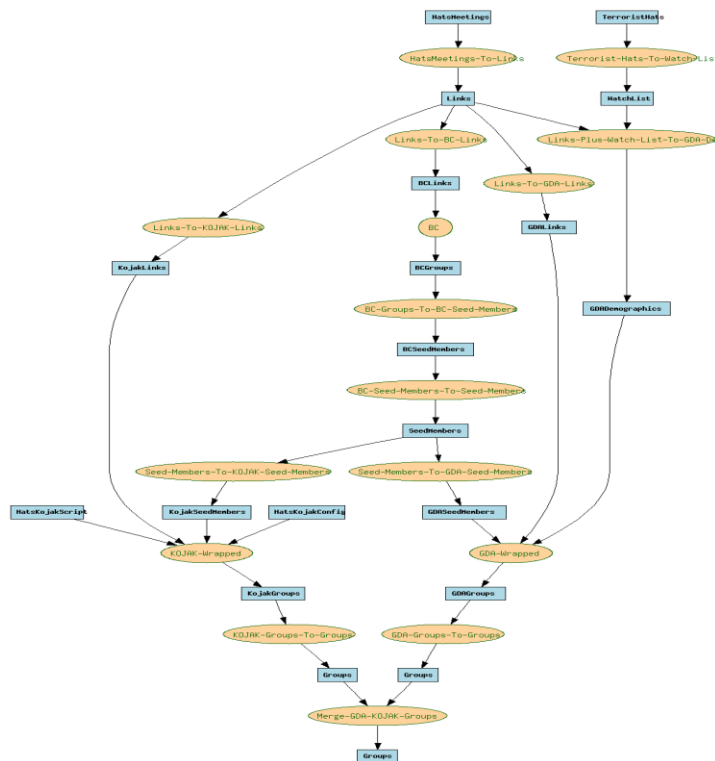


Figure 17. Workflow Sample

It includes several group finding algorithms (Betweenness Centrality, Kojak, and GDA) that first form seed groups given the watchlist and then expand the seed groups with additional group members. A parallelized version of this workflow was able to process thousands of entities in a few hours. This group finding procedure could be combined with the imagery processing process described earlier to form more comprehensive surveillance and warning processes.

6.5 Important Findings and Conclusions

The most important findings in this project were published in refereed conferences and journals. This section highlights these findings as a list of publications, including a short description of the results reported.

- Scaffolding Instructions to Learn Procedures from Users. Groth, P., and Gil, Y. 2009. In Proceedings of the AAAI Spring Symposium on Agents that Learn from Human Teachers, Stanford, CA.

Humans often teach procedures through tutorial instruction to other humans. For computers, learning from natural human instruction remains a challenge as it is plagued with incompleteness and ambiguity. Instructions are often given out of order and are not always consistent. Moreover, humans assume that the learner has a wealth of knowledge and skills,

which computers do not always have. Our goal is to develop an electronic student that can be made increasingly capable through research to learn from human tutorial instruction. Initially, we will provide our student with human understandable instruction that is extended with many scaffolding statements that supplement the limited initial capabilities of the student. Over time, improvements to the system are driven and quantified by the removal of scaffolding instructions that are not considered to be natural for users to provide humans. This paper describes our initial design and implementation of this system, how it learns from scaffolded instruction in two different domains, and the initial relaxations of scaffolding that the system supports.

- Analyzing the Gap between Workflows and their Natural Language Descriptions. Groth, P. T., and Gil, Y. 2009. In Proceedings of the IEEE International Workshop on Scientific Workflows (SWF), 299-305, Los Angeles, CA.

Scientists increasingly use workflows to represent and share their computational experiments. Because of their declarative nature, focus on pre-existing component composition and the availability of visual editors, workflows are often seen as more “natural” than programming or scripting languages for representing data analysis procedures. However, there is still a significant gap between the naturalness of workflow representations and natural language. In this paper, we aim to identify key constructs that intelligent workflow systems could support to allow for more natural workflow representations. These constructs are identified through a comparison of bioinformatics workflows and their associated descriptions obtained from the virtual research environment myExperiment.

- A Scientific Workflow Construction Command Line. Groth, P. T., and Gil, Y. 2009. In Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI), 445-450, Sanibel Island, FL.

Workflows have emerged as a common tool for scientists to express their computational analyses. While there are a multitude of visual data flow editors for workflow construction, to date there are none that support the input of workflows using natural language. This work presents the design of a hybrid system that combines natural language input through a command line with a visual editor.

- Leveraging Social Networking Sites to Acquire Rich Task Structure. Gil, Y.; Groth, P.; and Ratnakar, V. 2009. In Proceedings of the IJCAI Workshop on User-Contributed Knowledge and Artificial Intelligence: An Evolving Synergy (WikiAI), held in conjunction with the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI), Pasadena, CA.

Task management is a core part of knowledge work. However, intelligent assistance for task management is hampered by the lack of large amounts of structured knowledge about user tasks. In this paper, we present a novel approach, Social Task Networks, for obtaining rich user contributed task information by integrating task management with social networking sites.

- Social Task Networks: Personal and Collaborative Task Formulation and Management in Social Networking Sites. Gil, Y.; Groth, P.; and Ratnakar, V. 2010. In Proceedings of the AAAI Fall Symposium on Proactive Assistant Agents, Arlington, VA.

Personal tasks are managed with a variety of mechanisms from To-Do lists to calendars to emails. Task management remains challenging, since many tasks are interrelated, some may

depend on other people's tasks to be accomplished, and their priority and status changes over time. While many tasks could be automated by services and agents available on the Web, To-Do lists often capture tasks in textual form that combine non-automatable aspects. We have designed a new approach that combines techniques for task representations in hierarchical planning and for sharing in social networking sites. We broaden the notion of assistance so it is not just provided by computers but also by other people.

- Towards the Integration of Programming by Demonstration and Programming by Instruction using Golog. Fritz, C., and Gil, Y. 2010. In Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR), Atlanta, GA.

We present a formal approach for combining programming by demonstration (PbD) with programming by instruction (PbI)—a largely unsolved problem. Our solution is based on the integration of two successful formalisms: version space algebras and the logic programming language Golog. Version space algebras have been successfully applied to programming by demonstration. Intuitively, a version space describes a set of candidate procedures and a learner filters this space as necessary to be consistent with all given demonstrations of the target procedure. Golog, on the other hand, is a logical programming language defined in the situation calculus that allows for the specification of non-deterministic programs. While Golog was originally proposed as a means for integrating programming and automated planning, we show that it serves equally well as a formal framework for integrating PbD and PbI. Our approach is the result of two key insights: (a) Golog programs can be used to define version spaces, and (b) with only a minor augmentation, the existing Golog semantics readily provides the update-function for such version spaces, given demonstrations. Moreover, as we will show, two or more programs can be symbolically synchronized, resulting in the intersection of two, possibly infinite, version spaces. The framework thus allows for a rather flexible integration of PbD and PbI, and in addition establishes a new connection between two active research areas, enabling cross-fertilization.

- Assisting Scientists with Complex Data Analysis Tasks through Semantic Workflows. Gil, Y.; Ratnakar, V.; and Fritz, C. 2010. In Proceedings of the AAAI Fall Symposium on Proactive Assistant Agents, Arlington, VA.

To assist scientists in data analysis tasks, we have developed semantic workflow representations that support automatic constraint propagation and reasoning algorithms to manage constraints among the individual workflow steps. Semantic constraints can be used to represent requirements of input datasets as well as best practices for the method represented in a workflow. We demonstrate how the Wings workflow system uses semantic workflows to assist users in creating workflows while validating that the workflows comply with the requirements of the software components and datasets. Wings reasons over semantic workflow representations that consist of both a traditional dataflow graph as well as a network of constraints on the data and components of the workflow.

- Social Task Networks: Personal and Collaborative Task Formulation and Management in Social Networking Sites. Yolanda Gil, Paul Groth, and Varun Ratnakar. 2010. In Proceedings of the AAAI Fall Symposium on Proactive Assistant Agents, Arlington, VA.

Personal tasks are managed with a variety of mechanisms from To-Do lists to calendars to emails. Task management remains challenging, since many tasks are interrelated, some may

depend on other people's tasks to be accomplished, and their priority and status changes over time. While many tasks could be automated by services and agents available on the Web, To-Do lists often capture tasks in textual form that combine non-automatable aspects. We have designed a new approach that combines techniques for task representations in hierarchical planning and for sharing in social networking sites. We broaden the notion of assistance so it is not just provided by computers but also by other people.

- TellMe: Learning Procedures from Tutorial Instruction. Gil, Y.; Ratnakar, V.; and Fritz, C. 2011. In Proceedings of the ACM International Conference on Intelligent User Interfaces, Palo Alto, CA.
- Want World Domination? Play Risk! Matching To-Do items with How-Tos from the Web. Vrandečić, D.; Gil, Y.; and Ratnakar, V. 2011. In Proceedings of the ACM International Conference on Intelligent User Interfaces, Palo Alto, CA.
- To-Do lists are widely used for personal task management. We propose a novel approach to assist users in managing their To-Dos by matching them to How-To knowledge from the Web. We have implemented a system that, given a To-Do item, provides a number of possibly matching How-Tos, broken down into steps that can be used as new To-Do entries. Our implementation is in the form of a web service that can be easily integrated into existing To-Do applications. This can help users by providing them with an approach to tackle the To-Do by listing smaller, more actionable To-Dos. In this paper we present our implementation, an evaluation of the matching component over two sets of To-Do corpora with very different characteristics, and a discussion of the results.
- Interactive Knowledge Capture in the New Millennium: How the Semantic Web Changed Everything. Gil, Y. 2011. Knowledge Engineering Review, 26(1):.

The Semantic Web has radically changed the landscape of knowledge acquisition research. It used to be the case that a single user would edit a local knowledge base, that the user would have domain expertise to add to the system, and that the system would have a centralized knowledge base and reasoner. The world surrounding knowledge-rich systems changed drastically with the advent of the Web, and many of the original assumptions were no longer a given. Those assumptions had to be revisited and addressed in combination with new challenges that were put forward. Knowledge-rich systems today are distributed, have many users with different degrees of expertise, and integrate many shared knowledge sources of varying quality. Recent work in interactive knowledge capture includes new and exciting research on collaborative knowledge sharing, collecting knowledge from web volunteers, and capturing knowledge provenance.

- A Formal Framework for Combining Natural Instruction and Demonstration for End-User Programming. Fritz, C., and Gil, Y. 2011. In Proceedings of the ACM International Conference on Intelligent User Interfaces, Palo Alto, CA.

We contribute to the difficult problem of programming via natural language instruction. We introduce a formal framework that allows for the use of program demonstrations to resolve several types of ambiguities and omissions that are common in such instructions. The framework effectively combines some of the benefits of programming by demonstration and programming by natural instruction. The key idea of our approach is to use non-deterministic programs to compactly represent the (possibly infinite) set of candidate programs for given

instructions, and to filter from this set by means of simulating the execution of these programs following the steps of a given demonstration. Due to the rigorous semantics of our framework we can prove that this leads to a sound algorithm for identifying the intended program, making assumptions only about the types of ambiguities and omissions occurring in the instruction. We have implemented our approach and demonstrate its ability to resolve ambiguities and omissions by considering a list of classes of such issues and how our approach resolves them in a concrete example domain. Our empirical results show that our approach can effectively and efficiently identify programs that are consistent with both the natural instruction and the given demonstrations.

- Human Tutorial Instruction in the Raw. Yolanda Gil. Submitted to ACM Transactions on Interactive Intelligent Systems.

Humans learn procedures from one another through a variety of methods, such as observing someone do the task, practicing by themselves, reading manuals or textbooks, or getting instruction from a teacher. Some of these methods generate examples, which require the learner to generalize appropriately. However, when procedures are complex it becomes unmanageable to induce the procedures from examples alone. An alternative and very common method to teach procedures is tutorial instruction, where a teacher describes in general terms what actions to perform and possibly includes explanations of the rationale for the actions. This paper provides an overview of the challenges of using human tutorial instruction for teaching procedures to electronic students. First, procedures can be very complex and involve many different types of interrelated information. Second, human tutorial instruction is naturally plagued with omissions, oversights, unintentional inconsistencies, errors, and simply poor design. The paper presents a survey of work from the literature that highlights the nature of these challenges, and illustrates them with numerous examples of instruction in many domains. The paper organizes these challenges along four major dimensions: 1) situating the instruction in the context of relevant objects and their properties, 2) describing the steps involved, 3) specifying the organization of the procedure in terms of relationships among steps and substeps, and 4) conveying control structures. The paper discusses how the complexity of the learning problem increases with the complexity of the procedure in terms of the information to be conveyed and in terms of the amount of omissions and errors that occur in each of these dimensions.

- Capturing Common Knowledge about Tasks: Intelligent Assistance for To Do Lists. Yolanda Gil, Varun Ratnakar, Timothy Chklovski, Paul Groth, and Denny Vrandečić. To appear in ACM Transactions on Interactive Intelligent Systems, 2012.

Although to-do lists are a ubiquitous form of personal task management, there has been no work on intelligent assistance to automate, elaborate, or coordinate a user's to-dos. Our research focuses on three aspects of intelligent assistance for to-dos. We investigated the use of intelligent agents to automate to-dos in an office setting. We collected a large corpus from users, and developed a paraphrase-based approach to matching agent capabilities with to-dos. We also investigated to-dos for personal tasks, and the kinds of assistance that can be offered to users by elaborating them based on sub-step knowledge extracted from Web sites that describe how to carry out procedures. Finally, we explored coordination of user tasks with other users through a to-do management application deployed in a popular social networking site. We discuss the emergence of Social Task Networks, which link users tasks to their

social network as we all as to relevant resources on the Web. We show the benefits of using common sense knowledge to interpret and elaborate to-dos. Conversely, we also show that to-do lists are a valuable way to create repositories of common sense knowledge about tasks.

6.6 Significant Developments

There were two major contributions of ISI's research to the Bootstrapped Learning project. First, we demonstrated that non-programmers can use our approach to effectively teach complex procedures through tutorial instruction. Second, we were able to develop a framework that integrates a teacher's tutorial instruction with demonstrations (examples) of procedures. This second contribution was published in an article submitted to the ACM International Conference on Intelligent User Interfaces, a premier venue for publication of user teaching and demonstration, where it was recognized and selected as finalist for Best Paper Award.

We describe both key contributions in this section.

Can Non-Programmers Use TellMe to Teach Procedures from Instruction?

To evaluate our approach, we conducted initial formative user studies. The goal was to collect feedback on the overall approach and to find out whether there were any major barriers for users to communicate procedural knowledge with our interface.

We tested six subjects with ages ranging from 11 to 55. None of the subjects had programming experience, except one of the younger ones who had used Scratch. Each subject spent between 15 minutes and 1 hour using the system in total.

We wanted to avoid giving the subjects extensive training on our system. If our goal is to show that it is natural to give this kind of instruction, then any substantial training would defeat that purpose. The instructions to the subjects were limited to one page that described the different features and buttons of the TellMe interface, and are available on our web site¹.

We also wanted to avoid giving them descriptions of what procedures to teach because then they would likely just utter to the system the procedure in the way we would describe it to them. Therefore, we choose a Web site not developed by us that describes activities that are carried out by pilots that detect oil spills that pollute the sea off the coast of Belgium, without describing explicitly the procedures to be followed². We coded basic actions and object types, as well as the paraphrases based on the text that appears on that site. We asked the subjects to begin by looking at the actions that were already defined in the system. We then asked them to think of different kinds of procedures that they would think are reasonable for a pilot to carry out, and to teach them to the system. As a result, the procedures that different subjects created were unique and not comparable. There was wide variation on the size and nature of the procedures that were designed by the users themselves. For example, some procedures record images at different heights, while others focus on sending alerts and as early as possible. We noted that some subjects redesigned the procedure as they went along rather than beforehand, so in those cases

¹ <http://www.isi.edu/ikcap/tellme/instructions.html>

² <http://www.mumm.ac.be/EN/Monitoring/Aircraft/methods.php>

they had to adjust portions of the procedure that they had previously created, which they were able to do.

Our first goal was to check whether users would be able to use TellMe to specify procedures using natural language. For most subjects, the first few utterances were hard to get right, but after a few tries they learned to express what they wanted. This kind of interaction was typical across subjects. We also noted that the interaction with the command line helps the user construct interactively utterances that are already conformant to paraphrase patterns that the system understands. Here is an example of an initial interaction of one of our subjects, with the utterances that were rejected by the system marked in red:

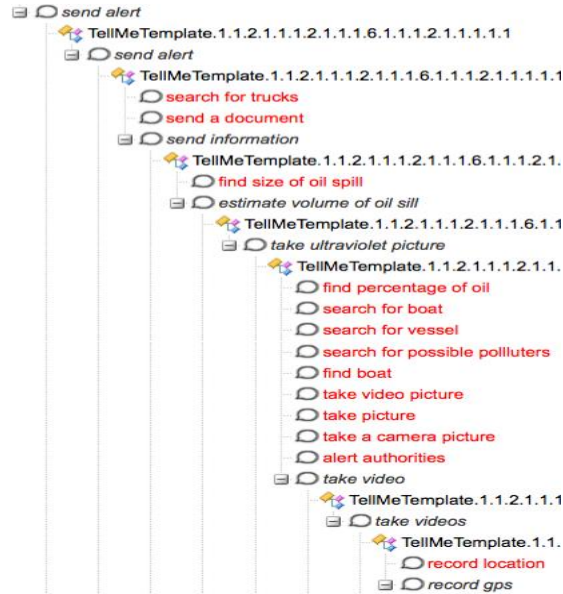


Figure 18. Initial Interaction

The subject was learning about what is possible to say, and what commands were understood by the system. After a few minutes, the same subject articulated the following instruction:



Figure 19. Articulated Instruction

where all the utterances were understood by the system.

None of the procedures created was a linear sequence, rather all procedures had an interesting structure with some parallel steps and with multiple step dependencies. The shortest procedure created by any subject had 4 steps. Below are two examples of the procedures that were created with TellMe by two different subjects:

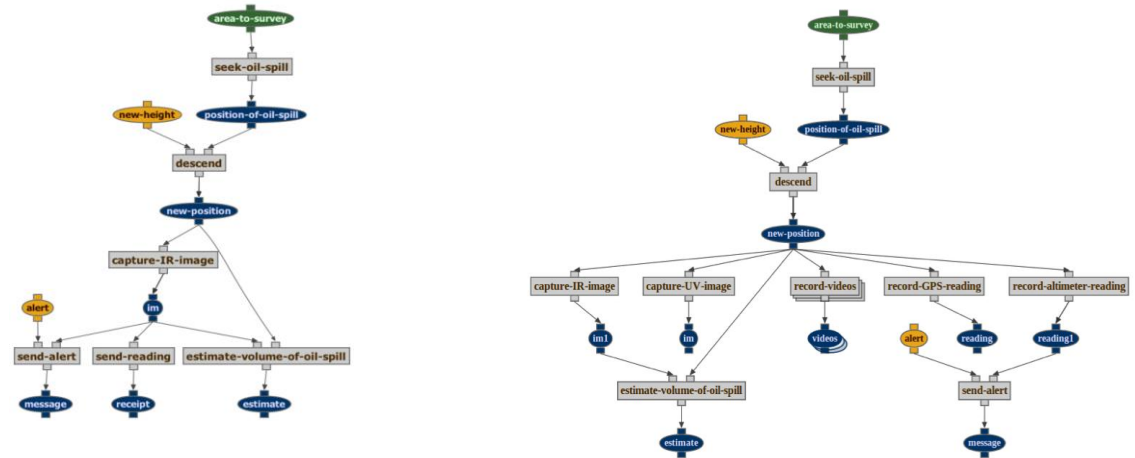


Figure 20. Two examples of procedures

Particularly interesting was the ease with which users created iterations that were implicit in the instruction but were not directly expressed. Below are two examples of instructions followed by the procedure learned and entered by two different subjects:

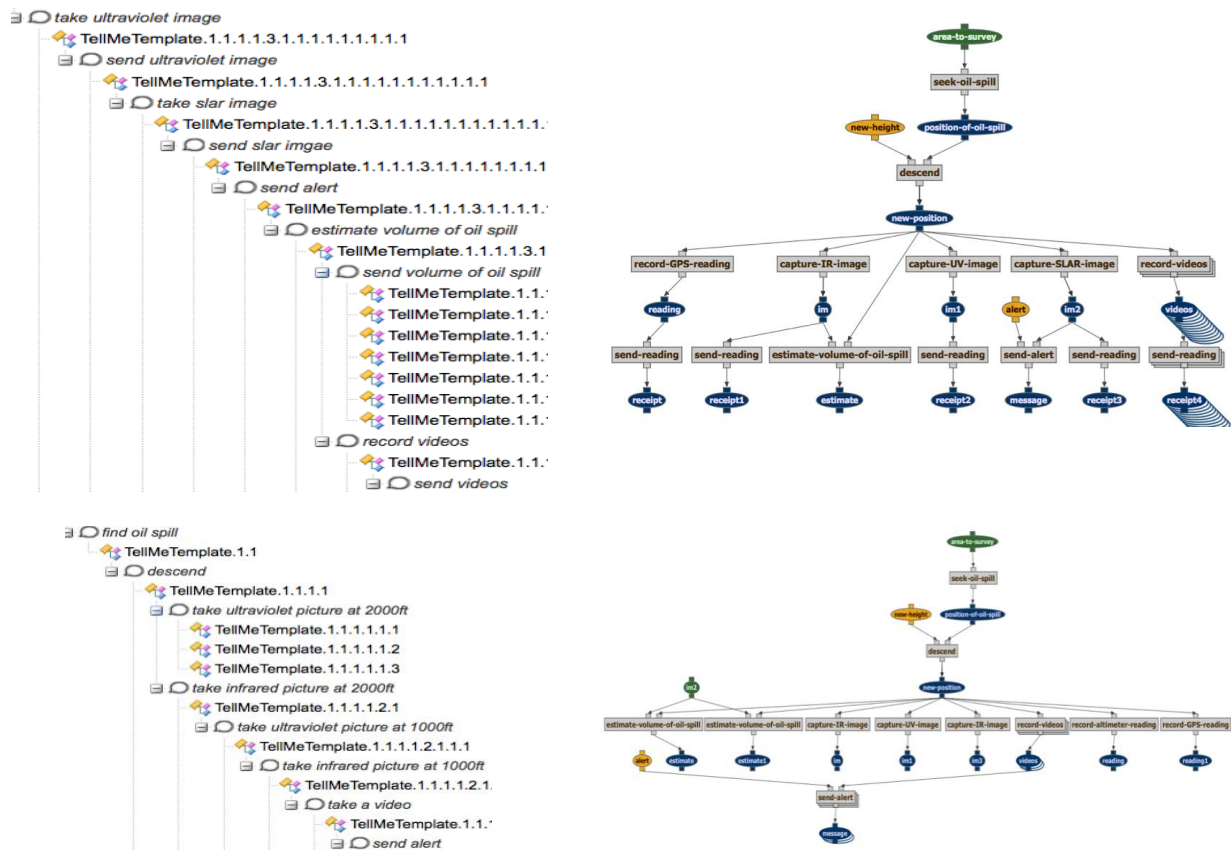


Figure 21. Examples of two sets of instructions entered by two different subjects

In the first one, “record videos” is followed by “send videos”. Since the send action can only send one document at a time, the system created an iteration where for each video there is a send action. In the second one, “take a video” followed by “send alert”, where there is no plural, resulted in an iteration where an alert is sent for each video.

All subjects were able to make most of their instructions understood. The table below shows the number of utterances entered by several subjects for one of their procedures, and the subset of those utterances that was accepted by the system:

Table 1: Subject Findings

Subject	Total utterances	Accepted
S1	9	6
S2	15	14
S3	39	17
S4	3	3
S5 (A)	41	19
S5 (B)	15	13
S6	27	10

There is wide variability across subjects. There is also great improvement as the user becomes more familiar with what can be expressed, as shown by the two procedures shown for subject S5.

We also wondered if users would be able to interact with the system to correct the system’s chosen interpretation when several are possible. In the following example of instruction, the subject inspected three alternative options and selected the third one, which represented what they had intended to convey:

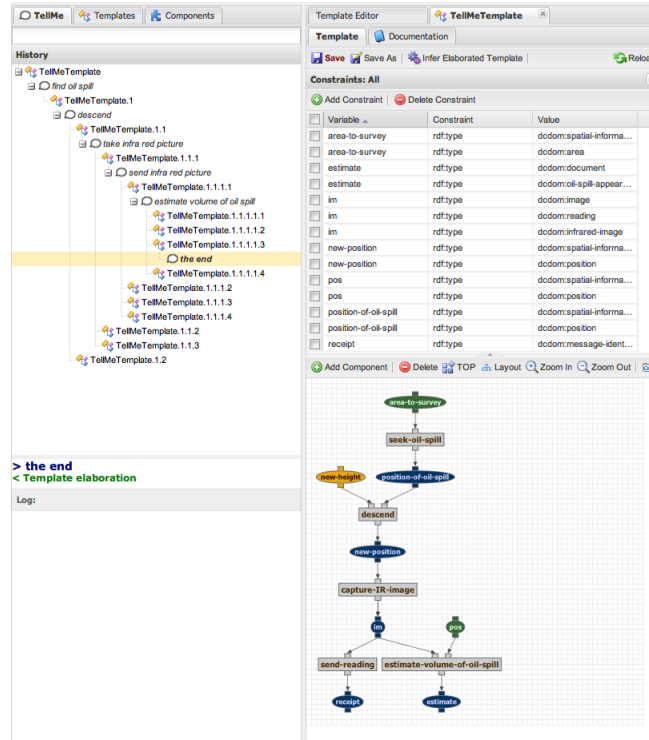


Figure 22. Subject chooses option

Also visible in that screen shot are the constraints that TellMe learns about the objects in the procedure, together with the topology of the procedure itself. These constraints are a significant part of what is being learned by the system.

The majority of subjects understood the procedures as the interface was showing them. The dataflow across the steps was not initially intuitive, but after seeing a few examples they would create a procedure that had all the steps interconnected. Some users never stumbled across such examples, so their procedures contained steps that were not integrated with the rest. Below is an example:

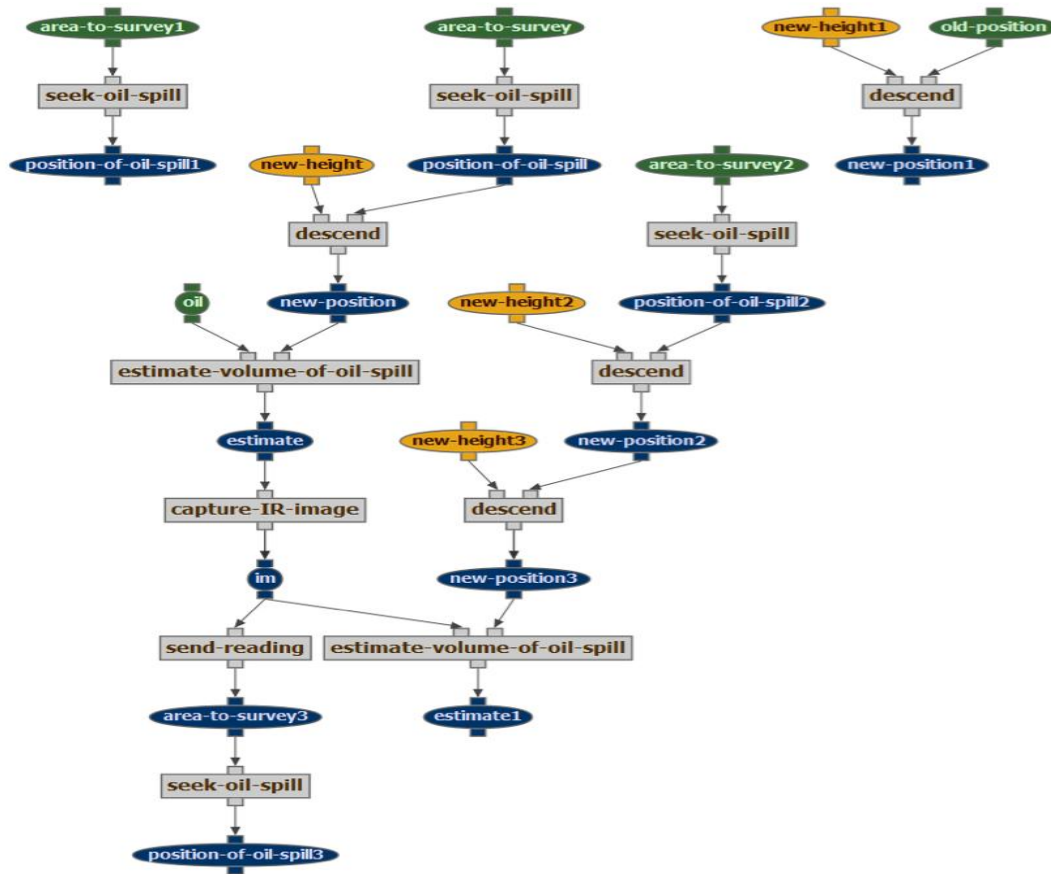


Figure 23. Non-integrated steps (screenshot)

We believe that this issue can be understood by users better but that we need to ensure that they are aware of the dataflow connections among steps. The tightening heuristic helped significantly in exposing subjects to these dataflow links. In some cases, subjects left inputs of steps unconnected to the outputs of other steps, in effect making them inputs of the overall procedure which was not intended. We believe a good remedy for this issue is that when the user is finished specifying the procedure, TellMe should explicitly mention what the current inputs of the procedure are, and have the user correct that if needed.

Can We Develop a Framework to Integrate Teaching through Tutorial Instructions and Teaching through Demonstrations?

This second contribution was published in an article submitted to the ACM International Conference on Intelligent User Interfaces, a premier venue for publication of user teaching and demonstration, where it was recognized and selected as finalist for Best Paper Award.

We developed a novel approach and implemented a system that produces the non-deterministic programs the hypothesis integration operates on. We redefined and implemented the semantics of the hypothesis integration in terms of program transitions, rather than evaluations ("transition semantics"). This provides more flexibility w.r.t. incremental program execution, range of algorithms that can be used for computing the program refinement (now allowing others than depth-first search), and synchronization of programs.

We developed an implementation to learn procedures from tutorial instruction that produces the non-deterministic programs that hypothesis integration operates on. We also developed an implementation to learn procedures from examples that produces the non-deterministic programs that hypothesis integration operates on. These two systems produced hypotheses that could be integrated by our framework.

We set up a basic harness for evaluation of hypothesis integration approach in the Stratagus domain, which is also used by the BL learning from demonstrations team, facilitating demonstration of the integration. We carried out several experiments to illustrate learning in the Stratagus domain. The procedures learned were quite complex in nature.

6.7 Implications for Future Research

In this project, we developed a novel approach and an implemented system to learn procedures through tutorial instruction using a natural interaction with a human teacher. Our research resulted in an interface that non-programmers were able to use to specify procedures of reasonable complexity. Many challenges remain, including handling incorrect instruction, using more complex procedure representations, and managing large numbers of procedure hypotheses. In addition, we would expect significant benefits from the integration of different modalities to learn procedures, concepts, and functions, within the same integrated framework.

In future work, we envision an electronic student equipped with an extensive catalog of strategies and heuristics that human students use to learn from natural and imperfect tutorial instruction. TellMe illustrates some of these heuristics, but many more could be investigated. For example, teachers do not always introduce all the objects that will be referenced in the instruction. When an object is introduced, the student would turn it into an input to the procedure. But when an object is referenced but not introduced, the student could adopt a strategy of making the object be one of the inputs to the procedure. The student should then apply such strategies and heuristics to 1) extract, 2) infer, and 3) hypothesize procedure models from the stated instruction. Because learning may never be completed and yet the learned procedure may be in good enough shape to be used, the student would apply these strategies to conduct self-assessments of the quality of the learned procedure models. Through this process the student would develop increasingly more refined (detailed) models of the procedure that are consistent with the instruction. The student also would need to reason about model constraints in order to detect whether a model is inconsistent and therefore discard it from further consideration.

Longer term, we could extend this work to develop an improved framework for an electronic student that enables it to learn to be a better student by discovering new strategies/heuristics as it goes through more lessons. This meta-capability for learning to improve student skills is omnipresent in human learning, and has received very little attention in the research literature.

7. LEARNING BY NOTICING

7.1 Task Objectives

The University of Arizona³ Bootstrapped Learning team, led by PI Cohen, had as its task objectives the following four goals:

- (1) Understand mechanisms underlying the ability to “notice” implicit instruction and fill in knowledge not explicitly provided by the instructor. This was also referred to as “learning by indirect cues” during the program.
- (2) Understand the dimensions of natural human instruction in the context of interacting with a capable electronic student.
- (3) Develop a prototype interface that permits natural instruction by a human teacher.
- (4) Learn procedural knowledge based on teacher demonstration.

The four task objectives evolved over the course of the program in response to challenges and results of earlier stages. These impacted the development of the task objectives as well as the methodology used to achieve them.

7.2 Technical Problems

There were several technical problems that the Arizona team encountered during the project. The following two were the most significant:

- (1) Consistent simulator behavior: During the program several domain simulators were used to provide data of unfolding events in scenarios. Each of the different simulators provided their own challenges, but in general all had problems in producing consistent behavior. Examples of inconsistent behavior included perceptual information arriving in different orders from one scenario to the next; some perceptual information being dropped or appearing in some scenarios but not others in which it was expected; and generally there was difficulty in reproducing simulator behavior, due to complex interactions with the Mable system and Interlingua representation. The turn-around time for addressing these issues was slow. The adoption of the XPlane simulator in the third phase helped address many of these issues.
- (2) Difficulties using Interlingua: The SRI team and Cycorp made great progress in providing better support for Interlingua over the course of the program, but in general Interlingua was difficult to work with and often felt more like an impediment to research than enabling integration. From the perspective of the Arizona team, there were several sources of these problems. First, the semantics of Interlingua was ill-defined and changed throughout the phases of the program, at times with no warning. Also, it was unclear what the requirements of the Interlingua Virtual Machine design were. In general, the Arizona team made progress by treating Interlingua simply as a syntax, defining an LbN-specific semantics within the LbN module. This is not necessarily the

³ Note: The Arizona team was based at USC/ISI in the first year of the project.

fault of the design or developers; rather, the design, implementation and use of any standardized knowledge representation is quite hard, and the BL program certainly was pushing these technologies in a new direction.

The primary impact of these challenges was delay in overall system integration, and perceived slower development of the overall working system. The Arizona team made most effective progress when not focusing on integration issues, using in-house simulators and representations. Overall, the Arizona team believes the development of the Phase 3 alternative of the iBL system, adopting RAPS (an already developed with many years of practical use in robust applications, and with a well-defined semantics) as the procedure language, and integration of the XPlane simulator environment were very promising and addressed a number of impediments. However, it was difficult to fully integrate with iBL in Phase 3 as much of the new framework was under development. In hindsight, it would have been ideal to start with that framework, and we believe it would be great as the focus system in a follow-on program.

7.3 General methodology

In Phases 1 and 2 of the program, the primary objective of the University of Arizona team was the development of the “Learning by Noticing” module in the Mable system. Learning by Noticing is the process of identifying and making explicit knowledge or information that is part of the background structure of current or past lessons and is relevant to the current lesson target concept, but not explicitly stated or referenced in the teacher’s instructions during the lesson. Were this information to not be made explicit, the performance of the other learning modules might: (1) not succeed in inferring the target concept, (2) might require more instances in order to make their inference (tantamount to requiring the teacher to make the implicit content explicit), or (3) the learned concept might be error prone or lower quality.

The UofA Team’s approach in Phase 1 to LbN development focused on transforming Interlingua descriptions of a scene (or sequence of scenes) and the teacher’s statements into a relational graph representation, and then applying a set of LbN heuristics to identify patterns in the graphs. The relationships the heuristics were designed to identify ranged from purely associational (e.g., correlation) to causal (e.g., the state of variable *X* modulates or controls the state of *Y*). The process of identifying these relationships is analogous to what statisticians do during exploratory data analysis: manipulating data to identify trends and patterns that may indicate important relationships that explain and predict other events. Once particular patterns were identified, the heuristics would then execute rules, such as labeling the pattern, introducing an explicit relation, editing the graph, or calling other heuristics.

While the heuristics developed in Phase 1 were interesting, it was unclear when if ever the products of the heuristics would be used by the other learning modules. In general, the lessons were constructed to exercise different modes of learning, but were not rich enough to require many of the noticing heuristics. In Phase 2, focus of LbN development shifted to providing functions that the other, now developed learners could use. Specific LbN tasks included identifying lesson episode boundaries, identifying the type of lesson being provided, and identifying candidate action preconditions and effects in order to construct action models that were then used by other learning modules (e.g., in the “MakeSo” scenarios, where a request for an action outcome was made without explicitly providing the action signature).

Throughout Phase 2, a number of questions were raised concerning how “natural” the scripted teaching episodes are compared to how humans might actually give instruction, and specifically, how humans might instruct a computational student. The findings of Dr. Dewayne Perry’s (University of Texas at Austin) study of human teachers attempting to instruct the Mable system provided some insight. A key observation of the transcripts that Dr. Perry’s study captured is that as soon as the teacher recognized that Mable is limited in the kind of instruction it can accommodate, the teacher tended to reduce instruction to a style more like direct programming than natural instruction. This pattern was observed in all of the transcripts collected.

In response, the University of Arizona team changed its basic methodology in order to investigate these questions, with the goal of better understanding how humans deliver instruction to an electronic student, to inform the design of the human instruction interface. The methodology included running experiments with humans using various interfaces to teach, and then using lessons learned based on analyses of the usage transcripts to inform design of a more natural interfaces as well as develop technologies for learning from demonstration.

In the fall of 2009, the Arizona team conducted a small pilot study that extended Dr. Perry’s study. The goal of the study was to construct an instructional environment that approximates what might appear in a natural field-programming scenario, where a human teacher must instruct an intelligent electronic student to carry out some task. In order to approximate a capable electronic student, the Arizona team used human participants as proxies for the electronic student in a “Wizard of Oz” (WOZ) protocol. In all cases, the teacher was told they are interacting with an intelligent electronic student, not another human. Also, both teacher and student did not have prior experience with the task before participating in the study. The teacher was given instructions about what to teach, but not how to teach it. Both student and teacher were given short training periods to get used to the interfaces used in the study. The studies were conducted in two conditions, one in which the student is a softbot in a virtual world, WubbleWorld, and the teacher instructs the student on how to complete simple tasks; in the other condition, the student controls a small physical robot and the teacher observes the robot’s behavior. In both conditions, the teacher issued simple commands in free-form text through a chat interface.

Analysis of the transcripts collected in the pilot study led in Phase 3 to the development of the University of Arizona Bootstrapped Learning User Interface, BLUI. In this phase, the Arizona team adopted the XPlane simulator domain as the domain in which to develop the teaching interface and teaching scenarios. The teaching interface was carefully designed to accommodate the variety of interaction modes observed in the pilot study while also constraining interaction so that teacher actions in the interface would be machine-readable. A follow-on WOZ study was conducted using the BLUI system.

During Phase 3, based on lessons learned from the WOZ studies and in parallel to the development of the BLUI system, the UA team also pursued two other tracks of work aimed at accommodating natural instruction in the BLUI system: Development of a framework for learning simple procedural knowledge base on demonstration, and developing a control framework for student interaction with the teacher. Both of these tracks were exploratory and there was not time to integrate them into the BLUI UAV ISR domain or iBL. However, their development was directly inspired by findings from the BLUI studies and the resulting system will be the subject of future work that integrates them into a student system.

The Arizona team developed a novel framework in which qualitative models of the semantics of an important class simple procedures, such as “follow me to the kitchen,” can be learned, and these models can perform two tasks: Executing the procedure, and recognizing when another agent has performed the procedure. This framework is based on a qualitative, relational model of the procedure using a modified finite state machine called a VFSM. The VFSM supports the recognition task natively, and we combined it with modern planning techniques to support execution of the model in complex environments. We developed methods for learning VFSM models through natural interaction with a human teacher in an apprenticeship learning paradigm. To demonstrate the efficacy of the framework, we conducted experiments whose results show rapid learning and high performance on both the recognition and execution tasks.

7.4 Technical Results

Phase 1 work on the Learning by Noticing module resulted in the creation and exploration of three broad categories of noticing heuristics.

(A) *Transformations*: transform data into other forms useful for further analysis or to expose patterns. E.g., (1) transform Interlingua utterances and perceptions into relational graph representations; (2) Identify chains (repeated structure) in graphs; (3) Identify “endpoints” (leaves) of graph patterns (boundaries); (4) Intersect and (5) Union graphs and (6) generalize non-matching nodes; (7) Compare two or more variables along dimensions; (8) transform continuous variable into discrete variable; (9) identify or propose complements and (10) inverses; (11) Elements that change together can be treated as the same; (12) Ignore (remove) elements that don’t change over time.

(B) *Discriminant*: functions and statistical tests that make decisions about the significance or strength of relationships between variables. E.g., (13) Chi-square test of two discrete variables; (14) Degree of pattern compression compared to uncompressed data; (15) Importance of special numbers (0, 1, 2, but not more) in patterns; (16) Compare distributions for similarities and differences (KL-distance); (17) identify episode boundaries and (18) relevance by locality by explicit teacher cues or patterns (e.g., the “current goal” is relevant to actions during the current demonstration, but not to other lesson instances).

(C) *Predictive*: heuristics for generating models from data and prior transformations for use in predicting other variables. E.g., (19) Regression; (20) Compare compressed representation of pattern to new observations and generate and test model instances not previously observed (ask if unlabeled stack of blocks is a tower; try building positive and negative instances of block arrangements and ask if they are towers); (21) Look for special names or functional roles of components of a collection (e.g., if a Tower is three blocks, that’s the role of the bottom, middle or top; are they named?)

Most of the LbN work in Phase 1 was done outside of the Mable system as much of the Mable architecture as well as the Interlingua Infrastructure were still under construction. The Arizona team created a set of demonstrations that used early versions of the Blocksworld and Robocup scenario output. Interlingua statements were processed syntactically as the Interlingua Virtual Machine was not yet complete. Arizona worked with the UT Austin, University of Wisconsin, Oregon State University and USC/ISI-Learning by Telling teams to identify potential use cases of LbN module outputs. Examples of specific demonstrations included identifying repeated

structure and identifying inverse relationships (on-top-of and below) in Block stacking in Blocksworld, and identifying an approximate predicate for “near” in Robocup (based on analysis of histograms of distances conditioned on ball possession).

In Phase 2, development of the LbN module specialized in response to specific Mable needs. At the end of Phase 2, the LbN module provided three key pieces of functionality: (1) Identifying preconditions and effects of actions based on examples to produce candidate action signatures in interlingua; these were used in conjunction with the Learning of Syntax, Learning by Telling and Learning of Constraints modules to solve a number of lessons; an extension of this functionality was developed into a facility to learn simple STRIPS planning operators and was reported in [Morrison *et al.* 2009]; (2) Identifying lesson boundaries, allowing for a relaxation trajectory in which lessons boundaries did not need to be explicitly indicated by the teacher, moving toward more natural instruction; (3) Identifying the type of learning needed to accommodate the lesson presented by the teacher by noticing the type of Interlingua instructions and their patterns of presentation, again moving toward more natural instruction in which the teacher does not need to explicitly identify the type of learning the student should engage in. The LbN module was successful in achieving these goals with high accuracy.

The experiment protocol and transcripts from the initial WOZ pilot study performed in Fall 2009 were published as a technical report in [Morrison *et al.* 2010]. In the softbot condition, participant Teachers and WOZ Students were able to communicate with one another using free text through a chat interface. The following observations were made based on analysis of the transcripts: (1) Modes of instruction were tightly interleaved; (2) Teacher’s did not explicit denote the beginning or end of a procedure, instead relying on contextual cues such as asking the student questions about their activity or asking the student to try again; (3) Teachers sometimes demonstrated actions themselves, instructing the student to watch, while other times they told the student what to do step by step; (4) Teacher feedback ranged from “good” or “no” to more elaborated explanations, such as “No, the blocks need to be on the ground, not the top”; (5) It was common for Teachers using free text to use multiple terms for the same object without explicitly noting the there different terms were being used; (6) Students asked questions of clarification, and Teachers gave definitions and reasons for actions or for labeling conditions; and (7) in some cases, Teachers provided background summaries of what they were going to teach before providing demonstrations or examples (e.g., “Let’s imagine that the sign post is an observer to hide from”). The robot condition had two variations, one in which Teachers could still use free text, but Students were restricted to selecting from a small number of utterances, and in the other, Students were not given any utterance capability. The following observations were made: (1) Similar to the softbot condition, all of the different teaching modes were observed and were tightly interleaved; (2) In the variant where the Student was not able to respond to the Teacher, Teachers tended to revert to the style of simple direct programming (delivering sequences of individual commands, waiting for the Student to respond after each command), as observed in Dr. Perry’s study with humans teaching Mable; (3) In the canned response variant, however, while the Teacher’s expressions tended to be simpler than in the full softbot free-text condition, the Teachers tended to ask more questions or conduct small tests of the Student’s state of knowledge; (4) Under both variants it was observed that when the Student followed instructions with no mistakes, the complexity of instructions increased, whereas the complexity decreased when the Student failed to follow directions correctly. Results of the pilot study were reported in [Kaochar *et al.*, 2011b].

The Arizona Bootstrapped Learning User Interface (BLUI) was designed to accommodate the patterns of instruction observed in the pilot studies while restricting all Teacher actions to selections from different instruction modes rather than free text. The BLUI provided interfaces for the following instruction modes:

- (1) Teaching by demonstration: The teacher could perform a sequence of commands and then group them into a procedure, and also label the sequence as either a good (positive) or bad (negative) example of a procedure. The teacher could either explicitly indicate they are beginning a sequence of commands as part of the procedure demonstration, indicating the end when finished, or the Teacher could later select a set of prior commands from a timeline and indicate they were part of a positive or negative instance of a procedure.
- (2) Teaching concepts by example: The teacher could define concepts by pointed to and labeling objects in the interface map
- (3) Teaching by reinforcement: The teacher can give feedback to the student using a scale indicating several levels from positive to negative.

Importantly, these Teacher was given free control to select any teaching mode at any time, including interleaving modes (e.g, starting a procedure, then labeling a concept, followed by issuing some commands, providing feedback, then ending the procedure example). The BLUI was constructed to use the XPlane flight simulator in conjunction with a map with objects that could be used to define UAV ISR scenarios. Participant Teachers were asked to teach simple tasks and concepts to the simulated Student controlling the UAV.

The BLUI was used in a more rigorous WOZ study with a total of 44 human Teacher participants. The experimenters controlled the XPlane and Student interface to simulator the electronic student. The following general observations were made based on the analysis of the transcripts:

- (1) We split the transcripts into three equal phases, beginning, middle and end, where each part had the same number of commands issued by the Teacher. Counting the number of times the different instruction modes were used, we found that across all transcripts, demonstration was used in equal amounts in the beginning, middle and end phases, while teaching concepts by example tended to be greater in the middle and end phases compared to the beginning, and Reinforcement increased steeply and linearly from beginning to end phases.
- (2) A comparison of the instruction mode usage between participants revealed that there were at least two classes of Teaching styles permitted by the interface: Most participants made use of the interface features to teach by demonstration, although a portion tried to teach the task to the student exclusively through the teaching concepts (providing object labels) by example, ignoring the demonstration facility.
- (3) Detailed analysis of the use of the demonstration facility revealed that there were several different ways the facility was used: (a) Teachers often used the demonstration mode as a means to label objects; (b) In all but two of the teachers, procedures were clearly intended to be taught but the procedure facility was not used; (c) procedure boundaries were often ill-defined, such as beginning a procedure demonstration, then “closing” the procedure example prematurely while still completing the procedure

- (4) Most Teachers were observed to explicitly test the Student on concepts, and follow the text with explicit feedback. Testing was also observed to occur most frequently in the last third of the transcript.

Further results and analysis of the WOZ study using the BLUI are reported in [Kaochar *et al.*, 2011a; Peralta *et al.*, 2011a, 2011b]

In the exploratory work involving the learning of simple procedures the Arizona team developed a model that can serve as a classifier of an instance of the procedure representing simple activities based on observed behavior, as well as model for executing the procedure in particular contexts. Early development of some of the precursor ideas for the framework are reported in [Fasel *et al.*, 2010, Walsh *et al.*, 2011].

In the work on learning simple procedures from demonstration, the Arizona team developed the VFMS model and methods for learning the VFMS structure in an apprenticeship learning framework, as well as employed planning technology to execute the model in complex environments. The protocol for teaching the procedure in the apprenticeship framework involved the following protocol: (1) The teach first selects a procedure and configures the environment to present the start state of the procedure; the teach binds each of the start state arguments to an object in the scenario and asks the student to execute the result model in the environment; (2) The agent uses the model along with a planner (a combination of the UCT algorithm with heuristics) to produce and execute a policy that enacts the model; the teach labels the execution as successful or unsuccessful; (3) If the agent's behavior is judged by the teacher to be unsuccessful, the teacher provides a *demonstration* of the procedure in the same environment; and (4) the agent then updates its set of positive examples to include its own execution of the procedure (if successful) and/or the teacher's demonstration (if the teacher produced one); The VFMS model for the procedure is then rebuilt by application of one of a set of construction methods.

The VFMS is a descriptive representation that does not explicitly prescribe a particular course of action to the robot. Thus, the robot must search the space of possible action sequences to find one that creates a sequence of changes in the qualitative state of the environment matching a path through the VFMS. We developed ways to solve this search problem by combining the VFMS with simulation and randomized sample-based planning techniques, enabling procedure execution in complex environments. Solving the search problem is made efficient by the introduction of a heuristic derived from the VFMS that estimates the number of steps required for completing the procedure. The marriage of a qualitative, relational procedure model and a robot control system is a major contribution of this work.

The VFMS learning and execution methods were evaluated by teaching the a set of different procedures that each included 10 to 20 variations in environment conditions in which demonstrations were presented. The experiments were run in two robot simulation domains, Gazebo and a version of the Wubble World domain that was used in the softbot WOZ pilot studies. We evaluated the performance of the VFMS on a set of 7 procedures, in the environments shown in parentheses: (1) go(Robot,Target): Travel to the target location. (Gazebo, WW2D); (2) go-via(Robot,Waypoint,Target): Travel to the target location via the waypoint location. (Gazebo, WW2D); (3) follow(Robot,Human,Target): Reach the target location after the human does. (WW2D); (4) intercept(Robot,Enemy,Target): Make contact with the enemy robot

before it reaches the target. (WW2D); (5) avoid(Robot,Enemy,Target): Reach the target without touching the enemy robot. (WW2D); (6) deliver(Robot,Item,Target): Navigate to and pick up the item, if necessary, travel to the target location, and then drop the item there. (Gazebo); and (7) go-around(Robot,Obstacle): Move around the object, ending up on the other side of it. (Gazebo).

Performance was compared against a recent Maximum-Likelihood procedure learning model based on teach demonstration, developed by Kollar et al. (2010). Overall, the VFSM was able to master almost all the procedures within 10-15 training episodes, especially with the most advanced learning method based on signatures. In virtually every learning condition tested, the VFSM was able to outperform the baseline method of Kollar et al. (2010) in procedure execution. Recognition performance generally mirrored execution performance, which is expected given that the same semantic representation drives both tasks in our framework. This result confirms the feasibility of achieving high performance in both tasks with a common representation, an important goal of our framework.

7.5 Important Findings and Conclusions

An important lesson of the Phase 1 LbN heuristic development was that the LbN approach has a number of dependencies that requires significant development and coordination with many other aspects of the Mable system. In particular, the following: (1) the need for consistent, rich domain scenario simulator environment output; this was a constant struggle in Phases 1 and 2, and in many cases the Arizona team was required to cleanse the output and provide very specific simulator behaviors in order for the heuristics to be successful; we never achieved a full online demonstration of the operation of the Phase 1 heuristics working with novel simulator behavior, but may be possible with additional development of the heuristics and simulators; (2) In order for there to be enough structure available for the LbN heuristics to function, there also needed to be additional Interlingua background knowledge that wasn't present in the framework; Arizona works with Cycorp and SRI to provide some of this background knowledge; (3) In order for the results of the LbN module to be used by the rest of the system, there needed to be a central, coordinated knowledge representation within Mable that can accept and store the outputs, as well as (4) specific development of the other learning modules so that they can make use of the knowledge in their learning.

In general, the development of the Mable system and focus on other learning goals did not make the more general purpose LbN heuristic behavior practical in the first two years, and in Phase 2 LbN development shifted to very specific learning need identified by several of the other development teams.

The teaching interface we built for this study is an exploratory system, designed to accommodate interleaving of multiple modes of instruction during teaching. Our goal was to use this system as a testbed to observe the types of teaching behavior that are elicited when given the opportunity to utilize multiple instruction modes. To the best of our knowledge, our system is one of the first of its kind but there is much room for improvement. While we have attempted to simplify the interface and reduce the number of teaching tools available, the complexity of the system can initially overwhelm a non-expert user. Although our teacher participants were given a 30 minute tutorial on the system usage, the complexity can be distracting and may have played a role on the productivity and structure of teaching. One of the teaching tools, the procedure definition facility, was particularly difficult for our participants to grasp and elicited unexpected behavior

such as implicit object labeling. However, we do believe that the interesting teaching behaviors we have observed cannot all be ascribed to interface flaws; rather they highlight issues that should be considered in building a successful human instructable electronic student.

Our analysis has shown that teachers often give instructions that require considerable interpretation and are not amenable to straightforward, direct identification. Humans make extensive use of implicit instructions, often fail to clearly indicate the beginnings and ends of lessons, appear to rely on meaningful linguistic names for objects and procedure labels, and frequently interleave testing and feedback, followed by new examples (and often do not give direct feedback after tests).

We found that a relatively simple set of rules can track aspects of the human Teacher instruction, but this was only successful at completely tracking teacher intention if the teacher follows a structured teaching style in which they used the interface as expected. Such a style makes teacher intent tracking “easy” because there is (1) a near 1-1 mapping between teacher instruction and intention, (2) most information relevant to intent is in local time context (based on very recent Student actions) and (3) no environmental context disambiguation necessary. This style of teaching, however, was the least common among our teacher participants and thus, a successful instructable robot will need to deal with the more subtle forms of teaching (often violating the criteria described) we found frequently in our transcripts.

In the learning of procedures by demonstration exploratory work, we use a simple demonstration protocol in which a VFSM for a given procedure can be learned with minimal burden on the teacher. We proposed a few increasingly powerful methods of constructing a VFSM from positive examples, culminating in a method based on procedure signatures. While learning an arbitrary finite-state machine from only positive examples is intractable, these methods are able to provide efficient learning by restricting the structure of learnable VFSMs. This trade-off further restricts the class of procedures the framework can learn to procedures with essentially sequential (rather than iterative or hierarchical) semantics. We also developed methods by which VFSMs can be composed to create more complex procedures by defining new procedures in terms of existing ones. We evaluated the performance of the entire framework, measuring execution and recognition performance of the VFSM for several procedures in two simulated robot environments, with a recent method from the human-robot interaction literature serving as a baseline. Our experiments demonstrated that high-quality VFSMs for several procedures can be quickly learned from interaction with a human teacher in the apprenticeship learning paradigm. In addition, we performed experiments probing the ability of the VFSM to represent procedures with primarily negative or goal-based semantics, revealing some limitations of the framework.

7.6 Implications for Future Research

There is still much to explore in the LbN heuristic exploratory data analysis framework. The idea of building a general purpose exploratory data analysis system has great appeal and we think the work on the Phase 1 LbN heuristics demonstrated promise. However, as identified above, further progress would require a consistent and rich domain simulator environment, a fair amount of existing background knowledge, and a central knowledge representation and learning control system prepared digest and make use of the knowledge uncovered by the heuristics. In general, this drives home that the entire student needs to be much more of a goal-directed

learning agent than a loosely federated collection of sub-modules each contributing independent knowledge to a passive store.

To our knowledge, the WOZ studies provide the first look at how human teachers instruct what they believe to be a capable computational student. The BLUI prototype provided a vehicle for studying behavior using an interface with multiple modes of instruction that may be freely interleaved, as appeared natural when humans were given the opportunity for free-text chat in the pilot study. The BLUI is quite primitive compared to the more sophisticated but constrained interface of the iBL system. However, the BLUI and its studies have achieved their goal of demonstrating the direction to head in to achieve more natural interaction. A key lesson is that there are significant challenges to overcome in tracking teacher intent, even when all of the individual commands available to the Teacher are individual machine-readable. In particular, the interface will need to accommodate automatic identification of implicit object labeling and implicit procedures as well as teacher-specified procedure boundaries needs to be improved and tested. These behaviors make the teacher intent tracking problem challenging and in order to address these issues, an instructable student system will need to include some nontrivial mechanisms to: (1) track environmental changes in the teaching session, (2) identify what context (local or global) is relevant to teacher's instruction (restricted language comprehension capability may be necessary), and (3) handle interleaving teacher instructions. Not only will this help robots learn from natural instructions, it will also be useful for helping the robot actively guide the interaction through robot-human feedback.

We believe the VFMS procedure learning framework is very promising, and there are a number of next steps to take this work in order to move beyond its current shortcomings: (1) The ability to accommodate more robust hierarchical composition is necessary if a robot is to achieve true bootstrapping and build arbitrarily complex procedures from simpler ones; (2) The VFMS is currently able to represent only a simple kind of conditional semantics through its branching structure. As a finite-state machine, the VFMS could in principle represent the semantics of procedures that include complex control flow structures such as backtracking or iteratively repeating subsequences of behavior. However, learning arbitrary FSM structures in a tractable manner requires posing set equivalence queries to the teacher that fall outside the scope of natural instruction. Strictly speaking, these limitations apply only to exact identification, and so heuristic methods, such as those developed for grammar induction, might enable the robot to learn a reasonable approximation of complex semantic structures; (3) By representing a sequence of qualitative states in the environment, the VFMS implicitly represents the manner in which a procedure is performed. Some procedures specify a goal without constraining the manner in which the goal is achieved. The VFMS learning methods start by assuming all aspects of manner are critical, and then selectively remove aspects of manner that vary across examples, such as coincidental relations or preliminary states. For goal-based procedures, learning the underlying goal is thus error-prone and time-consuming, as it requires "un-learning" nearly all of the assumed procedure structure. However, the alternate position, searching for goals directly and ignoring manner by default, presents a different set of issues, including the challenging problem of goal inference. A heterogeneous approach may prove fruitful: The learning problem could be simplified through the development of heuristics that could identify the class of procedures (goal-based, manner-based, causal, etc.) suggested by the variance in a given set of examples, and learning a model that suits that particular class.

7.7 References

- [Fasel *et al.*, 2010] I. R. Fasel, A. Wilt, N. Mafi and C. T. Morrison. (2010). Intrinsically Motivated Information Foraging. In *Proceedings of the International Conference on Development and Learning (ICDL)*.
- [Kaochar *et al.*, 2011a] T. Kaochar, R. T. Peralta, C. T. Morrison, I. R. Fasel, T. J. Walsh and P. R. Cohen. (2011). Towards Understanding How Humans Teach Robots. In *Proceedings of the 19th Conference on user Modeling, Adaptation and Personalization (UMAP 2011)*.
- [Kaochar *et al.*, 2011b] T. Kaochar, R. T. Peralta, C. T. Morrison, T. J. Walsh, I. R. Fasel, S. Beon, A. Tran, J. Wright and P. R. Cohen. (2011). Human Natural Instruction of a Simulated Electronic Student. In *Working Notes of the AAAI Spring Symposium Series Workshop: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration*.
- [Morrison *et al.*, 2009] C. T. Morrison, D. Bryce, I. R. Fasel and A. Rebguns. (2009). Augmenting Instructable Computing with Planning Technology. In *Proceedings of the ICAPS'09 Workshop on the International Competition for Knowledge Engineering in Planning and Scheduling (CKEPS)*.
- [Morrison *et al.*, 2010] C. T. Morrison, I. R. Fasel and P. R. Cohen. (2010). Fall 2009 Human-Instructable Computing Wizard of OZ Studies. *University of Arizona Computer Science Technical Report TR10-05*.
- [Peralta *et al.*, 2011a] R. T. Peralta, T. Kaochar, I. R. Fasel, C. T. Morrison, T. J. Walsh and P. R. Cohen. (2011). Challenges to Decoding the Intention Behind Natural Instruction. In *Proceedings of the 20th IEEE International Symposium on Robot and Human Interactive Communication (Ro-Man 2011)*.
- [Peralta *et al.*, 2011b] R. T. Peralta, T. Kaochar, I. R. Fasel, C. T. Morrison, T. J. Walsh and P. R. Cohen. (2011). Challenges to Decoding the Intention Behind Natural Instruction. *Workshop on Agents Learning Interactively from Human Teachers (ALIHT)*, at IJCAI 2011.
- [Walsh *et al.*, 2011] T. J. Walsh, D. Hewlett and C. T. Morrison. (2011). Blending Autonomous and Apprenticeship Learning. In *Proceedings of the Twenty-Fifth Annual Conference on Neural Information Processing Systems (NIPS 2011)*.

8. CONTROL OF LEARNING PROCESSES (UTAH STATE UNIVERSITY)

8.1 Task Objectives

- Objective 1: Design and Implement a MABLE control module to select learning targets and learning strategies based on Natural Instruction Methods and learning target type.
- Objective 2: Design the iBL intention elicitation module that represents knowledge about the learning target and derives Knowledge Acquisition Goals to ensure that the learned code modification can be implemented as a change to the performance system.

8.2 Technical Problems

- Objective 1: One of the main technical problems with designing a control module for MABLE is the definition of an upper ontology that can express knowledge about concepts, the curricula, the capabilities of learning strategies, and the status of learning the concepts. The second technical problem is performing the necessary reasoning to decide which concepts to learn with what curricula and which learning strategies.
- Objective 2: The iBL intention elicitation module faces many of the same challenges as the MABLE control module, but requires additional fine-grained reasoning about concepts and the generation of knowledge acquisition goals (i.e., questions). Namely, the intention elicitation requires defining the representation of an intention, the criteria for when it is complete, and the approach to acquiring the missing information that makes the intention incomplete.

8.3 General Methodology

- Objective 1: The approach taken by the MABLE control module is to divide reasoning about learning into two sub-modules: a concept hierarchy manager, and a learning strategy planner. The concept hierarchy manager tracks the hierarchy of learned concepts and how they are composed of other concepts (through Bootstrapping). Within the hierarchy, information about the quality of learned concepts is tracked to identify new learning targets. In this sense, the concept hierarchy manager is a type of goal monitor. The learning goals and curricula are input for the learning strategy planner, which sequences learning strategies to achieve the learning goals. The planner attempts to generate learning plans that are robust to receiving unpredictable results from the learning strategies.
- Objective 2: The approach taken by the iBL intention elicitation module is to represent the user's intention by a set of constrained attribute time intervals, which denote intended performance system actions or state variables in the ultimately modified code. The process of intention elicitation resembles mixed-initiative planning, where the module identifies missing aspects of the intention (as defined by the completeness criteria), generating possible options for the missing information from a partial domain model or allowing the specification of new options, and then integrating the selected options into the intention to continue the process.

8.4 Technical Results

- Objective 1: The MABLE control module was used to successfully pass go-no-go evaluations in each of the three Bootstrapped Learning program phases. The techniques developed in the learning strategy planner to construct robust plans have been integrated into a domain-independent automated planner, called DeFault.
- Objective 2: The iBL intention elicitation module has been successfully implemented and demonstrated on a number of UAV ISR mission tasks where the ISR procedures require modification.

8.5 Important Findings and Conclusions

- Objective 1: The MABLE control module and learning strategy planner have been used as the basis for findings discussed in the following publications:
Daniel Bryce and Christopher Weber, "Planning and Acting in Incomplete Domains", ICAPS, 2011.
Roger Mailler, Daniel Bryce, Jiaying Shen, and Ciaran O'Reilly, "MABLE: A Modular Architecture For Bootstrapped Learning", AAMAS, 2009.
- Objective 2: The iBL intention elicitation module has been used for the findings discussed in the following publication:
Daniel Bryce and Michael Freed, "Interactive Bootstrapped Learning for End-User Programming", AAAI Workshop on Smarter Living: The Conquest of Complexity, 2011

8.6 Significant Developments

N/A

8.7 Implications for Future Research

- Objective 1: The DeFault planner is the first planner to construct plans that are robust to missing or incomplete domain models, and provides an avenue toward more capable Bootstrapped Learning systems.
- Objective 2: The iBL intention elicitation module serves as a reference implementation of a tool for mixed-initiative planning and knowledge acquisition that can be used in future Bootstrapped Learning systems.

9. CONTROL OF LEARNING – PART 2 (UNIVERSITY OF MASSACHUSETTS)

9.1 Task Objectives

The objective of DARPA's *Bootstrapped Learning* (BL) program is the creation of trainable computing systems that enable behaviors to be taught rather than programmed. More specifically, this objective involves building an *electronic student* that is an *active* learner, continually posing questions and hypotheses to itself and actively looking for answers to those questions and for evidence that confirm or disconfirm those hypotheses. A diverse collection of *curricula* expressed in a standardized format is used to instruct the electronic student. When the electronic student is given a new concept to learn, it is told which prerequisite concepts are to be used in learning the concept.⁴ Concept instruction (in the form of training instances or other

⁴ Therefore, the electronic student does not need to identify the concepts to be learned or the precedence relationship between them, which is a major focus of work such as that of Datta and Kibler (Datta and Kibler 1993).

teaching modalities) is conveyed to the learning system in the formal instruction language (Mailler et al. 2009).

The overarching objective for the UMass effort was assisting SRI in building the control module and the knowledge repository for SRI’s MABLE electronic student.⁵

9.2 Technical Problems

The principal challenge faced in the UMass portion of the BL effort stemmed from the very small amount of training and testing given to the electronic student. Unlike traditional data-intensive machine learning methods, the focus of the BL program was demonstrating the feasibility of training the computing system without the use of extensive training curricula. MABLE is provided with very limited instruction (such as a few training instances) for each concept. This causes even the best of learners to make assumptions about the concept being taught (Hoffbeck and Landgrebe 1996; Zhou and Huang 2001; Zhang, Lin, and Zhang 2001). Those assumptions can be incorrect, forcing a learner to either hypothesize a single candidate for the concept being taught or avoid premature elimination of reasonable assumptions by proposing multiple candidates. Each of these proposed *concept candidates* stems from different assumptions made that are consistent with the instruction. As a very simple example, consider the blocks-world concept of a stack. A few labeled examples of stack and non-stack block arrangements are provided—all of them using blue blocks. A learner could assume that block color does not matter, that a stack must consist of blocks of the same color, or that stacks must be blue. Since a learner cannot eliminate any of these assumptions given the instruction, it could choose to propose a color-unimportant candidate, a monochrome candidate, and a blue-only candidate.

A secondary challenge in MABLE involved integrating and managing the use of diverse *learning strategies* (methods and algorithms).

9.3 General Methodology

We began our Phase I effort by surveying the literature for state-of-the-art control techniques suited for many-learner settings. The goal of this survey was to inform the design and implementation of control mechanisms for MABLE’s Control Integration Module (CIM). The job of the CIM is to choose between alternative, competing, concept interpretations (“concept candidates”) for answering imperatives and to decide what concepts should be relearned when failures occur. Throughout Phase I we worked with Daniel Bryce (Utah State University) and Roger Mailer (SRI and then Tulsa University) in designing, implementing, testing, and analyzing the initial MABLE CIM.

Much of the focus for CIM during Phase I was on selecting which learning strategies (“learners”) should be applied to each type of concept lesson. MABLE was designed to enable an instructor use any of a number of teaching methods to convey concepts, and each of these *natural*

⁵ The original name for the SRI electronic student was PLATO (Phased Learning Using Active Thought and Observation), but the system name was changed to MABLE (Modular Architecture for Bootstrapped Learning Experiments) to avoid potential trademark infringement.

instruction methods could invoke one or more of MABLE’s learning strategies to learn the concept being taught (the *target* concept). We hoped that selecting the most appropriate learner(s) would increase the likelihood of learning the concept correctly.

In conjunction with our Phase I work, the UMass effort looked at research and practice in layered learning, multi-level learning, and multi-concept learning, as well as hierarchical reinforcement learning, searching for prior work with similar learning objectives and challenges as those of MABLE. What makes MABLE different from all previously researched learning techniques is the emphasis on getting each concept (or layer, or level) right. We soon recognized that MABLE presented unexplored control challenges in managing multiple learned concepts; concepts generated due to the limited training and test (examination) provided for each concept. Learning strategies in MABLE generate competing candidates for a concept because they are not given sufficient training and test examples to uniquely identify a correct candidate. (Each of these concept candidates may be able to pass the tests for that target concept but still be incorrect when used in other settings.) Only later, when the concept is used in higher-level learning, will potentially differentiating behaviors of a concept’s candidates become apparent. In some cases, a “correct” candidate may not even be among the originally generated concept candidates.

As a branch effort in Phase 2, the UMass effort began designing promising monitoring and control techniques for managing multiple candidates for learned concepts with the objective of enabling MABLE to eventually get every learned concept correct—not just passing the tests for each target concept or correctly learning only the highest-level concepts. We also identified challenges that must be addressed in order to avoid undesirable *compensatory learning*, where the learning of higher-level concepts attempts to adjust for incorrectly learned supporting (prerequisite) concepts. We believed that the control strategy in MABLE should identify those candidates that maintain correct results as a concept is used in higher-level concepts and, when needed, direct additional learning activities on a concept that appears to be missing a candidate suitable for higher-level concept use. Pursuing this line of research consumed most of our Phase 2 and 3 research effort. The resulting concept management and learner-invocation strategy leverages learning failure as a central aspect of hierarchical (bootstrapped) concept learning where there is limited instruction and testing.

9.4 Technical Results

MABLE is an example of an automated system for hierarchical concept learning (Rivest and Sloan 1994; Nguyen et al. 2004) that uses one or more learning algorithms (“learners”) that learn higher-level *target* concepts using *prerequisite* concepts that were learned previously or provided to the system as primitive, base-knowledge concepts. In addition to the limited instruction provided for each concept, MABLE is given a *concept mastery exam* consisting of small number of test “questions” (as few as one).⁶ A mastery exam involves answering a question or performing a skill that demonstrates the concept has been learned. For example a mastery exam

⁶ Discussion of exam “questions” and “answers” to them are used loosely in this report. For example, a “question” might be a command to perform an action in order to demonstrate mastery of a target concept and its “answer” would be the sequence of primitive low-level actions to be executed.

question could be “Is this a stack?” Only a single candidate’s answers to the exam questions for the target concept can be submitted by MABLE for grading. This one-time-only-per-concept grading is the only supervised testing that is available. Therefore, we recognized that it is important that MABLE use grading to the best advantage.

The concept exam can be used to eliminate some of the candidates generated for the concept. If an answer is graded as incorrect, all candidates that produce that incorrect answer can be discarded. Answers graded as correct, on the other hand, only eliminate candidates if the question is known to have a single correct answer (based on the concept type or background knowledge), in which case all candidates that answered differently can be discarded.

Another constraint on MABLE is that its learners can use only the provided instruction for the concept. So, the graded answers and other candidate-use information (to be discussed later) cannot be given to a learner in an attempt to improve the candidates that it generates. The only change allowed is to invoke learners with the same instruction (lesson), but with different prerequisite concept candidates

Concept candidates that are not eliminated by the concept exam can still cause problems when used to learn higher level target concepts. Learners can fail to learn given a prerequisite candidate, producing no candidates for the target concept. Or, target candidates can be learned, but all of them are eliminated by the target concept’s exam. Such prerequisite candidate *use failure* can be addressed only by learning the target concept again using a different candidate for one or more of the target concept’s (direct or indirect) prerequisite concepts. An important insight in our work is that *use failures provide an indirect avenue for eliminating problematic candidates*. In other words, use failures (especially those encountered early in the use of a concept candidate) should be appreciated, rather than disparaged.

Some learners are able to “fix” prerequisite-candidate problems by learning to compensate for a prerequisite candidate’s undesired behavior. For example, suppose a learned candidate for the target concept of multiplying two integers incorrectly produces a negative result if both integers are negative, but operates correctly otherwise. A learner of a higher-level target concept using this incorrect candidate when learning a higher-level arithmetic operation could compensate for this error by changing the sign of the result produced by the prerequisite candidate when both numbers supplied to it are negative. Such *compensatory learning* can produce a short-term benefit of learning the target concept and even passing the concept exam. Unfortunately, compensatory learning masks use failures, allowing problematic candidates to be used longer than they would be otherwise. Because compensatory learning is internal to a learner, it cannot be prevented by any candidate-management strategy. An important consideration in the design of our concept-management strategy, was limiting the effect of compensatory learning by using candidates broadly, making it difficult for concept learners to compensate for problematic prerequisite candidates consistently.

The Concept Candidate Management (CCM) strategy (Zafar and Corkill, 2011) that we developed during Phase 2 and 3 of this effort controls the learning of a hierarchy of concepts in settings (like MABLE) where multiple candidates are proposed by learners for each concept. The objective is to maintain a single selected candidate for each concept and that this candidate does not have any observed failures in answering concept exam questions or when the candidate is used as a prerequisite in high-level concept learning. Exploring all possible combinations of

prerequisite concepts, and the concept candidates generated from them, is expensive. CCM limits this exploration without overlooking any viable candidates. The CCM strategy is based on three key ideas: 1) limiting the number of candidates generated by operating with a single selected candidate for each prerequisite concept at any time, 2) using failures in learning high-level concepts to select a different candidate for direct or indirect prerequisite concepts, and 3) using differences observed as candidates are used to guide candidate selection.

CCM uses observed behavioral differences to guide the selection of candidates when answering concept exam questions and in selecting prerequisites for learning higher-level concepts. Every learned candidate is given the concept exam, and the ungraded answers are used to place every candidate into an equivalence set where the observed behavior of all candidates is identical.

Further differentiation can be made as candidates are used as a prerequisite of higher-level concepts. Although we would have liked to record all uses, in MABLE we only had access to the inputs and outputs of prerequisite candidates calls when a high-level concept candidate answered an exam question. Each input given to the candidate was recorded as an unsupervised test question for the prerequisite concept, while the output was recorded as the candidate's answer. In addition, each newly recorded test question is answered by all other candidates of the prerequisite concept, and those answers are used to differentiate them further.

9.5 Leveraging learning failures

Our CCM strategy leverages two kinds of use failures: exam failures and prerequisite use failures.

Exam failures When a concept is first learned, CCM selects a candidate at random for grading from the equivalence set with the most number of candidates. By selecting from the largest set, CCM maximizes the number of candidates that will be eliminated if an answer is graded as incorrect. If a submitted answer is graded as incorrect, the entire equivalence set containing the selected candidate is eliminated, and an alternate candidate is selected randomly from one of the remaining equivalence sets.

Prerequisite-use failures When learning a target concept, the currently selected candidate for each of the target's prerequisite concepts is used by the learners. When a prerequisite use failure occurs, CCM goes into failure-resolution mode, changing one or more of the selected (direct or indirect) prerequisite candidates of the target concept being learned in an attempt to resolve the failure. Whenever a selected candidate is replaced with an alternate, all concept learning that used the previously selected prerequisite-concept candidate (directly and indirectly) will be discarded. Concepts that used those discarded candidates must be relearned, by reapplying the each concept's lesson with the learners using the modified set of prerequisite-concept candidates.

In failure-resolution mode, CCM first chooses the prerequisite concept in which to change the selected candidate. CCM prefers to choose the highest-level prerequisite concept that has previously unused candidates. Once all alternate candidate combinations have been tried at a level, alternate candidate combinations are tried for lower-level prerequisite concepts. This is for two reasons: 1) less work has been done with candidates at higher levels so fewer concepts have to be learned again and 2) the conjecture that the more a candidate has been used, the less likely it will cause a prerequisite use failure. If multiple prerequisite concepts are at the same level, ties are broken by preferring the prerequisite concept with the most number of equivalence sets

containing no previously used candidates, since it provides more choices of alternate candidates to select from that are likely to behave differently when used. If ties still persist, one of them is chosen randomly.

When selecting an alternate candidate for a prerequisite concept, CCM selects one at random from a different equivalence set than any previously selected candidate. If all equivalence sets have at least one previously used candidate, the set with the most number of candidates that have not been determined to be unsuccessful (defined shortly) is selected first, and one of the previously unused candidates in that set is selected randomly. If all candidates have been previously used, one of those candidates is selected randomly. When a use failure occurs, CCM records the *combination* of prerequisite candidates as previously used and will not use that particular combination again. A prerequisite candidate can still be used, however, with different combinations of other prerequisite candidates.⁷

Once an alternate candidate is selected, higher-level concepts that were learned using the previously selected prerequisite candidate are learned again, potentially resulting in additional concept exam or prerequisite use failures. Those are resolved in the same way as the original use failure but within the initial failure-resolution mode. Once all use failures have been resolved, CCM exits failure-resolution mode and all candidates that do not use the current selected candidates as prerequisite are discarded.

If the use failure is resolved, all previously used combinations are determined to be *unsuccessful* and never used again. A concept candidate is unsuccessful based on use failures if all combinations of direct or indirect prerequisite candidates have been determined unsuccessful.

If CCM is unable to resolve a use failure, the concept that triggered the failure-resolution mode is determined to be a *failed concept* because no combination of prerequisites can be used to successfully learn the concept. The failed concept is removed from the concept hierarchy, and CCM continues learning as many other concepts as possible. In this case, CCM reverts to the same state of selected candidates from before entering failure-resolution mode (continuing as if the use failure never occurred).

9.6 Important Findings and Conclusions

CCM is a highly effective strategy for managing the generation and selection of concept candidates using information gained as the candidates are used in learning higher-level concepts. By differentiating candidates based on their behavior and by associating unsuccessful learning of higher level concepts with the prerequisite candidates used to learn them, CCM can thoughtfully select alternative prerequisite candidates and control when unsuccessfully learned concepts have to be relearned. Using the CCM strategy, MABLE learned concepts that were not successfully learned otherwise—without any additional training or testing and without any changes to learning algorithms.

⁷ If a concept candidate is reselected, previously learned high-level concept candidates that were learned using it and the other selected prerequisite candidates do not need to be learned again.

We compared the performance of our CCM strategy with three alternative strategies:⁸ 1) a baseline strategy that does not change the selected candidate once higher-level learning begins; 2) a “learner selects candidate” (LSC) strategy where each learner is allowed to select the candidate for each prerequisite concept that it uses when learning a concept; and 3) CCM using no candidate differentiation (CCMno) beyond the initial elimination of candidates based on the concept exam.

The baseline strategy The baseline strategy obtains answers to the mastery exam from each candidate generated for the concept by the learners. It then selects one candidate at random and submits its answers for grading, and the graded answers are recorded. If that candidate’s answers pass the exam, it remains the selected candidate for the concept thereafter. If the candidate fails the exam, the selected candidate and all others whose answers were also incorrect are eliminated as unsuccessful. One of the remaining candidates, if any remain, is chosen at random to be the concept’s selected candidate thereafter. If all candidates are eliminated as incorrect, learning of the concept fails.

The LSC strategy The LSC strategy is an improved version of the control strategy originally used in MABLE. This original MABLE strategy eliminates only the concept candidate whose submitted answer fails the mastery exam. The improved LSC version eliminates all candidates whose answers to the exam are incorrect (identical to the exam-failure elimination used in the baseline strategy). Both LSC and the original MABLE strategy allow each learner to select any single candidate (from those that have not been eliminated by the concept mastery exam) for each prerequisite concept when learning a higher-level concept. The graded mastery exam answers for each prerequisite concept are also available to the learner. The learner creates a single candidate for the higher-level concept (using any combination of prerequisite candidates it deems appropriate), but other learners can select different prerequisite candidates in producing their candidate for the same higher-level target concept.

We analyzed the performance of each strategy in four learning domains in which MABLE was applied:

- ATF (Advanced Tactical Fighter), learning military formation strategies
- ISS (International Space Station), learning to simulate computer machine instructions
- ISS2, learning to simulate a larger and more complex set of computer machine instructions
- UAV (Unmanned Aerial Vehicle), learning to operate a UAV for monitoring suspect behavior

The detailed analysis results are presented in a technical report by Zafar and Corkill (Zafar and Corkill 2011), and we summarize them here:

⁸ We used the curricula, learners, and the control strategy from the 12/22/2010 version of MABLE for this evaluation.

Table 2. Concept Learning Performance

Domain	Concepts Learned Successfully/Unsuccessfully (Failed)			Runtime Factor	
	Baseline	LSC	CCM	LSC	CCM
ATF	25/8 (5)	35/3	38/0	1.2	1.9
ISS	28/7 (2)	31/6	32/5	1.1	1.8
ISS2	10/13 (13)	13/11 (12)	14/10 (12)	1.2	3.0
UAV	40/7 (4)	48/3	50/1	1.3	2.0

The number of concepts learned unsuccessfully (i.e., all candidates learned for the concept were unsuccessful) and the number of concepts where learning failed (no candidates could be generated by the learners) are also shown. (Both CCM and CCMno learn the same concepts successfully, so only the CCM values are shown in the table.) The learners in MABLE did a fairly good job of generating successful candidates in all domains except ISS2, where the learners failed to generate any candidates (even unsuccessful ones) for over one-third of the concepts (this generation-failure count is shown in parentheses in the table). The additional runtime required by LSC and CCM compared to the baseline strategy is also shown. The additional runtime required with the LSC strategy is due to the extra work performed by each learner in selecting the prerequisite concepts used in learning. The additional runtime required with CCM stems from the work that must be performed when a candidate that has been used as a prerequisite becomes unsuccessful. This additional runtime with CCM includes the effort spent exhaustively searching all combinations of prerequisite candidates (direct and indirect) when no learner is able to generate a successful candidate using any of them.

When the CCM strategy replaces a selected prerequisite candidate (direct or indirect), concepts that were learned using that unsuccessful candidate are learned again using each proposed replacement candidate. Figure 9 shows the total number of candidates that are selected by the baseline, CCM, and CCMno strategies and the number of times concept learning was performed. (The LCS strategy is not shown in Figure 9 because all candidate selection beyond that of the baseline strategy is performed internally by each learner and, as with the baseline strategy, concepts are not learned again.) In each learning domain, the CCM strategy performs nearly twice as many candidate selections as the baseline strategy (Figure 24(a)) and, due to relearning using newly selected prerequisite candidates, about four times as many concept learning invocations (Figure 24(b)). These totals exclude counts associated with the effort spent exhaustively searching all combinations of prerequisite candidates (direct and indirect) when no learner is able to generate a successful candidate using any of them. Figure 24 also shows that CCM is able to significantly reduce the amount of additional work, as compared with CCMno, by taking into account concept candidate behavior differences obtained as selected candidates are used as prerequisite candidates in higher-level concept tests.

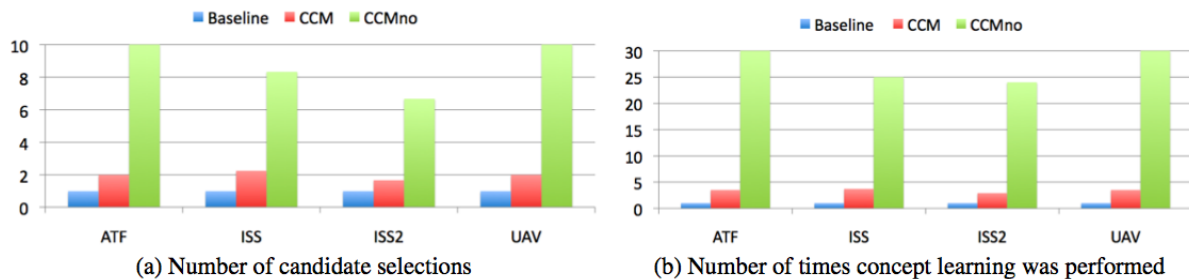


Figure 24. Additional Work Performed by CCM and CCMno Strategies (normalized to the baseline strategy).

9.7 Significant Developments

The main contribution of our BL research is the CCM strategy (discussed above) that manages multiple concept candidates and multiple learners effectively throughout open-ended hierarchical concept learning (where new target concepts may be supplied and taught to the system at any time in the future). The CCM strategy is important in settings like MABLE because multiple concept candidates are inherent given such limited instruction, no matter how much effort is applied in trying to improve the learners.

9.8 Implications for Future Research

The MABLE learners in our CCM experiments were used unmodified. (In our Phase 2 & 3 research, we treated these learners as true “black boxes.”) However, this approach limited exploration of several interesting research directions. First, the inputs and outputs of prerequisite candidates as they were used during concept learning were not available to the CCM strategy (only the inputs and outputs as prerequisite candidates were used in answering higher-level concept examination questions could be recorded). We would have liked to also record the inputs and outputs of prerequisite candidates as they were used by learners as they learned higher-level concepts. This additional source of candidate differentiation might improve CCM's ability to select successful alternative candidates. Second, each MABLE learner proposed only a *single* “best” candidate when invoked to learn a concept. If each MABLE learner had returned multiple candidates for a concept—especially when the learner's confidence among those candidates was close—concepts that were unlearnable (due to the lack of a suitable prerequisite candidate) might have been learned successfully given the additional candidates. Finally, if learners could annotate the candidates that they generate with their confidence in them, CCM could use those confidence values in candidate selection (replacing the random choices used in the absence of such information).

Another interesting, unexplored research direction is allowing a learner to use the inputs and outputs that CCM has recorded for a concept's candidates when the learner is invoked to learn the concept again. This information could be used by the learner as an additional source of unsupervised instruction stemming from how previously generated candidates were used in learning and testing higher-level concepts.

9.9 References

- Ali, K. M., and Pazzani, M. J. 1996. Error reduction through learning multiple descriptions. *Mach. Learn.* 24(3):173–202.
- Datta, P., and Kibler, D. 1993. Concept sharing: A means to improve multi-concept learning. In *Proc. of the 10th ICML*, 89–96.
- Hoffbeck, J. P., and Landgrebe, D. A. 1996. Covariance matrix estimation and classification with limited training data. *IEEE Trans. Pattern Anal. Mach. Intell.* 18:763–767.
- Mailler, R.; Bryce, D.; Shen, J.; and O'Reilly, C. 2009. MABLE: A framework for learning from natural instruction. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 393–400. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.
- Nguyen, S. H.; Bazan, J.; Skowron, A.; and Nguyen, H. S. 2004. Layered learning for concept synthesis. In Peters, J. F.; Skowron, A.; Grzymala-Busse, J. W.; Kostek, B.; Swiniarski, R. W.; and Szczuka, M. S., eds., *Transactions on Rough Sets I*, volume 3100 of *Lecture Notes in Computer Science*, 187–208. Springer Berlin / Heidelberg.
- Rivest, R. L., and Sloan, R. 1994. A formal model of hierarchical concept learning. *Inf. Comput.* 114:88–114.
- Zafar, H.; Corkill, D. D. 2011. Leveraging Failures to Enhance Hierarchical Concept Learning when Training and Testing are Limited. Computer Science Technical Report #2011-026, University of Massachusetts Amherst.
- Zhang, L.; Lin, F.; and Zhang, B. 2001. Support vector machine learning for image retrieval. In *Proc. IEEE Int. Conf. on Image Processing*, 721–724.
- Zhou, X. S., and Huang, T. S. 2001. Small sample learning during multimedia retrieval using biasmap. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 1:11.

10. SERVICE FOR CONCEPT INTERPRETATION (UNIVERSITY OF TULSA)

10.1 Task Objectives

The University of Tulsa (TU) became a subcontractor to SRI International on the DARPA Bootstrapped Learning program late in the first phase as a result of hiring Dr. Roger Mailler. Initially, TU was tasked with supporting the transition of the technical lead of the project from Dr. Mailler to Dr. Michael Freed. During the remainder of this phase, Dr. Mailler was involved in many of the technical details of the completion of the phase one system and in the interpretation and presentation of the results to DARPA.

As phase two planning began, TU's role in the program shifted to working with SRI to develop one of the core components of the system, the Concept Interpretation Monitor (CIM). The CIM

plays a critical role in determining the systems focus of attention and maintaining the current coherent set of concept fragments when complex lesson plans were being presented.

Within phase three of the program, TU began to work on a very specific problem that was identified during phase two. Namely, how can the CIM take fragments of knowledge and verify that they are consistent with one another. TU began working on an automated truth maintenance system based on a satisfiability (SAT) solver.

10.2 Technical Problems

Each phase of the program presented different technical challenges. In phase one, the key challenge was to complete a system that met the learning challenges of unknown problem domains. In phase two, the problem was to integrate and organize knowledge in the system to identify conceptual holes that needed to be filled in order for a lesson to be completely understood. In phase three, the challenge was to figure out how to develop a SAT solver that was capable of representing a wide array of knowledge and was responsive to new knowledge being added and incorrect knowledge being removed. It was particularly important to be able to find constraints that could collapse the solution space in order to promote conceptual coherence.

10.3 General Methodology

Our general methodology was to exploit known techniques and expand and adapt them to work when learning from natural instruction. The CIM was designed to merge the features of a version control system with a knowledge management system. The basic design involved tracking how fragments of knowledge are created, merged, and corrected by the various learning components in the MABLE system. To do this, target concepts were introduced to the system and were later matched up with concept interpretations. The target concepts formed hierarchies based on the laddering of BL curriculums. As concepts were paired with targets, new targets would emerge and that would be used to focus the attention of the learning modules. Sometimes, revisions to concepts were required and when this occurred, new branches in the search tree would become available. Several branches may actually lead to a complete concept and it was also the responsibility of the CIM to rapidly determine the correct one among the choices during student evaluation. The CIM turned out to be an important component in the success of the phase two student.

Following the phase two success, TU's attention turned to a fundamental problem in the CIM. Namely, when knowledge was refined with MABLE, the CIM abandoned entire branches of the revision graph. This forced the system to relearn concepts that may have only been affected slightly by the refinement. It was decided that the best way to fix this problem was to introduce an automated truth maintenance (ATM) system. With an ATM, concepts could be refined and added to the CIM and then checked for consistency with the existing knowledge. At the core of this module would be a constraint solver.

TU began implementing a state-of-the-art solver based on the forward-checking, conflict directed backjumping (FC-CBJ) algorithm (Prosser 1993). The FC-CBJ algorithm is a very efficient method for solving CSPs because it combines prospective (forward checking) and retrospective (conflict directed backjumping) techniques to rapidly prune branches in the search tree. The algorithm (see Figure 25) works by iteratively assigning values to variables using the fc-cbj-label

method (see Figure 26). During labeling, a variable is assigned a value from its currently valid domain. Once assigned, the algorithm forward checks neighboring variables and removes any of their domain elements that are invalidated by the assignment. If during forward checking, the assigned variable is determined to be inconsistent, then the next value is checked. If a consistent value is found, the method returns the index of the next variable to label. Otherwise, it sets the consistent flag to false and returns the current variables index.

```

procedure fcbj
  consistent  $\leftarrow$  true;
  status  $\leftarrow$  "unknown";
  i  $\leftarrow$  1;
  while (status = "unknown") do
    if consistent do
      i  $\leftarrow$  fc-cbj-label(i, consistent);
    else
      i  $\leftarrow$  fc-cbj-unlabel(i, consistent);
    if i > n do
      status  $\leftarrow$  "solution";
    else if i = 0 do
      status  $\leftarrow$  "impossible";
    end do;
  end fcbj;

```

Figure 25: FC-CBJ Algorithm

```

procedure fc-cbj-label(i, consistent)
  consistent  $\leftarrow$  false;
  for each val  $\in$  current-domain[i] do
    v[i]  $\leftarrow$  val;
    for j  $\leftarrow$  i + 1 to n do
      consistent  $\leftarrow$  check-forward(i, j);
      if !consistent do
        current-domain[i]  $\leftarrow$  current-domain[i] - val;
        undo-reductions(i);
        conf-set[i]  $\leftarrow$  conf-set[i]  $\cup$  past-fc[j-1];
        break;
      end do
    end do
    if consistent return i + 1 else return i;
  end fc-cbj-label;

```

Figure 26 FC-CBJ Labeling

When a consistent value cannot be found during labeling, the algorithm calls the fc-cbj-unlabel method (See Figure 27). This method essentially performs a backjump by first identifying the last variable in the search with a constraint violation and jumping to it. It also performs the needed bookkeeping to maintain the state of the search.

```

procedure fc-cbj-unlabel (i, consistent)
  h  $\leftarrow$  max_element(conf-set[i]  $\cup$  past-fc[i]);
  conf-set[h]  $\leftarrow$  (conf-set[h]  $\cup$  conf-set[i]  $\cup$  past-fc[i]) - h;
  for j  $\leftarrow$  i downto h + 1 do
    conf-set  $\leftarrow$   $\{\emptyset\}$ ;
    undo-reductions(j);
    update-current-domain(j);
  end do;
  undo-reduction(h);
  current-domain[h]  $\leftarrow$  current-domain[h] - v[h];
  consistent  $\leftarrow$  current-domain[h]  $\neq$  nil;
  return h;
end fc-cbj-unlabel;

```

Figure 27: FC-CBJ unlabeled

With an appropriate static solver in place, we began to investigate the impact of adding and removing elements of the constraint set. To do this we developed a test harness to measure the

impact that modifications have to the solution space of a constraint problem and ran experiments to measure the minimum distance between solutions that result from various dynamic changes. Because of the computation time needed to run these experiments, we developed the harness to run on the University of Oklahoma super computer (OSCER).

10.4 Technical Results

The result of our effort on the CIM in phase two was a working system that rarely, if ever, chose the wrong concept during the evaluation of the system. Being a combination of search and heuristic strategies, this was a great result.

Because of the magnitude of the problem, the results of our phase three experiments are still only partially complete. What we were able to determine is that at the critical junction between having many solutions and no solutions at all, that the solutions space rapidly collapses with solutions becoming quite separate from one another. On a number of problems in this critical, but desired region, there were significant distances between solutions when even small changes to the constraints were made. Overall, this is an encouraging result, because it indicates that the CIM within MABLE could indicate when it is likely that a solution to the problem is near by measuring aspects of the constraints without having to conduct a difficult search. It requires that during the incremental adding of constraints that consistency checks be performed, but it isn't until the threshold that a hard search needs to be conducted.

We are conducting further studies to exploit these findings.

10.5 Important Findings and Conclusions

At the conclusion of our efforts, we have discovered that as constraints are added to a ATM, that it eventually reaches a critical point where the solution space is expected to collapse onto a few, testable solutions. We also found that it isn't until this region that small changes in the constraints yield significant changes in the overall solution. For systems, like MABLE, this is good news because it indicates that in many instances, adding knowledge does not require extensive search to take place in order to maintain consistency. It also shows that simple measure may be enough to flag when a solution might be near.

10.6 Significant Developments

The two most significant developments were the CIM and the algorithms and test harnesses used to study the algorithms involved in knowledge management within MABLE. We made considerable progress in understanding how adding and refining knowledge in a knowledge base impacts the solution set. In the future, we hope to develop knowledge based metrics that can further this important study.

10.7 Implications for Future Research

The implications of these finding have significant impact on other problems that may be encountered in the future. They indicate that there is likely to be a much closer relationship between constraint satisfaction and machine learning particularly when it comes to measures of solution space and the impact that knowledge has on it.

We have already begun to exploit the new solver technology in our lab to improve upon some of our distributed problem solvers. These solvers are used to address a number of challenging problems facing the DoD including airspace management and satellite tracking.

11. A PROGRAMMING BY INSTRUCTION SYSTEM (SRI INTERNATIONAL)

11.1 Objectives

The iBL Workbench is an interactive system for programming by instruction. The term “workbench” emphasizes the goal of providing a work environment for modifying software behavior. iBL is not intended to showcase all technologies developed under the BL program but rather a subset that define a coherent use model and user experience. Key, high-level objectives of the iBL Workbench effort include: clear utility, demonstratability and extensibility.

Clear Utility. The iBL should exceed user expectations about how much control over software behavior can be achieved without help from software engineers. Changes to software should be achievable quickly in comparison to typical software revision processes.

Demonstratability. It should be possible to demonstrate iBL capabilities on demand using a single laptop computer. Demonstrations should encompass a range of distinct UAV-ISR scenarios, methods of instruction and code modification types. Demonstrations should cover all stages in the iBL use model (see Users and Use Model section below) or any selected subset.

Extensibility. Developers wishing to test/extend iBL capabilities should be able to acquire, install, run and learn the system quickly.

Comparison of iBL user experience to traditional programming

The iBL implements a form of Programming by Instruction (PbI). Informally, PbI is intended to mimic how people instruct other people. However, that is not likely to be achievable in a strong sense for the foreseeable future, so it is useful to identify what aspects of human instruction are important to capture. The iBL approach emphasizes aspects related to differences between formal knowledge requirements for programming vs. teaching. In particular, iBL is intended to impose...

1. Minimal need for users to know the performance system code base in advance or to examine it in the process of making a change. Example: users won't need to know whether a desired change in behavior requires adding a new procedure step or changing a parameter of an existing step.
2. Minimal need for users to address issues of integrating new code with pre-existing code. E.g. users won't directly identify required ordering constraints between new and old procedure steps.
3. Minimal need for users to reason about disruption to correct pre-existing code from newly integrated code. E.g. users won't directly specify cleanup actions that restore conditions violated by spliced in new code.

4. Reduced need for users to anticipate the full range of conditions in which newly added or altered code might be executed. E.g. the user won't need to independently decide to limit the scope of the change to compatible conditions.

Comparison of iBL to BL phase 2 evaluation system

The iBL Workbench is an attempt to package and mature BL technologies into a complete end-to-end system. A complete system can be seen as having three parts: (1) A user-facing capability for obtaining instruction content. This appears to require mixed-initiative dialog and an ability to combine different instructional methods. (2) A core learning capability that derives concepts reflecting teacher intent from instructional content plus background knowledge. (3) A code-facing capability for translating a learned concept into specific changes to a pre-existing code base that might be complex and tightly coupled.

The phase 2 BL system emphasized the second of these (core learning capabilities) in accord with the central research objective of the program: discovering how to adapt machine learning approaches to distill concepts from sparse but well-chosen and well-structured instructional inputs. Parts (1) and (3) were handled in a simplified way. For example, instructional dialog was largely non-interactive, allowing little flexibility for the student system to ask questions or otherwise influence instruction to address knowledge gaps. Similarly, code-facing issues were finessed by limiting programming language expressiveness and by limiting the kinds of changes allowed to the existing code base.

iBL is an attempt to address all three problem areas in order to deliver a complete PbI capability.

Users and Use Model

Users are assumed to be highly knowledgeable in the performance domain (UAV-ISR for present purposes) and generally more knowledgeable than original system developers regarding operating conditions, evolving mission needs and actual system capabilities. Users are also expected to be technically savvy, able to develop expertise with the iBL equivalent to that of a Microsoft Office power user. However, they are not expected to become familiar with the code base to be modified or to know how to program.

The iBL use model starts with some goal for altering system behavior or enhancing system performance. In the UAV-ISR domain, such goals are likely to originate with end users of the ISR information products (e.g. analysts, decision-makers) whose needs are not being met. The iBL user would then carry out the following steps:

1. Load or create a simulation scenario that produces goal-relevant conditions.
2. Run the simulation until these conditions occur. This sets a context for teaching. Typically, a user will watch the scenario and pause at an appropriate moment.
3. Start a new lesson and engage in dialog to specify instructional intent and core lesson content.
4. Resume the simulation, pausing to add additional instruction (for instance, to point out instructive examples) as needed. Teaching interactions for an "open" lesson may be non-contiguous and span multiple simulation trials. Interactions related to other lessons may be interleaved.

5. Periodically review a text summary of iBL's interpretation of the lesson to check if it reflects user intent. When satisfied, the user marks the lesson complete, causing a modification to the simulated system code base.
6. Run simulation and watch to see if the code modification has produced the desired effect. This may be done over many simulation trials to check the mod effect in different operating conditions.
7. Once the modification has been validated within the iBL workbench, it could then be exported for review by whatever vetting process the user's organization requires and eventually uploaded to the real system.

11.2 iBL Overview

Approach Summary

Teaching a system using the iBL takes the form of instructional dialog whose output is a *code transformation* (CT), a set of point modifications to performance software (PS) that together achieve teacher intent. CTs range from simple changes – e.g. modifying a single parameter value of a single procedure step – to arbitrarily complex modifications affecting and possibly refactoring multiple procedures. The current iBL approach generates CTs from a set of relatively simple code transform templates (CTTs). Dialog management and machine learning capabilities in the iBL can be viewed as support for selecting and specifying a CT template.

An iBL dialog consists of several phases beginning when the user presses the "New Lesson" button. First, a structured interaction driven by questions from the student is used to frame the lesson – i.e. to characterize teacher intent to the degree possible and set the stage for additional instructional content. The initial dialog may be adequate to select and full specify a CTT, potentially completing the lesson. Second, the user provides additional instructional content by one or more natural instruction methods (telling, example, feedback). Third, the user reviews an English language summary of what was learned and chooses to either accept (apply) the CT, continue the lesson or cancel.

Instructional dialogs are a sequence of dialog episodes (DEs), segments of teacher-student interaction in which some defined progress is made towards CT specification. The iBL approach defines a set of DE types, each associated with a knowledge acquisition goal (KAG) and one more or more methods for achieving the goal. KAGs fall into three general categories based on what sort of uncertainty they aim to reduce: **what** alternative behavior (or new concept) is desired, **when** the new behavior should apply; and **how** the change should manifest as modified code. Achieving a KAG reduces the space of hypotheses about what CT is needed, either by narrowing down which CTT will be used or narrowing down one or parameters of the selected CTT.

Dialog Episode (DE) methods for achieving a KAG include both questions from student to teacher and teacher-initiated actions to inform the student. The first phase of dialog (framing) consists mainly of questions from the student. These are meant to elicit as much as possible from the teacher about intent (satisfying what and when KAGs) and to exploit their domain knowledge to fill in knowledge gaps the iBL needs to realize this intent as a code mod (how KAGs). The second phase (instructional content by natural instruction methods) consists mainly of teacher-initiated actions. A dialog sequencing component in iBL figures out how to string DE

methods together to make steady progress towards fully instantiating a CT. Early versions of iBL will provide limited sequencing flexibility.

Example Scenario

Here we describe the general flow of iBL dialog using the *iBL_demo_2* scenario as an illustrative example. The scenario starts with a user noting that video imagery collected by the UAV during a circular observation pass is at too oblique an angle to be useful. The user engages in dialog leading to specification of one of two possible CTTs: *modify-step-goal-parameter* or *add-achieve-precondition-step*.

In the first scenario variant, we assume that the procedure for observing a target already includes a step to set altitude, but the specified altitude is too high. Teaching results in modifying the appropriate step parameter. In the second, we assume that the code does not include a step for setting an altitude. Instead a cruise altitude is established prior to the observation pass and remains at that level. In this case, teaching results in several additions to the observe-hot-suspicious-target procedure: (1) a step to descend, (2) ordering constraints between the new step and pre-existing steps and (3) a "cleanup" step to restore cruise altitude.

For reference, here is the relevant part of the PS procedural call stack:

```
> fly-isr-mission (includes takeoff, ascent to cruise alt, return to base,...)
>> fly-data-collection-itinerary (planned path plus contingencies)
>>> observe-hot-suspicious-target (a contingency handler)
>>>> make-observation-pass (circular path after iBL_demo_1)
>>>>> fly-circular-path
```

ANNOTATED DIALOG

Visible at start (see 11.3.1, 11.5.2): sim main view, camera view, narration view (showing high level event trace), teaching UI, sim control bar. The teaching UI includes a transcript of teacher and student “utterances” and elements for conducting the dialog. The user can “preview” what has been learned in a lesson so far and can choose to apply the lesson code change when done.

Setup: User runs simulation to validate the change made in *iBL_demo_1* and notices an excessively oblique view of the target in camera view. User pauses the sim, clicks "New Lesson" button and selects the option to "give feedback on undesirable current condition."

iBL (student): What do you want to give feedback about? Click on a visible object or object on the world object list and then pick an attribute of the object. (*user clicks on “this-image” icon in camera view causing inspect view of the image object to appear and then on vertical-view-angle attribute.*

User (teacher): vertical-view-angle of camera-image-1

iBL: Should a better vertical view angle be achieved by changing altitude or target-distance? (*user clicks on “altitude”*)

User: altitude.⁹

⁹ This step identifies an attribute the PS can directly control to influence the feedback attribute (e.g. view-angle). Candidates are generated by analysis (possibly recursive) of the function used to compute the attribute: view-angle =

iBL: When should altitude have been different to achieve a more desirable vertical-view-angle? Select start and end times from events listed in the narration view.

User: From start of the activity-23 (make-observation-pass) to the end of activity-23 (make-observation-pass).^{10,11}

iBL: The current altitude was set to *cruise-altitude(location)* 12 minutes before the timepoints you just selected. Which of the following applies?

- (a) Altitude should be changed for a good view angle then changed back to cruise altitude.
- (b) Altitude should be changed for a good view angle and not changed back.
- (c) Altitude should not have been set to cruise altitude but to make a goof view angle instead.
- (d) Cruise altitude should be maintained, not changed for a better view-angle. (abort lesson).

User: (a) changed and then changed back.¹²

iBL: In what conditions should altitude be adjusted to achieve a better vertical view angle? Option (a) is more conservative.

- (a) When performing make-observation-pass in order to observe-hot-suspicious-target?
- (b) When performing make-observation-pass regardless of why?

User: (a) in order to observe-hot-suspicious-target

iBL: Should altitude be adjusted for a better view angle always in those conditions or sometimes?

User: Always.

iBL: I know of one way to control altitude which is *set-vnav-altitude-restriction(altitude, location)*. Is it ok to use this or should we define an alternative method?

User: OK to use this.

iBL: Will *set-vnav-altitude-restriction* achieve the desired altitude immediately or gradually?"

f(altitude, lat-distance).

¹⁰ This establishes three candidates for where to modify the existing code base: (1) In observe-hot-suspicious-target, the procedure that called make-observation-pass as a step (or in some other procedural "ancestor" in its dynamic scope); (2) In the procedure make-observation-pass itself; (3) wherever altitude was last set (to cruise altitude). Subsequent dialog selects which based on the desired scope (applicability) of the desired change with the first option most restrictive and the last one least.

¹¹ This scoping step could reference environment events as well as internal events such as task start/end points.

¹² This excludes the third candidate for where to modify the code (see footnote 2) and sets up a future dialog goal of determining how/when to reinstate cruise altitude.

User: gradually.¹³

iBL: Will additional action be needed to keep altitude at the desired value once it is achieved or will it stay stable while performing make-observation-pass?

User: stable.¹⁴

iBL: Should additional action be taken if altitude deviates from the target value while performing make-observation-pass or not?"

User: not.¹⁵

iBL: What altitude should be achieved before starting make-observation pass when performing the activity observe-hot-suspicious-target? You can tell altitude directly or give feedback and examples.

User: tell.

iBL: Should the same altitude be selected in all conditions or should it vary depending on the situation?

User: same. 2000 ft. *(text entry field with units combo box appears after user selects "same")*¹⁶

iBL: Earlier you said that after the completing make-observation-pass the value of altitude should be restored to current-cruise-altitude(). Should this be restored immediately after or is it better to wait until other steps of observe-hot-suspicious-target are complete?

User: immediately.

iBL: Is it better to restore altitude to the previous value of current-cruise-altitude (i.e. its value just prior to being changed by set-vnav-altitude-restriction) or to use whatever the current value of current-cruise-altitude?

User: current.

iBL: No more questions for now. You can declare the lesson complete or elaborate on it.

Completion: The CT template for this code mod is now fully specified and thus eligible to be applied as a code transform. This is indicated in the lesson status window. The user examines an English language paraphrase of the CT and decides whether the lesson should be considered complete. If satisfied with the iBL lesson summary, the user presses the OK button to apply the

¹³ This is an example of filling in the iBL model of the PS code base in order to determine *how* to instantiate a code mod. In particular, it helps determine whether a temporal delay/constraint is needed between the added descent step and the start of make-observation-pass.

¹⁴ This determines whether it is sufficient to insert a precondition-achievement step or whether a maintenance step (enforce invariant) is also needed. Either way, the action model is annotated to indicate that altitude is "protected" while make-observation-pass is active, preventing subsequent learning from overriding this without first querying the teacher.

¹⁵ At this point, the iBL can determine that the CTT for inserting a procedure step will be used, that the step will invoke set-vnav-altitude-restriction with some not yet specified altitude, and that it will be inserted along with ordering and timing constraints to run prior to the make-observation-pass step of the procedure observe-hot-suspicious-target.

¹⁶ "Same" is selected for simplicity. Realistically, altitude should be a function of minimum lateral distance from target (which can vary due to airspeed, terrain obstacles, etc). Direct entry of the numeric value is also simplistic. Users would more likely want to express intent (a desired view-angle) and have altitude inferred.

CT. The lesson status window indicates that this lesson is now closed. The user may not be sure if the lesson is adequate as-is and may choose to apply it to see the effect in simulation. If needed, the lesson can be re-opened for further refinement.

Alternately, the user may know that the lesson needs refinement. This is where student-initiated dialog ends and where teacher-initiated instruction based on natural instruction methods (NIMs) developed in BL phases 1 and 2 of BL take over¹⁷. For example, the user may realize that 2000 ft. will not always be the correct altitude to observe at, nor will any fixed value. The best altitude depends on geography and on minimum lateral distance from the observation target (which in turn varies with many other factors). The user may find it much easier to specify altitude by example in a range of situations rather than try to formally specify a function. Similarly, the right way to reach the goal altitude may be more complicated than simply entering a new value into VNAV due to operational rules and situational factors. The user may choose to define the right descent procedure by telling (if they can recall all the relevant steps and rules) or by feedback (if not).

System Components

Major iBL components are depicted in Figure 28. Note that the diagram is not intended to show actual system architecture, but is instead intended as a guide to major functions/services, representations and dependencies. One important difference is the use of a general message-passing framework to mediate both data interchange and control; thus, none of the depicted connection arrows represent direct communication pathways.

The system is divided into three general component layers: application, simulation and teaching. The application layer covers all general iBL services including the UI management, overall system management and software infrastructure. The simulation layer includes the simulation component itself (X-Plane), all simulation object states and controllers including the PS (code eligible to be modified by instruction) and instrumentation to support logging, learning by noticing and instructional dialog. The teaching layer includes lesson status representations, student-teacher dialog management functions and all core BL capabilities for learning from instructional content and translating what is learned into code modifications.

¹⁷ The other central role for NIMs in the iBL framework is to create new building-block concepts including procedures and procedure elements (functions, predicates,...). The user can click the new-lesson button at any time to initiate such lessons, including as a digression from an ongoing lesson.

This overviews key functions, representations and dependencies. Not to be treated as a system architecture



- bg background
- CT code transform
- CTT code transform template
- DEO dialog episode operator
- KA knowledge acquisition
- iBL interactive Bootstrapped Learning system
- ILP inductive logic programming
- LbN learning by noticing
- NIM natural instruction method
- PS performance system
- RAP reactive action package

11.3 Application Layer

The application layer provides general infrastructure and services for simulation and teaching layer tools. It includes: UI, user workflow management, project file management, system mode control, component integration, data transport, data management, initialization and installation.

UI

The application layer manages the user interface components associated with all layers. On startup, the user sees a layout as depicted in Figure 29. Primary application and simulation layer controls are provided on the control bar widget (Figure 30). Situation awareness information needed to set lesson context and validate lesson impact (steps 2 and 6 of the iBL use model) are provided by main pane views and the Narration View. The main pane can be switched between a Plan View Display (mission context), Flight View (tactical context) and Camera View (sensor context). The Teaching Interface provides lesson management controls (creating and switching) and a set of instructional support elements starting with the primary dialog element.

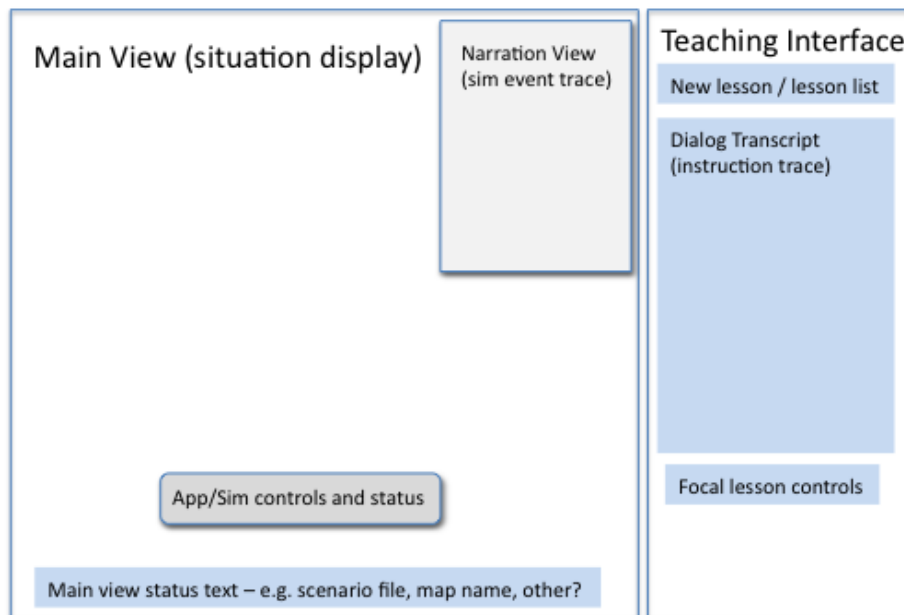


Figure 29. UI Layout

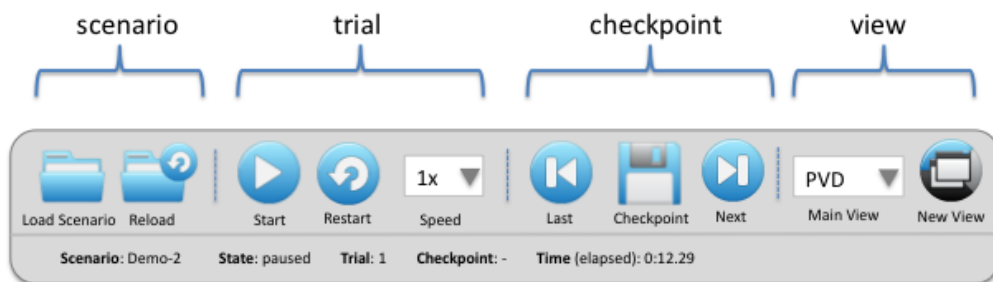


Figure 30. Control Bar

Scenario Definition Files and Ingredients

A scenario definition file (SDF) results from the user setting up a simulation run or series of runs to produce a teaching context. A SDF should include the following:

- Imports of scenario objects from external libraries including both native simulator libraries (e.g. X-Plane aircraft, maps, autopilot) and custom libraries for mission environments (aircraft sensors, ground objects).
- Imports of behavior descriptions for all active sim objects. These will typically be RAP representations with one RAP per file.
- Initial values for all non-varying object attributes (including world attributes such as time of day).
- If there are multiple simulation trials, a set of independent variables corresponding to initial sim object attribute values, a method for selecting IV values for the next trial and a list of dependent values to log separately.
- An initialization function that instantiates all objects with initial values at the start of a sim trial (or stored values if loading from a checkpoint).

Future iBL versions may include scenario creation tools that automatically generate SDFs and support (lib) files.

Checkpoints

Checkpoints are saved simulation states that can be reloaded as an alternative to running a simulation trial from the initial state defined by an SDF and trial settings. Because some simulation objects are likely to have complex internal state (especially the PS) and functions to support restoring state upon checkpoint load are likely to be brittle, early versions of the iBL may not support checkpointing.

Version Control

Here we refer, not to version control for the iBL code base, but to a version control system component within the iBL to manage key scenario files. In particular, version control will be used for performance system files modified by instruction. This component will maintain prior code versions to manage rollbacks and allow users to reopen closed lessons to remedy teaching and learning errors. Early versions of the iBL may not include a PS version control component.

11.4 Simulation Layer

The simulation layer includes all components needed to generate domain-level context for instruction.

X-Plane

X-Plane v9.5 was selected for UAV-ISR simulation. It is a supported commercial product available from Laminar Research for \$29.00 (<http://www.x-plane.com/>) and runs on OS X, Windows and Linux.

Simulation ingredients can be acquired from the Downloads page on the X-Plane site including maps/terrain, aircraft, autopilots, sensors, ground entities and visualization tools. Members of the X-Plane user community have made additional model elements available on other sites (some free and some for purchases) and members of that community may be hired to develop additional content.

Performance System and RAPs

The key outputs of the iBL are instruction-driven changes to a target code base. We refer to that code base as the Performance Software (PS) as distinct from iBL software, simulator software and software controlling simulation objects other than the PS-controlled object (the instructable system).

In principle, the PS could be written in any executable language including general purpose programming languages (Java, C++, etc). For iBL, it is useful to focus on a single language, ideally one that meets all of the following criteria:

- **powerful** enough for demanding UAV-ISR systems
- **mature** enough to have the kinks worked out and to include dev tools
- **high-level** enough to avoid needing a lot of iBL work on language minutiae
- **widely-used** enough for iBL capabilities to be seen as ready for action

RAPs, a plan/procedure language developed for AI agents, was selected as the best fit for iBL in the short-medium run.¹⁸ This choice maximizes power and high-level expressiveness at the cost of wide use. In particular, RAPs has a rich language for procedures that includes concurrency, reactivity, multi-task management, time management and other capabilities needed for control-oriented domains such as ISR data-collection. It also includes syntax for modeling code behavior intent (e.g. post-conditions) that is very useful for driving teaching dialog and specifying code transforms. RAPs is mature as AI frameworks go, having been continuously used, maintained and extended for more than 20 years.

A RAP (Reactive Action Package, see Figure 31) is a procedure, more or less in the traditional programming language sense, that combines one or more alternative methods for performing a task. The INDEX clause contains the name of the RAP and input parameters. The SUCCESS clause specifies the desired outcome. If the outcome is already achieved when the RAP begins execution, execution terminates successfully without further action. Each METHOD clause defines an alternative way to meet the success condition. CONTEXT checks whether the method applies and may contain preference criteria. If context clauses for more than one method are satisfied, the first in lexical order will (normally) be selected. The TASK-NET defines a set of steps constituting the main body of the method. Each step names either a primitive action or

¹⁸ More detailed PS language selection criteria: (1) powerful, high level language (temporal constraints, durative actions, concurrency, contingency-/exception-handling, interruption management, complex conditions, pre-/post/invariant conditions in steps, explicit control over loops/branches/decomposition); (2) mature (cross-platform, Java-friendly, embeddable, testing support, logging, dynamic code modification (hot swap), stable, debug tools, documentation, licensing); (3) wide-use (AI community awareness, general software community awareness)

another RAP. The task net can include various control structures for, e.g., branching, event- and time-based triggering, concurrency and iteration. Self-modeling notations include the FOR clause which specifies post-conditions and causal links to other steps. These were originally intended for use by deliberative AI algorithms (planners) but are needed in iBL to drive dialog and parameterize code transforms.

```
(DEFINE-RAP
  (INDEX (load-into-truck ?object))
  (SUCCESS (location ?object in-truck))
  (METHOD
    (CONTEXT (or (not (size-of ?object ?size))
                  (and (size-of ?object ?size)
                        (<= ?size arm-capacity))))
    (TASK-NET
      (t1 (pickup ?object)
           ((holding arm ?object) for t2))
      (t2 (putdown ?object in-truck)))
    (METHOD
      (CONTEXT (and (size-of ?object ?size)
                     (> ?size arm-capacity)))
      (TASK-NET
        (t1 (pickup lifting-aid)
             ((holding arm lifting-aid) for t2))
        (t2 (pickup ?object)
             ((holding arm ?object) for t3))
        (t3 (putdown ?object in-truck))))))
```

Figure 31. Example RAP from Firby (1989) in the domain of robotic trucks

More information about RAPs can be found in papers available online – e.g.:

- RAPs language and interpreter (Jim Firby's 1989 dissertation)
citeseer.ist.psu.edu/article/firby89adaptive.html
- Condition detection
<http://www.cs.cmu.edu/~aphillips/publications/cera-aaai-04.pdf>
- Agent architecture incorporating RAPs
<http://www.kipr.org/papers/jetai.pdf>

Learning by Noticing

As noted, iBL uses a model of the PS and its task environment to determine how to modify the existing code base given some well-defined expression of instructional intent. This model comes from three places: (1) static analysis of the code base (e.g. extracting RAPs FOR clauses to model post-conditions); (2) dynamic analysis of PS behavior over a series of simulation trials (e.g. learning causal rules to help connect feedback attributes such as view-angle to controllable attributes such as altitude); and (3) asking questions of the user, leveraging their domain knowledge to fill in gaps in the existing model of code norms, requirements and logic (e.g. asking the user whether a VNAV descent occurs instantly or gradually). Dynamic analysis is a form of learning by noticing. Specific objectives in this area are still being set.

11.5 Teaching Layer

The teaching layer includes all components for acquiring instructional content from a user, defining a change in PS behavior implied by that content, generating a code transform and applying that transform to modify the PS.

Lessons

An iBL lesson is an interaction between iBL and user as well as a data object representing the state of that interaction. State information includes a record of interactive dialog, a representation of the hypothesis state of the concept being learned (i.e. possible CTTs and specifying values), evidence gained so far for reducing the space of valid hypotheses, a set of knowledge acquisition goals for reducing the space further and lesson status information (focal, open, closed; transform ready, not ready).

Open lessons are instructional interactions that are still underway. Several lessons may be open at the same time, though only one is the *focal* lesson, the subject of current dialog. *Closed* lessons have been declared complete, causing the lesson code transform to be applied. They can be re-opened by the user, optionally rolling back the code change resulting from the already-applied transform.

Note that this sense of lesson differs from the use of the term in earlier, non-interactive BL systems where it referred to prepared instructional content for a specified concept and was one element of a “curriculum.”

Teaching Layer UI

The teaching interface (see Figure 32) includes components for managing lessons, carrying out general dialog, specifying objects/values/relations of specialized types (by “gesture”) and carrying out dialog for particular natural instruction methods.

Lesson management (top control bar in Figure 17 includes functions for starting a new lesson, switching focus to a different open lesson and reviewing (and potentially re-opening) completed lessons. Lesson management controls are always visible and show the current focal lesson. Users can start a new lesson at any time; this may occur as a digression from an ongoing lesson in order to define a needed building block concept (e.g. a function or procedure).

Below the lesson management bar is the instruction content pane. Normally, this shows a dialog transcript for the focal lesson, an English paraphrase of most recent iBL questions and user answers/teaching actions. During the initial phase of dialog (framing the lesson), student questions appear in the Dialog Transcript view. The user can answer most questions by clicking on underlined text in the question text.

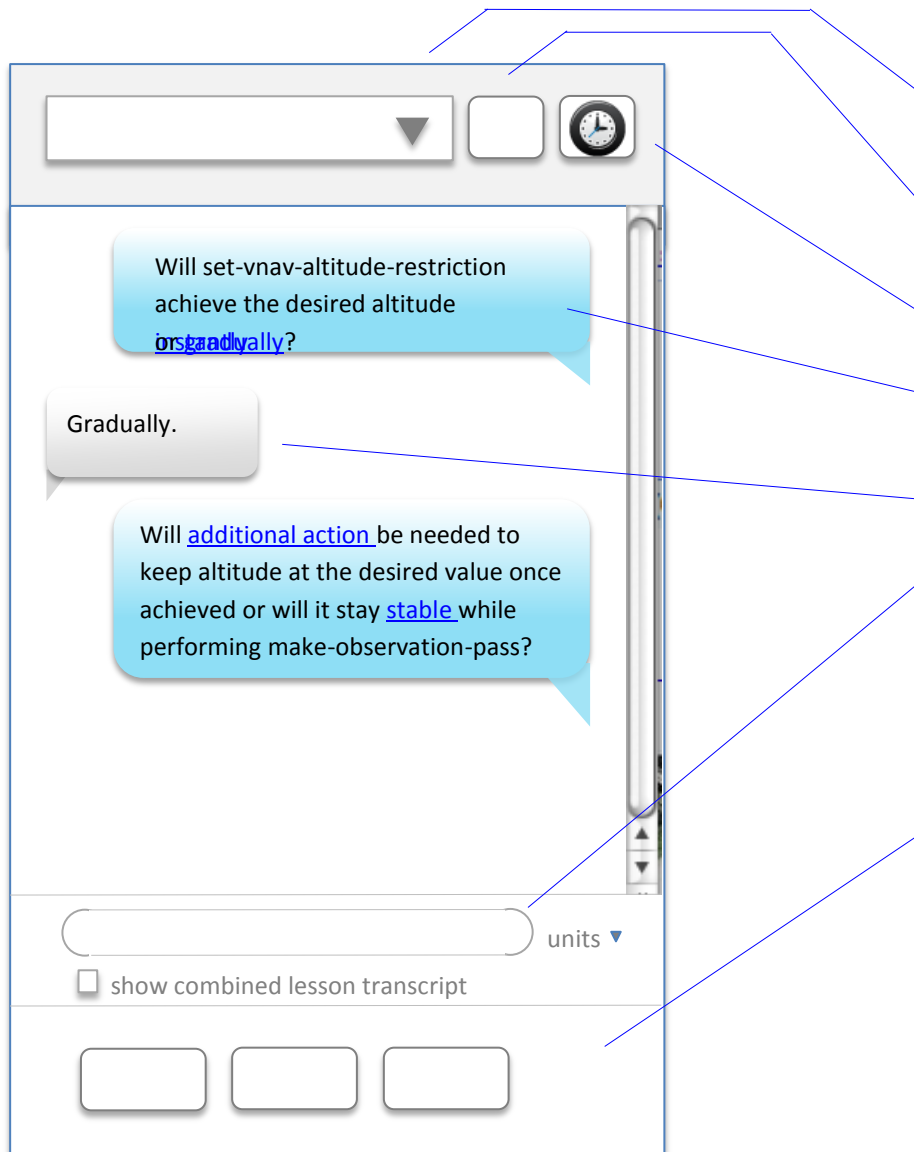


Figure 32. Teaching UI

Alternately, the user may need to answer questions by clicking an object in the simulation main view (e.g. selecting a sensor image or ground object), typing into a text field (e.g. to specify numeric values) or by bringing up and interacting with any of several specialized data views. Specialized views include a hierarchical list of simulation objects (including those not visible in the main view), attribute value/histories of specific objects, the Narration View (a chronological simulation trace useful for selecting timepoints), the Procedures View (a user-readable list of PS procedures that may be modified), the Activities View (the PS agenda, essentially its call stack) and others.

Finally, the Teaching UI will include NIM-specific components – e.g. for specifying procedures by Telling (the TellMe system) and for demonstrating procedural behaviors. UI and user interaction designs for these are pending.

Code Transformations

Code transforms are fully specified modifications to the performance system code base that result from iBL instructional dialog. They consist of one or more point modifications – i.e. lexically localized changes or additions to the code base such as substituting one expression for another or inserting a line of code (step) into a procedure.

Code Transformation Templates (CTT) define a class of code transformations that collectively represent a typical set of related changes needed to produce a single conceptual change to code behavior. For example, when inserting a new procedure step represented in a language where lexical order does not define execution order, a CTT will specify point modifications both for adding the step and for declaring when to carry it out (in relation to other steps and/or exogenous events).

When a CTT is fully specified – i.e. all of its parameters have values – it is eligible to be applied as a CT. The user examines a natural language description of the proposed change and decides whether it is ready or if additional CT refinement is needed. Student-driven dialog will often be enough to produce a fully specified “first draft” version of the CT. This sets a baseline version for further refinement by Natural Instruction Methods and associated invocation of ML capabilities developed under the BL program.

Example: *CTT:change-step-parameter* consists of a single point modification that replaces a RAP step parameter with a new expression. Specifying this CTT means setting five parameters: (1) the RAP to be modified, (2) the method within the RAP, (3) the step within that method, (4) the parameter within the step and (5) a new expression. (1) is indexed by name; (2)- (4) by lexical position. Example:

```
(define-rap
  (index (collect-isr-data ?obs-target-list))
  (method
    (context (threatcon green))
    (task-net
      (sequence
        (t1 (turn-on camera-1))
        (t2 (vnav-altitude ?cruise-altitude))
        ...))
  )
```

To change the target altitude from `?cruise-altitude` to `(2000 ft)`, iBL would apply CTT-change-step-parameter (rap method step arg expr) with rap = collect-isr-data, method = 1, step = t2, arg = 1 and expr = (2000 ft). We assume that these changes are not applied directly to a text level representation of the code, but to parse trees of individual RAPs (with attached model annotations). The code transformation component then derives a new version of any modified procedure from the modified parse.

Dialog Episodes

An iBL dialog encompasses all interactions that contribute to a lesson including not only language-like “utterances”, but also gestures and interactions with specialized UI elements to produce instructional content. A dialog episode (DE) is a fragment of a larger dialog that produces one unit of lesson content – e.g. the answer to a question from the student or an

indicated example of the concept being taught.¹⁹ A DE may be initiated by the user or by the iBL system. If initiated by the system, the episode takes the form of a question-answer pair. The question is generated to achieve or advance a knowledge acquisition goal (KAG) generated by the iBL system. If initiated by the user, the episode has the form of a supported natural instruction method (NIM) addressing a user-specified KAG.

Knowledge Acquisition Goals

KAGs fall into three general content categories, differentiated by what aspect of a PS modification they help specify: **what** the performance system should do (behavior), **when** this behavior should be carried out (scope) and **how** the code base should be modified to achieve teacher intent (what+when) without breaking correctly specified current code (implementation). Lessons that modify PS code behavior include dialog on all three topics. Lessons that create building blocks (e.g. new functions, procedures and conditions) without aiming to immediately affect PS behavior cover what and how but not when.

Information that satisfies a KAG can be directly applied to reducing the space of hypotheses about what to learn. The initial hypothesis space for a lesson is the set of all possible parameterizations of all available code transform templates. Conceptually, a DE reduces the set of candidate CTTs and/or reduces the set of possible CTT parameter values. However, this does not mean that early dialog is used to select a CTT and later dialog used to specify parameters. That would mean dialog about how precedes dialog about what and when.

Instead, early dialog establishes teacher intent (what+when) without reference to PS internal structures and state representations. This can be approximately expressed in terms of a rule where the LHS represents scope (when) and the RHS represents desired behavior (what).²⁰ Using the example from 11.2.2, this might be paraphrased as "IF circling a hot-suspicious observation target THEN view angle should be 45 degrees or less."

Further dialog refines this to align with PS internals – i.e. scope specified in terms of procedural context and/or states testable by control structures; behavior in terms of attributes that the PS can control directly and functionally parameterized methods for doing so. The result in our example might be paraphrased as "IF circling to make an observation pass for the purpose of observing a hot suspicious target THEN VNAV should be used to make altitude low enough to achieve a view angle of 45 degrees or less."

When the user selects an instruction method to start a lesson the Lesson State Update component creates an initial set of KAGs. In our example, the method is to provide feedback on an unsatisfactory current state. This creates initial KAGs to specify: parameters of that state including a state variable (attribute and object), a violated teacher preference on its value and timepoints defining when the violation occurred (see Figure 33a).

¹⁹ Dialog might also include exchanges of other sorts including meta-conversation about the lesson (starting, ending, pausing, assessing progress, interrupting, resuming), agreement on transient language in support of a lesson (e.g. "foo" = all procedures that change altitude; "the truck" = truck-32) and information exchanges as decision support for the user in determining what to teach (e.g. a look at all active mission tasks).

²⁰ Handling the general case requires more expressive notations such as used for constraint-based temporal planning – e.g. CAIP (Jonsson and Frank).

Unsatisfied KAGs index Dialog Episode Operators (see next section) to engage in dialog episodes that achieve the goal. This may result in new KAGs being created to ensure that, once all are achieved, the resulting teacher intent representation defines a new PS policy (see Figure 33b). For example, when the user selects vertical-view-angle of the observation video as the state variable of interest, iBL determines that this is not something the PS can directly control. It then generates a new KAG to identify a directly controllable state variable providing indirect control of view angle.

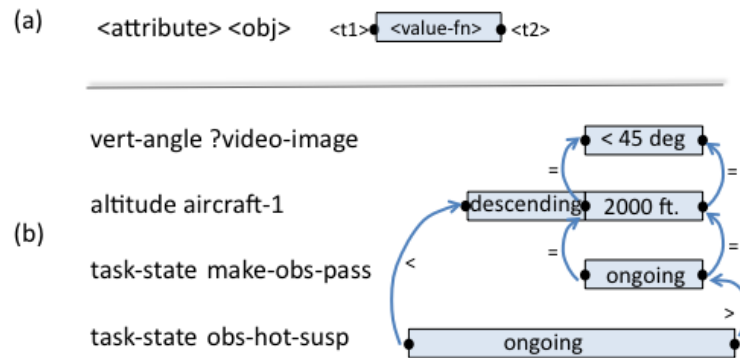


Figure 33. (a) initial KAGs lead to (b) (partially) specified user intent

When enough is known about intent, dialog about how to implement that intent becomes possible.²¹ In particular, iBL needs “how” information to select a CTT, map

what/when information to CTT parameters and specify additional CTT parameters for non-disruptively splicing in new code. For example, the question in 11.2.2 about whether to change from cruise altitude to observation altitude in flight or to be at observation altitude from the beginning (never bothering to reach cruise altitude) is implicitly a question about which CTT to use (change parameter of existing step vs. add a new step). Similarly, the question about whether to restore cruise altitude once observation is complete is implicitly about software requirements that the current code base already meets (regarding the desired persistence of an existing altitude change step) and thus whether the add-step CTT needs to be parameterized to ensure that the requirements stays met.

Note that none of these “how” questions refer to code. Instead, they elicit domain knowledge from the user that helps fill in gaps in a model of the code, enabling iBL to make intrinsically code-level decisions such as CTT selection.

²¹ Dialog about implementation can be interleaved through the conversation rather than waiting for all aspects of intent to be specified. This is the case in the example in 11.2.2 where CTT selection is made about halfway through. Interleaved dialog will probably be perceived as more natural. More generally, it will sometimes be necessary to switch dialog back and forth between what, when and how topic areas since progress in one area will tend to reveal options and constraints in another.

Dialog Episode Operators

For every type of knowledge acquisition goal there are one or more Dialog Episode Operators (DEOs) for achieving it. DEOs include at least the following information:

- **who:** initiated by student or teacher
- **index:** knowledge need (KAG) served by executing the DEO
- **preconditions:** DEO applicability test
- **priority:** preference for this DEO over eligible alternatives (even for same KAG)
- **procedure:** an action or sequence of actions, at least one each for iBL and user. Includes UI actions (e.g. text of questions to user), info retrieval (e.g. find name of procedure that caused specified event), analysis (e.g. determining that view angle is function of altitude and lateral-distance), error handling (e.g. what to do if user enters invalid response)
- **postconditions:** formatted info provided for lesson state update
- **new KAGs:** rules run after the procedure to generate additional KAGs if needed

Dialog episodes initiated by the system start by matching a KAG to the index clause of a DEO and then checking its preconditions to see if it is eligible to execute. Sometimes eligible matches will be found for several KAGs. Moreover, there may be more than one match for a single KAG representing alternative methods. For example, in cases where the KAG might be satisfied by information in the PS code model, a DEO may be used to make the query/inference; if the query fails, the alternative DEO is used instead. Alternative DEOs may also be available for different NIMs. Once a DEO is selected, its procedural content is executed.

Example. The following Dialog Episode (a single Q&A pair) appearing in the sample scenario (section 11.2.2) would result from executing the DEO specified below.

iBL: Will set-vnav-altitude-restriction achieve the desired altitude immediately or gradually?"

User: gradually

```
who: iBL
index: KAG=augment-action-model:effect-latency-type
preconditions: known(?x = control-action), known(?y = controlled-
attribute)
priority: 4
procedure:
  1.    prompt: will ?x achieve the desired ?y /option(immediately)
         or /option (gradually) ?
  2.    wait for click-option-select = ?choice
  3.    echo: ?choice
postconditions: (post-condition-latency ?x ?choice)
new-KAGs: if ?choice=gradually then new-goal(predict-duration ?x ?y)
```

When a DEO executes successfully, the output is delivered to lesson state update mechanisms to make the lesson hypothesis more complete or more accurate.

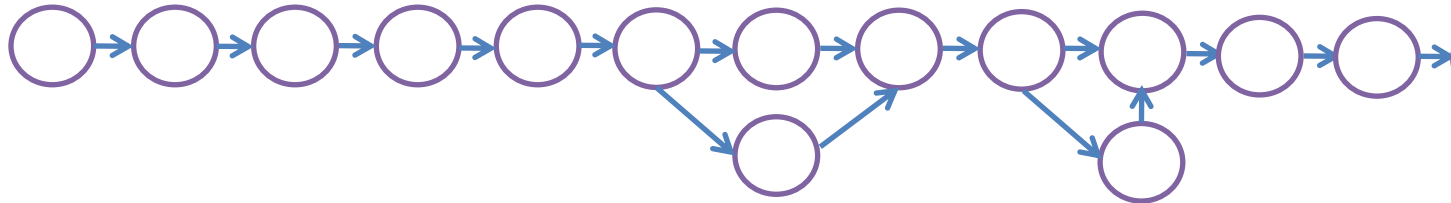
Dialog Sequencing

There are at least two major dialog sequencing issues: managing a mixed-initiative interaction and choosing which KAG generated by iBL to work on next.

The current approach to mixed-initiative dialog control assumes that iBL will always have at least one active KAG and corresponding DEO until the lesson is eligible to be declared complete – i.e. iBL will always “have something to say” until that point. This gives a possible overall structure to the dialog where iBL initiates all dialog to the point where there is a “first draft” hypothesis for the learned concept (a fully specified CTT), at which point the user initiates all further dialog to refine that hypothesis. In this model, users might take initiative during the first phase for two reasons. First, they might start a new, digressive lesson to teach a building block concept needed to make further progress at the main lesson. Second, they might elaborate on a just-covered topic to further refine that aspect – e.g. after dialog establishing that 2000 ft. would have been a good altitude in the current context, the user might want to provide examples of good altitudes in other contexts to help iBL generalize the hypothesis about target altitude.

The previous section implies one method for sequencing which KAG to work on next – determine what DEOs apply to all current KAGs and do whichever has the highest (fixed) priority. This is probably too simple an approach since it ignores factors related to good conversational flow, makes no provisions for bridging continuity gaps resulting from shifts of initiative and, in general, fails to take context into account. Users would likely find dialog sequenced without context-sensitivity to be disjointed and unnatural.

However, even that simple approach is more sophisticated than we can likely handle in early versions of iBL, not because the sequencing method is complicated, but because it requires a more fleshed out representation of lesson state than we currently have. Instead, the initial approach will use a simple decision-tree for sequencing (Figure 34).



- | | |
|---|--|
| 1. Select:lesson-start (→ contrast desired/actual state) | 9. Augment-action-model:effect-latency (→ gradual) |
| 2. Select:outcome-attribute (→ image-vertical-view-angle) | 10. Augment-action-model:effect-stability (→ stability) |
| 3. Select:controllable-attribute (→ aircraft-altitude) | 11. Select-invariant-enforcement-specification-method (not used) |
| 4. Select:episode-interval (→ start(proc-obs-pass),end(proc-obs-pass)) | 12. Select:invariant-violation-recovery-policy (→ no-action) |
| 5. Select:preference-conflict-resolution (→ late-override-and-restore) | 13. Select:parameter-specification-method (→ tell-constant) |
| 6. Select:relative-task-scope (→ immediate-call-context(proc-obs-pass, call-stack)) | 14. Describe:simple-value-with-unit (→ 2000 ft.) |
| 7. Select:control-behavior (→ set-vnav-altitude-restriction) | 15. Select: relative-start-time (→ end(proc-obs-pass), +0) |
| 8. Define:new-procedure (not used) | 16. Select-value-update-policy (→ recompute-on-start) |

Figure 34. Dialog Episode Operators and sequence for iBL_demo_2

Lesson State Updates

Lesson state updates translate new lesson content into concept hypothesis revisions and new knowledge acquisition goals. The effect of particular content depends mainly on what KAG it was acquired to achieve. For example, starting with the initial KAGs represented in Figure 18a, we can see the effect of new information gained in each dialog episode in the example scenario.

<attribute> <obj> <t1> • <value-fn> • <t2>

iBL (student): What do you want to give feedback about? Click on a visible object or object on the world object list and then pick an attribute of the object.

User (teacher): vertical-view-angle of camera-image-1

vertical-view-angle camera-1 <t1> • <value-fn> • <t2>

iBL: Should a better vertical view angle be achieved by changing altitude or target-distance?

User: altitude.

vertical-view-angle camera-1 t1 • <value-fn> • t2
 =
 altitude aircraft-1 <t3> • <value-fn> • <t4>

iBL: When should altitude have been different to achieve a more desirable vertical-view-angle? Select start and end times from events listed in the narration view.

User: From start of the activity-23 (make-observation-pass) to the end of activity-23 (make-observation-pass).

vertical-view-angle camera-1 t1 • <value-fn> • t2
 =
 altitude aircraft-1 t3 • <value-fn> • t4
 =
 task-state make-obs-pass <t5> • ongoing • <t6>

Information from new dialog episodes elaborates the representation of user intent. New KAGs driving new DEs are generated at each step if new unknowns are introduced. These fall into several categories. First, there are parameters of each “token” including state-attribute, state-object, start and end time points and value – all of which can be defined functionally (or equivalently, in terms of constraints). In addition, each token either represents something to be enforced (i.e. it is part of the “what” or behavior part of the intent description) or to be waited for (i.e. it is part of “when” or scope). Different KAGs apply to each type.

For example, specifying vertical view angle as part of intended behavior causes the lesson update component to test whether this is something the PS can control. If so, no new KAG is needed. If that is unknown, the updater generates a KAG to find out and enhance the code model

accordingly. If not, as is the case, a new KAG is generated to find a controllable attribute that produces the desired influence over view angle. In this case, that leads to elaborating the intent to include a behavior token for altitude and to resolve the KAG for view-angle time points to be equal to the timepoints of the new altitude token. To define timepoint for altitude, the intent description is further elaborated to include a token on the task-state of an activity, make-observation-pass. This token defines scope rather than behavior, so its timepoints do not need to be controlled, just detected. That constrains the timepoints for altitude and generated a KAG to determine when the activity is in the ongoing state. Note that specified token parameters are functional and therefore may contain variables – as long as those variables are tied to waited for events such as PS activity start and end times (or any dynamic value assignment process).

New KAGs come from initial state and from intent elaborations as described. Additional KAGs arise when intent is specified enough to select a CTT, each of which includes a set of associated unknown parameters. For example, the add-new-precondition-step parameter CTT selected about halfway through the dialog in 11.2.2 includes parameters for where to insert the new step, whether a cleanup action is needed and whether to enforce the precondition as an invariant once the supported step begins. Each of these drives questions that appear in later dialog.

Finally, new KAGs may be generated by the user to add evidence relevant to an intent attribute. For example, the user may have specified early in the dialog that, in the current situation, desired altitude is 2000 ft. But later, noticing that the lesson summary is treating 2000 ft as if that was always the correct value, the user may want to give examples of different altitudes in different situations. Initiating dialog to specify a new altitude example generates a new KAG. Updating the lesson involves applying ML methods to generalize from known examples. In the case of ILP trying to learn a function, this may identify additional factors (function parameters) – e.g. time of day or presence of nearby mountainous terrain – which will require elaborating the intent structure with new tokens on new state variables.

Lesson updates are made in response to satisfied KAGs and do not, in themselves, determine when a lesson is complete. However, the user can review a summary of the lesson state at any time, an English summary of how behavior will change if the lesson is applied. If the user finds this acceptable, they can declare the lesson complete, causing the lesson code transform to be applied to the PS code base.