Meta-Reasoning Over Goals: A Summary of the GAIA Project

Spencer Rugaber SPENCER@CC.GATECH.EDU

Ashok K. Goel GOEL@CC.GATECH.EDU

Design & Intelligence Laboratory, School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30308, USA

Lee Martie LMARTIE@UCI.EDU

Department of Informatics, University of California at Irvine, California 92697, USA.

Abstract

We present a summary of GAIA, an interactive environment for designing game-playing agents. A game-playing agent in GAIA contains not only knowledge about the game world, but also a model of its own goals, methods and knowledge. If and when the agent fails to achieve a goal, it uses its self-model to reason about its goals, identifies repairs to its design, and retrospectively adapts itself. We illustrate GAIA through experiments in designing agents that play a version of the turn-based strategy game called Freeciv.

1. Introduction

The GAIA project explores the use of teleology in modeling agents that play turn-based strategy games. Its overall hypothesis is that the use of teleology supports the diagnosis and revision of such agents to better adapt them to their game environments. Specific research questions include how to represent agent goals, how to relate the goals to the means by which the goals are accomplished and how to reason over the models in support of diagnosis and adaptation.

GAIA itself is an interactive environment for designing game-playing agents. In GAIA, a designer interactively designs a game-playing agent in a high-level agent modeling language called TMKL2. The designed agent contains not only knowledge about the game world, but also models of its goals, methods and knowledge. Given the model of the agent, GAIA automatically compiles the agent program code and executes the agent in the game world. If and when the agent fails to achieve a goal specified in the model, a meta-reasoning component called REM uses the agent's model to reason about its goals and failures, identifies repairs to its design, and retrospectively adapts the agent. We evaluate GAIA, TMKL2 and REM through experiments in designing agents that play parts of the turn-based strategy game called Freeciv[1]. In this paper, we

[1] http://freeciv.wikia.com/

present a summary of the GAIA project in order to build a connection with the newly formed Cognitive Systems and Goal Reasoning communities.

2.TMKL2

TMKL2 is an agent modeling language that is used to express the teleology of an agent's design and the means by which the teleology is realized. TMKL2 includes vocabulary for specifying agent goals, the mechanisms to accomplish the goals, and the agent's knowledge of its internal design and its external environment. GAIA realizes TMKL2 models using an interpreter. When used to model a Freeciv agent, the interpreter is capable of executing a model in conjunction with Freeciv's server program to play the game. GAIA assumes that the division between the agent's model and external parts is clearly defined, and that this division takes the form of an (Application Programming Interface) API to the external code. GAIA also makes an explicit distinction between adaptation-time modeling of the agent and run-time execution of the adapted agent: Adaptation takes place on a model of the agent and then the model is interpreted to effect agent behavior in the game world.

1.1. Goals

TMKL2 comprises three sublanguages for modeling `Goals`, `Mechanisms` and `Environment`, corresponding to the Tasks, Methods and Knowledge portions of the earlier TMKL language [Murdock & Goel 2008], respectively. The first sublanguage describes the agent's goals. A `Goal` expresses a reason that the agent does what it does, in terms of its intended externally visible effects on the agent's world. `Goals` may be parameterized, enabling the agent to target specific elements of its Environment, such as, for example, a specific city. A `Goal` is expressed via a pair of logical expressions describing the precondition for `Goal` accomplishment (called its `Given` condition) and the expected effect of `Goal` accomplishment on the agent's `Environment` (its `Makes` condition). The final element of a `Goal` specification is a reference to the means by which the `Goal` is to be accomplished. In this version of TMKL2, each `Goal` is associated with the single `Mechanisms` by which it is to be achieved.

1.2. Mechanisms

The `Mechanism` portion of a TMKL2 model describes how the agent accomplishes its `Goals`. There are two kinds of `Mechanisms`, `Organizers` and `Operations`, that are each defined in terms of two logical expressions describing their precondition for execution (`Requires` conditions) and their effect (`Provides` condition). An `Organizer` mechanism is defined as a finite state machine comprising `States` and `Transitions`. `Start`, failure and success `States` are all explicitly indicated. `States`, in turn, refer to `subGoals`, enabling hierarchical refinement of an agent's specification. Transitions may be conditional (dependent on a `DataCondition`) with respect to the agents current perception of the world, as expressed in its `Environment`. The other kind of mechanism is an `Operation`. Operations are parameterized invocations of computational resources provided to the software agent via its API to external software, such as the Freeciv server. That is, each `Operation` models one of the agent's computational capabilities.

1.3. Environment

A TMKL2 program includes a description of the agent's understanding of itself and the world in which its exists. In particular, the agent's Environment comprises a set of typed Instances and Triples (3-tuples) relating the Instances to each other. In order to describe Instances and Triples, TMKL2 provides two modeling constructs, Concepts and Relations. A Concept is a description of a set of similar Instances. It is defined in terms of a set of typed Properties. Moreover, Concepts are organized in a specialization hierarchy promoting compositionality and reuse. There is a built-in Concept called Concept. When a TMKL2 model is constructed, Instances of Concept are automatically added to it for each defined Concept, enabling reflection by the agent over its own definition. A Relation describes a set of Triples allowing the modeling of associations among Concepts. In particular, an Instance of one Concept can be related to Instance of another via a Triple.

1.4. Semantics

A TMKL2 model of an agent connects the Goals of the agent to the Mechanisms by which the Goals are accomplished. The program is declarative in the sense that all behavior is defined in terms of logical expressions (Given, Makes, Requires, Provides). Consequentially, one semantic interpretation of a TMKL2 program is that it declaratively describes the behavior that a software agent must exhibit in order for it to accomplish a set of top-level Goals. TMKL2 programs are not just descriptive, however: They can be used to actually control the modeled agent. This requires an operational semantics of TMKL2: A TMKL2 model prescribes the detailed behavior of the agent in the world. Operationally, a TMKL2 Model can be interpreted as a hierarchy of finite state machines controlling communication with the external software with which the agent interacts. Superior state machines in the hierarchy effect the accomplishment of superior Goals. FSMs corresponding to Goals without any subGoals are called *leaf* FSMs. All state machines execute synchronously; that is, at any given time, each machine is in a specific State. At the next virtual clock tick, all pending DataConditions for active leaf machines are evaluated, and the outgoing Transitions evaluating to true are traversed, resulting in entry into new States. Upon entry into a State, the corresponding subGoal and its Mechanism are interpreted. Mechanism interpretation ultimately resolves into Operation invocations and updates to the Environment. After all invocations have been processed, the Environment is updated to reflect any changes to the agent's run-time data structures made by the invocations. Interpretation terminates if the Organizer for the top-level Goal enters either a success or failure State.

1.5. An Example of a TMKL2 Model of a Freeciv Agent

Figure 1 presents part of the model of an agent, called Alice, capable of playing a simplified version of Freeciv. The figure illustrates a visual syntax for TMKL2. The partial model includes Alice's top level Goals and Organizers. In particular, the top (green) rectangle of the diagram denotes Alice's top-level Goal of collecting gold pieces. Contained within this rectangle is another (slate blue), depicting an Organizer comprising three States—an initial State (black circle), a subGoal reference (gray) and a final State (yellow). The subGoal itself is shown as the rightmost of the two (orange) rectangles on the second level. Its Organizer, in turn, refers to two subGoals—one that continually mints more gold until enough has been produced and the other determining when to end the game. The bottom two rectangles contain

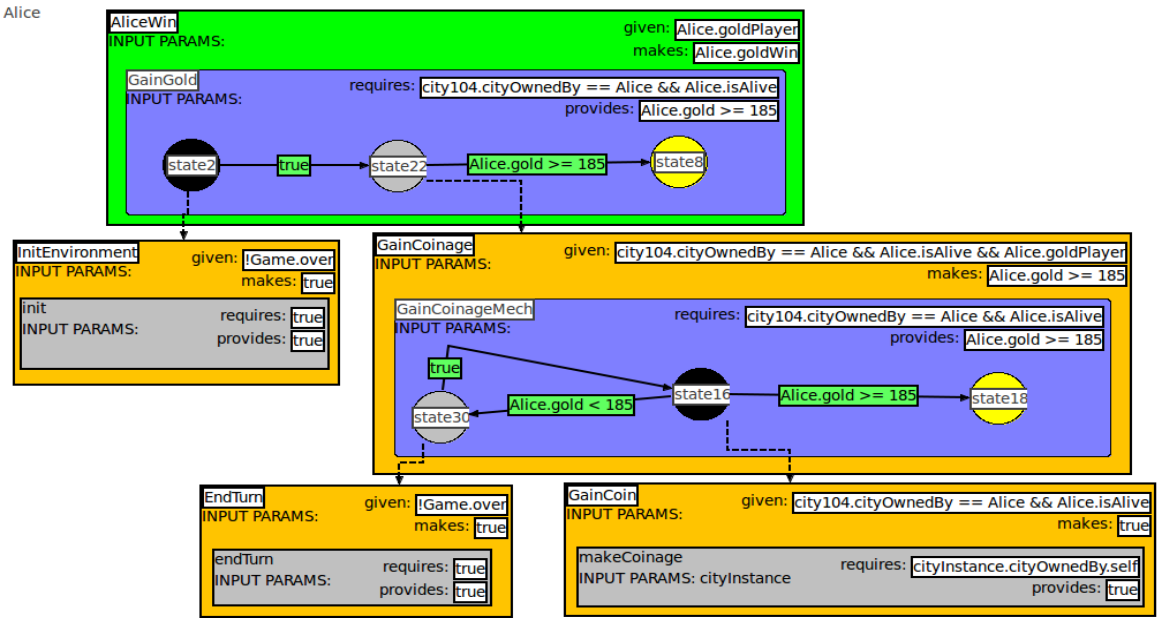

Figure 1: A portion of the TMKL2 model of Alice, an agent that plays a part of Freeciv.

`Operations` (gray), responsible for interacting with the Freeciv server. Complementing the `Goals` and `Mechanisms` shown in the figure is Alice's `Environment` (not shown). Example `Concepts` represented by `Instances` in the `Environment` include `City`, `Tile`, `Player`, and `Unit`.

2. GAIA

TMKL2 models can be constructed, modified and executed using the GAIA interactive development environment. GAIA is written in the Java programming language and is built using the Eclipse[2] software development environment.

2.1 The GAIA Architecture

The conceptual architecture for GAIA is presented in Figure 2. In the center left of the figure is SAGi, the GAIA user interface. REM is the reasoning module responsible for adapting models. Also part of GAIA is the model manager responsible for encapsulating access to agent models and persisting them to permanent storage. In-memory representation of TMKL2 models take the form of Java objects that are interpreted by the TMKL2 interpreter, which interacts with the world via the Runtime Communications Manager and associated queues.

SAGi invokes the TMKL2 interpreter to execute a model and thereby interact with the Freeciv server. The interpreter walks the TMKL2 tree of state machines iteratively until the agent either succeeds or fails to achieve its top-level `Goals`. When the interpreter attempts to accomplish a `subGoal` whose `Mechanism` is an `Operation`, it must place into the Operation Request Queue a request to the Freeciv server to execute a game action, encoding parameters as necessary.

[2] http://www.eclipse.org/

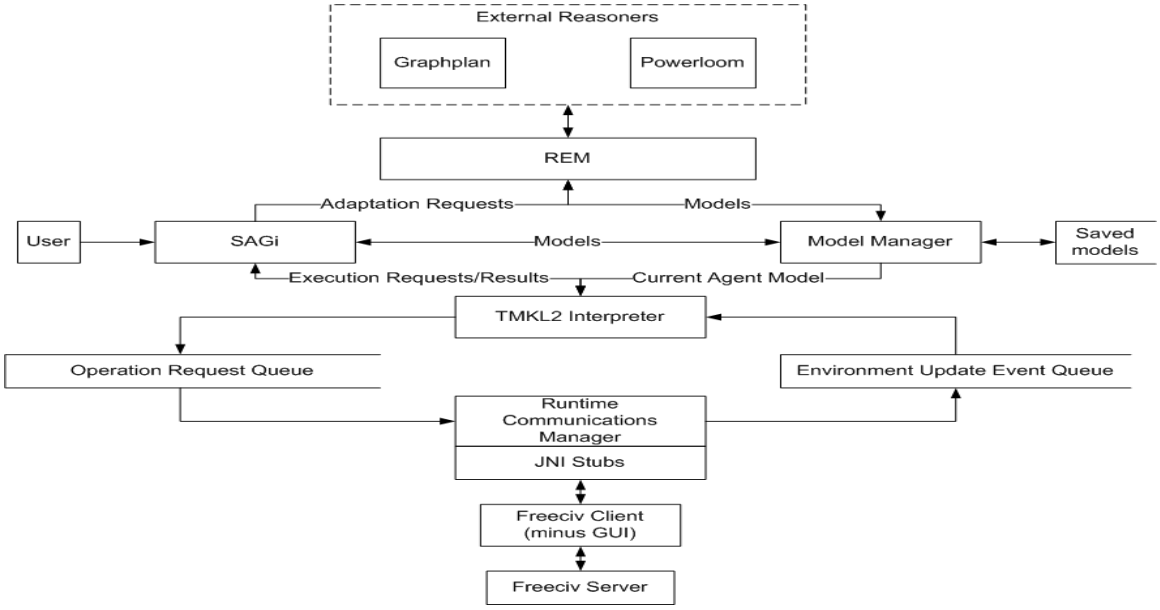

Figure 2: The Conceptual Architecture of GAIA.

REM is the other major component of GAIA. REM, when given an agent model and a situation—either a failed `Goal` or an altered `Environment`—produces an updated agent model engineered either to successfully accomplish the `Goal` or to take advantage of the new knowledge in the `Environment`. To achieve retrospective adaptation, REM performs three steps: localization (determining which of an agent's `subGoals` and associated `Mechanisms` were inadequate to accomplish the agent's overall `Goal`), transformation (devising an alternative `Goal`), and realization (providing/altering a `Mechanism` to accomplish the alternative `Goal`). Localization is accomplished in REM using a heuristic to find a low-level `State` in an `Organizer` such that the state's `Provides` condition suffices to accomplish the failing `Goal`. Further, the detected `State` must have a failing precondition (`Requires` condition). The presumption is that the `State` had not been reached, and, if it had been reached, then the agent would have succeeded.

Realization and transformation are accomplished by matching the failing situation against a library of adaptation plans, choosing a candidate transformation from the library and applying the result to the agent's model to produce a revised model. REM sits atop the Powerloom[3] knowledge representation and reasoning system. Powerloom supports automatic classification (truth maintenance) as well as natural deduction. TMKL2 logical expressions are easily mapped to/from Powerloom, and REM algorithms are easily expressed in Powerloom's variant of first-order logic.

3. Adaptation Scenarios and Results

To validate our approach to model-based adaptation, we have conducted several experiments, each involving variants of the Alice agent depicted in Figure 1. In the experiments, Alice plays a simplified variant of Freeciv against other agents. The simplified game consists of two agents.

[3] http://www.isi.edu/isd/LOOM/PowerLoom/

Each agent controls a civilization and is responsible for its government, economy, citizen morale, and military. Each civilization has one city, citizens in that city, and a number of warriors. Each game tile yields a quantity of food, production, and trade points each turn of the game. Food points feed a city's civilians; production points are used to support existing warriors or generate new warriors; and trade points are distributed among luxury, tax, and science resources. Initially both players start out in year 4000BC, with fifty gold pieces, zero warriors, and one worker that collects resources from a nearby tile. A city is either producing a warrior or collecting gold pieces on any given turn. Alice can win by collecting a specified number of pieces of gold, and the other agent can win by capturing Alice's city. An experiment consists of running Alice against its opponent and noting the results. Then, REM is tasked by the experimenter to adapt Alice, and the game is rerun. In order to best understand the results, FreeCiv was run deterministically throughout the experiments.

3.1. Experiment #1

The purpose of the first experiment we conducted was to test whether REM could make a trivial adaptation to improve Alice's performance versus Freeciv's built-in robot player. In general, a player of this reduced game has to make a decision about allocating resources between collecting gold and creating warriors to defend its city. At the beginning of the first experiment, Alice's strategy was to devote all of her resources to the former pursuit. An obvious adaptation is to adjust Alice to balance her resource allocation, and the first experiment tested whether REM could make this adaptation. In the experiment Alice played against Freeciv's robot player, which we call Frank, configured at its highest skill level. Although Alice had knowledge that Frank could win by capturing her city, she was unaware that Frank had more powerful weaponry and more production capacity than she had. When played against Frank, unadapted Alice directly succumbed to his attacking chariots, legions, and horsemen. Before losing, Alice was able to acquire 175 units of gold and lived for 3075 years. However, Alice failed to acquire sufficient gold to accomplish her `Goal`, thereby requiring retrospective adaptation. In this experiment, no transformation was needed. That is, the failure was that an `Organizer` rather than a `Goal` was flawed. Realizing a replacement `Organizer` took place by interjecting a new `State`, whose success would satisfy the preconditions of a problem `State`. The new `State` was created by first searching a small library of generic `Goal` patterns to see if any satisfy the preconditions of the problem `State`. After an instantiated `Goal` pattern was found it was assigned as the `Goal` of the new `State`. This new `State` was then inserted into the localized `Organizer` just prior to the problem `State`. This guarantees the problem `State's` precondition is satisfied upon its visitation. In Experiment #1, the new `State` was added with a `Goal` to build additional warriors. This `Goal` increases the defense of Alice's city if she is visibly outgunned on the game map. After performing this adaptation, the new agent, Alice', was tested against Frank. While still outgunned, Alice' fared better in longevity and defense. She lasted 3125 years and killed one of Franks powerful attacking units. Because some of her resources had been allocated to defense, she fared worse in gold acquisition, acquiring only 147 units. The lesson learned was that compensating for a well-understood limitation could be accomplished by making use of a simple heuristic alteration of a TMKL2 agent model and a small library of patterns.

3.2. Experiment #2

The previous experiment was an example of *retroactive* adaption in which a failure was mitigated. In Experiment #2, *proactive* adaption was attempted to take advantage of a slightly altered game rule. In particular, it now takes more gold units for Alice to win a game. Tests were run on Alice to see if Alice's model was still valid after the rule change. REM tested if each `Mechanism's Provides` condition satisfies its parent `Goal's Makes` condition; that is, if the `Mechanism` was capable of accomplishing the new `Goal`. If one of these tests failed, REM then located the responsible `Mechanism`. In this experiment, REM localized Alice's GainGold `Organizer`. Next, a replacement `Organizer` was created to achieve the new win condition. To do this, REM used an external planning tool, called Graphplan[4]. REM translated the initial game `Environment` into a Graphplan *facts* file. Then all `Organizers, Operations`, and game rules were translated into a Graphplan *operators* file. After pruning out operators with no effects, the resulting Graphplan file contained 10 operators. Next, REM ran Graphplan on the facts and operators files. Graphplan generated a three-stage plan capable of accomplishing Alice's top-level `Goal`. This plan was then translated back into an `Organizer` to replace GainGold. The lesson learned from this experiment was that for a simple numeric change, a no-longer valid TMKL2 `Organizer` can be located and adapted using an external planner.

3.3. Experiment #3

The first experiment described above was *off-line* in the sense that the adaptations were made after a game was completed. Experiment #3 is an *on-line* adaption in that Alice is changed while she is running. Moreover, her opponent, Barbra is also adapted during the game. In this experiment, both Alice and Barbra were reconfigured into two parts, one *allopoietic* and the other *autopoietic*. These term are borrowed from the literature of self-organizing systems and denote, respectively, the part of a system that changes and the part that does the changing. Alice's allopoietic part used a parameter, alpha, to determine how Alice should allocate her resources between obtaining gold or producing warriors. The autopoietic part of Alice adapted the allopoietic part by adjusting alpha to produce gold only if she had sufficient defensive capability to fend off Barbra's visible attackers. Similarly Barbra's allopoietic part used a parameter, beta, to determine the number of warriors with which to attack Alice's city. The autopoietic part of Barbra adapts the allopoietic part by adjusting the number of warriors Barbra attacks Alice. For both agents, the autopoietic part was itself a (meta-) agent. In particular, the meta-agent's `Environment` consisted of a description of the allopoietic part, including `Goals, Mechanisms` and (allopoietic) `Environment`. By monitoring game status, the meta-agent could make appropriate adjustments to the base agent's parameter by executing (meta) `Operations`. Running Alice versus Barbra resulted in the agents engaging in an arms race. Eventually Alice was able to defeat Barbra. In winning, Alice collected 186 gold units, Barbra had 6 dead warriors, Alice had 3 live warriors and never lost a battle. Barbra adapted herself 4 times, and Alice adapted herself 6 times. The lesson learned was that TMKL2 models allow for simple real-time adaptations by using meta `Operations` to control the agent strategy.

4. Discussion and Future Work

The GAIA development environment provides infrastructure enabling exploration of teleological modeling and reasoning over goals and the means to realize them. The experiments conducted so

[4] http://www.cs.cmu.edu/~avrim/graphplan.html

far have been a limited proof of concept. As such, many questions have been raised and future topics for research suggested.

- **Language extensions:** TMKL2 as described is limited to one `Mechanism` for each `Goal`. Alternative, possibly concurrent, `Mechanisms` and the reasoning necessary to choose among them might be provided. Also, as currently designed, TMKL2 is primarily aimed at expressing achievement `Goals`. The addition of invariants, along with augmenting REM to directly support truth maintenance, would extend GAIA modeling range. Much more ambitious is the ability to deal with non-functional concerns, such as performance.
- **Game design:** In addition to FreeCiv, we have used GAIA with several other games. For example, we have built an agent model to play TicTacToe and a TicTacToe game server. We then watch GAIA adapt the agent to play variants, such as *misère* (play to lose) and allowing players to use either X or O on any move. We have added some abilities to GAIA to support the designer in specifying game server APIs and generating stub `Operations`. Nevertheless, adding a new game still requires significant work on the part of the designer.
- **Adaptations:** The experiments described above illustrate but a few of the many kinds of adaptations imaginable. We have cataloged about a dozen such adaptation types, but the list is ad hoc. What is lacking is a unifying framework for them using which REM could specialize its adaptation capabilities. Also, the framework would enable a more systematic compilation of adaptation patterns.
- **Reasoning:** As it exists, GAIA makes use of PowerLoom, which is a truth maintenance reasoner. As described above, we have also hooked GraphPlan into GAIA, albeit on an ad hoc basis. The question then remains as to what reasoning capabilities are required by teleological and specifically adaptive questions. Among the possibilities are machine learning for better localizing failures from execution logs. Needed also are specialized capabilities for determining what existing `Mechanism` might best be applied (or adapted) to realize a modified `Goal`, and how to formulate a high-level `Mechanism` that combines existing low-level `Mechanisms`.
- **Meta-Reasoning:** Experiment #3, described above, was a very simple example of meta-reasoning, amounting to just parameter tweaking. Nevertheless, the generality of TMKL2's modeling capabilities are such that much more general schemes are possible. In particular, imagine a (meta-)agent formulated to deal with specific adaptation opportunities, including detection, localization, etc. That is, the process by which the experimenter designed Experiment #3's meta-agent could itself be coded in TMKL2 to deal with this particular class of adaptations.
- **Evaluation:** The experiments described above do not constitute a thorough evaluation of GAIA. Such a study would explore a variety of further questions such as: How robust is TMKL2 in dealing with different games? What is the relationship between class of adaption and required reasoning power? Important also are issues such as reasoning performance and usability of GAIA as a design environment. Ultimately, the key question will be the extent to which our approach to teleology, as manifest in the tight connection between `Goals` and `Mechanisms`, can address the general problem of adaptation.

5. Related Research

In this summary paper, we will only briefly cover closely related research; our technical papers cover related research in more detail. Our work perhaps is most directly related to research on meta-reasoning (Cox & Raja 2011). Much of the research on meta-reasoning for self-adaptation

has used self-models of agents that help localize modifications to the agent design, e.g., (Anderson et al., 2006; Fox & Leake 2001; Jones & Goel 2012; Murdock & Goel 2008). We can trace several themes in model-based self-adaptation in intelligent agents. For example, self-adaptations can be retrospective, (Anderson et al. 2006; Fox & Leake 2001; Jones & Goel 2012), i.e., after the agent has executed an action in the world and received some feedback on the result, or proactive (Murdock & Goel 2008), i.e., when the agent is given a new goal similar and related to but different from its original goal. As another example, self-adaptations may pertain to domain knowledge (e.g., Fox & Leake 2001; Jones & Goel 2012) or reasoning processes (Anderson et al. 2006; Murdock & Goel 2008). The GAIA architecture supports both kinds of adaptations.

Our earlier research on model-based reflection and self-adaptation (Murdock 2008) suggested that (1) goal-based models of intelligent agents that captures the teleology of the agent design can help localize the changes to the agent design needed for classes of adaptations, and (2) hierarchical organization of the goal-based models of the agent designs helped make the above localization efficient. The work reported here extends earlier work on self-adaptation in two ways. Firstly, the agent specification language TMKL2 has a better defined syntax and semantics than its predecessor TMKL (Murdock & Goel 2008). This adds clarity, precision and rigor. While many agent specification languages specify the goals, mechanisms, structure and domain knowledge of agents, TMKL2 explicitly organizes the agent's mechanisms and domain knowledge around its goals. Together, the goals and the mechanisms that achieve them specify the teleology of the agent's design. Goel & Rugaber (2014) describe GAIA in more detail.

6. Conclusions

While much of earlier work on model-based reflection and self-adaptation pertained to agents that operated in small, fully-observable, deterministic, and largely static worlds, GAIA operates in large, complex, dynamic, partially observable and non-deterministic worlds such as Freeciv. The experiments in self-adaptation described here cover a small range of retrospective and proactive agent adaptations. They demonstrate that (i) it is possible in principle to design game-playing agents so that their teleology can be captured, specified and inspected, (ii) the specification of the teleology of the agent's design enables localization of modifications needed for the three experiments in self-adaptation, and (iii) this self-adaption in turn enables the agent to play an interactive game, monitor its behavior, adapt itself, play the game again, and so on.

Acknowledgements

We thank Andrew Crowe, Joshua Jones, and Chris Parnin for their contributions to this work. We also thank the US National Science Foundation for its support for this research through a Science of Design Grant (#0613744) entitled *Teleological Reasoning in Adaptive Software Design*.

References

Anderson, M., Oates, T., Chong, W., & Perlis, D. (2006) The metacognitive Loop I: Enhancing Reinforcement Learning with Metacognitive Monitoring and Control for Improved Perturbation Tolerance. *Journal of Experimental and Theoretical Artificial Intelligence*, 18(3), 387–411, 20.

Cox, M., & Raja, A. (2011) *Meta-Reasoning: Thinking About Thinking.* MIT Press, 2011.

Fox, S., Leake, D. (2001) Introspective Reasoning For Index Refinement In Case-Based Reasoning. *Journal of Experimental and Theoretical Artificial Intelligence*, 13:63-88, 2001.

Goel, A., & Rugaber, S. (2014) Interactive Meta-Reasoning: Towards a CAD-Like Environment for Designing Game-Playing Agents. In *Computational Creativity Research: Towards Creative Machines* T. Besold, K-U. Kuehnberger, M. Schorlemmer & A. Smaill (editors), Chapter 17, *347-370*, Atlantis ress.

Jones, J., & Goel, A. (2012) Perceptually Grounded Self-Diagnosis And Self-Repair Of Domain Knowledge. *Knowledge-Based Systems*, 27:281-301, 2012.

Murdock, J., & Goel, A. (2008) Meta-case-based reasoning: Self-improvement through self-understanding. *Journal of Experimental and Theoretical Artificial Intelligence*, 20(1), 1-36.