Chapter 6 Goal-directed Metacontrol for Integrated Procedure Learning

Jihie Kim, Karen Myers, Melinda Gervasio and Yolanda Gil

Developing systems that learn how to perform complex tasks presents a significant challenge to the artificial intelligence community. As the knowledge to be learned becomes complex, with diverse procedural constructs and uncertainties to be validated, the system needs to integrate a wide range of learning and reasoning methods with different focuses and strengths. For example, one learning method may be used to generalize from user demonstrations, another to learn by practice and exploration, and another to test hypotheses with experiments. The POIROT system pursues such a multistrategy learning methodology that employs multiple integrated learners and knowledge validation modules to acquire complex process knowledge for a medical logistics domain (Burstein et al., 2008).

For a learning system of such complexity, activities of participating agents must be coordinated to ensure that their collective activities produce the desired procedural knowledge. This kind of control is inherently *metalevel* (Anderson & Oates, 2007; Cox & Raja, Chapter 1) in that it requires the system to reflect on what it is doing and why, to monitor its progress, and to make adjustments to its

behaviour when performance falls short of expectations. Without such introspection, effective coordination and prioritization of the base-level learning and reasoning components would not be possible. This type of introspection corresponds to a form of metareasoning centered on "stepping back" from the system to analyze its behavior, as discussed by Perlis (Chapter 2). As such, it contrasts with the majority of work to date on metareasoning, which has focused on the problem of bounded rationality, as described by Zilberstein (Chapter 3).

Developing a metalevel reasoner for such a complex, integrated learning system poses several challenges, including

- Assessing the progress of learning over time;
- Systematically addressing conflicts and failures that arise during learning;
- Addressing gaps and shortcomings of the individual and aggregate learning results;
- Supporting flexible interactions among agents that pursue different learning strategies.

We describe a metalevel framework for coordinating the activities of a community of learners to create an integrated learning system. The metalevel framework is organized around *learning goals*, which are formulated through introspective reasoning to identify problems and requirements for the ongoing

learning process. These learning goals are posted to a shared blackboard to direct the other components in the system. Goals can be either *process* or *knowledge* oriented.

Process goals define specific tasks to be performed as part of the learning process and are used to coordinate the activities of the various learning and reasoning components. Examples of process goals for task learning include hypothesis creation, hypothesis merging, explanation of observations, and hypothesis validation through experimentation.

Knowledge goals provide the means for a component to convey the need for additional information to further the learning process. In particular, the quality of learned knowledge could be compromised by missing critical information and the efficiency of learning may be impaired by ambiguity arising from insufficient knowledge.

In the succeeding sections of this chapter, we describe the modules within our metalevel framework that are responsible for addressing process (Maven) and knowledge (QUAIL) goals.

Maven (Moderating ActiVitiEs of iNtegrated learners) formulates and achieves metalevel process goals to support integrated learning. Maven's design is based on our prior work on metalevel goals and reasoning for interactive knowledge capture (Kim & Gil, 2007; Gil & Kim, 2002). Maven explicitly

represents plans for achieving learning goals along with high-level strategies to prioritize learning goals. By generating assessment annotations on learned knowledge, Maven keeps track of learning progress and makes decisions on learning goals to pursue (Kim & Gil, 2008).

QUAIL (Question Asking to Inform Learning) addresses knowledge goals by managing a process of selecting and posing questions to a human expert to fill identified knowledge gaps (Gervasio & Myers, 2008; Gervasio et al., 2009). Question selection trades off the utility of missing knowledge with the cost of obtaining it.

Figure 6.1 shows how our goal-oriented metalevel framework for integrated learning maps into a more general model of metareasoning described by Cox and Raja (Chapter 1). The base-level actions correspond to the performance tasks for which procedural knowledge is being learned. Learning occurs at the reasoning level, while the metalevel supports control of learning through two essential mechanisms: metalevel process management (realized by Maven) and metalevel management of learned knowledge (realized by QUAIL). The metalevel influences the components at the reasoning level by posting appropriate learning goals and information to direct their activities.

[Figure 6.1 near here]

Metalevel process management tracks progress toward current learning goals by monitoring the performance of base-level components and their results; it also initiates additional goals to drive the system toward achieving overall learning objectives. Metalevel management of learned knowledge identifies knowledge gaps by introspection over the current state of the learned knowledge, takes actions to eliminate gaps by posing questions to the human demonstrator, and then provides information back to the base-level learners and reasoners to address unresolved knowledge goals. Although not yet supported in QUAIL, conceptually it could also coordinate with the metalevel process management module to initiate additional activity by the base level as a means of addressing knowledge gaps (as opposed to relying solely on the human demonstrator to provide answers).

Background: A Multiagent Learning System

The POIROT system is designed to learn complex process models for a medical logistics domain. 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 attempts 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* that contain step orderings, branches, loops, preconditions, task decompositions, and object selection criteria, adopting the style of hierarchical-task network methods (Ghallab et al., 2004). Several different types of learning approaches (embodied as *agents*) participate in learning these complex workflows in POIROT. In the following list, we describe the capabilities of the agents that were modeled in our system. The details of individual agents are out of the scope of this paper.

- 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 to create 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/Meta-

AQUA uses a set of explanation patterns for building such domain methods (Cox, Chapter 9; Cox & Burstein 2008).

- hypothesis integrators: integrate domain methods created by different learners and detect potential conflicts or ambiguity in the domain methods. Stitcher provides this capability (Burstein et al., 2008).
- workflow constructors: 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 this planning capability (Nau et al., 2005).
- workflow executors: test the constructed workflows by execution. SHOPPER provides this capability (Burstein et al., 2008).
- knowledge validation by experiments: test alternative hypotheses by designing and performing experiments. CMAX (Morrison & Cohen, 2007) provides this capability.

Individual agents, which are described in (Burstein et al., 2008), communicate through a shared 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, primarily web service definitions for primitive actions. To learn complex workflows from such input, the system needs to coordinate the learning agents effectively.

Managing Process Goals: Maven

Maven has explicit representations of process-oriented learning goals and a set of plans to prioritize and accomplish those goals. Maven follows the general BDI (Belief, Desire and Intention) agent model (Rao & Georgeff 1995). In a BDI architecture, *beliefs* represent 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.

External State

As shown in Figure 6.2, the shared knowledge base (a blackboard) where all the agents post results forms an external state for Maven. Maven monitors the results of the participating agents including issues in generating process knowledge. From what it monitors, it forms models of the current workflow knowledge and reasons about what to do next, i.e., what learning goals to generate. Maven *intend*s some of them using a set of goal selection strategies. As an effect, Maven posts intended learning goals that can be achieved by the agents and initiates *AgentTasks*. The results from AgentTasks including new learned knowledge are stored in the shared knowledge base. If issues such as ambiguity or conflicts are found, they are also reported by the agents as a part of the results. The new result can lead to further learning goals.

[Figure 6.2 near here]

Bottom-up control can result when learning agents create new learning goals 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 goals that Maven itself initiated.

Workflow Models

Maven keeps track of hypotheses from the participating agents as *workflow models*. Workflow models represent process knowledge formed or changed by the

agents during generation of domain methods including alternative method hypotheses and their constructs such as step orderings.

Maven creates assessment annotations on individual workflow models with respect to issues found (e.g., unknown conditions for a branch), coverage (e.g., covered medical evacuation scenarios), and validation status (e.g., whether the model was validated with simulation). Maven also relates workflow models using superseding relations (i.e., one model supersedes another) and competing relations (i.e., a set of models for alternatives).

Learning Goals and Plans

Maven uses explicit learning goals and plans for achieving them. Table 6.1 shows some of the process goals and plans used by the system. The set of goals and plans reflects both the capabilities that are supported by the participating agents and the knowledge constructs that need to be learned. They can be extended as new agent capabilities are introduced.

[Table 6.1 near here]

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∈ *G* can have a set of parameters $param_g$ that describes desired goals. For example, the LearnWorkflowFromDemoTrace goal is posted/desired with respect to the demo trace and problem description, and the given background knowledge. Each plan has 1) *trigger conditions* for determining when to execute the plan, 2) *achievement conditions* for detecting achievement of related learning goals, and 3) substeps for achieving the goal. 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 6.3 (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.

[Figure 6.3 near here]

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 6.3 are described below.

Control Strategies for Learning

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 strategies include:

- In the initial phase of learning, prefer domain method creation goals to issue resolution goals;
- For a given piece of knowledge, prefer issue resolution goals to knowledge validation goals;
- When there are multiple issue resolution goals, prefer those for 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 & Gil, 2007; Kim & Gil, 2003; Gil & 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 versus a cautious learner that tests learned knowledge from the beginning and produces more validation goals early on.

Learning Goal Life Cycle

The top-down and bottom-up control cycles imply that learning goals can take several different paths in their life cycle. This is shown in Figure 6.4. 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 never be intended by Maven, and agents do not follow up on them.

[Figure 6.4 near here]

Belief Knowledge Base

The Belief Knowledge Base (KB) in the blackboard (BB) represents the shared beliefs 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. Hypotheses are represented as workflow models and can be annotated including how they *supersede* other hypotheses. For example, the analysis of experiment results may tell us that 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 using goals desired and achieved over time as follows.

Maven procedure <*P, G, C, S*>
While (not *done*) {
  retrieve_new-results *(BB, learning_ history)*
  cs ← update_workflow_models()
  ∀ $g \in G$
    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 *belief KB*
}

In the above procedure, retrieve_new_results (*BB, learning_history*) accesses the results using the sensors employed in *BB*. Maven then updates the learning state by creating and updating workflow models. 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 having capabilities to achieve *p*. prioritize&select (*goals*, *S, learning_state*) prioritizes desired goals and intend a goal 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 depends on the learning control strategies *S* that are employed. Following the strategies 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 submethods).

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 for a medical evacuation task is posted on the blackboard, Maven detects it as the trigger condition of the goal LearnWorkflowFromDemoTrace. As shown in Figure 6.3, 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 GeneralizeTrace. Maven monitors BB and recognizes that CG1-1 is achieved. In the meantime, additional learning goals are desired to address additional learning requirements. In particular, 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 workflow model does not yet cover the whole demonstration trace.

At that point, XPLAIN creates a set of domain methods but produces a failure in explaining some of the steps, including why the expert did not choose the closest airport (K3). XPLAIN has the capability to generate 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 activity 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 in workflow models. 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 that multiple alternative domain methods (i.e., competing workflow models) are 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 (workflow models) 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 workflow models are given lower priorities.

Learning Phases: Validation by 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 a LookUpFlight web service call for a patient does not 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 also have developed a model of cost and benefit of performing an experiment for deciding which experiment to perform.

Assessing Progress of Learning and Determining Completion

Once the detected issues are resolved, the system creates a 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. If the planner cannot produce a complete workflow, then it will post an appropriate learning goal to address the source of the problem.

When a full 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 and b) workflow models representing knowledge that is created and modified over time. For example, when all the significant intended goals are achieved, Maven can announce that learning is 'done'.

Question Asking for Knowledge Goals

We formulate the question-asking process in terms of three separate activities: *question nomination, question selection,* and *question posing.*

Question nomination refers to the generation of questions that can potentially provide value to the learning process. Within the POIROT framework, question generation is performed by the individual hypothesis formers and evaluators, as they are best positioned to determine their information needs.

Question selection is the process of choosing from among nominated questions those that should be posed to the user. The creation of appropriate selection strategies lies at the heart of our work and is discussed further below.

Question posing refers to the interactions with the user to obtain answers. Relevant issues include modality (not addressed in this paper) and timing (discussed below).

Question Catalog

We have defined a catalog of questions to inform the learning by demonstration process, details of which can be found in (Gervasio & Myers, 2008). The catalog covers areas such as the function and causality of elements in the demonstration trace, abstraction, alternatives and justifications, limitations on learned knowledge, and the process of learning. The question catalog was derived from an analysis of the general task-learning process. We identified the different types of generalizations required to induce a general set of problem-solving methods from the trace of a successful execution of a plan for a specific problem and then

systematically analyzed the demonstration trace and the generated hypotheses to identify the relevant portions about which questions may arise.

Question Selection Strategies

Question selection strategies must balance the need for information to further the learning process with the following considerations.

A. Individual questions contribute different levels of value to the learning process.

B. Answering questions imposes a burden on the user.

C. The role of question asking is to inform learning, rather than to replace it by a knowledge acquisition process.

These considerations lead naturally to the formulation of the question selection problem in terms of a cost-benefit analysis, drawing on models of *utility* and *cost* for individual questions.

Utility Model for Questions

The *base utility* of a question is determined by the component that nominates the question. The factors that affect base utility are highly dependent on the specifics of the nominating component. For example, learning components might measure utility in terms of the expected change in the theoretical lower bound on the

number of examples needed to learn the target concept. Components faced with ambiguity might factor in the expected reduction in the number of valid alternative hypotheses.

Since our framework involves a community of learners, factors beyond base utility must be considered to determine overall question utility. First, base utility estimates must be normalized to make them comparable across components. Second, the relative importance of contributions from different components must be considered. Finally, multiple learners may benefit from the answer to a given question.

Given these considerations, we propose to model overall question utility relative to a set L of learners as follows:

$$Utility\ (q) = \sum_{l \in L} w_l \times Utility(q,l)$$

$$Utility\ (q,l) = w_B \times BaseUtility(q,l) + \quad w_{LG} \times LearningGoalUtility(q,l)$$

The utility of a question for an individual learner l, denoted by $Utility\ (q,l)$, is defined to be a weighted sum of the base utility $BaseUtility(q,l)$ assigned by the learner and the utility $LearningGoalUtility(q,l)$ of the associated learning goal that motivated the question by the learner. Here, $BaseUtility(q,l) \in [0,1]$ while $w_B + w_{LG} = 1$. The overall utility for question q, denoted by $Utility\ (q)$, is a weighted sum of the utilities assigned by the individual learners, with the w_l encoding the

relative weightings assigned to the individual learners subject to the constraint that $\sum_{l \in L} w_l = 1$.

Cost Model for Questions

Prior work on mixed-initiative systems has identified five cost factors for consideration by an agent when deciding whether to interact with a user (Cohen et al., 2005):

1. Inherent difficulty of the question
2. Level of disruption given the user's current focus of attention
3. User's willingness to interact with the system
4. Timing of the interaction
5. Appropriateness of the question

Within a learning by demonstration setting, the user would be expected to focus on interacting with the system and be tolerant of questions that may seem ill motivated or have obvious answers. As such, we can disregard factors (2) through (5) above, focusing instead on the inherent difficulty of the question as the basis for the cost model.

With this perspective, the cost model should measure the cognitive burden incurred by the expert in answering the question. This can be an arbitrarily complex quantity to measure, involving not only readily observable factors such as time to answer and brevity of answer but also potentially more complex

metrics such as difficulty in understanding a question. However, we can use certain heuristics to approximate cognitive burden. Typically, certain question formats will be easier to answer than others. For example, yes/no questions usually will be less costly to answer than multiple-choice questions, which in turn would be less costly than open-ended questions.[1] Similarly, a question grounded in the demonstration trace most likely will be easier to answer than the same question asked about a hypothetical situation.

With these observations in mind, we define the cost of a question by

$$Cost(q) = w_F \times FormatCost(q) + w_G \times GroundednessCost(q)$$

where $FormatCost(q)$ denotes the cost associated with the format of q, and $GroundednessCost(q)$ is either $C_{concrete}$ or C_{hypo}, depending on whether the question relates to concrete elements in the demonstration trace or a hypothetical situation. Both $FormatCost(q)$ and $GroundednessCost(q)$ are constrained to lie in the interval [0,1]. The weights w_F and w_G allow relative weighting of the two cost factors, subject to the constraint that $w_F+w_G=1$.

Control Strategies for Question Posing

We consider two types of control for managing when to pose questions: *asynchronous* and *synchronous*.

Asynchronous Control. An asynchronous control strategy lets questions be asked continuously during the learning process. Asynchronous strategies could possibly lead to faster learning as they would enable early elimination of incorrect or irrelevant hypotheses, leading to a more focused search. For example, questions to resolve substantial ambiguity during learning may better be asked as they arise, rather than waiting until a complete initial hypothesis has been formed.

However, asynchronous control complicates the management of question asking, since the decision of whether or not a question is worth asking has to be made without knowledge of any other questions and possibly even without a hypothesis space against which the value of a component's contribution can be measured. Management of continuous questioning has been considered previously. Techniques to address this problem generally can be categorized as *backward* or *forward looking*.

Backward-looking approaches incorporate historical information about previous interactions into their cost models, as a way to prevent question overload. For example, (Cohen et al., 2005) uses cost-benefit analysis to determine when the expected utility increase for asking a question outweighs the associated costs. The cost model includes both the associated *base bother cost* for a question and an *accumulation bother cost* derived by summing over costs

associated with prior question initiations, scaled by a decay factor that takes into account the temporal distance from prior questions to the present.

In contrast, forward-looking approaches use Markov Decision Processes (MDPs) to reason about future expected costs and reward in order to determine when to initiate interactions. Forward-looking approaches have been used to support questions about both action selection strategy (e.g., Scerri et al., 2002) and learning user preference models (e.g., Boutilier, 2002).

Forward-looking approaches require detailed models of the likelihood and utility of expected question outcomes, which will be difficult to obtain in practice. For this reason, a backward-looking approach has greater appeal for our question-asking framework.

Synchronous Control. A synchronous control strategy lets questions be asked only at a fixed point during the learning process. Synchronous control may sacrifice some learning efficiency as components may not be able to get critical clarifications as early as desired. But it could potentially lead to better use of the resources for question asking as the questions could be considered in groups, all pertaining to some stable state of the hypothesis space.

The synchronous question selection problem can be formulated as follows. We assume the following functions defined for a collection of questions Q.

$$Cost(Q) = \sum_{q \in Q} Cost(q)$$

$$Utility(Q) = \sum_{q \in Q} Utility(q)$$

As noted above, question selection imposes a burden on the demonstrator. Furthermore, our goal is to provide question asking in *support of learning,* rather than devolving into a knowledge acquisition process that obtains extensive procedure knowledge through system-initiated interactions. For these reasons, we impose a budget on question asking to restrict access to the user.

Definition (Synchronous Question Selection). Given a collection of questions $Q=\{q_1 \ldots q_n\}$ and a budget B, determine a subset $Q' \subseteq Q$ with $Cost(Q') \leq B$ such that there is no $Q'' \subset Q$ for which $Cost(Q'') \leq B$ and $Utility(Q'') > Utility(Q')$.[2]

This framing of synchronous question selection maps directly to the *knapsack* problem (Kellerer et al., 2005). Although the knapsack problem is NP-complete, dynamic programming algorithms can generate solutions that run in time $O(nB)$. Given a reasonable number of nominated questions and budget, we anticipate acceptable performance.

Synchronous question selection could be applied at the end of the learning process, thus enabling individual learners to refine their initial hypotheses prior to generation of the final learning output. Another possibility is to support a fixed number of synchronous question selection sessions during the learning process,

thus enabling the effects of the answers to be propagated through the system. The available budget for question asking would be distributed across the different sessions, in a manner that provides the most benefit to the learning process. This *multiphase synchronous question selection* provides some of the benefits of asynchronous question selection but with simpler control and better-understood selection methods.

Current Status

Initial prototype versions of Maven and QUAIL have been designed and implemented to support metalevel control of integrated learning for the POIROT system.

The initial prototype of Maven was developed with explicit hierarchical relations among goals and subgoals in ontologies of goal trees using OWL (OWL, 2009). Maven (1) assesses learning status using workflow models that keep track of knowledge generated from agents, (2) triggers learning goals based on the assessment, (3) selects learning goals by applying several control strategies, and (4) pursues selected learning goals using metalevel plans. Whenever new results are posted on the blackboard by the agents, Maven follows these steps and creates new tasks for the agents. The prototype supports a simulation of various agent activities during workflow learning including hypothesis generation, hypothesis

integration, and hypothesis validation through experiment. We plan to investigate different learning behaviors using simulation results.

Our initial QUAIL prototype implements a synchronous strategy for question selection. We are using this prototype to understand better how to model question utility and costs, how to distribute resources across multiple-phase synchronous question asking, and how learning performance (speed, quality) can be improved through appropriate question asking. Our initial focus has been on instantiating the question-asking framework for a particular type of learner, specifically a lexicographic preference learner (Yaman et al., 2008). Experimental results show that, generally speaking, judicious question asking can improve learning performance (Gervasio et al., 2009). However, the results make clear the importance of understanding the value of different types of information for learning in different contexts.

Related Work

In models of cognitive systems (both models of human cognition and artificial intelligence systems), memories play a 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 these strategies for coordinating the activities of integrated learners.

Goal-driven approaches for learning systems (Ram & Leake, 1995) include Meta-Aqua (Cox & Ram, 1999), Pagoda (desJardins, 1995), and Ivy (Hunter, 1990). While most of these systems focus on a uniform learning method, our work supports a wider range of learning methods with different strengths.

Ram and Hunter (1992) introduce the notion of explicit knowledge goals to capture gaps in the system's knowledge, defining knowledge goals as comprising both the specification of the missing knowledge and the task enabled by it. In addition, they propose augmenting knowledge goals with a utility measure to help drive the inference process.

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 metareasoning in intelligent systems (Anderson & Oates, 2007; Cox, 2005; Raja & 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 & Wefald, 1991; Schut & 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.

Unlike in other metacontrol approaches that control learning with a set of performance measures (as described by Epstein and Petrovic in Chapter 4), in our framework, the learning is driven by learning goals that represent gaps and issues in generating complex workflow knowledge. The goals are mapped to various capabilities that participating agents can support.

Summary and Lessons Learned

We introduced a goal-oriented approach for metalevel coordination of learning agents designed to acquire complex procedural knowledge. This framework employs 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. Question-asking capabilities enable knowledge goals to be addressed by exploiting the knowledge of the domain expert in order to fill gaps in learned knowledge.

The metareasoning that we employ for coordinating base-level reasoning and learning components is a form of introspection designed to enable a system to analyze and then adapt its behavior at execution time in order to improve overall performance. Our experience on this effort has validated our belief that this kind of introspective metareasoning is essential for effective problem-solving in complex systems, and that more research is required to understand how to manage this kind of control process effectively. The focus for most work on metareasoning to date, by contrast, has emphasized the related but distinct problem of bounded rationality.

Our goal-directed approach combines selection from among predefined problem-solving strategies (for process goals) with question asking managed by cost/benefit analysis (for knowledge goals). While our work has shown that these approaches can be effective for metalevel control of a complex system, it proved more difficult than anticipated to formulate the domain-specific background models necessary to support these methods, namely the control logic for process selection and the utilities for question asking. One important direction for future work is to enable development of these models in a more flexible and less time-consuming manner. For example, the control logic for process selection could be derived from high-level principles, rather than being explicitly hardwired. Such principles would be grounded in objectives for learning procedural knowledge, such as coverage, coherence, and confidence; general-purpose strategies would

define approaches for achieving and trading off these objectives that could be tailored to specific procedure learning situations. Utility models for question asking could be learned via experimentation that would assess the effectiveness of different questions in different contexts.

Another area for future work is to generalize the notion of processing for knowledge goals to support mechanisms other than posing questions to the expert user. In particular, the experimentation performed by CMAX could potentially be applied to address knowledge gaps, as could the invocation of other learning mechanisms.

Currently, annotations on workflow models are limited to supersede and compete relations. Learning agents can provide more information including how methods are derived, such as the 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.

We expect that our metacontrol framework can be useful for capturing procedural knowledge or workflow knowledge in other applications. For example, interactive acquisition of process models requires reasoning on user-provided process information. To provide useful guidance, the system needs to keep track of issues and gaps in the knowledge being built (Kim et al., in press). Some of our

learning goals, both process and knowledge goals, can be mapped to the issues that arise during such an acquisition process and our metacontrol approaches may be useful in driving the interaction with the user.

Acknowledgments

This work was supported by the Defense Advanced Research Projects Agency and the United States Air Force through BBN Technologies Corp. on contract number FA8650-06-C-7606.

References

Anderson, M. & Oates, T. (2007). A review of recent research in metareasoning and metalearning. *AI Magazine*, *28*(1), 12-16.

Boutilier, C. (2002). A POMDP formulation of preference elicitation problems. *Proceedings of the 18th National Conference on Artificial Intelligence*, (pp. 239-246). Menlo Park, CA: AAAI Press.

Burstein, M. H., Laddaga, R., McDonald, D., Cox, M. T., Benyo, B., Robertson, P., Hussain, T., Brinn, M., & McDermott, D. (2008). POIROT - Integrated learning of web service procedures. In *Proceedings of the 23rd National Conference on Artificial Intelligence* (pp. 1274-1279). Menlo Park, CA: AAAI Press.

Cohen, R., Cheng, M., & Fleming, M. W. (2005). Why bother about bother: Is it worth it to ask the user? In *Proceedings of the AAAI Fall Symposium on Mixed-Initiative Problem-Solving Assistants* (pp. 26-31). Menlo Park, CA: AAAI Press.

Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence*, *169*(2), 104-141.

Cox, M. T., & Burstein, M. H. (2008). Case-based explanations and the integrated learning of demonstrations. Künstliche Intelligenz (Artificial Intelligence,) 22(2), 35-38.

Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence, 112*(1-2), 1-55.

desJardins, M. (1995). Goal-directed learning: A decision-theoretic model for deciding what to learn next. In A. Ram & D. Leake (Eds.), *Goal-driven learning* (pp. 241-249). Cambridge, MA: MIT press.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the Third International Workshop on Multiple Classifier Systems* (pp. 1-15). New York: Springer Verlag.

Ghallab, M., Nau, D. & Traverso, P. (2004). Hierarchical Task Network Planning. Automated Planning: Theory and Practice. Morgan Kaufmann.

Gervasio, M. & Myers, K. (2008). Question asking to inform procedure learning. In M. T. Cox & A. Raja (Eds.), *Proceedings of the AAAI Workshop Metareasoning: Thinking about Thinking* (pp. 68-75). Technical Report WS-08-07. Menlo Park, CA: AAAI Press.

Gervasio, M., Myers, K., desJardins, M. & Yaman, F. (2009). Question asking for preference learning: A case study. In *Proceedings of the AAAI Spring Symposium on Agents that Learn from Human Teachers* (pp. 56-62). Menlo Park, CA: AAAI Press.

Gil, Y. & Kim, J. (2002). Interactive knowledge acquisition tools: A tutoring perspective. In *Proceedings of the Annual Conference of the Cognitive Science Society* (pp. 357-362). Mahwah, NJ: Lawrence Erlbaum Associates.

Hunter, L. E. (1990). Planning to learn. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 261–276). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Kellerer, H., Pferschy, U., & Pisinger, D. (2005). *Knapsack problems*. Berlin: Springer Verlag.

Kim, J., Gil, Y., & Spraragen, M. (in press). Principles for interactive acquisition and validation of workflows. To appear in *Journal of Experimental and Theoretical Artificial Intelligence.*

Kim, J. & Gil, Y. (2008). Developing a meta-level problem solver for integrated learners. In M. T. Cox & A. Raja (Eds.), *Proceedings of the AAAI Workshop, Metareasoning: Thinking about Thinking* (pp. 136-142). Technical Report WS-08-07. Menlo Park, CA: AAAI Press.

Kim, J. & Gil, Y. (2007). Incorporating tutoring principles into interactive knowledge acquisition. *International Journal of Human-Computer Studies*, *65*(10), 852-872.

Kim, J. & Gil, Y. (2003). Proactive acquisition from tutoring and learning principles. In *Proceedings of the International Conference on Artificial Intelligence in Education*, (pp. 175-182). Amsterdam: IOS Press.

Morrison, C. & Cohen P. (2007). Designing experiments to test planning knowledge about plan-step order constraints. In *Proceedings of the ICAPS Workshop on Intelligent Planning and Learning,* (pp. 39-44*)*. Menlo Park, CA: AAAI Press.

Morrison, C. & Kim J. (2008). Meta-reasoning for experiment control. *Personal Comunication*.

Nau, D., Au, T., Ilghami, O., Kuter, U., Munoz-Avila, H., Murdock, J., Wu, D., & Yaman, F. (2005). Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, *20*(2), 34-41.

OWL. (2009). Web Ontology Language, http://www.w3.org/TR/owl-features

Raja, A. & Cox, M. (Eds.) (2007). *Proceedings of the Workshop on Metareasoning in Agent-Based Systems*, International Conference on Autonomous Agents and Multiagent Systems. AAMAS-07.

Ram, A. & Hunter, L. (1992). The use of explicit goals for knowledge to guide inference and learning. *Journal of Applied Intelligence*, *2*(1), 47-73.

Ram A. & Leake, D. (1995). *Goal-driven learning,* Cambridge, MA: MIT press.

Rao A. & Georgeff, M. (1995). BDI-agents: from theory to practice. Proceedings of the First International Conference on Multiagent Systems, San Francisco, 1995.

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

Scerri, P., Pynadath, D., & Tambe, M. (2002). Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, *17*, 171-228.

Schut, M. & Wooldridge, M. (2001). Principles of intention reconsideration. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (pp. 340-347). New York: ACM.

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

Winner E., & Veloso, M. (2003). DISTILL: Learning domain-specific planners by example. In *Proceedings of the International Conference on Machine Learning* (pp. 800-807). Menlo Park, CA: AAAI Press.

Yaman, F., Walsh, T.J., Littman, M.L., & desJardins, M. (2008). Democratic approximation of lexicographic preference models. In *Proceedings of the International Conference on Machine Learning* (pp. 1200-1207). Menlo Park, CA: AAAI Press.

Yaman, F. & Oates, T. (2007). Workflow inference: What to do with one example and no semantics. In *Proceedings of the AAAI Workshop on Acquiring Planning Knowledge via Demonstration* (pp. 46-51). Menlo Park, CA: AAAI Press.

(a) Sample learning goal types

-- KNOWLEDGE CREATION --
GeneralizeTrace
ExplainTrace
CreateWorkflowWithDomainMethods
-- ISSUE IDENTIFICATION --
IdentifyOrderingAmbiguity
IdentifyUnexplainedSteps
-- ISSUE RESOLUTION --
ResolveAmbiguousStepOrderingHypotheses
ResolveUnknownBranches
…
-- KNOWLEDGE VALIDATION --
ValidateWorkflowKnowledge
  EnsureWorkflowGeneratability
  EnsureTraceReproducibility
  ValidateKnowledgeWithExperiments

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*)
  FindAppropriateStepOrderingHypos to confirm or modify *orderings*, or suggest more experiment
- Achievement condition: Step orderings modified or confirmed

Plan:CreateWorkflowWithLearnedMethods
(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 …

(b) Sample Maven plans for achieving learning goals

---

[1] Question format and cognitive load are not perfectly correlated (e.g., the halting problem constitutes a particularly difficult yes/no question). Our question catalog has been designed to enforce this correlation, with simple question formats used only for questions with low expected cognitive load.

[2] This formulation of the problem assumes that the questions in Q are independent of each other, i.e., obtaining the answer to one question does not impact the utility of answering the others.