Developing a Meta-Level Problem Solver for Integrated Learners

Jihie Kim and Yolanda Gil ({jihie,gil}@isi.edu)
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292 USA

Abstract

A learning system may need several methods that follow different learning strategies in order to learn how to perform complex tasks. For example, a learning method may be used to generalize from user demonstrations, another to learn by practice and exploration, and another to test hypotheses with experiments. In such an integrated learning system, there is a need for systematically coordinating the activities of the participating learning agents especially to ensure that the system creates appropriate procedural knowledge.

In this paper, we describe an approach for developing a meta-level problem solver that coordinates different agents in an integrated learning system based on their capabilities and status of learning. The system is called Maven. Maven is cast on a BDI-style framework with explicit representation of learning goals, a set of plans for achieving learning goals, and high-level learning strategies that prioritize learning goals. By supporting both top-down and bottom-up control strategies, Maven supports flexible interaction among learners. The status of desired learning goals and goal achievement history enables assessment of learning progress over time. Maven is being used for coordinating learning agents to acquire complex process knowledge.

Keywords: meta-level reasoning, integrated learning

Introduction

Developing systems that learn how to perform complex tasks is a challenging task. As the knowledge to learn becomes complex, with diverse procedural constructs and uncertainties to be validated, the system needs to integrate a wider range of learning methods with different strengths. The Poirot system pursues such a multi-strategy learning methodology that employs multiple integrated learners and knowledge validation modules (Burstein et al., 2007).

In such an integrated learning system, activities of participating agents have to be coordinated systematically, especially to ensure that the overall system acquires the desired procedural knowledge. This can be done through a meta reasoning capability that monitors and controls the learning process (Anderson and Oates 2007; Cox 2007). The meta-level reasoner may solve a given learning problem with a set of meta-level plans for coordinating and prioritizing the activities of the agents based on their capabilities and on the status of learning.

Developing such a meta-level reasoner for an integrated learning system poses several challenges:

- supporting flexible interactions among agents that pursue different learning strategies.
- systematically addressing conflicts and failures that arise during the learning.
- assessing the progress of learning over time.

In this paper, we describe an approach for developing a meta-level reasoner that coordinates activities of different learners in an integrated learning system. The system is called Maven (Moderating ActiVitiEs of iNtegrated learners). Maven is developed based on our early work on learning goals and meta-level reasoning for interactive knowledge capture (Kim and Gil 2007; Gil and Kim 2002). The system explicitly represents learning goals, a set of plans for achieving learning goals, and high-level learning strategies that prioritize learning goals. Maven includes both top-down and bottom-up control strategies, and therefore supports flexible interactions among learners. The status of desired learning goals and goal achievement history enables assessment of learning progress over time. We describe our work in the context of developing a meta reasoner for Poirot.

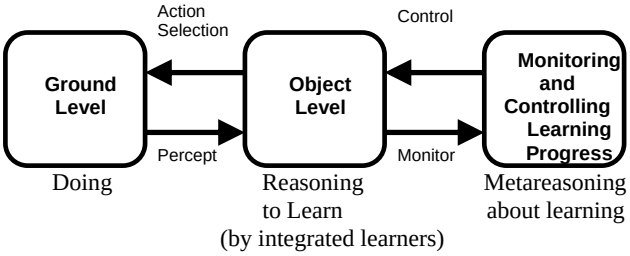

Figure 1: Meta reasoning for integrated learners

Figure 1 shows a mapping between the Maven framework and a general model of metareasoning (Cox and Raja 2007). The groud level actions corresponds individual learning activities of participating learning agents. The object level represents various learning and reasoning capabilities that support integrated learning. Maven monitors the learning process and keeps track of the progress including when learning is done.

Background: A Multi-Agent Learning System

The Poirot system is developed to learn complex process models in a domain of medical evacuation. Users invoke

web services to plan the evacuation of a set of patients, expressed as a *workflow*. Given a sequence of expert demonstration steps (called a trace) that shows how to create evacuation plans for moving patients from existing locations to desired hospitals, the Poirot system needs to learn a general workflow that can solve similar evacuation problems. Each step in the trace is either a web service invocation (e.g. look up patient requirement, find an airport) or object selection action (e.g. select a flight from a proposed flight list).

The learning process constructs *domain methods*, essentially hierarchical-task network methods that contain:

- step orderings
- branches and loops
- preconditions
- task decompositions
- object selection criteria

There are several different types of learning approaches (or *agents*) that participate in learning such complex workflows in Poirot.

- trace-generalizers: generalize information in the given demonstration trace and build domain method hypotheses (domain methods for short) for representing step sequences, loops, branches and preconditions. Such methods can be used in creating workflows. WIT uses a grammar induction approach in creating a finite state model of trace steps (Yaman and Oates, 2007). DISTILL learns general procedure loops and conditions for each step (Winner and Veloso, 2003).
- trace-explainers: build domain methods that explain the given top-level task goal against the given demo trace. Xplain uses a set of explanation patterns for building such domain methods (Cox 2008).
- hypothesis integrators: integrate domain methods created by different learners and detect potential conflicts or ambiguity in the domain methods. Stitcher provides such capability (Burstein et al., 2007).
- workflow constructor: For a given problem goal and an initial state, these create a workflow from a set of domain methods and primitive action definitions. SHOP2 provides such planning capability (Nau et al., 2005).
- workflow executors: test the constructed workflows by execution, such as SHOPPER (Burstein et al., 2007).
- knowledge validation by experiments: test alternative hypotheses by designing and performing experiments, such as CMAX (Morrison and Cohen 2007).

These agents communicate through a blackboard. The learning problem given to Poirot consists of a single demonstration trace and a problem description (a top-level task goal and an initial state). The system currently has limited background knowledge such as web service definitions for primitive actions. In order to learn complex workflows from such input, the system needs to coordinate the learning agents effectively.

Approach

Maven has explicit representations of learning goals and a set of plans to prioritize and accomplish those goals. Maven follows the general BDI (Belief, Desire and Intention) agent model. In a BDI architecture, *belief* represents what the agent believes to be true about the current state of the world, *desires* consist of the agent's goals, *intentions* are what the agent has chosen to do, and *plans* describe how to achieve intentions. The BDI reasoner matches goals with plans that decompose them into subgoals, and turns those subgoals into desires. It then decides which goals to intend based on other plans. The BDI reasoner checks the external state with sensing actions, and may suspend intended goals or replan its behavior accordingly. This framework supports both reactive and goal-oriented behavior. For Maven, the BDI model supports flexible interaction with various system components, including interleaving of top-down and bottom-up control.

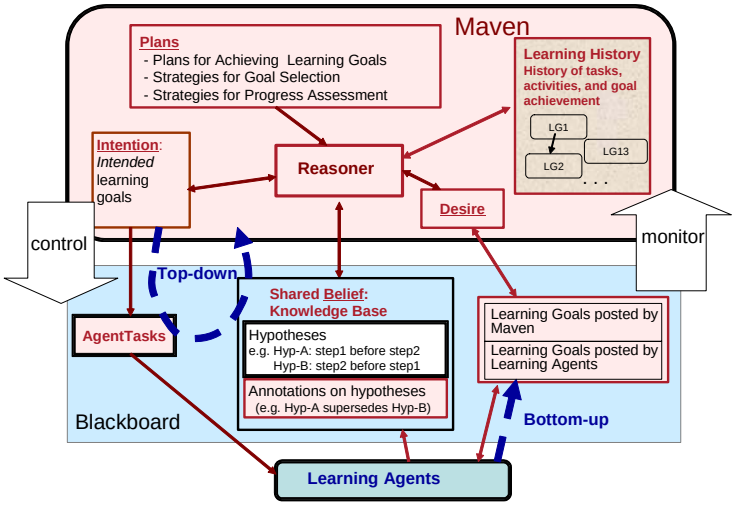

Figure 2: Maven interaction with other agents

Flexible Control

Among (shared) learning goals that are desired, Maven *intend*s some of them using a set of goal selection strategies, and initiates *AgentTasks* for the intended learning goals.

The **top-down control cycle** in Figure 2 consists of a) Maven intending several learning goals, b) Maven creating AgentTasks for intended learning goals and c) learning agents responding to the tasks and reporting the results. The results from AgentTasks including new learned knowledge are stored in the shared knowledge base. If there are issues such as knowledge gaps or conflicts found, they are also reported by the agents as a part of the results. The new result can lead to further learning goals.

The **bottom-up control cycle** results when learning agents create new learning goals by themselves and post the goals as shared learning goals. The agents can pursue the goals asynchronously without Maven's intervention. Maven responds to the goals created by the agents as well as the ones that Maven itself initiated.

These top-down and bottom-up control cycles support flexible interactions among the participating agents and between the agents and Maven. The details of these cycles are described below with some examples.

Learning Goals and Plans

Maven uses explicit learning goals and plans for achieving the goals. Table 1 shows some of the learning goals and plans that are used by the system. The set of goals and plans reflect both the capabilities that are supported by the participating agents and the knowledge constructs that need be learned. They can be extended as new agent capabilities are introduced.

-- KNOWLEDGE CREATION --
GeneralizeTrace
ExplainTrace
CreateWorkflowWithDomainMethods

-- ISSUE IDENTIFICATION --
IdentifyOrderingAmbiguity
IdentifyUnexplainedSteps

-- ISSUE RESOLUTION --
ResolveAmbiguousStepOrderingHypotheses
ResolveDomainMethodGaps
ResolveUnknownBranches
ResolveUnexplainDomainMethodHypotheses

-- KNOWLEDGE VALIDATION --
ValidateWorkflowKnowledge
EnsureWorkflowGeneratability
EnsureTraceReproducibility
ValidateKnowledgeWithExperiments

(a) sample learning goal types

Plan:LearnWorkflowFromDemoTrace (DemoTrace&ProblemDesc *tr*, BackgroundKnowlege *k*)
- trigger condition: a new demonstration trace and a problem description (with the top-level task goal and initial state) given
- substeps: GeneralizeTrace (*tr*, *k*) and/or ExplainTrace (*tr*, *k*), to create domain methods ms, and then CreateWorkflowWithDomainMethods (*tr*, *ms*)
- Achievement condition: No remaining issue on created workflow

Plan:GeneralizeTrace (DemoTrace&ProblemDesc *tr*, BackgroundKnowlege *k*)
- trigger condition: No generalized domain methods for trace.
- substeps: create an AgentTask for trace-generalizers (WIT & DISTILL) with the current trace information
- Achievement condition: A set of domain methods for the trace is successfully created by the trace-generalizers

Plan:IntegrateKnowledge(DomainMethods *m1*, DomainMethods *m2*)
- trigger condition: more than one domain method hypotheses exist for the same trace steps.
- substeps: create an AgentTask for trace-integrators with the alternative methods
- Achievement condition: The methods are successfully integrated

Plan:ValidateCausalHypotheses (DemoTrace&ProblemDesc *tr*, StepOrderings *orderings*)
- trigger condition: notices step sequence ambiguity
- substeps: achieve subgoals in sequence
  DesignExperimentForCausalHypotheses (*tr*, *orderings*)
  to produce experiment packages {*pkg*}
  SelectExperimentsToRun ({*pkg*}, *tr*) to select pkg_i
  RunDesignedExperiments (pkg_i, *tr*)
  FindAppropriateStepOrderingHypossWithExpResults
  to confirm or modify *orderings*, or suggest more experiment
- Achievement condition: Step orderings modified or confirmed

Plan:CreateWorkflowWithDomainMethods (DomakinMethods *ms*, DemoProblemDesc *pr*)
- trigger condition: new domain methods for achieving the top-level task goal created and there are no unresolved issues for the domain methods to use …

Plan:EnsureTraceReproducibility (Workflow *w*, DemoTrace&ProblemDesc *tr*)
- trigger condition: a workflow can be generated from learned knowledge
- substeps: achieve subgoals in sequence
  ExecuteWorkflowWithInitialState (*tr*, *w*)
  to produce execution result *exe_result*
  CheckCompatibilityWithDemoTrace (*tr*, *exe_result*)

(b) Sample Maven plans for achieving learning goals

Table 1: Example Maven learning goals and plans

The initial Maven knowledge base consists of a set of learning goals *G*, plans *P*, agent capabilities *C* for achieving primitive learning goals, and strategies *S* for selecting learning goals and progress assessment: <*P, G, C, S*>.

Each learning goal $g \in G$ can have a set of parameters $param_g$ that describes desired goals. For example, LearnWorkflowFromDemoTrace goal is posted/desired with respect to the demo trace and problem description, and the given background knowledge. Each learning goal g is associated with a plan with 1) *trigger conditions* 2) *achievement conditions* for detecting achievement of the learning goal and 3) substeps for achieving it. The substeps in the plan are described in terms of the parameters and the subgoals involved in achieving the goal.

$<tc_g\ (param_g),\ ac_g\ (param_g),\ substeps_g(param_g)>$

A learning goal is *desired* when its trigger conditions are satisfied. We introduced the achievement condition in order to keep track of goal achievement while supporting bottom-up control. Maven relies on a set of *sensors* that keep track of trigger conditions of all the learning goals and achievement conditions of desired learning goals. Goals can be achieved serendipitously or goals may fail unexpectedly even after associated plans are executed.

When an intended goal is decomposed into subgoals, its subgoals are desired as defined by the Maven plan. For example, in the initial phase of learning, when a new expert demonstration trace is detected by the Maven sensor, the goal of LearnWorkflowFromDemoTrace will be desired for the trace. The LearnWorkflowFromDemoTrace goal can be intended and decomposed into its subgoals, GeneralizeTrace, ExplainTrace and GenerateWorkflow from learned domain methods. The top portion of Figure 3 (history of desired goals) illustrates how the goals are related. For GeneralizeTrace, Maven will create an AgentTask for invoking trace-generalizers (a set of learning agents that create step orderings, branches and loops from a given demonstration trace). When all the subgoals are achieved and the achievement condition is satisfied (i.e. a workflow is successfully generated from the learned domain methods and there are no issues), the original goal to LearnWorkflowFromDemoTrace becomes achieved.

Some kinds of learning goals can be iteratively desired when their trigger conditions are satisfied. For example,

goals for validation experiments can be desired more than once until the experiment results provide enough information to confirm or disconfirm the tested hypotheses. The details of other goals in Figure 3 are described below.

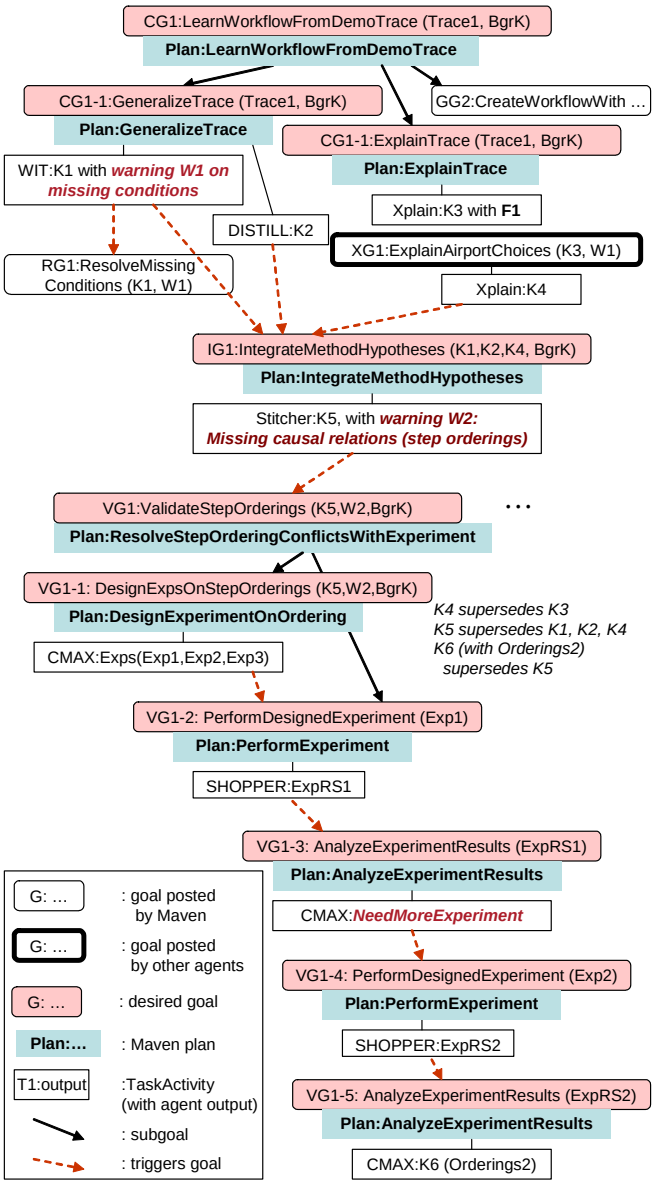

Figure 3: A goal/method decomposition of desired and intended learning goals

Learning control strategies

In following the top-down control cycle, Maven can adopt different goal prioritization strategies in selecting which goals to intend/pursue in the current situation. For example, Maven may choose to first create more knowledge and then validate the created knowledge. The current strategies used include:

- In the initial phase of learning, prefer domain method creation goals to issue resolution goals.
- For a given knowledge, prefer issue resolution goals to knowledge validation goals.
- When there are multiple issue resolution goals, prefer ones that resolve issues on more frequently used knowledge.
- Prefer finishing subgoals of existing plans instead of intending new goals.

Additional criteria such as confidence on the knowledge created and competence in solving related problems with learned knowledge, can be introduced to drive the learning process (Kim and Gil 2007; Kim and Gil 2003; Gil and Kim 2002). That is, the selection of which learning goals to pursue can be decided based on expected confidence and competence changes by achieving goals or subgoals.

Depending on the strategies employed, the system may present different behavior such as an eager learner that generates more hypotheses first vs. a cautious learner that tests learned knowledge from the beginning and produces more validation goals early on.

Learning goal lifecycle

The top-down and bottom-up control cycles imply that learning goals can take several different paths in their lifecycle. This is shown in Figure 4. A goal can be desired from a trigger condition of the goal or created by other agents, such as an agent posting a goal to resolve a gap. Some of the desired goals can be selected by Maven and intended. Such goals are achieved according to Maven plans. As described above, agents can pursue the goals themselves without Maven intervention. Some of the desired learning goals may be never be intended by Maven and agents do not follow up on them.

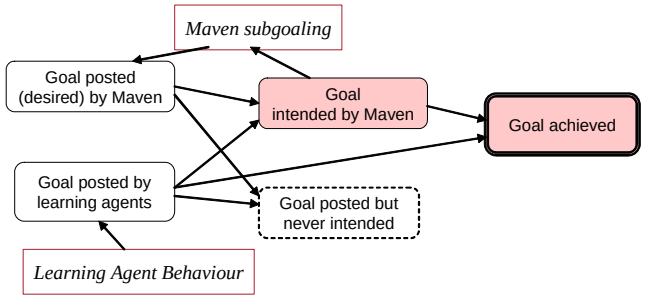

Figure 4: Learning goal lifecycle

Belief KB

The Belief KB in blackboard (BB) represents shared belief of the participating agents. The BB contains the given problem description (a demonstration trace, a top-level task goal and an initial state), and hypotheses that reflect learned knowledge so far. That is, when agents generate new knowledge, they post their results on the BB. Hypotheses can be annotated including how they *supersede* other hypotheses. For example the analysis of experiment results may tell us some step orderings supersede others.

Maven reasoner

The Maven reasoner is responsible for keeping track of sensors for goal trigger conditions and achievement conditions, desired learning goals, and selecting learning goals according to the learning control strategies. The assessment of the overall learning status is performed based on the goals desired and achieved over time, as described below.

Maven procedure <*P, G, C, S*>

While (not *done*) {
  cs ← update_learning_state_with_sensors*(BB,*
                                learning_ history)
  ∀ *g* ∈ *G*
      if $tc_g(cs)$ = true, create_desired_goal (*g, cs)*
  ∀ *a* ∈ desired_goals ()
      if $ac_a(cs)$ = true, set_achieved (*a*)
  if (desired_goals () ={} *or* significant_goals_achieved ())
      done ← true
  p ← prioritize &select (desired_goals(), *S, cs*)
  if (primitive_goal (*p*)) create_AgentTask (*p, C*)
  else *// subgoals desired and intended in following iterations*
      follow the substeps in $plan_p$, and update *beliefKB*
}

In the above procedure, update_learning_state_with_sensors (*BB, learning_history*) updates the learning state according to the sensors employed in *BB*. create_desired_goal (*g, learning_state*) creates a desired goal for *g* with respect to its parameters $Param_g$. desired_goals () finds the current desired goals that are not achieved, and create_AgentTask (*p, agent-capabilities*) creates tasks for agents who have capabilities to achieve *p*. prioritize&select (*goals*, *S, learning_state*) prioritizes desired goals and intend one according to the strategies *S* and *learning_state*. significant_goals_achieved () checks whether knowledge creation goals are achieved and there are no unresolved gaps or conflicts.

The selection of learning goals and AgentTasks to perform depend on the learning control strategies *S* that are employed. Following the strategies that are described above, Maven can pursue knowledge creation goals in the initial phase of learning, and select goals in a coherent manner by following existing plans where possible. Issue resolution goals are prioritized based on the domain methods involved, including how they are related with other domain methods (e.g. superseding relations, how methods are used as sub-methods).

Coordinating activities of learning agents: Examples

In this section, we provide a walkthrough of Maven behavior using examples from the Poirot system.

Initiate learning

When a demonstration sequence Trace1 that performs a medical evacuation task is posted on the blackboard, Maven detects it as the trigger condition of the goal LearnWorkflowFromDemoTrace. As shown in Figure 3, Maven creates a desired goal CG1 with Trace1 and associated evacuation problem descriptions, including the initial state W0 and the task goal PSG1 (move patients before their latest arrival times). CG1 is then intended by Maven and its subgoals (GeneralizeTrace and ExplainTrace) will be desired. Maven then intends the subgoals and creates AgentTasks for them. The following shows how the learning agents accomplish these tasks in Poirot.

Example of bottom-up behavior

WIT and DISTILL perform the task to GeneralizeTrace. Maven monitors BB and recognizes that CG1-1 is achieved. In the meantime, additional learning goals are desired due to knowledge gaps found in the WIT results. The WIT algorithm finds loops and branches but does not identify conditions for branches. Maven does not intend such goals yet since Maven control strategies provide higher priority for the knowledge creation goals during the initial phase of learning. It also recognizes that the current learned knowledge does not cover the whole demonstration trace yet.

At that point, Xplain creates a set of domain methods but produces a failure in explaining some of the steps including why the expert didn't choose the closest airport (K3). Xplain has a capability of generating its own learning goals and share them with other agents. It can pursue the generated learning goal XG1 and resolve it by learning new knowledge K4 for explaining the steps. This type of activities illustrates the bottom-up asynchronous control of the learning process. Note that the shared learning goals are accessible by any agents in the system and agents can proactively pursue learning goals.

Note that K4 supersedes K3 in that K4 resolves issues in K3. Maven keeps track of such superseding relations among the learned knowledge. Maven uses them in prioritizing learning goals. For example, learning goals on validating superseded domain methods are less important than the goals for superseding domain methods.

Example of top-down behavior

Maven notices there are multiple alternative domain methods created for the same trace steps, and triggers a goal for integrating created domain methods. When the knowledge integration goal (IG1) is intended, Stitcher can perform the associated task.

The domain methods created by WIT, DISTILL, and Xplain complement each other. For example, DISTILL produces conditions for branches that are missing in WIT output. On the other hand, DISTILL ignores steps without effect. Stitcher recognizes such gaps and integrates the original methods (K1, K2, K4) into a new domain method set (K5). K5 supersedes K1, K2, and K4. Any learning goals for superseded knowledge are given lower priorites.

Learning phases: validate learned methods with experiment

Both WIT and DISTILL rely on consumer-producer relations among the trace steps (which step produces an

effect and which step uses the result) in deciding step orderings, and some of the causal relations may not be captured. For example, when an LookUpFlight web service call for a patient doesn't return any result, it may cause the expert to find available planes. Maven can initiate validation goals for potential missing causal or step ordering relations.

CMAX has several experiment design strategies for testing different step orderings. Note that depending on the experiment execution result, CMAX may decide either to revise/confirm the orderings or perform further tests. This may involve the same type of goals triggered multiple times to perform tests and analyze the test results. Modified or confirmed orderings supersede existing knowledge.

We have developed a model of cost and benefit of performing an experiment (Morrison and Kim 2008). For example, experiments that cover more expert trace steps may collect more information about the trace. Testing steps that have multiple relevant issues may be more valuable than testing steps with only one issue since the former may address multiple issues with the same experiment. On the other hand, for some experiments, setting the initial state for executing the selected steps is expensive. Maven can use such cost and coverage model for choosing which experiment to perform first.

Assessing progress of learning and determining completion

Once the detected issues are resolved, the system creates workflow using learned domain methods that are not superseded by other alternative domain methods. SHOP2's planning mechanism is adopted in creating workflow that achieves the given problem solving goal PSG1, given the initial state W0. The planner may identify gaps in the learned knowledge, if it cannot produce a complete workflow, and post them as learning goals.

When a workflow is generated, further validation can be done by executing the workflow with the given problem description and comparing the execution results with the provided demo trace. When no further issues are found, no more learning goals will be desired.

Maven can assess the status of learning by keeping track of a) what goals were desired and achieved, b) knowledge that are created and modified over time. For example, when all the significant intended goals are achieved, Maven can announce that learning is 'done'. Maven can also assess whether learning is progressing or not. For learning goals that are repeatedly desired without being achieved, Maven could detect *thrashing* behavior and remove the goal.

Current Status

We have designed meta-level plans for Maven. The initial prototype of Maven was developed on the Spark system that supports a BDI agent model (Morley and Myers 2004). The Maven process is triggered by a demo trace, and it follows a simple plan that decomposes the LearnWorkflowFromDemoTrace into several subgoals (GeneralizeTrace, ExplainTrace, IntegrateKnowledge, CreateWorkflow, and ExecuteWorkflow). In the prototype, Maven applies a very simple control strategy based on the capabilities of the participating agents, and intends the subgoals in a sequence. We are currently extending the control procedure to support more flexible prioritization of learning goals.

Related Work

In models of cognitive systems (both models of human cognition and artificial intelligence systems), memories play critical role in learning and problem solving (Tulving 1983). Especially, metacognitive strategies that promote reflection and self-assessment are known to increase the effectiveness of learning. We are adopting some of the strategies such as in coordinating the activities of integrated learners.

There have been several goal-driven approaches for learning systems (Ram and Leake 1995). Meta-Aqua, Pagoda, and Ivy are examples. Most of these systems focus on behavior of a uniform learning method. Our work generalizes the framework by supporting a wider range of learning methods with different strengths.

Ensemble methods have been used in classification tasks for combining results from multiple learners (Dietterich 2000). These methods show improved accuracy. However, learning complex 'procedural' knowledge requires more diverse capabilities from different agents, and more general strategies for exploiting their capabilities are needed.

Recently there has been increasing interest in control of computation and meta-reasoning in intelligent systems (Anderson and Oates 2007; Raja and Cox 2007). Some of the agent control approaches involve development of utility models or deliberation policies that determine actions taken by agents in an open environment (Russell and Wefald 1991; Schut and Wooldridge 2001). We expect that similar utility models can be developed based on several criteria such as confidence on the knowledge being built and cost of agent tasks, and be used in combination with our existing learning control strategies.

There have been several approaches including programming by demonstration techniques to learn general procedural knowledge (Cypher 1993). However, most of the existing approaches apply a uniform strategy for a limited number of learning methods. As the knowledge to learn becomes complex, with diverse procedural constructs and uncertainties to be validated, the system needs to integrate a wider range of learning methods with different strengths.

Interactive learning systems such as PLOW (Allen et al., 2007) can acquire complex knowledge from user instructions (Noelle 2006). Maven provides a complementary capability in that we systematically pursue different learning activities including knowledge validation, and keep track of learning status with explicit learning goals.

Summary and Future Work

We introduced an approach for coordinating activities of learning agents to acquire complex procedural knowledge. Maven follows a BDI model to support flexible top-down and bottom-up control. Based on the capabilities of participating learning agents and the characteristics of the knowledge to be learned, explicit learning goals and plans for achieving the goals are defined. The history of desired learning goals allows the system to keep track of progress while learning. The system employs a set of high-level learning control strategies for prioritizing learning goals.

Currently annotations on learned domain methods are limited to supersede relations. Learning agents can provide more information including how methods are derived, such as producer-consumer relations used in creating step orderings or assumptions made in finding loops. We plan to investigate how such intermediate results can be shared and facilitate further interaction among agents.

In general, Maven can pursue different strategies in validating knowledge at a given time depending on the cost and benefit of the validation task. For example, tests on the domain methods that are used more often may be preferred than the ones that are less frequently used (Morrison and Cohen, 2007). We plan to explore alternative learning control strategies and analyze how they affect the behavior of the overall system and the quality of learned knowledge. The new strategies will make use of a utility model that analyzes cost and benefit of goal achievement.

Acknowledgement

This research was funded by the DARPA Integrated Learning program with grant number FA8650-06-C-7606, subcontract to BBN Technologies.

References

Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M. and Taysom, W. 2007. PLOW: A Collaborative Task Learning Agent, *Proceedings of the National Conference on Artificial Intelligence*.

Anderson, M. and Oates, T. 2007. A Review of Recent Research in Metareasoning and Metalearning, *AI Magazine*.

Burstein, M., Brinn, M., Cox, M., Hussain, T., Laddaga, R., McDermott, D., McDonald, D. and Tomlinson, R. 2007. An architecture and language for the integrated learning of demonstrations, *AAAI workshop on Acquiring Planning Knowledge via Demonstration*.

Cox, M. T. 2007. Metareasoning, monitoring, and self-explanation. *Proceedings of the First International Workshop on Metareasoning in Agent-based Systems*.

Cox, M. T., and Raja, A. 2007. Metareasoning: A Manifesto, Technical Report, BBN TM-2028, BBN Technologies. www.mcox.org/Metareasoning/Manifesto.

Cypher, A. 1993. Watch what I do: Programming by demonstration. Allen Cypher, Ed. MIT press, 1993.

Dietterich, T. G. 2000, Ensemble Methods in Machine Learning. *Multiple Classifier Systems* 2000: 1-15.

Gil, Y. and Kim, J. 2002. Interactive Knowledge Acquisition Tools: A Tutoring Perspective, *Proceedings of the Annual Conference of the Cognitve Science Society*.

Kim, J. and Gil Y. 2003. Proactive Acquisition from Tutoring and Learning Principles, *Proceedings of AIED 2003*.

Kim, J. and Gil, Y. 2007. Incorporating Tutoring Principles into Interactive Knowledge Acquisiiton, *Int. Journal of Human-Computer Studies*, 65(10).

Morley, D. and Myers, K. 2004. The Spark Agent Framework, *Proceedigs of the International Conference on Autonomous Agents and Multiagent Systems.*

Morrison, C. and Cohen P. 2007. Designing Experiments to Test Planning Knowledge about Plan-step Order Constraints, *ICAPS workshop on Intelligent Planning and Learning*.

Morrison, C. and Kim J. 2008. Meta-Reasoning for Experiment Control, *Internal project report.*

Nau, D., et al. 2005. Applications of SHOP and SHOP2, *IEEE Intelligent Systems*, vol. 20, no. 2, pp. 34-41.

Noelle, D. C. 2006. Learning from advice. In L. Nadel (Ed.), *Encyclopedia of Cognitive Science*. London: Macmillan.

Raja, A. and Cox, M. 2007, *Proceedings of the AAMAS workshop on Metareasoning in Agent-Based Systems*.

Ram A., and Leake, D. 1995.Goal Driven Learning, MIT pr.

Russell, S. and Wefald, E. 1991. Principles of Metareasoning, *Artificial Intelligence* 49 (1-3): 361-395.

Schut, M. and Wooldridge, M. 2001. Principles of Intention Reconsideration, *Proceedigs of the International Conference on Autonomous Agents and Multiagent Systems.*

Tulving, E. 1983. *Elements of episodic memory*. New York: Oxford University Press.

Winner E., and Veloso, M. 2003. DISTILL: Learning Domain-Specific Planners by Example. *Proceedings of International Conference on Machine Learning.*

Yaman, F. and Oates, T. 2007. Workflow inference: What to do with one example and no semantics, *AAAI workshop on Acquiring Planning Knowledge via Demonstration.*