A Theory of Reflective Agent Evolution

J. William Murdock
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Ph.D. Thesis Proposal
Technical Report GIT-CC-98-27
12/9/98

Abstract

Intelligent agents typically operate in complex, dynamic environments. Such environments require that an agent be able to adapt to meet new demands. One promising strategy for such adaptation is to have an agent reason about itself using an explicit model of the tasks it performs, the methods it uses to perform those tasks, and the knowledge which those tasks and methods manipulate. I propose to investigate the application of this reflective self-modification process to agents situated in interactive domains. This investigation will consist of three major components: (i) implementing an agent architecture which provides autonomous support for model-based evolution, (ii) instantiating a variety of specific agents into this architecture, and (iii) systematically varying both the descriptions of the agents and the design of the architecture itself. I expect this research to provide insight into the nature of agent evolution and the potential role of model-based reflection.

1 Motivation

The real world is complex and dynamic. Much of traditional research in Artificial Intelligence and Cognitive Science has focused on highly simplified task domains. Recently, however, there has been a growing interest across many disciplines in the subject of reasoning in less restricted environments. One aspect of this interest is the *intelligent agent* paradigm, e.g., [Johnson & Hayes-Roth, 1997]. This approach typically involves systems which do autonomous reasoning in highly interactive situations. Such interactivity forces systems to respond to an external environment which can radically change over time as a result of the behaviors of human users and other intelligent agents.

Because the environment in which an agent operates does change, it is often necessary for the agent to *evolve* in response to these changes. The task of agent evolution raises questions of the knowledge used to facilitate evolution and the methods by which this knowledge is applied. There are many kinds of knowledge which may be of use in improving an agent's performance in some environment. Examples of useful areas of knowledge include the domain, the specific environment, the nature of other agents in the environment, etc. Because the task of evolution involves modification of the agent's reasoning mechanism, it seems reasonable to hypothesize that knowledge about that mechanism is particularly useful. For example, the web browsing agent learning about new external browsers would benefit from knowing that the overall goal is to display a file, that external browsers perform display actions, etc. Thus processes of *reflection*, i.e., a system's reasoning about itself, may be particularly valuable in supporting agent evolution.

AUTOGNOSTIC [Stroulia, 1994, Stroulia & Goel, 1994, Stroulia & Goel, 1996] and SIRRINE [Murdock & Goel, 1998] are two closely related systems which do reflective reasoning. These systems are based on a theory that elements of cognition can be viewed as abstract devices. Thus reasoning systems can be thought of as made up of distinct components which combine to produce the overall behavior of the system. From this perspective, reflective evolution becomes a redesign process: given a design of a system and a new requirement, the goal is to produce a modified design which satisfies that new requirement. Thus the mechanisms of evolution in these systems are inspired by earlier work on the design of electrical and mechanical devices [Goel, 1989, Goel et al., 1997b]. In particular, AUTOGNOSTIC and SIRRINE resemble earlier work on engineering design in that they use models which are functional (i.e., that take an intentional stance toward describing the effects of a system), causal (i.e., that show the mechanisms by which effects occur), and compositional (i.e., that show how the effects of elements are combined). This combination of capabilities has been shown to provide substantial leverage in model-based reasoning.

AUTOGNOSTIC uses these models for failure-driven learning; it makes changes to systems in response to specific, individual failures to accomplish a goal in some context. SIRRINE extends this paradigm into more agent oriented domains, and demonstrates evolution in response to more strongly interactive environments. However, SIRRINE too is largely oriented toward changes in response to particular system failures. I claim that these sorts of changes are a specific subset of the broader class of kinds of evolutionary development that may be supported by reflective reasoning.

A particularly interesting and valuable type of evolutionary development is the addition of *new capabilities* to a system. A system can be built with arbitrarily many capabilities during its initial construction, but a sufficiently dynamic environment will always demand more features and provide more opportunities. For example, consider an agent whose function is to retrieve and present information from the World Wide Web (i.e., a web browser with the intelligent reasoning capabilities). Web browsers typically have a certain set of file types which they are, themselves, capable of displaying and another (generally much larger) set of types for which they have access to external display programs. However, an agent browsing the web is likely to eventually encounter a type of file which for which it does not know of an external display program. Most recent web browsers (e.g., Netscape Navigator 4.x and Microsoft Internet Explorer 4.x) have built in mechanisms automatically configuring new viewers. If, however, this agent did not have such a feature, it would have to learn about new external viewers. There are many ways that such learning could occur. The agent could be given feedback regarding what it should have done after failing to display a file ("you should have run this viewer"). The agent could be given advance notification of the desired behavior ("when I click on this, I want you to run this viewer"). The agent can also learn by observation or by trial and error. Any of these approaches could be used to allow the agent to evolve to meet the demands of the unfamiliar file type.

The research proposed here is intended to extend the theories of reflective model-based reasoning developed in existing research to address the specific goal of adding new capabilities to an existing agent. This goal immediately raises a variety of topics. Figure 1 shows an overview of the highest level topics to be addressed in this research. The three major themes which I intend to study are agency, evolution[1], and reflection. The theme of evolution arises

[1]It is important to distinguish evolution here from the traditional AI subject of *learning*. The process of evolution is certainly a learning process, in that the system changes itself. However, evolution may encompass processes which are outside the boundaries of traditional studies of learning. An agent learns within the context of performing some task, and may require substantial analysis to decide what to learn and how. The context of reasoning and sometimes even the analysis have often been construed as outside of learning, *per se*. The research proposed here, like much of recent work in AI, does not view learning as a separate component but rather considers processes which operate across the traditional boundaries between learning, problem solving, planning, decision making, etc. The use of the term *evolution* here is essentially derived from the use of the term in the field of software engineering, to describe the entire process by which an existing software system is transformed into a new one.

from the need of agents to react to the dynamic natures of their environments. The theme of reflection arises from the demands of evolutionary reasoning for knowledge to act upon.

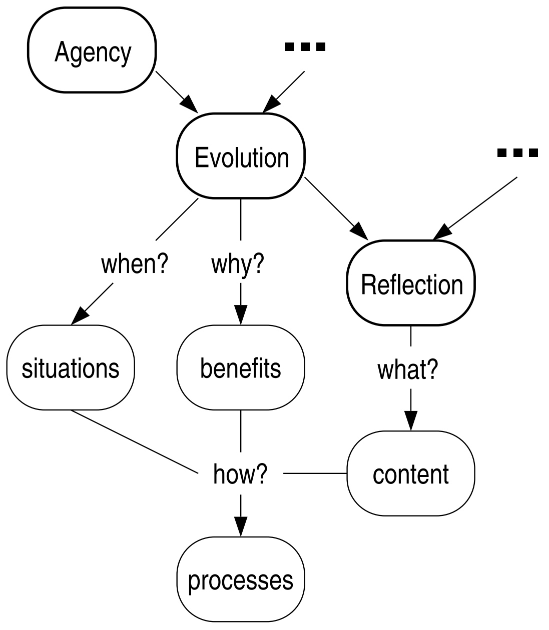

Figure 1: These are the key relationships among the major research topics. Note that the ellipses in the figure indicate that there are other contexts in which evolution and reflection are also important. Evolution may be crucial even in relatively static, traditional AI domains if a perfect algorithm is not known in advance. Similarly, reflection has a wide variety of uses outside the context of evolution.

As Figure 1 illustrates, the themes of agency, evolution, and reflection can be further analyzed as raising a variety of questions. Two crucial topics that directly arise from the notion of evolution are:

- *When* can evolution occur? What are the situations in which evolution is possible? For example, a web browsing agent may learn about a new external viewer after it fails to display a relevant file. It may also learn about such a viewer in advance by anticipating what documents a user will request. In this research, I will focus on evolution which occurs both before and after the execution of an agent.

- *Why* should evolution occur? What are the benefits to performing evolution? The external viewer example demonstrates the addition of a new capability. Evolution may also support improvement of the speed or quality of existing capabilities, etc. In this research, I will focus on the addition of new capabilities.

The idea of reflection raises an additional core topic:

- *What* is knowledge of self? What is the content of reflective knowledge? A web browsing agent might know what it does (presenting information from the web), how it does it (by retrieving documents and then displaying them on the screen), under what conditions it is useful (when connected to a network), etc.

The application of the mechanism of reflection to challenge of evolution raises an additional topic:

- *How* can reflective evolution occur? Given some reflective content and a set of evolutionary situations and benefits, what are some effective processes that enable evolution? For example, a web browser might add a new viewer by performing analogical transfer from the mechanisms which activate the existing viewers.

The topics of situations, benefits, content, and processes all constrain each other. For example, if a desired benefit is an improvement in speed, it may be necessary for the content to include information about the speed of various portions of the agent. The topics of processes and content are particularly tightly related; decisions regarding what content to include are made on the basis of what the processes require and decisions regarding what processes are feasible are made on the basis of what the content can support. The research proposed here involves the study of all four of these topics, each in the context of the others.

2 Example

TIM, or Tiny Intelligent Mosaic, is a simple simulated web browsing agent which is intended to initially imitate the behavior of the Mosaic web browser and to intelligently evolve new capabilities. TIM has been used as an example in SIRRINE and we expect to make further use of it in the proposed research.

The work on TIM in SIRRINE has focused on the external browser configuration problem. Consequently, the existing model represents issues relating to the selection and invocation of external browsers at a moderate level of depth. In contrast, the system deals with most other issues (such as receiving input from the user and accessing the network) in the minimum amount of depth needed to provide a context for the external browser portion of the model. As I investigate other issues with TIM, I expect to further develop the TIM model.

The initial task-method structure for TIM is shown in Figure 2. It was built by imitating both the functionality and structure of Mosaic for X Windows, version 2.4. At the onset of the model-building we had a basic understanding of the primary mechanism: the access and display of files from the internet; this understanding came from a variety of sources including use of the system, a cursory examination of the source code, and a preliminary analysis developed using the SAAM methodology [Abowd et al., 1997]. Our initial understanding was that the system involved the following steps:

- A URL is provided by the user (by typing into an "Open URL" dialog box, by clicking on a link, or by selecting an item from the "Hotlist").
- That URL is divided into pieces and those pieces are used to issue a request to the network.
- A MIME tag is extracted from the response generated by the network.
- If the MIME tag corresponds to a type which can be displayed internally (e.g., HTML), the system does so.
- Otherwise, if the MIME tag corresponds to a type for which an external browser is known, that external browser is invoked on the document retrieved.

The task-method decomposition illustrated in Figure 2 follows directly from this analysis. Some portions of the model are extremely simplified; for example, the task *communicate-with-server* is represented as a primitive step even though it covers the entire (moderately complex) task of requesting and receiving a document from the internet. This simplification is acceptable for our purposes because our focus is on external browsers, not on network communication. If we later wanted to use the model to address some issues relating to this communication, we could simply further decompose this task into lower level methods and subtasks.

In a problem given to SIRRINE, TIM is asked to retrieve the document at the following URL:

http://www.cc.gatech.edu/grads/m/Bill.Murdock/dcsp.pdf

This document is an Adobe Portable Document Format (PDF) file of MIME type application / pdf. Mosaic 2.4, on which TIM is based, does not have a mapping for this MIME type to an external browser program. Consequently, TIM fails for this task. SIRRINE then receives feedback from the user indicating that the command `acroread ~a` is appropriate to this problem. SIRRINE uses this knowledge to evolve TIM so that it is capable of invoking this command when it encounters documents of that MIME type.

The main reasoning process which SIRRINE uses to perform evolution in this example consists of the following steps:[2]

[2]At the most abstract level, the process described in this example (execute, blame, repair, execute) is the same as the main process used in AUTOGNOSTIC. In the work proposed here, I intend to explore a range of processes, of which this is one.

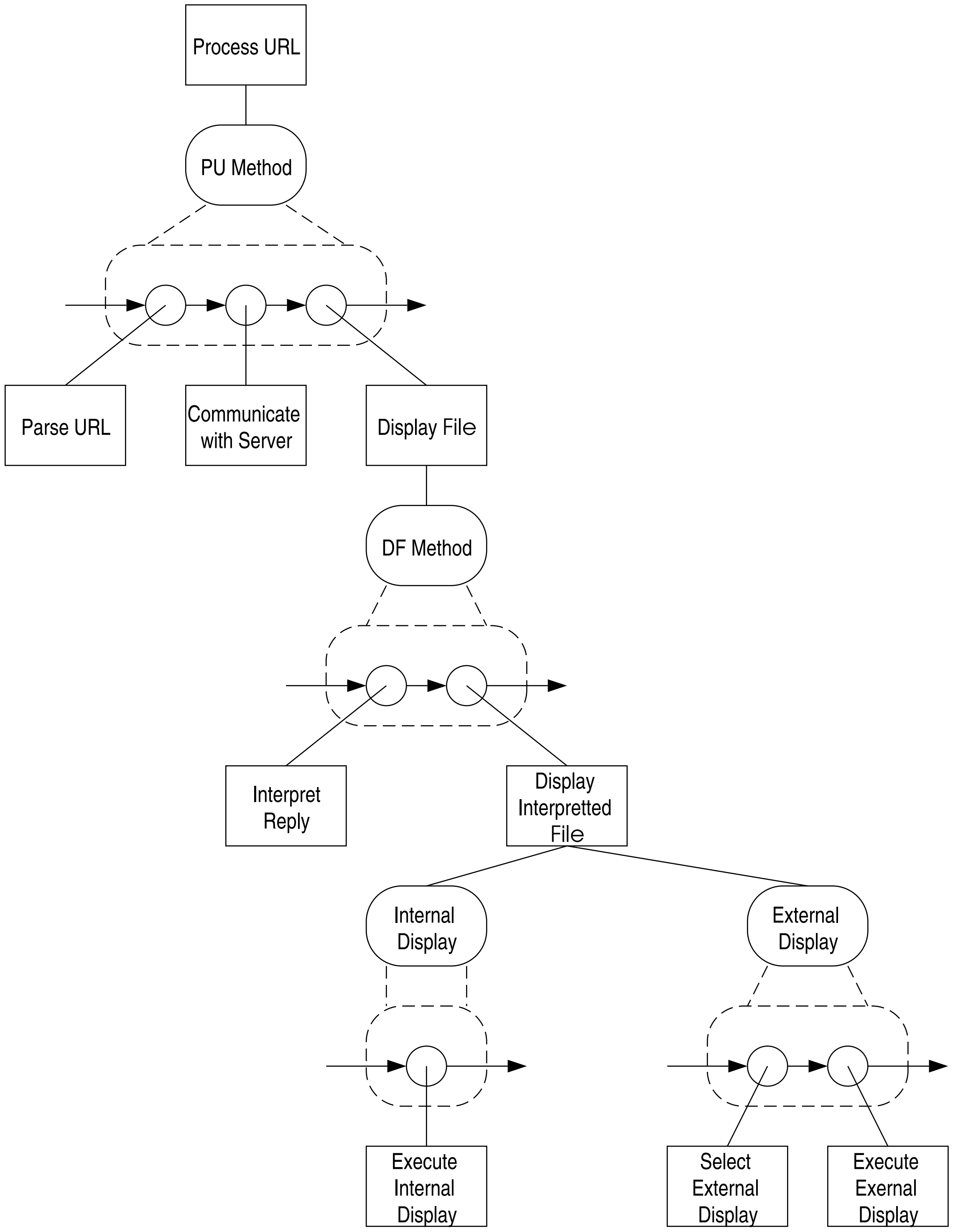

Figure 2: A model of TIM. Rectangular boxes indicate tasks while rounded boxes indicate methods. Lines from tasks down to methods represent the availability of methods to accomplish a task. Transition diagrams underneath methods indicate the control imposed by methods on their subtasks. Lines from the elements in the transition diagrams down to subtasks show that the indicated subtask accomplishes the indicated transition.

Execution: TIM runs and attempts to solve the problem as specified. During execution, a trace of reasoning is generated. The trace stores the tasks and methods executed and the knowledge states which act as input and output to those tasks and methods. Figure 3 shows a trace for TIM for the PDF example.

Blame Assignment: If TIM fails, as it does in the example, SIRRINE attempts to identify a particular procedural element (task or method) at which the failure may have occured. The order in which tasks and methods are analyzed is from top to bottom (i.e., most abstract to least abstract in order to prefer more general solutions where possible) and from right to left (i.e., from last executed to first executed in order to prefer analyzing more proximal causes of the failure). In general this process returns a list of possible assignments of blame since there may be many possible causes for failure. Figure 4 illustrates the order in which tasks and methods are analyzed in the PDF example. The element of particular interest in this example is the select-external-display task. This task should have produced a command but didn't; because SIRRINE was given a command as feedback, it posits that the failure of the system may be a result of this task not producing that command.

Repair: Given a particular assignment of blame, the repair mechanism is intended to make an alteration to the system which corrects the identified fault. The repair mechanism chooses a potential assignment of blame from the list produced by the blame assignment mechanism in the order in which they were assigned; i.e., the repair mechanism uses the same heuristics of abstraction and proximity that the blame assignment mechanism does to order the selection of repair candidates.[3] In the example, the select-external-display task is the first task for which the blame assignment mechanism has identified a failure for which the repair mechanism has an applicable strategy. Since the select-external-display task is implemented as a simple look-up table, the repair mechanism is able to perform the repair by simply adding another element to the table: one which maps MIME type application / pdf to command acroread ~a.

Execution: Finally, TIM is run again with the same URL request. This time the modified select-external-display task is successful and the program successfully displays the document.

[3]In general, one would generally want different rules for selecting items to repair than for ordering blame assignment. In particular, one might want to order potential candidates for repair on the basis of what kind of failure the blame assignment mechanism recognized. I intend to pursue this issue further as more agents and problems are analyzed.

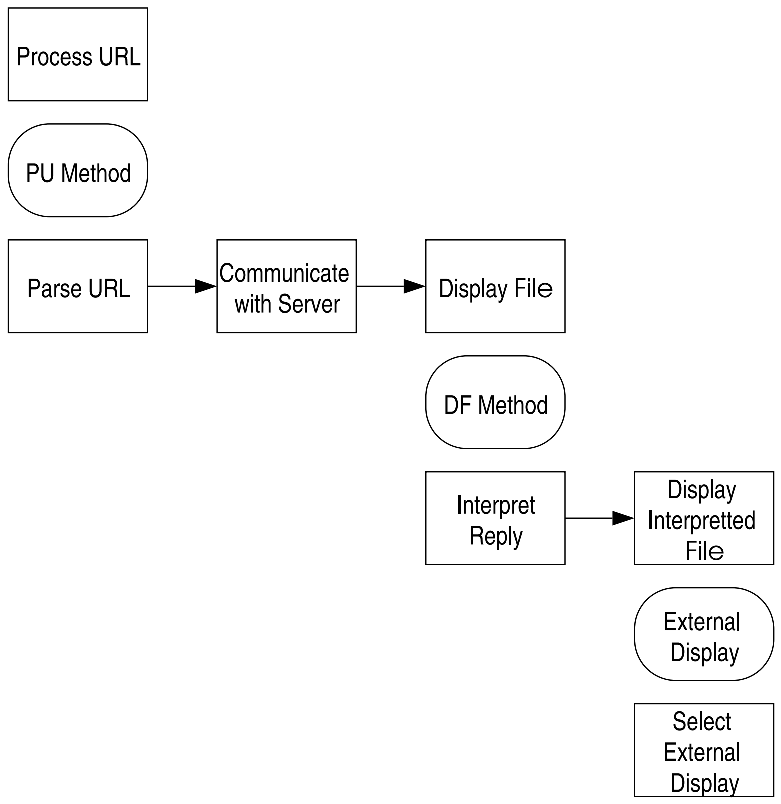

Figure 3: A trace of a sample execution of TIM which fails to invoke an external browser. Arrows represent the order in which execution occured. Vertical position indicates the different levels of abstraction which the trace represents.

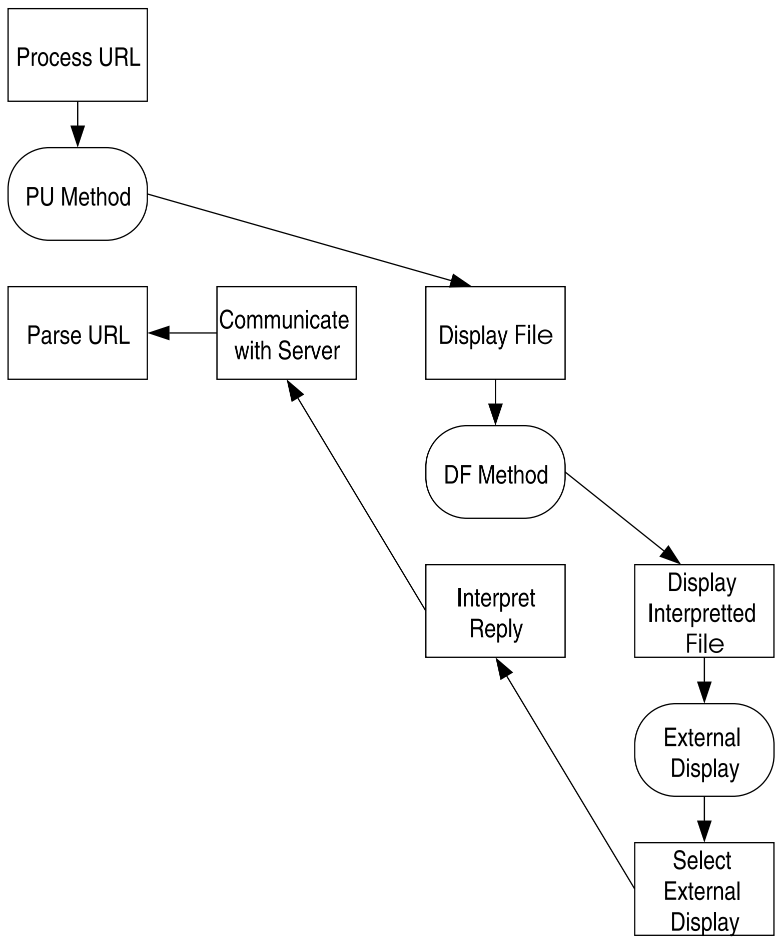

Figure 4: The progress of the blame assignment process through the trace of the failed external browser execution. Arrows here represent the order in which elements are evaluated for potential blame assignment.

3 Issues and Scope

Recall from Figure 1 that we have divided the themes of agency, evolution, and reflection into four major topics: situations, benefits, content, and processes. The topics of situations and benefits largely involve the *questions* to be addressed, i.e., the nature of the problems to be considered. The topics of content and processes largely involve the *answers* to be provided, i.e., the manner in which these problems will be solved. The former subject (questions) is discussed in this section. Section 4 addresses the latter subject (answers).

3.1 Evolutionary Situations

What are the situations in which evolution is desirable? AUTOGNOSTIC deals with several types of learning situations, generally driven by a response to a failure. One such situation involves detecting failures as they occur by recognizing that the specified relations over the output do not hold (i.e., that the agent did not accomplish its specified goal). Another involves direct feedback from a user (i.e., the agent thought it accomplished its goal but was later told that the result was inadequate). In the proposed research I intend to include failure-driven situations among those to be addressed; failures present particularly important situations for evolution because they identify circumstances in which some sort of change may be needed. However, I intend for failure driven learning of this sort to be only one aspect of a broader range of possible problems for evolution.

The goal of an evolutionary reasoning system is to take an agent and a problem and solve that problem. For any problem, either the given agent is already capable of solving the problem or the tool must evolve the agent in order to solve the problem. In SIRRINE, a problem is simply defined as a task[4] and a set of associations between abstract concepts and specific values. The problem described in Section 2 is specified as:

task: process-url

associations: {(url, "http://www.cc.gatech.edu/grads/m/Bill.Murdock/dcsp.pdf")}

Problems of this form are solved in SIRRINE first by executing the task and then if necessary redesigning and then re-executing that task. Such redesign and re-execution occurs if SIRRINE detects a failure and / or explicit feedback is given. When feedback is given, it is in the form of an additional association of a concept to a value; this association is not an input to the agent put rather is an association that the agent should have produced. In the example from the proposal, the feedback given is:

[4]Tasks and the additional knowledge structures which fully specify the behavior of a task (such as methods, concepts, etc.) are described in more detail in Section 4.1.

association: (command, "acroread ~a")

It is obvious, however, that a problem need not be simply a task and a set of associations, and that feedback need not be simply an association. There is an essentially unlimited variety of kinds of knowledge that could potentially be given to a system as input. In the following three subsections, I will attempt to more precisely characterize the particular kinds of inputs that I will consider in this research. In the first subsection, I present an ontology of the kinds of information that a user might want to provide to a reflective evolver (e.g., tasks, associations, etc.). This ontology is not powerful enough to represent all possible kinds of problems that one might want a reflective evolving agent to address, but it is powerful enough to represent a much broader class of problems than is feasible to study in the course of this dissertation project. The second subsection will thus use the ontology to define some specific classes of problems which will be studied during the course of this research. The third subsection then uses the same ontology to describe classes of user feedback which may be used to provide additional information during a problem solving episode.

3.1.1 Problem Ontology

Below are four general kinds of information which may be relevant to the specification of a problem from a user, either in advance of problem solving or after some problem has occured (as feedback). The general classes of information include:

Known Task: An agent might be told to solve a task which it is already familiar with. A known task is assumed to have its methods, subtasks, and submethods completely specified, including its primitive tasks; it corresponds to a functional description of an existing, working agent. An example of the use a known task would be a web browsing agent told to display a file from the web.

Partial Task: An agent might be given an partial task to solve, i.e., a task for which some information is not specified. An obvious example of a partial task is a task with no methods specified. Such a task may include a relationship between its input and output but would not indicate how this relationship is to be accomplished. A partial task corresponds to a hypothetical capability which has not yet been fully implemented. For example, a web browsing agent might be told to display a file from somewhere other than the web (and thus will have to adapt its web browsing capabilities to this new task).

Association: An agent might be given an association between a specific concept and a value. Information of this form typically is used to specify a set of inputs to an agent;

they constitute the initial knowledge state that the agent works from. For example, a web browsing agent might be given an association between the url concept and a specific URL such as http://www.gatech.edu/.

Relationship: An agent might be told that a particular relation holds over a particular a set of values. Knowledge of this form typically encodes declarative statements about the world or about the knowledge that an agent possesses. A logical combination of relationships is also a relationship. An example of a relationship would be the fact that one file type is similar to another.

We can thus see a particular cluster of information provided by the user, before, during, or after problem solving as being composed of zero or more of each of the above kinds of information, as described in Table 1.

	(K, P, A, R)
$K =$	a set of known tasks
$P =$	a set of partially specified tasks
$A =$	a set of associations
$R =$	set of relationships

Table 1: Information provided by a user describing a problem to be solved.

Table 2 provides a more detailed summary the contents of the elements of these four sets. The elements of K are fully specified tasks. An element of P can be described as a set of slot / filler pairs (an example of a slot is the makes slot which describes the kind of information produced as output by a system). A particular slot in a task is filled by a relationship over concepts (for example, a relationship provided to the makes slot of a web browsing task might be that the contents of a document have been placed on the screen). An element of A maps a concept to a value. A relationship is defined by a relation (e.g., "is a," "is similar to," "is broken," etc.) and one or more values for which that relation is asserted to hold.

$K_i =$	(Task T_i
	...)
$P_j =$	(Task T_j
	:slot$_a$ $(R_{P_j a}\ V_{P_j a1}\ V_{P_j a2}\ \ldots)$
	...)
$A_k =$	(C_k, V_k)
$R_l =$	$(R_l\ V_{R_l 1}\ V_{R_l 2}\ \ldots)$

Table 2: Detailed contents of the information types.

3.1.2 Problem Classes

Table 3 describes five classes of problems within the ontology provided in Section 3.1.1. Each of these five classes provides specific restrictions on the values of four sets (K, P, A, and R) which define a subset of the possible problems. The classes are non-overlapping and together cover only some of all the possible problems which can be addressed using the ontology.

Class 1:	$K = \{K_1\}$, $P = R = \{\}$
Class 2:	$K = \{\}$, $P = \{P_1\}$, $R = \{(\text{is-similar-to } P_1\ K_1)\}$
Class 3:	$K = \{\}$, $P = \{P_1\}$, $R = \{(\text{is-a } V_{P_1 a1}\ V_2)\}$
Class 4:	$K = \{\}$, $P = \{P_1\}$, $R = \{(\text{is-similar-to } V_{P_1 a1}\ V_2)\}$
Class 5:	$K = \{K_1\}$, $P = \{\}$, $\|R\| > 0$
Class 6:	$K = \{\}$, $P = \{P_1\}$, $\|R\| > 1$
Class 7:	$\|K\| + \|P\| > 1$

Table 3: A non-comprehensive taxonomy of problem classes. Classes 1–4 are the ones to be addressed in this research. Classes 5–7 are (non-comprehensive) examples of interesting problem classes which will not be addressed in this research but which could be interesting topics for further research.

None of the classes place any restrictions on A; each type of problem may start with an arbitrary set of associations of concepts to values (i.e., an arbitrary initial knowledge state). The four classes to be addressed each involve one task (either fully or partially specified) along with at most one relationship which specifies additional information about that task.

Class 1 is the problem addressed by SIRRINE and AUTOGNOSTIC. The inputs are a single, known task to be addressed and an arbitrary set of associations. This is the most fundamental use that an agent architecture can be put to: an existing agent is run on a specific set of inputs. In this case, no evolutionary reasoning needs to be done prior to the agent executing because a complete agent is already known. However, evolution may be necessary if the agent fails. Such failure can be identified either through explicit representations of failure states in the model or from additional feedback from the user (as in section 3.1.3 below). An example of a Class 1 problem involves telling a web browsing agent to display a file at a given URL, as in Section 2. Since Class 1 problems have already been addressed in SIRRINE and AUTOGNOSTIC, I do not intend for them to be a major focus of this research. However, since the work to be done in this project builds on that existing work, it is reasonable to expect that the ideas developed here will be applicable to Class 1 problems.

Class 2 involves solving a new, partially specified task. In addition to the partial specification of the task (which should generally include information about the output / effects of the desired computation), an explicit relationship is provided which asserts that the desired task is similar to some known task. Some evolutionary reasoning needs to be done before problem solving can begin (since the desired task has not been fully implemented);

the similar, known task provides a starting point for this evolution. An example of a Class 2 problem involves asking a web browsing agent to instead browse a local file system and telling it that browsing a file system is like browsing the web.

Classes 3 and 4 also involve partially specified tasks, but instead of including a fact about the task itself, they each include a fact about some knowledge element referred to by the task rather than a fact about the task itself. A Class 3 problem involves a knowledge element which is an instance of some broader class of knowledge elements. A Class 4 problem involves a knowledge element which is similar to some other knowledge element (about which more is known). Consider, for example, the local file system browsing problem from the previous paragraph. Instead of stating that that browsing a file system is like browsing the web, it is possible that a problem could state that browsing a file system involves displaying local files and that local files are instances of documents (Class 3) or that local files are like web documents (Class 4). Note that, as these examples suggest, Classes 2, 3, and 4 are capable of expressing very similar kinds of information and in some cases it may be possible to infer one kind of information from another. For example, given the Class 4 fact that local files are like web documents, it may be reasonable to infer the Class 3 fact that local files are documents and the Class 2 fact that browsing local files is like web browsing. Note also, however, that there are subtle and potentially significant differences between these three kinds of information. For example, knowledge that two instances are similar *may* indicate that they are members of the same class and visa versa, but they also may not.

Classes 5, 6, and 7 provide examples of classes which will not be addressed in this work; problems like these might make productive topics for later research projects. Class 5 problems involve fully specified tasks for which some additional information is also provided (such as classification or similarity information of the sorts provided in Classes 2, 3, and 4). For some Class 5 problems, it may be useful to simply run the existing task; for others, it may be useful to do some advance evolution. Class 6 problems include multiple, arbitrary relationships in their specification. Relationship information in Classes 2–4, is highly constrained and restricted to a few specific kinds of relations with reasonably well-defined semantics. Consequently, relatively strong functionally oriented methods should be applicable. In contrast, analysis and evolution mechanisms for Class 6 problems will presumably need to be relatively weak (i.e., less tied to the semantics of the information). Mechanisms like Genetic Programming [Koza et al., 1996] and Structure Mapping [Gentner, 1983] may be more appropriate. Class 7 problems involve solving multiple tasks. In some cases it may be possible to simply decompose a Class 7 problem into multiple, independent instances of Class 1 through 6 problems. In some cases, it may not, either because the subproblems don't fit into any of the classes or because the operation of the subproblems interfere with each other (for example, the tasks "put block A on block B" and "put block B on block A" typically can't have both of their desired results in effect simultaneously). All of these issues are interesting and should provide future research opportunities. However, since they are outside the scope of the proposed research, I will focus my attention on Classes 1–4 for the remainder of this document.

3.1.3 Feedback

Section 3.1.2 describes some kinds of input which may be given at the beginning of the use of a system. However, other kinds of inputs may be given during during or after a task is executed if the results of that task are not adequate. Such feedback can be provided by a user or by an external computerized agent acting as a user. It is possible to identify some specific classes of feedback that are of particular interest to this work. In providing feedback, Table 4 describes six classes of feedback to be addressed in this research; they are labeled with letters to avoid confusion with the numbered problem classes.

Class A:	$R = \{(\text{is-broken Task}_1\ \text{Knowledge State}_1)\}$
Class B:	$R = \{(\text{should-produce Task}_1\ \text{Knowledge State}_1\ \text{Knowledge State}_2)\}$
Class C:	$R = \{(\text{should-invoke Task}_1\ \text{Method}_1\ \text{Knowledge State}_1)\}$
Class D:	$R = \{(\text{is-similar-to Task}_1\ \text{Task}_2)\}$
Class E:	$R = \{(\text{is-a Value}_1\ \text{Value}_2)\}$
Class F:	$R = \{(\text{is-similar-to Value}_1\ \text{Value}_2)\}$

Table 4: A non-comprehensive taxonomy of problem classes. These classes are the ones to be addressed in this research.

Relationships, which encode declarative facts about the world (including facts about the agent itself and its knowledge), are particularly relevant to the job of providing feedback. Specifying known or partially specified tasks (as in the problem classes above) is relevant for telling the agent what the overall effect to be accomplished is. However, an agent receiving feedback later in the process already knows the desired effect (since it was specified at the beginning); it simply needs some additional piece of information. For this reason, each of the classes of feedback refers only to the R item in the general interface ontology (from Section 3.1.1). A piece of feedback may *refer* to a task, or a knowledge state, etc. but it does not directly *assert* that a particular task is to be done. A user who wants to assert that some other task should be attempted may do this by specifying a new problem rather than by providing feedback on an existing problem.

The first two classes of feedback correspond to the traditional forms of feedback provided in SIRRINE and AUTOGNOSTIC. The first class of feedback simply provides an indication that the task is broken, i.e., that the output generated was inadequate. In the existing work, this is expressed implicitly by simply calling the routines which modify a broken agent. Having an explicit relation to express the fact that the agent is broken makes the flow of information between the user and the evolver more clear and furthermore allows us to include feedback of this sort in a broader ontology of feedback classes. The second form of feedback provides more elaborate information: that a task which is given Knowledge State$_1$ should produce Knowledge State$_2$ (note, however, that a task description should involve not only knowledge states but also world states; see Section 4.1.3 for more details on this subject). This second form of feedback is also presented implicitly in SIRRINE and AUTOGNOSTIC; the user calls a repair routine with a knowledge state as an argument, and it is assumed that what is meant is that the most recently invoked main task or one of its subtasks should have produced that knowledge state given the knowledge state it received.

The four remaining classes are not supported in the existing systems. Class C feedback involves directly asserting that a particular task should invoke a particular method under a particular knowledge state. Feedback of this sort should generally be used to indicate that the task which was being solved (or one of its subtasks) should have been done using a different strategy altogether. For example a user might tell a web browser that it never should have tried to display a particular file within the browser window, but rather it should have known that the file should be displayed using an external display program. The problem of method selection is an important one for this research; evolving the conditions under which a method is selected is a key form of evolution.

Class D, E, and F feedback provide information which is essentially similar to the information provided in the Class 2, 3, and 4 problem types. Each of these types of feedback can be used to support similar evolutionary design strategies to those which would be used for the analogous type of initial problem. This evolutionary reasoning would also have the additional benefit of a reasoning trace from the initial problem solving attempt. Thus it may be possible for feedback of this sort to facilitate additional reasoning strategies.

3.2 Evolutionary Benefits

What are the benefits to evolution? The class of benefits which will be considered in this research is the ability to accomplish tasks which the system could not accomplish before.[5] This class can be divided into two, more specific types of benefits:

- The ability to solve a broader class of problems. For example, a web browser which evolves to support a new kind of external viewer for a new kind of file has increased the range of documents which it can successfully display.

[5]Other important benefits of evolution include increased quality of results and more effective use of resources (e.g., more speed, less memory, etc.). These may be interesting topics for later research but will not be studied in detail in this work. See Section 6.4 for a further discussion of this distinction between functional and non-functional evolution.

- The ability to solve a different class of problems. For example, a web browser which is designed to operate over the internet but is placed on a local area network (with different protocols, etc.) may need to evolve to operate in this different domain. In this case the system isn't *adding* functionality, but rather is *modifying* the existing functionality.

The issues relating to the different benefits of evolution are of relatively minor importance to SIRRINE and AUTOGNOSTIC; since the evolutionary situations which these systems involve only specific responses to individual failures, the system needs simply to worry about a change to one particular solution and thus the issue of whether the class of solutions is growing or simply shifting is of only incidental importance. I expect that the different kinds of evolutionary benefits will be considerably more important as we expand into a broader range of evolutionary situations.

4 Hypotheses and Solution

Given the account in Section 3 of what issues we expect to be addressed in this research, it now possible to present preliminary hypotheses regarding our intended results. This section describes both the language which will be used to encode knowledge of agents and the processing which will occur to instantiate evolution over this knowledge.

4.1 Reflective Content

Reflection is largely defined by the knowledge it uses: knowledge of a system's own reasoning. A core issue which I intend to address in the proposed research is that of the language which is used to encode knowledge of self. I have been developing a language known as TMKL, Task-Method-Knowledge Language [Murdock, 1998a], which I hypothesize will be able to support reflective reasoning processes such as agent evolution. Work on TMKL is an extension of the Task-Method-Knowledge (TMK) modeling framework [Stroulia, 1994, Goel et al., 1996, Griffith, 1997, Murdock & Goel, 1998]. Most of the existing work on TMK models, however, has largely focused on the content of such models. Our work on TMKL involves considerable incremental refinement of this theory of content, particularly in regards to issues of interactivity and external behavior. Furthermore, this refinement is done in the context of defining a language, *per se*; careful consideration has been given to issues such as readability, conciseness, orthogonality of syntax and semantics, etc.

The TMK framework was largely inspired by Structure-Behavior-Function (SBF) models of physical devices[6] [Goel, 1989, Bhatta, 1995, Goel et al., 1996, Goel et al., 1997b]. The basic idea behind SBF models is that the relationship between the physical construction of a device (i.e., the structure) and the intended effect of that device (i.e., the function) is described by a flow of causal interactions (i.e., the behavior).

There have been a number of theories which have viewed elements of cognition as inherently device-like in nature. The theory of Generic Tasks [Chandrasekaran, 1988] suggests that there are primitive "building blocks" of cognition such as classification and recognition and that complex reasoning strategies can be viewed as complex devices resulting from the synthesis of these components. This perspective suggests that perhaps past work in adaptive redesign of physical devices may be applicable, by analogy, to the adaptation of these proposed abstract cognitive "devices." The combination of the SBF approach to device modeling and the Generic Tasks perspective of reasoning systems as devices provides a foundation for the TMK theory and thus for TMKL.

TMKL models are *functional*, *causal* and *compositional*. These three key traits are derived from the earlier SBF paradigm. Functional models describe the effects of systems and their elements. Causal models describe how elements of a system interact. Compositional models describe how the interactions of elements of a system combine to for the effects of a system as a whole. The functional nature of SBF models is particularly useful in recognizing similarity between devices and relevance of components to specific overall effects. The causal nature of SBF models is particularly useful in performing detailed analysis of devices and verifying the potential applicability of modifications. The compositional nature of SBF models facilitates reasoning across functional and causal information. For example, a typical case-based design scenario using SBF might first retrieve a relevant case using functional indexes, search through a causal flow of transitions, identify and adapt a relevant component involved in that flow using that component's functional description, and then verify that the altered component has the desired overall effect by tracing back through the causal description. Thus the functional, causal, and compositional aspects of SBF models provide tightly interconnected leverage for the redesign process. The TMK paradigm and the TMKL language are intended to extend this leverage into the domain of software agents.

[6]In fact, some sources, e.g., [Stroulia, 1994, Stroulia & Goel, 1995, Stroulia & Goel, 1996], refer to the models which I am calling TMK models as *SBF models of reasoning systems*, in analogy to the SBF models of physical devices. I refer to them here as TMK models because their key elements are task, methods, and knowledge and they are conceptually very similar to the the models described as TMK in sources such as [Goel et al., 1996, Griffith, 1997, Murdock & Goel, 1998].

4.1.1 Elements of the Modeling Language

The development of the modeling language is one of the goals of this project and thus a complete description of the language is (obviously) not available yet. The TMK framework of Autognostic and Sirrine provides an initial hypothesis for a modeling language and some preliminary work has been done on TMKL [Murdock, 1998a].

Processes in TMK are divided into *tasks* and *methods.* A task is a unit of computation which produces a specified result. A description of a task answers the question: *what* does this piece of computation do? A method is a unit of computation which produces a result in a specified manner. A description of a method answers the question: *how* does this piece of computation work? Task descriptions encode functional knowledge; the production of the specified result is the function of a computation. Method descriptions encode causal knowledge; the mechanism whereby a result is obtained is the cause that a computation operates as it does. Each task is associated with a set of methods, any of which can potentially accomplish it under certain circumstances. Each method has a set of subtasks which combine to form the operation of the method as a whole. These subtasks, in turn, may have methods which accomplish them, and those methods may have further subtasks, etc. At the bottom level, primitive tasks are defined which may not be further decomposed.

An example of a task is accessing a document in a web browser. Examples of methods which accomplish this task are issuing request across the internet, retrieving the document from a cache (a faster method but less often applicable), accessing a proxy server, etc. Each of these methods sets up its own subtasks; for example, the internet request method involves getting an IP address from a name server, sending a request to that IP address, waiting for a response, etc.

Descriptions of knowledge in TMKL is done through the specification of *domains.* A domain identifies a set of *domain concepts*, i.e., kinds of knowledge, *task concepts*, i.e., applications of knowledge, *relations*, i.e., links between knowledge elements, and *constraints*, i.e., relations which must hold over knowledge elements. For example, a domain concept in the domain of web browsers would be a URL. Two task concepts might be the URL currently being viewed and the next URL to fetch. A relation over these task concepts might be that there is a hypertext link from the former to the latter. A constraint might be that while a link is being activated (e.g., by a mouse click), the relation just described must hold.

A system might have several interconnected domains. For example, a web browser might have a domain of web concepts (such as URL's, documents, etc.) and a domain of TCP/IP concepts (such as IP addresses, packets, etc.). Some tasks and methods might need access to only one or the other of these domains (e.g., the task of getting an IP address from the name server might need access to the TCP/IP concepts but not the web concepts). TMKL provides mechanisms for integrating multiple domains (e.g., indicating that a particular set of packets in the TCP/IP domain corresponds to a document in the web domain).

The process and knowledge portions of the TMKL models are intimately interconnected. Tasks and methods describe the domains in which they operate, the task concepts that they take as inputs and produce as outputs, etc. They further specify relations which must hold over the input values (i.e., applicability conditions) and relations which should hold over the outputs or between the inputs and outputs (i.e., effects). Thus the tasks and the methods refer not only to each other but also to the knowledge which they operate over.

4.1.2 Nature of the Modeling Language

What exactly is TMKL intended to be? What does it mean to have a model in the language being proposed here? Three key perspectives on what TMKL is encoding are:

- How an agent works. The claim that this perspective makes is that reasoning is done by selecting methods to address tasks and instantiating subtasks to accomplish methods. This is essentially the Router [Goel et al., 1994] perspective. In this sense, TMKL can be viewed as analogous to cognitive architectures such as ACT-R [Anderson, 1993], SOAR [Laird et al., 1987], blackboard architectures [Hayes-Roth & Hayes-Roth, 1979], etc., though obviously much more limited in scope than these projects.

- What an agent knows about itself. The claim that this perspective makes is that agents know about the tasks, methods, and knowledge they use (and further utilize this knowledge in adapting themselves). This is essentially the TMK / AUTOGNOSTIC view. In this sense, TMKL can be viewed as analogous to AI theories of reflection / meta-reasoning such as MOLGEN [Stefik, 1981], Theo [Mitchell et al., 1989], Meta-Aqua [Cox, 1996], etc.

- What a designer knows about a system being designed. The claim that this perspective makes is that descriptions of tasks, methods, and knowledge facilitate redesign of systems. This view has been advanced in the MORALE project [Abowd et al., 1997]. In this sense, TMKL can be viewed as analogous to architectural description languages such as those described in [Clements, 1996].

My working hypothesis is that these three perspectives are intimately interconnected. For example, to the extent that self-models are accurate, the first and second perspectives above are indistinguishable. Similarly, to the extent that adaptive learning can be viewed as a redesign process, the second and third perspectives also seem largely isomorphic. Are all three of these perspectives sufficiently similar that they can be addressed by a single language? Further development and instantiation of the proposed TMKL framework should provide empirical insight into this question.

4.1.3 Situated Reflection

As discussed in Section 1, a key element of this research project involves the study of agents which interact heavily with their environments. Both SIRRINE and AUTOGNOSTIC provide semantic descriptions of tasks as relationships between input knowledge and output knowledge. However, the operation of a task includes effects to and from the world state in addition to the agent's internal knowledge state. In the existing systems, this is handled by having the knowledge state include knowledge about the world state. Thus we could encode a task description which indicated: "given knowledge that Block X is not on Block Y, produce knowledge that Block X is on Block Y." However, this description does not indicate whether the agent is to put one block on the other or simply to change its beliefs about where the blocks are. Furthermore, there is much evidence (e.g., [Brooks, 1990]) that many kinds of very active tasks need not involve knowledge at all.

Consider two varieties of primitive tasks: primitive tasks which take place entirely in the mind and those which involve perception and / or action in the world. Existing work on TMK has largely ignored this distinction. A task-method analysis of cognition seems to have the potential to provide very effective accounts of reasoning which combines both internal processing and external interaction because it can build higher level tasks and methods from primitive elements of both varieties (as in REFLECS). However, I believe more care is needed to address some of the issues raised by the distinction between mental and embodied actions.

As an example of such an issue, consider the representation of *environmental variables*. In [Kirlik, 1998], q.v., environmental variables are characterized by two properties: accessibility to perception and accessibility to action. Accessibility in this context is defined as either proximal (things which can be seen or touched) or distal (things which can only be perceived or affected indirectly). Thus there are potentially four kinds of environmental variables in this framework (proximal perception + proximal action, proximal perception + distal action, distal perception + proximal action, and distal perception + distal action). To the extent that they can be viewed in this way, I would argue mental variables are inherently proximal perception + proximal action (since they are present in the mind). One way to address this distinction in TMKL is to define proximal variables as those for which primitive tasks exist which perceive or act upon them directly and distal ones as which only interact with non-primitive tasks. Note that this requires that we have a mechanism for interacting with variables at higher levels which cannot be interacted with at lower levels. TMK does not have such a mechanism; the only way to affect or make use of a variable in a high level task is to have it decompose into one or more subtasks which affect or use that variable.

The proximal / distal distinction is just one of many issues that need to be resolved in building task-method models of agents in situated task domains. An effective task-method theory of both reasoning *and* acting requires further research.

4.2 Reflective Evolutionary Processes

What are the processes which can be used to accomplish evolution in the context of reflective knowledge? Of the four key topics (situations, benefits, content, and processes), this is the one which requires the most development in the research proposed here. The potential situations and benefits of evolution are relatively intuitive. I have a preliminary account of reflective content in the form of TMK and TMKL. While I certainly intend to further develop this account, I believe that the central themes (a hierarchy of tasks and methods defined by the knowledge they affect, instantiation of generic elements, etc.) will probably not change tremendously during the course of this research. I expect that the bulk of the development work which will be done during the course of this research will focus on processes. Incremental revision to the hypotheses presented in the preceding sections will largely take place in the context of this extensive research on the subject of processes.

4.2.1 Architecture

Figure 5 provides an abstract view of the proposed architecture. Computation is divided into two pieces: Execute Task and Execute Method. Each of these pieces calls the other; tasks invoke methods and methods invoke tasks. These two pieces of computation operate over a single, coherent knowledge base which is divided into three large sections: a TMKL model of an agent, a TMKL trace of an execution, and a collection of domain knowledge available to the agent. The task and method executors take both the model and the domain knowledge as input and produce both the trace and new domain knowledge as output (in addition to having some effect on the external world state). Note that the three sections of the knowledge base are tightly interconnected. For example, elements of the trace describe invocation of tasks and methods in the TMKL model, and elements of the domain knowledge are linked to their abstract domain concepts (also in the TMKL model). The combination of the top level task execution and method execution elements along with the general knowledge base constitutes the *TMKL kernel*.

Figure 6 provides a more concrete view of the proposed architecture, operating on a particular system (the TIM web-browsing agent described in Section 2). This view shows how the *evolver* is implemented within the kernel, and TIM is, in turn, implemented within the evolver. The evolver is implemented as a set of tasks and methods in TMKL. The evolver

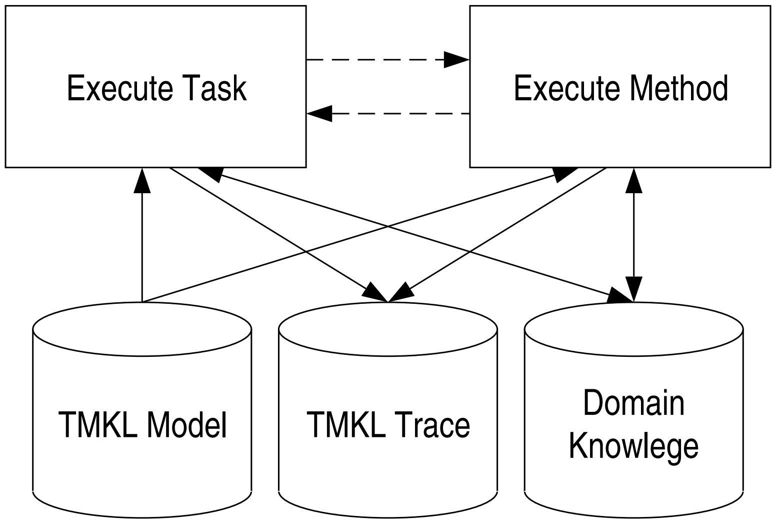

Figure 5: The top level proposed architecture. Dotted lines indicate control flow; continuous lines indicate data flow.

includes an Evolve Model component, as shown in the figure, which consists of several related tasks and methods for performing evolution. One aspect of evolution is the execution of an agent (i.e., the application of the Execute Task / Execute Method cycle). The domain knowledge that the Evolve Model task operates over includes the TMKL model and trace of the TIM system. In other words, TIM running under the kernel is recursively embedded in the evolver running under the kernel.[7]

Note that both the TMKL Model of Evolver and Domain Knowledge of Evolver knowledge collections in Figure 6 have ellipses at the bottom of them indicating additional information included within them. In addition to the embedding of the system being evolved, the evolver makes use of considerable additional knowledge. Included the TMKL Model of Evolver element are a variety of evolutionary tasks (e.g., credit assignment, modification, etc.) and evolutionary methods (e.g., analogical transfer, pattern instantiation, etc). Included in the domain knowledge of the evolver is the evolutionary problem to be solved (specified in the ontology developed in Section **??**), knowledge of TMKL design patterns, etc. The development and analysis of this evolutionary content to be implemented in the TMKL kernel is the focus of the bulk of the proposed research. The development of the language and the kernel are also research goals, but they are explored here essentially as a foundation on

[7]In principle, one could even make the TMKL model of the evolver be domain knowledge available to the evolver (i.e., replace "TIM" in Figure 6 with "the evolver"). In this way, the reflective evolution process could reflect on the reflective evolution process. I intend for the theory and tool developed in this research to support this sort of reflection. However, I do not intend to have this topic, second-order reflection, be a major focus of this research. The topic raises a variety of questions, such as when is second-order reflection used, what can it accomplish, should there be specialized mechanisms to support second-order reflection, etc. I expect for the proposed research to (incidentally) produce a framework for *asking* these questions but intend to leave the *answering* of these questions for later work.

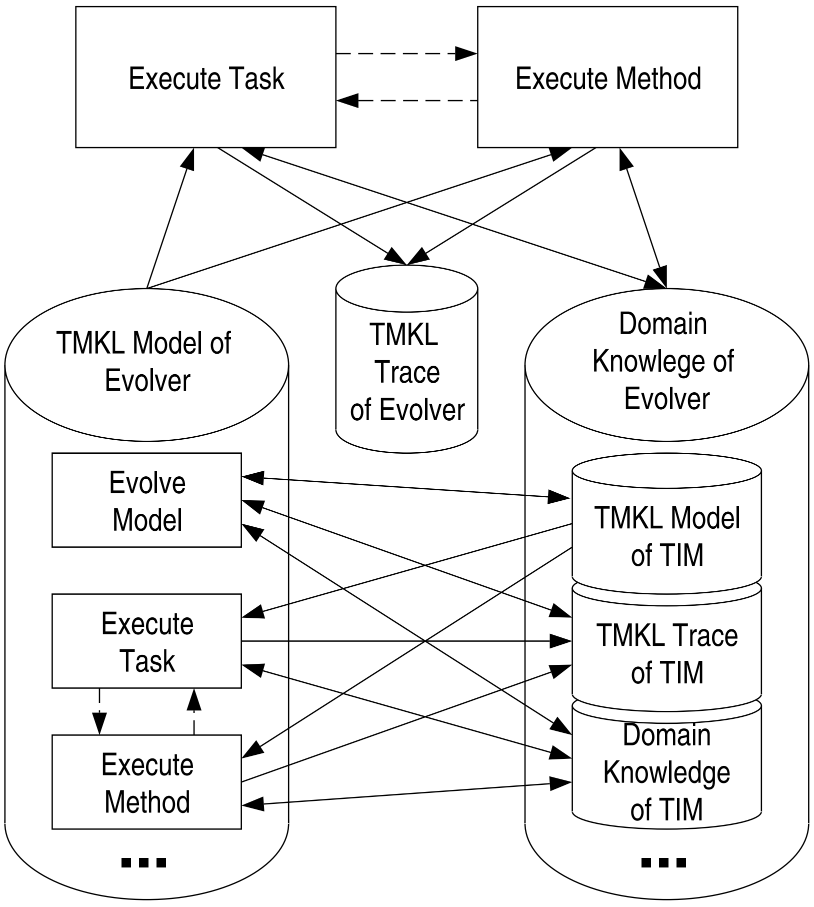

Figure 6: A detailed view of the proposed architecture instantiated on TIM. Dotted lines indicate control flow; continuous lines indicate data flow.

which to develop the reflective evolutionary capabilities; these evolutionary capabilities are the primary research focus.

4.2.2 Algorithms

Consider the four specific classes of problems described in Section 3.1.2, as well as the six classes of feedback described in Section 3.1.3. Applying each of the classes of problems with each of the classes of feedback, plus each of the classes of problems with no feedback provides a total of 28 potential varieties of usage. One could address these usage types with 28 distinct, unrelated algorithms. However, this would not result in one theory with a relatively broad range of coverage but rather with 28 separate theories with much more limited scope. One would hope, instead, that a single or small class of algorithms could be applied at the highest levels of abstraction which covered all of the included cases and that the distinctions between the different usage types would become increasingly significant at the lower levels of abstraction. Thus the higher level tasks and methods of the evolver would be common across broad classes of usage types while the lower level tasks and methods would be increasingly specialized. In particular, work with SIRRINE and AUTOGNOSTIC as well as my preliminary work on the new project suggests usage of the sort described in Section 3 may be supported by the following top level algorithm:

Hypothesized generic problem solving algorithm:

```
While Not Done
    If $K = \{K_1\}$
        If (execute-task $K_1$ under knowledge-state $A$) Succeeds
            Done [Succeed]
        Else If (acquire-feedback) Succeeds
            If (evolve-under-trace-and-relation) Fails
                Done [Fail]
        Else If (evolve-under-trace) Fails
            Done [Fail]
    Else If (evolve-under-relation) Fails
        Done [Fail]
```

Figure 7 presents a TMK sketch of this high-level algorithm. There are many ways in which a given computation can be divided into tasks and methods (see Section 5.2 for a discussion of the issue of variability in TMKL models of a given system). For example, the while loop in the pseudo-code could easily be implemented as a loop within a method rather than as a recursive call to the main solve-problem task. I have chosen the recursive approach in this case because it seems more descriptive of the actual intent; the process here is not about solving the problem over and over (as a loop would imply) but rather it is about transforming the problem and then trying to solve the transformed problem, i.e., the later part of the method really involves the main task itself.

In both the pseudo-code and the TMK sketch, five subtasks are left unspecified: execute-task, acquire-feedback, evolve-under-trace-and-relation, evolve-under-trace, and evolve-under-relation. The first of these, execute-task, is discussed in Section 4.2.1. The acquire-feedback task is presumably relatively straightforward; it simply prompts a user for some data.[8] The remaining three tasks, evolve-under-trace-and-relation, evolve-under-trace, and especially evolve-under-relation, are the primary focus of this research.

Table 5 provides functional descriptions for the key tasks depicted in Figure 7. The top level task, solve-problem, takes as input a problem, as specified in Section 3.1.2, and solves that problem. The execute-task subsystem takes a fully specified task (either provided by the user, in a Class 1 problem, or developed by the evolver, in a Class 2–4 problem) and executes it. The evolve-under-relation task essentially transforms a Class 2–4 problem (receiving a partially specified task, a relation, and a state in which to operate) into a Class 1 problem (which has a fully specified task). The evolve-under-trace task takes a fully specified task, which has been executed as well as trace and result information, and makes some modification to the task if needed (typically if the task failed). The evolve-under-trace-and-relation task is similar to evolve-under-trace but also contains explicit information which may require that evolution occur.

The evolve-under-trace-and-relation and evolve-under-trace tasks are ones which have been addressed in SIRRINE and AUTOGNOSTIC. As indicated in Figure 7, these tasks occur after an agent has run and has failed. If feedback is provided, the evolve-under-trace-and-relation task is invoked (the failed execution provided a trace, and each of the classes of feedback provides exactly one relation). If no feedback is provided, the evolve-under-trace task is invoked. Since both of these tasks have been addressed in earlier work, they will be less central to this research than the evolve-under-relation task. However, it will still be important to revise the methods for addressing these tasks so that they are suited to new issues in this

[8]A deeper analysis of the feedback acquisition, however, could explore a variety of issues. For example, if there are multiple users and / or computer-based knowledge sources available, there may be a question of which one (or ones) to solicit feedback from, and if feedback is received from multiple sources, which sources are considered more valuable / reliable / important. Another feedback issue involves parsing and understanding feedback; some sources of feedback may require natural language processing or at least conversion from some alternative knowledge representation format. Yet another feedback issue involves the question of whether it is worth soliciting feedback at all; a system shouldn't waste time as well as bother a user for feedback on a problem that it could solve without the feedback. All of these issues are interesting but outside the scope of this project. For this research, it is assumed that feedback is provided as needed by a single user in a format which is already understood by the agent.

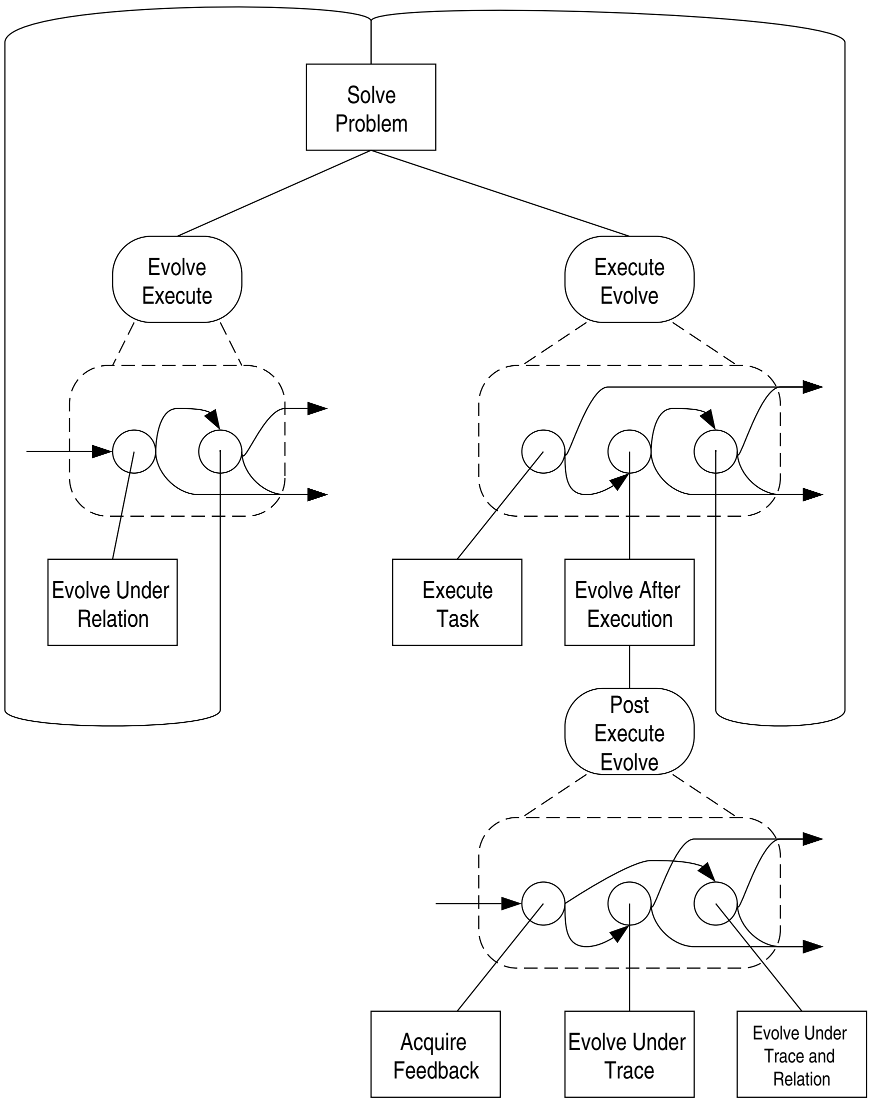

Figure 7: Top level sketch of a TMK model of the general problem solving process. Sharp cornered boxes represent tasks. Round cornered boxes represent methods. A dotted expansion of a method indicates the control flow which that method specifies. In these control flow diagrams, arrows coming upwards out of a state indicate success while arrows coming downward out of a state indicate failure.

Task solve-problem	
Given	Task K_1, or partially specified task P_1 and relation R_1
	Knowledge state A
Makes	Results of executing (possibly evolved) task under that state
Task execute-task	
Given	Task K_1
	Knowledge state A
Makes	Results of executing K_1 under A
Task evolve-under-relation	
Given	Partially specified task P_1
	Relation R_1
	Knowledge state A
Makes	Fully specified task which should operate under state A
Task evolve-under-trace	
Given	Task K_1
	Knowledge state A
	Results of executing K_1 under A
	Trace of executing K_1 under A
Makes	Evolved task which should successfully operate under state A
Task evolve-under-trace-and-relation	
Given	Task K_1
	Knowledge state A
	Results of executing K_1 under A
	Trace of executing K_1 under A
	Relation R_1
Makes	Evolved task which should successfully operate under state A

Table 5: Functional descriptions of the solve-problem task and its key high-level subtasks.

research (e.g., more reactive “agent” systems). The evolve-under-relation task is not one which has been addressed in existing work on TMK models. This task does not occur after a failure, but rather takes place before a task is executed. The evolutionary mechanisms in SIRRINE and AUTOGNOSTIC require that there be a trace of reasoning available to support localization of the pieces to be modified. Addressing the evolve-under-relation task will require strategies which rely on different kinds of localization strategies.

At first glance, it seems obvious that the evolve-under-trace-and-relation and evolve-under-trace tasks are appropriate to Class 1 problems (with or without feedback respectively) while evolve-under-relation is appropriate to Class 2–4 problems. In particular, because Class 1 problems have fully elaborated tasks provided, it is possible to execute a task immediately, as in the execute-evolve method in Figure 7. Class 2–4 problems do not have fully elaborated tasks but they do have additional relations specified which may provide direct localization information. For example, Class 4 problems indicate that two knowledge items are similar; this may suggest that portions of existing, known task structures which refer to one of these knowledge items need to be evolved to manipulate the other.

The mapping of Class 1 problems to the first two evolutionary tasks and Class 2–4 problems to the remaining task need not be this strict however. For example, if a Class 2 problem leads to an elaborated task which is then executed, but that execution fails, then it may be useful to invoke the evolve-under-trace task. Furthermore, it seems likely that there will be some overlap in the methods which are available to support these tasks. The set of applicable methods for the evolve-under-trace-and-relation should include all of the methods applicable to the evolve-under-trace and evolve-under-relation tasks, since one can simply ignore a trace or relation (and since a fully specified task is simply an extreme case of a partially specified task). Additionally, some of the methods across tasks may share common subtasks. As noted earlier, the evolve-under-relation task involves unique challenges for localization; in some cases methods for the evolve-under-relation task may contain unique localization subtasks which are not shared with methods for the other evolutionary tasks but may also contain common, shared subtasks for making modifications once that localization is complete.

The two broad classes of evolutionary mechanisms that this project will address are *analogical transfer* and *generic pattern instantiation*. I intend for each of the methods for the three evolutionary tasks identified to involve one or the other of these.[9] Analogical transfer involves drawing information from a specific instance of a TMKL model (or piece of a TMKL model) into another model which has been recognized as similar or related in some way. Generic pattern instantiation involves drawing information from abstract knowledge about TMKL models; such information can be seen as pre-compiled analogies over a set of TMKL models.[10]

[9]This would seem to indicate that analogical transfer and generic pattern instantiation are, in some sense "generic" problem solving methods. It might be useful for specific methods to have some sort of formal indication that they are instances of a particular class of methods in this sense. AUTOGNOSTIC has a similar notion of prototypical tasks but does not address the notion of generic methods. It is unclear at this time whether this topic should be addressed in this research. I do not intend to consider it further *unless* I encounter a specific situation where explicit representation of generic methods provides some specific added value.

[10] This assertion obviously suggests the question of how these analogies are pre-compiled. A machine learning technique that extracted patterns of reasoning over a set of models would be extremely useful; automated abstraction of generic TMKL patterns would seem to provide an approach for learning new kinds of changes to make to a system. This topic seems much too large to thoroughly address within the context of a dissertation project which also considers a number of other issues. Consequently, I have decided to drop this item from the research agenda for this proposal. I do, however, think that it is a very important issue for future research.

A wide variety of apparently heterogeneous evolutionary strategies may be classified as generic pattern instantiation. For example, consider the modification made to TIM in Section 2. A primitive task which is implemented by a table should produce a specified output given a specified input. Consequently, a new element is added to the table which maps that input to that output. This is clearly a pattern of modification which can happen again and again. In SIRRINE, this change is implemented by LISP code; the modification pattern is implicit in the procedures which add an entry to a table. In principle, however, it should be possible to create declarative knowledge structures which explicitly state that situations of this sort (specified input output pairs for a table primitive) can be addressed by modifications of this sort (adding a table entry). This would allow us a principled mechanism for developing, implementing, testing, comparing, and eventually reporting new generic evolutionary patterns. Furthermore, it could potentially provide a foundation for autonomous learning of generic repair patterns (to be examined in later work, as specified in Footnote 10). I absolutely intend to support explicit, declarative descriptions of generic evolutionary patterns in this research project.

Under this framework, most explicit repair strategies in AUTOGNOSTIC and SIRRINE would be classified as generic pattern instantiation. In fact, this approach seems very well suited to evolution using traces, addressed by these systems. Often the greatest difficulty in applying generic patterns involves localization. Because these patterns are generic, there should typically be a potentially overwhelming variety of patterns which are applicable to some part of a TMKL model. Evolution using traces is particularly well suited to problems where localization is a crucial issue since traces provide powerful support for localization. Generic pattern instantiation faces relatively little difficulty in actually making a modification once a location has been identified. Only a very limited set of generic patterns (often only one) will be applicable to any given assignment of blame and credit. A particular generic pattern will generally specify fairly precisely what modification needs to be made (e.g., the previously mentioned table entry pattern says to add one entry of the specified values, and not to remove any entries). There may be *some* reasoning and knowledge needed to identify which pattern to apply, and *some* flexibility to a given pattern, but these are generally not the core issues and thus generally not where the most power is needed.

In contrast, analogical reasoning strategies seem to be more suited to situations where explicit relations are available. Unlike a generic pattern, a specific source analogue already has a complete structure which strongly constrains the localization of matches and mismatches between existing and desired capabilities. Since there may be *some* alternatives for matches, localization is still an issue but it seems to be a more constrained, less challenging issue than it is for generic pattern instantiation. In contrast, the question of what modification needs to be made given a particular localization is a much more challenging one for analogical reasoning. While generic patterns clearly identify what elements of the pattern are used for recognition and what elements are used for modification, specific source analogues provide no such information. Evolution using relations may provide additional support for making these decisions since they elaborate the nature of the connection between the source and the target.

I have identified three crucial evolutionary tasks: evolution using traces, evolution using relations, and evolution using both traces and relations. Evolution using traces receives powerful support for determining *where* to make changes to a system but less support for determining *what* changes to make; consequently generic pattern instantiation methods, which are challenged by the former issue, are particularly suited to this task. Similarly, evolution using relations receives more powerful support for determining *what* to change than *where* to make changes; analogical reasoning methods, which require more intensive analysis of the former issue, may be better suited to this task. The task of evolution given both a trace and a relation may then be well addressed by either variety of method or even by a combination of the two.

5 Evaluation

The primary method for evaluating the hypotheses proposed here is the construction of computer programs and the systematic variation and analysis of these programs. In particular, I intend to construct a program which will act as a both as an interpreter of models in the TMKL language and as a reasoning mechanism for evolution of the models. Variation and analysis of this program (and by implication the underlying theory) will occur along four dimensions:

Variation of Context: I will test the system on a variety of different agents in a variety of domains.

Variation of Models: For a given agent, I will test a variety of different possible TMKL models of that agent.

Variation of Environment: For each model of each agent, I will provide a variety of operating environments which provide a variety of opportunities to evolve.

Variation of Reasoning: For each combination of context, model, and environment I will further test a variety of reasoning strategies.

The purpose of conducting these tests across these variations is to evaluate our hypotheses across the four main topics identified in Section 1: situations, benefits, content, and processes. I expect that each of the four dimensions of experimentation will be relevant across all four issues. However, some dimensions may have more relevance to some issues than others. Variation of context and environment are likely to largely determine the evolutionary situations and benefits which are relevant to those experiments. Variation of models is likely to be closely tied to the issue of reflective content. Similarly, variation of reasoning is likely to be closely tied to the reflective evolutionary processes.

The most interesting and useful results of these experiments are likely to be those which explore interactions across issues and dimensions of experimentation. For example, results regarding the modeling language which are derived from variation of models are likely to be most significant in the context of results regarding the processes derived from variation of reasoning. Most generally we would want claims of the form "modeling content A is useful because provides benefit B in situation C via process D." By experimenting across a range of dimensions, I hope to be able to make relatively general claims of that sort.

5.1 Variation of Context

A crucial element of the evaluation of this theory is the specific agent systems on which the experiments will be conducted. In SIRRINE, we have studied two such systems: the TIM web-browsing agent presented in Section 2 and a simple meeting scheduling agent. We intend to continue the work begun in the SIRRINE project on these two agents. Additionally, we intend to examine agents a number of additional reasoning domains in order to provide results over a broad range of systems.

5.1.1 Meeting Scheduling

One agent which has been modeled in SIRRINE is a relatively simple meeting scheduler, i.e., a program for determining a time when a group of people can meet given a set of schedules. This agent was built specifically as an example for use in that project and is deliberately very limited in order to provide a variety of opportunities for supporting evolutionary behavior.

Figure 8 presents SIRRINE's TMK model of the meeting scheduling agent. The top level task of the agent is the task of scheduling a meeting. It has one method which it uses, that of enumerating a set of slots and checking those slots against the schedules. The slot enumeration mechanism sets up three subtasks: finding a first slot to try, checking that slot against the list of schedules, and finding a next slot to try. It also defines a set of transitions which order these subtasks; in particular, it starts with the find-first-slot subtask and then loops between the check-slot-schedules and find-next-slot subtasks until either the the check-slot-schedules task succeeds, (i.e., a slot has been found which satisfies all of the schedules) and thus the method has been successful (as denoted by the arrow pointing diagonally upward out of the control structure) or the find-next-slot task fails (i.e., there are no more slots to consider) and thus the method has failed (as denoted by the arrow pointing diagonally downward out of the control structure). Both the find-first-slot and find-next-slot tasks are primitive, i.e., they are directly implemented by simple procedures. The check-slot-schedules task, however, is implemented by the schedule-enumeration method. This method steps through each individual schedule in the list of schedules and checks the proposed slot against each of them until a conflict is determined or all of the schedules have been checked.

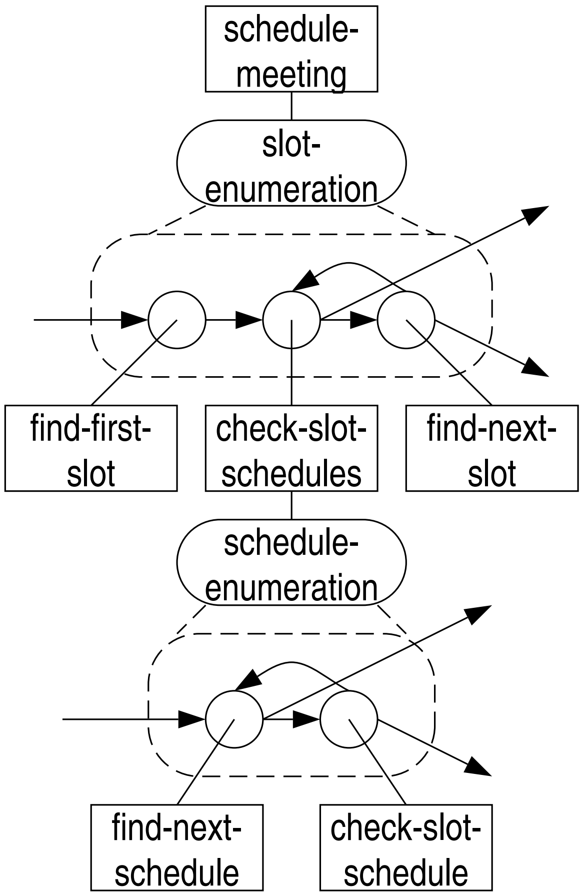

Figure 8: A TMK model of a simple meeting scheduling agent

The design of the meeting scheduling agent implicitly makes a number of simplifying assumptions. A few of the (many) assumptions include:

- All meetings must take place entirely between 9am and 5pm on Monday through Friday.
- All existing schedules are entirely immutable; there are no "soft" constraints on scheduling.
- All meetings must be as long as the specification requires; one cannot offer a shorter time slot as a "compromise."

- All of the prospective attendees must be able to attend a meeting for a time slot to be acceptable.
- All meetings are held weekly and all schedules describe only weekly events.

Any or all of these assumptions may be invalid in any given environment. Consequently, each of these assumptions presents an opportunity for learning whether, how, and to what extent, the environment violates these assumptions. Some experimentation with learning of this sort was done in SIRRINE and we expect to do more in this project.

5.1.2 Year 2000

The ZD tutorial [Allemang, 1997] presents an example of an employee time card system which suffers from a mismatch in Year 2000 compliance among its components. This example is analyzed from the perspective of TMKL in [Murdock, 1998a]; Figure 9 depicts the model described in that paper. The task which is modeled is that of an employee submitting a time card to a manager's computer. The method for this task divides the process into three stages: first the employee's time card program exports the data, then the network transports the data file to the manager's machine, and finally the manager's time card program imports the data.

Two main examples using this system are addressed in [Allemang, 1997]. One of these involves the formatting of data files (with items separated by commas or tabs). The other involves Year 2000 compliance (how dates are represented). Both of these problems involve inconsistencies between the operation of the export mechanism and the operation of the import mechanism. In the tab / comma problem, the distinction between the two mechanisms is considered to involve different kinds of functionality; hence the separation of the import task into two different methods which invoke a tab formatted importer and a comma formatted importer. Thus the challenge for this problem is to recognize which combinations of functional units will work (here, the system recognizes that the import mechanism must use the comma format because this format is the only one which the export format is capable of generating). In contrast, the date formatting in the Year 2000 problem is analyzed not as involving different functional elements but rather as involving specific constraints on the operation of these elements; i.e., the function of the export-comma task is to export tab-separated data and the date format used is simply an additional property of the task. The challenge for this problem is to propagate these additional properties through the system to observe interactions and inconsistencies among the elements.

The Year 2000 and tab / comma cases provide relatively simple examples of problems which may be solved by prior analysis rather than by responding to failures. In the Year 2000 problem, if the date format is explicitly represented in the model, it is possible to

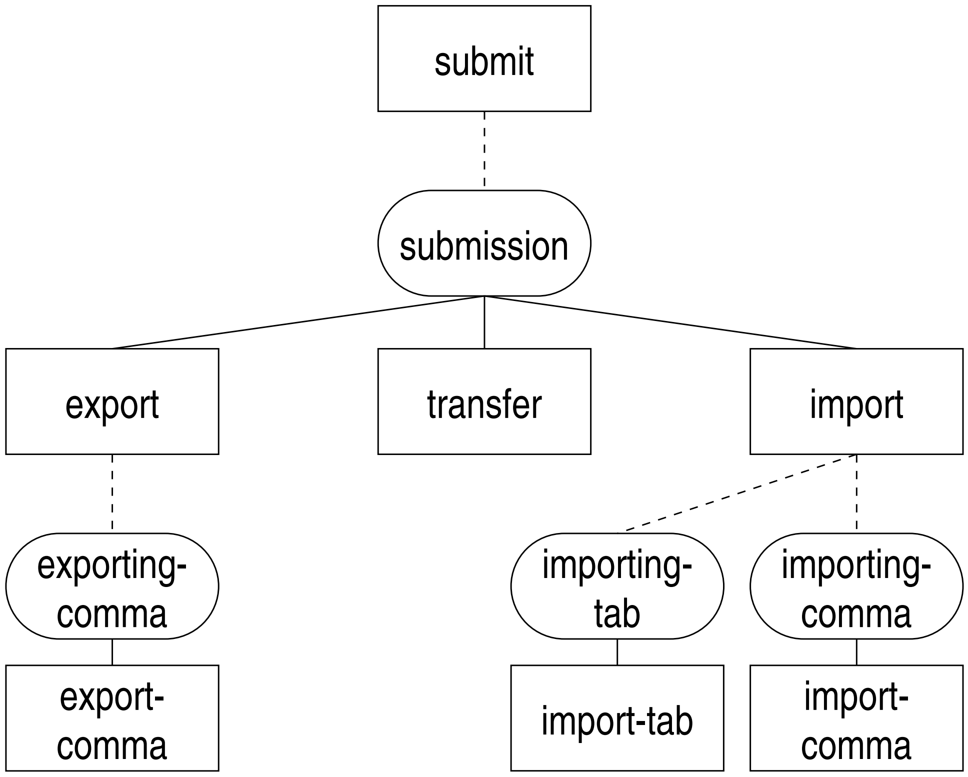

Figure 9: A TMKL model of the Year 2000 time card system

recognize the incompatibility by examining the model itself (propagating constraints on individual components to prove possibly self-contradictory constraints on the system as a whole). Furthermore, the employee time card system poses a variety of interesting and challenging issues for representation in TMKL (for example, the system consists of multiple components running independently on separate machines over a network). I intend to study this example further in the proposed research.

5.1.3 Other Domains

I intend to examine several reasoning systems in addition to the three which I have already begun to consider (TIM, the meeting scheduler, and the employee time card system). I intend to include the following systems in this research:

ADDAM ADDAM [Goel et al., 1997a] combines experience-based planning with simulated robotic action in the domain of disassembly of physical devices. It provides an opportunity to study the relationship between reasoning and acting in the context of an approximation of an external physical environment.

REFLECS REFLECS [Stroulia, 1994] combines Autognostic with a reactive robot navigation system. REFLECS is the Autognostic example which comes closest to the issues which we intend to address in the proposed research; it involves action in the real world, learning in response to the nature of the environment, etc. Consequently, I would expect a new analysis given the new research to provide substantially different results. At the very least, I would expect that a model of the REFLECS navigation system would be substantially clearer in TMKL since this language will have explicit mechanisms for representing external action (as opposed to AUTOGNOSTIC which essentially modeled external action as a form of computation and world states as a form of knowledge). It is reasonable to suspect that additional benefits may be gained in the form of greater reasoning efficiency and/or power; such gains should become apparent from our research.

In addition, I may examine several more systems as time and opportunity permits. New domains may be considered if they provide additional properties of relevance to the exploration of the theoretical issues described in Section 3.

5.2 Variation of Models

In addition to testing our theory across a range of agents, I intend to also study a variety of TMKL models for each agent. The purpose of this dimension of experimentation is to develop a deeper understanding of what the issues are in building a model. By studying concrete examples of varied descriptions of a given agent, I hope to see patterns of advantages and disadvantages to different styles of models.

For example, one way of varying a model is the adjustment of the division of computation into tasks and methods. Figure 10 illustrates three different models for a specific agent. This agent contains three steps: loading a file, altering it, and displaying it on a screen. The model on the left represents this agent as a task with a single method which has three primitive subtasks (load, alter, and display). The model in the middle describes the agent as a task with one method which has two subtasks: an access task, whose method is further divided into a load and an alter task, and a primitive display task. Similarly, the model on the right shows a task whose method has a primitive load subtask and a use subtask which is further decomposed into alter and display.

In this agent, it is clear that the way that the computation is divided does not affect its operation; all three models are "correct" in the sense that they accurately indicate what actions will be taken at what times. However, the evolution which can take place over these models may be very different. For example, an analogical evolutionary process which reuses a portion of a model in some other context might reuse the entire access task or the entire use task depending on which of these models were available. Presumably, the one which would be most likely to be most valuable as a candidate for reuse would be the one

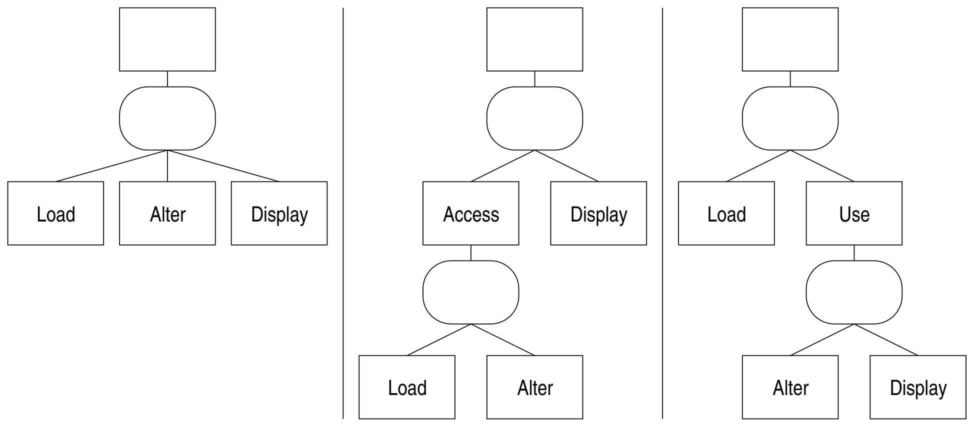

Figure 10: Three different models of a simple agent

in which the intermediate level tasks defined some functionally significant aggregation; if either access or use is really a meaningful abstraction, that task is more likely to be reusable. The hypothesis for this example would be that the most conceptually coherent model would provide the best support for evolution. Experimenting with a variety of models for each agent should provide insight into the generality of this hypothesis and allow deeper investigation of what constitutes a conceptually coherent model.

There is a virtually limitless assortment of ways that a model of a given agent can vary within a powerful modeling language like TMKL. Some of these variations (like the example above) provide identical problem solving behavior but result in different evolutionary capabilities. Others may result in differences in both of these processes. A few particularly interesting issues for modeling a given agent include:

- Which tasks are primitive and which require further decomposition.
- How the relevant knowledge elements are divided into domains of reasoning.
- For a given task:
  - What criteria are used to select methods.
  - Whether and to what extent properties of the required input are specified.
  - Whether and to what extent properties of the desired output are specified.
- For a given method:
  - How ordering constraints over subtasks are specified.
  - Whether and to what extent properties of the inputs and outputs are specified (as above).

- For a given concept:
  - What examples of the concept are known.
  - What relationships over instances of the concept are known.
  - What relationships are known to exist between instances of the concept and instances of other concepts.

What features of a model provide persuasive, conceptually meaningful descriptions? What features support effective model-based evolution? I intend to investigate these two major questions by providing a variety of responses to issues such as those listed above. I expect to find that the answers to these two questions are intimately interconnected.

5.3 Variation of Environment

Given a particular agent under study, one might devise a variety of environmental circumstances in which the agent is to operate. The external browser problem described in Section 2 is an example of such a circumstance. TIM receives a request from a user which involves accessing a page from the web; this particular combination of request and availability are such that TIM's performance is unacceptable (it doesn't display the document). Thus it is necessary for TIM to evolve.

Obviously, precise details of the variation of the environment will depend on the particular agents being studied. For a given context, a wide variety of environments may be of interest. For extremely simple domains, it may be possible to enumerate all possible environments of interest and exhaustively test our agent on them. For example, consider a stateless Tic Tac Toe agent which, given a board position, is intended to find a minimax optimal move. If we design an imperfect agent of this sort, we can exhaustively test it on every possible board position it could encounter, and on each one for which it performed suboptimally, have it evolve. If it eventually evolved to a perfect Tic Tac Toe player, we could claim to have evolutionary processes which provided complete coverage across all possible environments for that model of that agent in that domain.

However, as the Tic Tac Toe example illustrates, domains for which it is possible to completely prespecify all possible environments and the ideal behaviors in those environments are generally fairly uninteresting. In particular, there seems to be very little motive to build an evolving agent for these environments because one could simply build a perfect agent in the first place. The contexts which I intend to apply this research (described in Section 5.1) do not have exhaustively pre-ennumerable environments. It is only possible to test the system in a limited range of environments. The challenge for experimentation across this dimension is thus to choose interesting environments which illustrate important points about the issues being studied (most notably evolutionary situations and benefits, reflective content, and reflective evolutionary processes).

5.4 Variation of Reasoning

A major research tool in the field of Artificial Intelligence is the ablation study: pieces of a given system are removed and the behavior of the system without these pieces is compared to the behavior with these pieces. This methodology can be viewed as a specialized form of systematic variation of the reasoning process; for each piece ablated, the researcher addresses a specific system design decision, should this piece be included or not. In general, we may want to conduct experiments in which the decision is not a binary choice of inclusion or exclusion but rather a selection over a range of features. For example, one could vary the order in which various elements of the reasoning process occured.

I expect that the representation of the evolutionary processes themselves as TMKL models (as discussed in Section 4.2.1) should provide strong support for designing and conducting experiments which vary these processes. A few particular ways in which having a TMKL model may facilitate experimentation include:

- *In selecting pieces of the system to vary*: TMKL models identify what the different pieces of the system are and how they operate. Consequently, they may be used to choose elements to perform variations on. For example, ablation studies may be done simply by removing tasks or methods.

- *In providing a mechanism for doing the variation*: Since TMKL models are used to guide execution of a system, one can modify the system simply by altering its TMKL model. In addition to being more convenient than rewriting source code, this feature makes it more clear that exactly the desired element or elements are modified / rearranged / ablated / etc.

- *In systematizing variation*: Since TMKL provides similar representations for each of the different tasks and methods in a system, it is much easier to conduct a particular variation across such tasks. For example, an experiment which studied the effects of overspecifying applicability constraints on different learning processes could be done by adding these overspecified constraints to each of the different task or method representations. Compared to a comparable change to an arbitrary piece of source code, this approach would be both easier (since the specification would only have to be done once and then could be applied to each of the relevant pieces) and more rigorous (since it would be more clear that precisely the same change was made to each task).

- *In analyzing and reporting the results*: Since the experiments are conducted in the context of a principled account of what the elements of the system are, it should be much easier to draw and express conclusions regarding how changes to these elements affect the operation of the system as a whole.

Using the variation of the TMKL model of the evolutionary processes themselves as a mechanism, there are many specific process variations which may be conducted across each combination of agent, model, and problem. To a large extent, the process variations which will be of interest will depend on the specific processes which are developed during the research. However, it is possible to tentatively characterize some examples of relevant variations of reasoning. A few particularly interesting issues for varying the reasoning processes include:

- In what order tasks such as execution monitoring, credit / blame assignment, modification, etc. are processed.
- Which of these tasks are used in different situations.
- For the execution and monitoring processes:
  - How methods are selected.
  - Whether traces are generated.
  - If traces are generated, what information is recorded in them (tasks processed; methods invoked; alternatives considered; outcomes; initial, final, and intermediate knowledge states; etc.).
  - Whether the monitoring process forces intervention as soon as a problem is detected or waits until a final result is generated.
- For the credit / blame assignment process:
  - What classes of credit and blame are considered.
  - In what order elements are considered for assigning credit and blame.
  - Given multiple possible assignments of credit and blame, how one or more are selected as candidates for modification.
- For the modification process:
  - Which modification strategies are used.
  - Which modification strategies are considered applicable to which classes of credit and blame.

  - Given multiple possible modification strategies applicable to a situation, how one is selected.

Some of the above issues may be addressed by specific strategic biases. For example, one possible bias in the selection of candidates for modification is to alter the most general candidate for which an effective modification strategy is available (to provide the most broadly applicable learning mechanism). To the extent that these biases may be made explicit (perhaps as declarative knowledge structures or as decision making reasoning methods), they provide particularly useful candidates for systematic variation (by adjusting which biases are used and how conflicts among them are resolved).

By representing our reasoning processes in TMKL it should be possible to precisely and systematically test a wide range of system configurations. Such variations may occur over biases or any of the other sorts of issues listed above. By exploring a range of configurations, I hope to discover general principles about evolutionary reasoning processes.

6 Discussion

Agents which interact with dynamic environments present a wide variety of interesting challenges. This proposal focuses on the topic of the addition of new capabilities to an exiting intelligent agent. The proposed mechanism for addition involves reflection over functional, causal models. Sections 6.1, 6.2, and 6.3 below serve to place this proposed research in a broader context. Section 6.1 discusses the relevance of this work to a potential broader theory of reflection. Section 6.1 relates this work to the study of human cognition (since humans are, themselves, intelligent agents). Section 6.2 briefly summarizes the expected contributions of this work beyond the existing body of research. The proposal then concludes with Sections 6.1 and 6.5. The former discusses the breadth of applicability of this work and the latter asserts the fundamental claim to be assessed in the proposed project.

6.1 Applications of Reflection

I have hypothesized that reflection can enable a system to evolve. This use of reflection is the one which is most central to the research proposed here. However, there are a number of other applications of self-knowledge which I want to consider. A few particularly important uses of reflection are:

Multi-strategy reasoning: Explicit representations of functional elements potentially enables dynamic, flexible selection of these elements within a reasoning episode.

Interoperation: An agent that understand what it does can inform other agents what service it provides and may be able to recognize when it requires a specific service from another agent.

Explanation: An agent that understands how it operates can explain how it obtained some result. This can be useful both for teaching a novice user how to do something and for convincing an expert user that a decision or action is correct. Models similar to the TMKL models proposed here have been used for explanation tasks of both sorts [Goel et al., 1996].

Resource Allocation: Explicit knowledge of the costs and benefits of reasoning tasks and methods can be used to allocate resources such as processing time. This approach is typical of work such as [Boddy & Dean, 1989, Russell & Wefald, 1991].

Prediction: In many environments, particularly ones involving competition, it can be important to predict the actions of others. One way to do this is to attempt to reason about how they would reason. This process could potentially use knowledge of a system's own reasoning process to predict the reasoning processes of others. Mini-max game playing algorithms implicitly rely on this sort of reasoning (i.e., I expect that if I make move X, my opponent will make move Y, because that's what I would do in that situation).

Error Prevention: Knowledge about reasoning processes may be used to avoid situations in which errors might take place. For example, if something is known about the reliability of certain methods, a reflective system might reason about the combined reliability of a set of methods, and decide that a process must be attempted twice to obtain a desired overall reliability.

Error Recovery: Many researchers in cognitive science have observed that explicit representation of processes can be particularly useful in recovering from mistakes after they occur [Suchman, 1987, Kirsh, 1991]. When someone reaches a reasoning state which is clearly erroneous, they are motivated to reason about how they got to that state, even if the process which got them their was largely non-reflective at the time.

In this research I intend to focus on the above uses of reflection only as needed to support our primary interest, agent evolution. For example, I will certainly need to consider the first of these applications, multi-strategy reasoning; our approach to evolution revolves largely around modifications to the organization and composition of reasoning elements and thus I will need to work with models which reflect this organization and composition. However, this will not be the primary focus of this work. I expect that existing accounts of multi-strategy reasoning within a task-method framework, e.g., [Goel et al., 1994, Stroulia, 1994, Punch et al., 1996] will cover most of the work I want to do in this research and I will expand on these accounts only to the extent to which I need to do so to support the strategies of evolution which I develop or to cover the systems I wish to model.

6.2 Human Cognition

The research proposed here focuses on the development of theories of intelligence; the question of whether these theories are of how humans *do* reason or of how computer programs *can* or *should* reason is viewed as secondary (though still important). The particular research program proposed here uses development and analysis of software systems as its primary experimental method. Consequently, it would be reasonable to expect that the results obtained will be considerably more convincing as evidence about artificial intelligence. However, I do expect that the theory being developed will have relevance to the study of human cognition.

I believe that a truly general, powerful computational theory of intelligence is inherently a useful and interesting hypothesis regarding human cognition.[11] Consider the space of all *possible* mechanisms for representing and manipulating self-knowledge. It is intuitively obvious that this space is extremely large; virtually any kind of knowledge and inference could potentially be in memory. In contrast, consider the space of all *sufficient* mechanisms for this phenomenon. This space is clearly far more restricted; the mechanism must be shown to be both powerful enough to allow exploration to occur, parsimonious enough to form a credible theory, and general enough to support a wide range of exploratory reasoning. To the extent that these are strong restrictions, it is apparent that the space of sufficient solutions is very small with respect to the space of possible solutions. Furthermore, as the breadth and depth of the sufficiency is increased, it is clear that the space of sufficient solutions becomes so small that any sufficient solution approximates any other sufficient solution; i.e., a truly powerful computational system approximates human cognition. Thus I argue that to the extent that our theories have been shown to provide a broad and interesting range of capabilities, they constitute a plausible hypothesis for a cognitive model.

A deep investigation of these claims of relevance to human cognition is beyond the scope of the research proposed here. I expect to develop a plausible cognitive hypothesis for a range of closely related phenomena but do not intend to experimentally test this hypothesis. There is some existing and ongoing work which does provide evidence of the cognitive plau-

[11]Note that this claim is fundamental to the role of AI and computational modeling in the study of human cognition. The argument presented here is present, either implicitly or explicitly, in an enormous range of research, e.g., [Newell & Simon, 1963, Minsky, 1975, Anderson, 1983]. Furthermore, it is largely isomorphic to traditional philosophical views on the nature of all scientific theories, e.g., [Kant, 1781].

sibility of TMK models, e.g., [Griffith et al., 1997, Griffith, 1997, Griffith & Murdock, 1998, Murdock, 1998b]. To the extent that TMKL and the reasoning methods which operate over TMKL are extensions of the TMK theory, this provides us with further evidence that the research here may have cognitive validity. The application of the results that this research produces to the subject of human cognition should be a productive topic for future research.

6.3 Contributions

The work proposed here touches on a wide variety of topics, including agency, evolution, reflection, etc. There is much existing research on these topics and the potential contribution of this work lies in extending the frontier of research in these areas both by combining existing capabilities from separate projects and by developing new capabilities which no existing projects currently support. There are a variety of projects, listed below, which define the frontier which I intend to extend. Note that I do not intend for this to be "the ultimate theory" which subsumes all of the following projects. I do, however, intend for the research to make new, unique contributions such that it is not, itself, subsumed by any of these projects or even all of them in combination.

AUTOGNOSTIC: AUTOGNOSTIC forms a crucial piece of the theoretical foundation on which the research proposed here is based. The Structure-Behavior-Function (SBF) models of knowledge systems in [Stroulia, 1994, Stroulia & Goel, 1994, Stroulia & Goel, 1996] divide descriptions of these systems into tasks, methods, and knowledge elements and thus constitute an earlier version of the Task-Method-Knowledge modeling framework on which TMKL is based. Furthermore, we expect that much of the reasoning which AUTOGNOSTIC does in the service of failure-driven learning will be applicable to evolution in more interactive domains; early work on SIRRINE (q.v., below) seems to support this belief.

I expect the research proposed here to extend and expand upon AUTOGNOSTIC in numerous ways. Three of the most significant are (i) situatedness in highly interactive domains, (ii) evolution in non-failure driven contexts, and (iii) meta-reflection, i.e., the ability of the system to reflect on the reflection process itself. Other proposed improvements include the addition of new features to the modeling language and the development of new techniques for modifying systems.

SIRRINE: SIRRINE exists largely as a bridge between AUTOGNOSTIC and the research proposed here. The reasoning methods of SIRRINE are essentially a subset of those found in AUTOGNOSTIC, and the modeling language of SIRRINE, while possessing a refined, more coherent syntax, has only minor incremental revisions in the content. On the other hand, the domains to which SIRRINE has been applied, meeting scheduling and web browsing, are more characteristic of the proposed research than of AUTOGNOSTIC. To some extent, this combination has been a difficult fit. This has been a major research contribution of SIRRINE: identification of areas where the existing knowledge representation and reasoning strategies are particularly challenged by intelligent agent domains. The issues and experiments described in preceding sections of this document are, in part, inspired by the demands which SIRRINE has not been able to meet.

ToRQUE: ToRQUE [Griffith, 1997, Griffith & Murdock, 1998] is another system which uses TMK models. Unlike, AUTOGNOSTIC, SIRRINE, and the work proposed here, ToRQUE is not intended as a general purpose agent architecture. Rather, it is intended to deal with a specific problem solving system (a scientific reasoning program which analyzes a particular problem involving torsion in springs). ToRQUE and this proposal can be viewed as complementary. This work could potentially be used to place the particular reasoning system in ToRQUE (which is already decomposed into tasks and methods) into an architecture which provides reflective evolution, an issue which ToRQUE does not address. ToRQUE uses TMK models to support exploratory reasoning which is an issue which is largely outside the scope of this proposal. Furthermore, it more deeply addresses issues of cognitive modeling (since it is based on an analysis of a human problem solving protocol). ToRQUE provides a cognitive theory of exploratory scientific reasoning processes; the work proposed here could potentially provide an account of the evolution of these processes.

ZD: ZD [Liver & Allemang, 1985, Allemang, 1997] is a functional representation for describing computational systems (as is TMK). ZD has largely been used for *analysis* of systems, particularly with regards to viability of configurations of components. Consequently, it is particularly strong in providing formal structures and mechanisms for representing and propagating properties of information states. However, ZD has not been used to support autonomous *learning*. Thus, TMK generally provides more elaborate support for this sort of task (e.g., richer knowledge structures, executable models, etc.). In the development of TMKL, I have been attempting to combine elements of the powerful analytical support in ZD with the support for learning and execution in existing TMK research. The work proposed here should provide insight into the viability of this combination. A more detailed analysis of the relationship between ZD, TMK, and TMKL may be found in [Murdock, 1998a].

Other Projects: During the course of this research, I intend to carefully analyze the results obtained with those derived from a wide range of additional systems. It is expected that close relationships will be found with projects such as ACT-R [Anderson, 1993], SOAR [Laird et al., 1987], Theo [Mitchell et al., 1989], Prodigy [Veloso et al., 1995], Meta-Aqua and Meta-TS [Cox, 1996], PLUTO [Shippey et al., 1998, Murdock et al., 1997], etc. Additionally, we will want to compare the approach as a whole with approaches from a variety of related fields such as Automated Debugging [Ducassé, 1993] and Software Architectures [Clements, 1996]. It is expected that the research proposed here will provide insights which complement many of those from this related research.

6.4 Applicability

All theories and tools have a limited scope; they are applicable to some kinds of issues and problems, and not others. Any kind of problem which can not be formalized as a member of one of the four relevant problem classes described in Section 3.1.2 can not be addressed by this theory. More generally, the following problem traits are particularly indicative of applicability of the theory and tool being developed in this research:

Modular Systems: It is an established principle of software engineering that programs which are divided into distinct modules with separate, unique functionality are inherently more easy to evolve than those which aren't. A system which haphazardly interleaves low level commands which accomplish distinct aspects of high level functionality are difficult, even for a human programmer, to modify [Rugaber et al., 1995]. This principle seems particularly applicable to agent evolution of the sort being done in this research. An agent which logically divides into distinct tasks, methods, subtasks, submethods, etc. is far more easily modeled in TMKL and is far more likely to produce useful results when it is modeled in this way.

In practice, most systems seem to be highly modular at the highest levels, but less modular at lower levels. For example, the Mosaic code is divided into separate files which seem to have relatively distinct purposes. Mosaic is further divided into C functions which are reasonably well modularized, although their invocation is often intermixed with direct calls to individual commands. At the lowest level, individual lines of code, the modularity is fairly bad; different lines of code which have different impact on the overall behavior of a block of code are often interleaved in what ever way is convenient for the programmer. At those lower levels where modularity is poor, our TIM system is a relatively poor imitation of Mosaic. In general, it seems clear that it is only possible to provide an effective TMKL model of a system down to those levels at which a system is well modularized. Evolutionary problems which require analysis and modification at lower levels than this are not likely to be well addressed by this project.

Functional Evolution: It should be possible to evolve an agent to achieve both new capabilities and more efficiency for existing capabilities. However, it seems like TMKL, at least in its present form, is more suited to the former goal, i.e., that of functional evolution. In particular, TMKL explicitly represents information about the function of pieces of the agents (in terms of the task that they accomplish), but does not directly encode information about the efficiency of different pieces of the code. I do believe that it is possible to enhance the TMK paradigm to include non-functional (e.g., efficiency) information, and that information of this sort could be used to support non-functional evolutionary problems. However, a systematic, carefully considered study of this topic (extending this approach to non-functional problems) could be a very large project on its own. Thus, in the interests of having a more thoroughly constrained thesis project, I will be specifically excluding problems of non-functional evolution from this research.

Incremental Enhancement: Evolution, by definition, is about incremental change. This theory is intended to address problems in which some new capability is required of an agent. However, this new capability must have some relationship to existing capabilities. It is reasonable to evolve a web browser so that it is able to display new kinds of documents, or to browse documents over a different kind of network, or to display documents in a different way, etc. It is probably not reasonable to evolve a web browser so that it is able to provide navigational control for a mobile robot; such functionality has little or nothing to do with web browsing and thus should involve a different agent all together. More precisely, I claim that this project should involve the following sorts of incremental enhancements:

Parameter Adjustment: A system parameter is adjusted to some new value. For example, if a web browser has an explicit limit to the size of documents that it may attempt to display, that limit can be increased or decreased as needed. Modifications of this sort may be either done through the instantiation of a simple (often broadly applicable) generic pattern or as a highly restricted form of analogical transfer (transferring a single value).

Reconfiguration: Some kinds of evolution may take place by reordering or otherwise reconnecting computational elements (for example, a method which is known to be applicable to two different tasks may be altered so that it is only invoked by one of those tasks). These sorts of modifications may also be done using patterns (e.g., when the actions of one task invalidate the preconditions for another, try reordering them) or by analogy (e.g., two tasks are similar to two tasks in a similar system which are executed in the opposite order, so it is reasonable to try reordering them).

Generic Functionality: In addition to patterns of adjustments (used to change parameters or configuration), there may be entire patterns of functionality. For example, many systems have the ability to save a result and use it in a later reasoning episode is broadly applicable. A generic pattern based on this observation might be used to evolve a simple robot navigation system this pattern to allow it to recall past routes and reuse them.

Analogous Functionality: The addition of new functionality may be possible even without prior knowledge of that sort of functionality *if* similar functionality is present in exiting tasks and methods. For example, consider the case of a web browser trying to operate over a different kind of network. Even if there is no generic pattern for browsing over networks, the fact that browsing over one kind of network is similar to browsing over another makes it possible to perform this evolution using analogical reasoning.

6.5 Thesis

The primary claim to be investigated in this research is that **reflection over functional, causal, and compositional models enables effective addition of new capabilities to an intelligent agent**. The TMKL language will constitute an account of functional, causal models of an intelligent agent. As the conceptual architecture presented here is refined, it will develop into an account of evolutionary processing over these models. The language and processing together will form a unified theory of reflective evolution. The primary method for developing, elaborating, and presenting this unified theory will involve developing an agent architecture tool which interprets and evolves TMKL models. The primary method for testing, evaluating, and justifying this theory will involve modeling a variety of preselected agents using the tool and systematically varying not only the information presented to the tool but also the nature of the tool itself in the context of these agents. Through these methods, I intend to explore how and when functional reflective knowledge may be used in the evolutionary development of new agent capabilities.

References

[Abowd et al., 1997] Abowd, G., Goel, A. K., Jerding, D. F., McCracken, M., Moore, M., Murdock, J. W., Potts, C., Rugaber, S., & Wills, L. (1997). MORALE – Mission oriented architectural legacy evolution. In *Proceedings International Conference on Software Maintenance 97* Bari, Italy.

[Allemang, 1997] Allemang, D. (1997). ZD tutorial. Unpublished.

[Anderson, 1983] Anderson, J. (1983). A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, 22.

[Anderson, 1993] Anderson, J. R. (1993). *Rules of the Mind.* Hillsdale, NJ: Lawrence Erlbaum Associates.

[Bhatta, 1995] Bhatta, S. (1995). *Model-Based Analogy in Innovative Device Design.* PhD dissertation, Georgia Institute of Technology, College of Computing.

[Boddy & Dean, 1989] Boddy, M. & Dean, T. (1989). Solving time-dependent planning problems. In N. S. Sridharan (Ed.), *Proceedings of the 11th International Joint Conference on Artificial Intelligence* (pp. 979–984). Detroit, MI, USA: Morgan Kaufmann.

[Brooks, 1990] Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6, 3–15.

[Chandrasekaran, 1988] Chandrasekaran, B. (1988). Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *Knowledge Engineering Review*, 3(3), 183–219.

[Clements, 1996] Clements, P. C. (1996). A survey of architecture description languages. In *Eighth International Workshop on Software Specification and Design* Germany.

[Cox, 1996] Cox, M. T. (1996). *Introspective multistrategy learning: Constructing a learning strategy under reasoning failure.* PhD dissertation, Georgia Institute of Technology, College of Computing.

[Ducassé, 1993] Ducassé, M. (1993). A pragmatic survey of automated debugging. In *Proceedings of the First International Workshop on Automated and Algorithmic Debugging* Linköping, Sweeden.

[Gentner, 1983] Gentner, D. (1983). Structure mapping: A theoretical framework for analogy. *Cognitive Science*, 7.

[Goel, 1989] Goel, A. K. (1989). *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving.* PhD dissertation, The Ohio State University, Dept. of Computer and Information Science.

[Goel et al., 1994] Goel, A. K., Ali, K., Donnellan, M., Gomez, A., & Callantine, T. (1994). Multistrategy adaptive navigational path planning. *IEEE Expert*, 9(6), 57–65.

[Goel et al., 1997a] Goel, A. K., Beisher, E., & Rosen, D. (1997a). Adaptive process planning. Poster Session of the 10th International Symposium on Methodologies for Intelligent Systems.

[Goel et al., 1997b] Goel, A. K., Bhatta, S., & Stroulia, E. (1997b). Kritik: An early case-based design system. In M. L. Maher & P. Pu (Eds.), *Issues and Applications of Case-Based Reasoning to Design.* Lawrence Erlbaum Associates.

[Goel et al., 1996] Goel, A. K., Gomez, A., Grue, N., Murdock, J. W., Recker, M., & Govindaraj, T. (1996). Explanatory interface in interactive design environments. In J. S. Gero & F. Sudweeks (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence in Design* Stanford, California: Kluwer Academic Publishers.

[Griffith & Murdock, 1998] Griffith, T. & Murdock, J. W. (1998). The role of reflection in scientific exploration. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society.*

[Griffith et al., 1997] Griffith, T., Nersessian, N., Goel, A. K., & Clement, J. (1997). Exploratory problem solving in scientific reasoning. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*: Lawrence Erlbaum Associates.

[Griffith, 1997] Griffith, T. W. (1997). *A Computational Theory of Dynamical Modeling in Scientific Reasoning.* PhD thesis proposal, Georgia Institute of Technology, Cognitive Science Program, College of Computing.

[Hayes-Roth & Hayes-Roth, 1979] Hayes-Roth, B. & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science*, 3, 275–310.

[Johnson & Hayes-Roth, 1997] Johnson, W. L. & Hayes-Roth, B., Eds. (1997). *Proceedings of the 1st International Conference on Autonomous Agents*, New York. ACM Press.

[Kant, 1781] Kant, I. (1781). *The critique of pure reason.* On-Line Edition: Virginia Polytechnic Institute and State University. translated by J.M.D. Meiklejohn.

[Kirlik, 1998] Kirlik, A. (1998). The ecological expert: Acting to create information to guide action. In *Proceedings of the 1998 Conference on Human Interaction with Complex Systems, HICS-98* Dayton, OH.

[Kirsh, 1991] Kirsh, D. (1991). Today the earwig, tomorrow man? *Artificial Intelligence*, 47(1-3), 161–184.

[Koza et al., 1996] Koza, J. R., Goldberg, D. E., Fogel, D. B., & Riolo, R. L., Eds. (1996). *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA. MIT Press.

[Laird et al., 1987] Laird, J., Newell, A., & Rosenbloom, P. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33.

[Liver & Allemang, 1985] Liver, B. & Allemang, D. T. (1985). A functional representation for software reuse and design. *International Journal of Software Engineering and Knowledge Engineering*, 5(2), 227–269.

[Minsky, 1975] Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The Psychology of Computer Vision.* McGraw-Hill.

[Mitchell et al., 1989] Mitchell, T. M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., & Schlimmer, J. C. (1989). Theo: A framework for self-improving systems. In K. VanLehn (Ed.), *Architectures for Intelligence.* Erlbaum.

[Murdock, 1998a] Murdock, J. W. (1998a). *Modeling Computation: A Comparative Synthesis of TMK and ZD.* Techincal Report GIT-CC-98-13, Georgia Institute of Technology, College of Computing.

[Murdock, 1998b] Murdock, J. W. (1998b). Prolegomena to a task-method-knowledge theory of cognition. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society.*

[Murdock & Goel, 1998] Murdock, J. W. & Goel, A. K. (1998). A functional modeling architecture for reflective agents. In *Proceedings of the AAAI98 Workshop on Functional Modeling and Teleological Reasoning* Madison, Wisconsin.

[Murdock et al., 1997] Murdock, J. W., Shippey, G., & Ram, A. (1997). Case-based planning to learn. In *Proceedings of the Second International Conference on Case-Based Reasoning* Providence, RI.

[Newell & Simon, 1963] Newell, A. & Simon, H. (1963). GPS, a program that simulates human thought. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and Thought.* R. Oldenbourg KG.

[Punch et al., 1996] Punch, W., Goel, A. K., & Brown, D. (1996). A knowledge-based selection mechanism for strategic control with application in design, diagnosis and planning. *International Journal of Artificial Intelligence Tools*, 4(3), 323–348.

[Rugaber et al., 1995] Rugaber, S., Stirewalt, K., & Wills, L. M. (1995). The interleaving problem in program understanding. In *Proceedings of the Second Working Conference on Reverse Engineering* (pp. 166–175). Toronto, Ontario, Canada: IEEE Computer Society Press.

[Russell & Wefald, 1991] Russell, S. & Wefald, E. (1991). *Do the right thing: Studies in limited rationality.* Artificial Intelligence. Cambridge, Mass: MIT Press.

[Shippey et al., 1998] Shippey, G., Murdock, J. W., & Ram, A. (1998). The pluto system: Planning as a control strategy in multistrategy learning. Student abstract under submission.

[Stefik, 1981] Stefik, M. (1981). Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16(2).

[Stroulia, 1994] Stroulia, E. (1994). *Failure-Driven Learing as Model-Based Self Redesign.* PhD dissertation, Georgia Institute of Technology, College of Computing.

[Stroulia & Goel, 1994] Stroulia, E. & Goel, A. K. (1994). A model-based approach to reflective learning. In F. Bergadano & L. D. Raedt (Eds.), *Proceedings of the 1994 European Conference on Machine Learning* (pp. 287–306). Catania, Italy.

[Stroulia & Goel, 1995] Stroulia, E. & Goel, A. K. (1995). Functional representation and reasoning in reflective systems. *Journal of Applied Intelligence*, 9(1), 101–124. Special Issue on Functional Reasoning.

[Stroulia & Goel, 1996] Stroulia, E. & Goel, A. K. (1996). A model-based approach to blame assignment: Revising the reasoning steps of problem solvers. In *Proceedings of the National Conference on Artificial Intelligence - AAAI96* Portland, Oregon.

[Suchman, 1987] Suchman, L. (1987). *Plans and Situated Actions: The Problem of Human Machine Communication.* Cambridge University Press.

[Veloso et al., 1995] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental Artificial Intelligence*, 7(1).