Consistency Maintenance in the Design of Autonomous Agent Representations

Glenn Wasson, Anand Natrajan, Jim Gunderson, Gabriel Ferrer, Worthy Martin, Paul F. Reynolds, Jr.

Technical Report No. CS-98-06
March 18, 1998

Contact: wasson@virginia.edu

Web: ftp://ftp.cs.virginia.edu/pub/techreports/CS-98-06.ps.Z

Consistency Maintenance in the Design of Autonomous Agent Representations

G. S. Wasson, A. Natrajan, J. P. Gunderson, G. J. Ferrer, W. N. Martin, P. F. Reynolds Jr.
{wasson, anand, gunders, ferrer, martin, reynolds}@virginia.edu
keywords: agent architectures, plan execution, reactive control, situated reasoning, robotics

Abstract

Multi-representation models are an attractive design solution for layered autonomous agent architectures in terms of managing complexity of internal representations and separation of design concerns. However, the representations maintained by the multiple layers can become inconsistent with one another due to the nature of the layers' concerns and capabilities. Consistency maintenance in multi-representation models is an emerging concern in many domains, such as military simulations. We present an approach to consistency maintenance in multi-tier agent architectures. We draw on experience and techniques from the multi-representation modeling community. The benefit of this approach for autonomous agent designers is a conceptual framework through which to organize systems that must deal with temporal and mapping inconsistencies.

Introduction

This paper provides a conceptual framework for designing autonomous agents which maintain multiple internal representations. Many of today's agents are designed in accordance with the principles of various robot architectures (Bonasso *et al* 1997)(Gat 1992)(Wasson *et al* 1997). Such architectures adopt a multi-tier strategy in which different operations take place at different layers of the architecture based on the operation's time scale, and detail of information required. While we will state briefly what makes a particular representation effective for use by a particular level of an agent's architecture, the thrust of this paper is maintenance of internal consistency between the differing representations of the environment that exist within the agent, not the issue of maintaining consistency between internal representation and the world (Brill *et al* 1998).

The discussion of our design framework has been inspired by principles used by the multi-representation modeling community (Davis & Hillestad 1993)(Reynolds *et al* 1997). We will describe parallels between the needs of agent designers and those of multi-representation modelers. We begin by describing the needs of various layers of an agent architecture and what impact this has on the representation used by that layer. Next, we introduce similar issues confronted by the multi-representation modeling community. Lastly, we present design solutions from this community in the context of an example agent designed to operate in a simulated domain called BridgeWorld.

Multi-Layered Representation

The concerns and capabilities of agents operating in dynamic environments have led designers to create multi-layered architectures. A particular task may be divided into sub-tasks that can be assigned to specific layers of the architecture, based on amount of inference required, time-scale, level of abstraction, bandwidth, etc. (Bonasso *et al* 1997). Although the number and nature of layers differs between various architectures, typically there is some inference engine or planner, and some perception/action (PA) layer.

While there has been debate about the role of state, or representation in the PA layer of an architecture (Brooks 1986), previous work has shown that a PA layer can be more effective with representation (Brill *et al* 1998). However, the planner and PA layer representations are epistemologically linked, i.e., the representations of the planner are "grounded" in the representations of the PA layer (Agre & Chapman 1987). The representations of the PA layer are abstracted into the representations used by the planner. When we say "the representation used by a layer of an architecture", we refer to the data structures that the layer uses to describe its environment. For example, a planner's representation could be the encoding of a map. Examples of representations used by other layers of various architectures include lisp-like predicate structures (Firby 1987) and indexical-functional identifiers (Agre & Chapman 1987)(Wasson *et al* 1997).

The PA layer "binds" abstract representations currently needed by the planner to entities in the world via the PA's well-maintained representation. Maintaining representation means keeping it up-to-date with the state of the world, i.e., continually checking the data's validity. Note that this is different than merely storing the representation, which allows the information to become stale. It is this dependency between planner and PA layer representation that allows inconsistencies to arise.

An important consideration in designing a system of representation for an agent is that representations used by the PA layer must be time-efficient, i.e., contain specific information and be straightforward to update. Representation for the planner must be algorithmically efficient, that is, specifically geared toward the planning algorithm to be used. In the remainder of this section, we will discuss the characteristics of planner and PA representations in more detail and show various inconsistencies that can arise between the two.

Representation Requirements

It is computationally infeasible for a planner to have a complete model of its world and all operations that might change that world, represented at sufficient detail to allow optimum planning. One traditional approach to this problem is to develop a hierarchical plan using fewer, more comprehensive operators applied to abstract entities (Sacerdoti 1974). Planner representation must express enough detail for the planner to be aware of important environmental changes, yet be abstract enough to make planning both feasible and effective.

The perception/action layer is the primary interface between the agent and the world it inhabits. It senses the environment and uses effectors to modify it. This requires that the PA layer's representation be maintained using current sensor values and little inference. Exactly what can be represented and how that representation can be maintained depends on the sensor capabilities of the agent.

Maintaining representation places a heavy computational burden on the PA layer and so its representation can not be as abstract/comprehensive as the planner's representation. PA representation must be simple and direct, often representing details that are not considered important to the planning process, yet are crucial for controlling effectors.

Inconsistencies Between Representations

The planner and PA layer operate with different representations of time, and use significantly different representations of the world. Potentially the two world views can become inconsistent. Consider the following example of how inconsistency can result in plan failure. The planner issues instructions to open a door. Its representation of "door" is based on the domain knowledge that doors are a certain height and width and have handles in a certain position. The PA layer may locate an object of appropriate height and width, but may not locate a handle if it is angled from its assumed position. The representations are inconsistent because the planner has beliefs about this door, yet the representation of the PA layer does not substantiate those beliefs. Communication is required to resolve whether the PA layer should treat this object as a door. As another example, the planner issues instructions for an assembly to be attached to the center of a base plate. However, the actual attachment turns out to be off-center. The PA layer can represent this fact via the positions of the individual parts. However, the planner can only represent "attached" or "unattached". Here the inconsistency is with the spatial resolution to which the two layers understand attachment. Clearly, the planner's representation of the world can become inconsistent with that of the PA layer. In the next section we will see how similar inconsistencies arise from the coupling of differing types of simulations used in the multi-representation modeling community.

Multi-Representation Modeling

Multi-Representation Modeling (MRM) or Cross-Resolution Modeling is concerned with resolving conceptual and representational differences that arise from multiple representations joined for a common objective (Davis & Hillestad 1993). Much work has been done on MRM in military simulations, primarily training simulations. A common scenario is the coupling of a simulation that models military forces at an abstract level, (e.g. platoons, battalions or corps), with a simulation that models individual battlefield entities (e.g. tanks or wheeled vehicles). The MRM problem arises if simulation A models a given platoon that is also modeled as its constituent tanks in simulation B. If there is a way to maintain consistency within multiple, concurrent representation levels of an abstraction, Reynolds *et al* recommend Multiple Representation Entities (MREs) as the approach for capturing it accurately (Reynolds *et al* 1997).

A number of problems arise when multiple representations co-exist, many of which are eliminated by the MRE approach to MRM (Reynolds *et al* 1997). However, the remaining key issues that are addressed by this paper are mapping inconsistency and temporal inconsistency. Temporal inconsistency arises in military simulations in a number of ways. If a tank is perceived to have fired in two directions simultaneously, there is a temporal inconsistency since the tank's turret could not have possibly faced two directions at any time. If an entity expends ammunition on another dead entity because it did not perceive that the second entity was dead, there exists a simulation error arising from a temporal inconsistency. Mapping inconsistency occurs in military simulations due to a loss of information between two levels. For example, when tank positions are instantiated from a platoon's positions, the tanks are placed according to military doctrine. However, a rapid aggregation-disaggregation sequence may cause the tanks to "jump" position in a physically infeasible manner.

The MRE approach advocates representing an entity at all levels at all times. While Reynolds *et al* discuss the notion of a core set of components from which complete representations can be generated on demand (Reynolds *et al* 1997), here we use a scheme that stores all components at all layers at all times (note: this does not imply maintenance of all information at all times).

A Design Example: BridgeWorld

In this section we will use MRM techniques to design an agent that maintains internal consistency between its representations. Our agent's task is to build a bridge in a simulated, but dynamic and unpredictable domain, called BridgeWorld. BridgeWorld is populated with boards, nails, screws, nuts, brackets and tools such as hammers and screwdrivers. The agent creates a plan to build a bridge, including

fetching raw materials, constructing small assemblies and combining them into larger assemblies. The agent is equipped with two arm/hand-like effectors and a toolbelt capable of holding a number of small objects, such as nails and screws. The agent's vision system is simulated, but subject to noise. While the agent's actions can fail and its perceptions can be incorrect, the fidelity of the simulated environment is not of primary importance here; we focus on maintaining internal consistency.

Various inconsistencies can occur in the agent's representation of the BridgeWorld domain. Temporal inconsistency occurs when two layers of the architecture have differing views of some aspect of the simulation at the same time. For example, if the agent has nailed two boards together, its planner will have a **connected-boards** representation, allowing it to reference the entire assembly. However, manipulating the assembly requires the PA to have representations of the individual boards in order to place its effectors properly. Now suppose the boards were not nailed together securely and when the agent lifts the assembly, it falls apart. The PA layer is tracking the positions of the individual boards in its hands, and the information is present in the board representations to realize that they are no longer proximate. However, the planner still has the **connected-boards** representation. This represents a temporal inconsistency, which requires planner/PA layer communication to resolve.

Mapping inconsistencies arise when one layer represents some fact which another layer does not (or cannot). For example, consider the task of screwing a bracket to a board. Suppose the planner represents the necessary tool merely as **screwdriver**. The PA layer needs to know about specific perceptual qualities of the screwdriver (to find and grasp it) which may require the PA system to know the screwdriver's head style, phillips or flat. Even if the types of screwdrivers were differentiated by handle color, the PA layer would be representing head style, but in a less direct way. A mapping inconsistency exists here because one layer (the PA layer) represents a fact that another layer (the planner) does not. In other words, there is information loss between representations of the same object at different layers. If the screw to be used is a phillips head screw, then either a phillips or flat head screwdriver may be used and this information loss does not affect the agent. However, if the plan calls for a flat head screw, then only a flat head screwdriver can be used. The agent must use a specific tool and so the planner's representation is inadequate. We will discuss how to resolve both of these mapping inconsistencies in the next section.

Another kind of mapping inconsistency arises from the difficulty of maintaining state in a dynamic world. If the agent nails together two boards as above, it will have a planner representation of **connected-boards** and PA layer representation for each board. However, any differences between the actual positions of the connected boards and the positions ascribed to **connected-boards** cannot be captured in the planner's representation. This represents a mapping inconsistency. The difficulty is if the agent subsequently moves to perform another task, say to find more lumber, the agent no longer needs to maintain the representation for the individual boards it just nailed together. Since the task has changed (from a hammering task, to a navigation task) the PA layer will not expend the computational resources to maintain the representations from the last task. When the agent returns with more lumber, the agent's task will again require it to manipulate the connected boards. The planner can tell the PA layer about the **connected-boards**, but to create representations for the individual boards again, the planner must use domain knowledge of the meaning of **connected-boards** to initialize the PA layer representation. However, this domain knowledge will not necessarily correspond to the information that was lost when the original board representations were dropped. This may cause the PA layer to be unable to recognize the individual components because the PA layer's perception system may require pose information for identification.

Task Analysis

Dealing with representational inconsistencies begins with the agent's design. The first step is to identify what the agent will represent at each level and what inconsistencies exist between these representations. This is determined by starting with a task analysis. In layered agent architectures, tasks are decomposed into more and more concrete sub-tasks at each layer. This decomposition is partially based on the capabilities of the agent's sensors/effectors and the amount of autonomy that the planner is able to give to the other layers. The BridgeWorld agent has a three-layered system where the planner specifies a series of steps to achieve some goal, and each of those is broken into a set of actions by a middle layer (called the task executor or TE). Each action is performed by the PA layer. For each task (or sub-task) that a layer performs, there exist various *roles* in that task, which the agent must fulfill. Each of these roles forms an entity that should be represented by that layer.

For example, suppose the planner generated the step "assemble **T-joint$_2$**". The TE may decompose this task into the sub-tasks of placing a board appropriately, placing another board across one end of the first board at a right angle, and nailing the boards together. In the planner's notion of the task, something must play the role of **T-joint$_2$**. Once the T-joint is assembled, it can be "bound" to **T-joint$_2$** (the planner's representation) via the PA layer's representation. Each of the sub-tasks also contain roles. For example, nailing the boards together requires objects to fulfill the roles of **board$_1$**, **board$_2$**, **hammer**, and **nail**. Each of the roles in the tasks executed by the PA layer can be represented. Using the planner and PA layer representations, the designer can de-

termine where inconsistencies can exist. For example, the agent must maintain internal consistency between the representations for the components that make up the assembled T-joint and the planner's representation for **T-joint$_2$**.

Since the representational needs of the different layers of the architecture differ significantly, some amount of inconsistency is inevitable, and not necessarily undesirable. Therefore, when the designer decomposes the agent's task and analyses the task roles, the design goal is not total elimination of inconsistency, but rather "bounded inconsistency". This means the agent designer must decide which potential inconsistencies between the different layer's representations are tolerable given the application, and which are not. Consider the task of hammering a nail. Since the PA layer can use either a claw or ballpeen hammer to pound the nail, it is reasonable for the planner to not represent types of hammers, even though the PA layer may need to know such facts to identify and manipulate actual hammers that exist in the environment. However, in the previous example of selecting an appropriate screwdriver for a flat head screw, the planner did need to represent the type of screwdriver. In that case, the inconsistency was outside of acceptable bounds.

Dependency Graphs

Once the entities to be represented at the various layers are known, potential inconsistencies must be identified. We propose the use of dependency graphs (DGs) as a tool for capturing consistency in multiple representations (Natrajan *et al* 1995). Our DGs are conceptual tools that focus a designer's attention on consistency maintenance aspects of multiple representations through a set of dependencies called accumulative, distributive and interaction.

Figure 1 depicts a simple sub-assembly (a T-joint) from BridgeWorld as well as representations used by the planner and PA layer. Figure 1a shows the configuration of the T-joint (two boards that have been nailed together). The planner maintains a representation of the T-joint that it can reason about and treat as a single object. However, the PA layer deals only with the environmental aspects which its sensors can directly acquire: boards, brackets, nails, screws, etc.

Figures 1b and 1c show the same DG with various dependencies highlighted. In the DG, the planner and PA layer have different representations of the T-joint that must be maintained consistent with each other. The components of the representation are the nodes of the DG and the dependencies between components are the arcs between the nodes. We use the notation **object{property list}** to denote the attributes that a particular layer stores for a particular object.

Figure 1b highlights some interaction dependencies below the PA layer's representation. Interaction dependencies capture changes to the representation from the environment

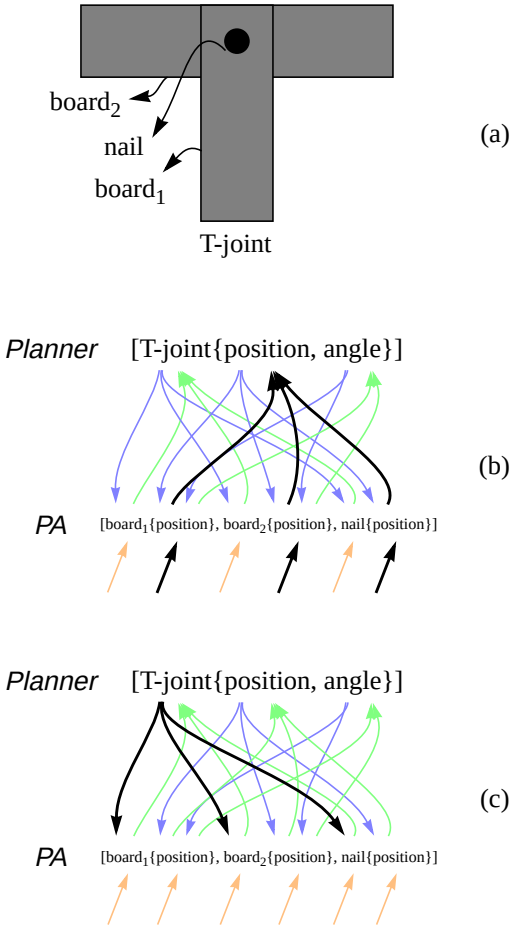

Figure 1. Dependency Graphs for an object

or other objects. Also, Figure 1b highlights accumulative dependencies between the PA layer's representation and the planner's representation. Accumulative dependencies capture the notion that a certain node is the accumulation of some other nodes. For example, **T-joint{position}** may be regarded as an accumulation of **board$_1${position}** and **board$_2${position}**. In this case, the accumulating function may be a bounding box, but there may exist other components with different accumulating functions such as summation, averaging or centroid. Figure 1c highlights distributive dependencies. Distributive dependencies capture the notion that if an action is required on a planner node, that may require actions on a set of PA layer nodes. Some nodes may be the distribution of a single node. The accumulative and distributive dependencies capture the notions of "is-part-of" and "has-part", respectively. Changes to the multiple representations originate from the interaction dependencies and propagate via the directed graph to other components. Conceptually, each node is affected by an incoming dependency. The node's effects are propagated via outgoing dependencies to other nodes with different semantics depending on

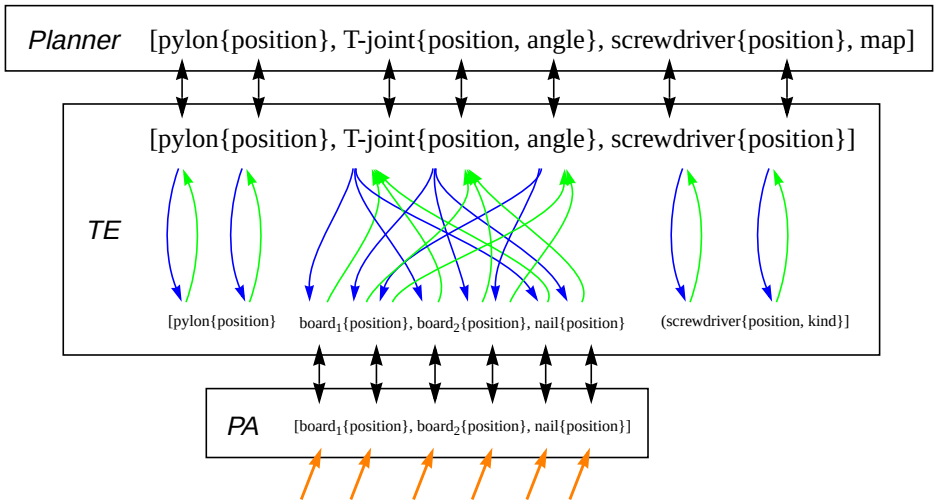

Figure 2. Correlation between planner and PA layer representations

which kind of dependency is followed. The result of this graph traversal is to keep the representations internally consistent in the face of dynamic changes to the components of the representation.

Figure 2 depicts the autonomous agent for BridgeWorld as a Multiple Representation Entity. The planner's representation includes **T-joint** and some other objects that the planner must know about in BridgeWorld. The TE maintains a dependency graph (DG) for the two representations. The bidirectional arrows from the planner to the TE and the PA layer to the TE imply that the corresponding attributes are the same and need not be duplicated in implementation. The PA layer represents only those entities important to its current task. In figure 2, only a portion of the planner's representation is being maintained by virtue of the PA layer's current task and hence its current representation. The maintenance of planner representation is facilitated by the DGs stored in the TE.

Consistency Maintenance with DGs

As outlined earlier, changes to representations originate from interaction dependencies. These changes are propagated to other representations via the other dependencies. In this respect, consistency maintenance resembles a graph traversal. The origin of the traversal is when one or more representation components are affected by changes in the environment detected by the PA layer's sensors. The traversal is continued along accumulative or distributive dependencies. Whenever an accumulative dependency is traversed, the corresponding accumulative function must be applied in order to gain the new state of the affected representation. The accumulative function might use the state of other representations. For example, when the accumulative dependency between **board$_1${position}** and **T-joint{position}** is traversed, an accumulative function that computes the new position of the **T-joint** may be invoked. This function may require the position of the second board and the position of the nail. Whenever a distributive dependency is traversed, the corresponding distributive function is applied in order to gain the new state of the affected representations. When designing a means for a planner to have a T-joint moved, the distributive dependency maps the desired T-joint location to desired locations for the individual components. It is clear then that the DG maintains temporal consistency between the two representations by virtue of propagating every change to all representations.

Mapping inconsistency is harder to solve without domain-specific information. In many cases, mapping inconsistency may be resolved by proper partitioning of the domain space. For example, in figure 2, if the planner never cares about the type of screwdriver, then the mapping inconsistency between the PA layer's and the planner's representations of the screwdriver does not matter. In other words, the planner must never alter its behavior based on the kind of the screwdriver, though the PA layer may. In other cases, a mapping inconsistency may be turned into a temporal consistency issue by representation augmentation. If the planner did not store **T-joint{angle}**, there would be no way to reflect the angle between boards at that level. This loss of information represents a mapping inconsistency since this fact is easily computable at the PA layer. However, if we augment the planner's T-joint representation to contain the board angle attribute (and add appropriate accumulative and distributive dependencies), we have changed a mapping inconsistency to a temporal one. Note that we used domain specific information to resolve this inconsistency; under our current understanding, generally this is true.

Related Work

A number of agent architectures have relied on representational abstraction. The first of these, ABSTRIPS (Sacerdoti 1974), was designed to address the combinatorial explosion resulting from representing the world in as detailed a manner as possible. The model of maximum detail was referred to as the ground space, and the spaces of decreasing detail were called the abstraction spaces. Each level in the abstraction space hierarchy was characterized by the level of detail used to specify preconditions for operators. Plans were generated by first planning in the more abstract space, and then adding preconditions at each level down in the hierarchy. Any inconsistencies that existed got resolved as the plan got refined in each abstraction space. Similar ideas are found in numerous other planners from the literature (see (Tate *et al* 1990) for a review). Note that the representations in different abstraction spaces are qualitatively different from the representations that exist at different levels of a combined planning/PA architecture. The different abstraction spaces still share the same fundamental assumptions of the planner representation, that is, they assume perfect knowledge, a discrete time scale, that operators always succeed, and that changes in the world state are solely the result of operators. None of these assumptions are held by a PA system.

Davis and Hillestad (Davis & Hillestad 1993), Reynolds *et al* (Reynolds *et al* 1997) and Natrajan *et al* (Natrajan *et al* 1995) have explored issues of consistency maintenance for conjoined military simulations. We believe ours to be the first work to address these concerns for autonomous agents with multi-layered architectures.

Conclusions

The design of multi-tier autonomous agents presents many challenges due to the nature of inconsistencies that can arise between the multiple representations within the agent. These representations must be reconciled with each other despite the different characteristics of the different layers. We drew on the experience of the multi-representation modeling community to show that representation inconsistencies can be categorized as either temporal or mapping inconsistencies. Our design framework shows how these inconsistencies can be found and addressed.

Multiple Representation Entities have been used with military simulations to internalize consistency maintenance among multiple representations of the same entity. Dependency Graphs encode solutions by making the relationships between representation components explicit. Dependency Graphs have proven useful in identifying potential temporal and mapping inconsistencies in BridgeWorld and other examples. This paper identifies important components of a design strategy (task analysis and dependency identification) that reduce or eliminate inconsistency in autonomous agents.

References

[1] Agre, P.E. and Chapman, D. 1987. Pengi: An Implementation of a Theory of Activity. *AAAI-87*: 268-272.

[2] Brill, F., Wasson, G., Ferrer, G. and Martin W. 1998. The Effective Field of View Paradigm: Adding Representation to a Reactive System. *Engineering Applications of Artificial Intelligence issue on Machine Vision for Intelligent Vehicles and Autonomous Robots*. to appear.

[3] Bonasso, R., Firby, R., Gat, E., Kortenkamp, D., Miller, D. and Slack, M. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2).

[4] Brooks, R.A. 1986. A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, RA-2(1):14-23.

[5] Davis, P.K. and Hillestad, R.J. 1993. Families of Models that Cross Levels of Resolution: Issues for Design, Calibration and Management. *Proceedings of 1993 Winter Simulation Conference*.

[6] Firby, R.J. 1987. An Investigation into Reactive Planning in Complex Domains. *AAAI-87:* 202-206.

[7] Gat, E. 1992. Integrating Planning and Execution in a Heterogeneous Asychronous Architecture for Controlling Real-World Mobile Robots. AAAI-92: 809-815.

[8] Natrajan, A., Reynolds Jr., P.F. and Srinivasan, S. 1995. Consistency Maintenance Using UNIFY, UVa Computer Science Technical Report 95-28.

[9] Reynolds Jr., P.F., Natrajan, A. and Srinivasan, S. 1997. Consistency Maintenance in Multi-Resolution Simulations. *ACM TOMACS*, July 1997, 7(3): 368-392.

[10] Sacerdoti, E.D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5: 115-135.

[11] Tate, A., Hendler J. and Drummond, M. 1990. A Review of AI Planning Techniques. In Readings in Planning: 26-49.

[12] Wasson, G., Ferrer, G. and Martin, W. 1997. Systems for Perception, Action and Effective Representation. *FLAIRS-97*: 352-356.