Spring 4-15-2005

A Comparison of Planning Based Models for Component Reconfiguration ; CU-CS-995-05

Naveed Arshad
University of Colorado Boulder

Dennis Heimbigner
University of Colorado Boulder

Department of Computer Science
University of Colorado
Boulder, CO 80309-0430

A Comparison of Planning Based Models for Component Reconfiguration

Naveed Arshad
Dennis Heimbigner
{arshad,dennis}@cs.colorado.edu

Technical Report CU-CS-995-05

April 15, 2005

Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

Abstract. Dynamic Reconfiguration is the process of changing a component-based system with minimum disruption and maximum automation. This process consists of three phases i.e. sensing the present state, planning for a new state and acting on the system to move it to the new state. Although, there are techniques and models for the sensing and acting phases, planning is a relatively neglected phase in this process. We have been using planning for the dynamic reconfiguration of component-based systems. However, planning systems provide different modeling languages. These languages have their own characteristics. Not all of these characteristics are important for the dynamic reconfiguration of software systems. In this paper we compare and evaluate the strengths and weaknesses of two planning languages: PDDL (Planning Domain and Definition Language) and AML (Aspen Modeling Language). Although the original target of both of these languages is not dynamic reconfiguration, we will discuss how their features can be used to model planning the dynamic reconfiguration of component based systems.

1. Introduction

Dynamic reconfiguration is the process of dynamically loading, moving and unloading of components so that the system can quickly adapt to changing environmental conditions, while maintaining its consistency [8]. Dynamic reconfiguration of component-based systems (CBS) is necessary because of the increased automation associated with component-based systems. Indeed, the process of adding, removing and updating of components cannot be avoided in any system. However, it is not possible to stop the whole system to make any change(s) in the system. Dynamic reconfiguration is required in large systems, where it may not be possible or economical to stop the entire system to allow modification to part of its hardware or software [16].

The next generation of CBS relies mostly on self-configuring systems that reconfigure themselves as much automatically as possible. IBM autonomic computing, HP's Utility Computing, Sun's N1 and Microsoft's Dynamic Systems Initiative (DSI) are all geared towards providing more self-configuring ability to the systems. The basis for self configuring systems is dynamic reconfiguration.

Although, the idea of dynamic reconfiguration to provide self-reconfiguring ability in CBS is promising, there are certain issues related with the process of dynamic reconfiguration that makes it difficult. Component-based systems are complex in terms of number of components and their inter-relationships. Maintaining a consistent state introduces an added complexity. Such systems are often heterogeneous [24] and often distributed. Lastly, there may be a large number of possible reconfigurations from which to choose[14], and the optimal choice may not be easy to determine. All of these factors make reconfiguration a difficult process.

In an attempt to address these complexities, we are applying automated planning systems from the Artificial Intelligence community to this problem [3]. Planning systems can automatically construct plans for the reconfiguration of a system. They can search the space of possible reconfigurations to find one that is optimal or that at least meets acceptable criteria for the new configuration.

The focus on planning came from the observation that the process of dynamic reconfiguration looks similar to the "sense-plan-act" (SPA) approach in control-based systems. Sensing is the monitoring of the state of the system and its associated components. Planning creates a plan for moving the system from one state to another. A plan is a sequence of steps that takes the system from one configuration state to the desired reconfigured state. Acting is the phase of actually carrying out the steps from the plan to reconfigure the system.

Traditionally, the AI planning systems used to solve toy problems like blocks world. However, now these systems are powerful enough that they are now being used in planning for the areas like spacecraft missions, intelligent manufacturing, operations research and others. Also, in past the planners could only plan for STRIPS-like domains where there are no numeric or temporal constraints. However, now these planners can take into account constraints like resource usage, time availability, cost etc. Moreover, instead of producing sequential plans these planners can now generate parallel plans.

The key to successful use of planning is the development of specifications of the planning *domain* and of the initial and final states (configurations) of the system. The planning domain describes the structure and constraints of a particular problem area. In our case, the domain must describe the system to be configured: its components, inter-connections, and the constraints on those components. The initial and final states describes how the components are connected (initial state) and how they should be connected (goal state).

There are various planning specification languages available for describing the domain and states, and each of these planning languages has its own strengths and weaknesses. As far as we can determine, this is the first

attempt to use the planning specification language for planning for dynamic reconfiguration in CBS. Therefore, it is necessary to find out the strengths and weaknesses of these languages in terms of dynamic reconfiguration modeling.

In this paper we are going to examine the strengths and weaknesses of two planning languages. These languages are PDDL (Planning Domain and Definition Language) and AML (Aspen Modeling Language). PDDL is the standard language for the International Planning Competition (IPC) for the last four years. PDDL mostly has a first order logic syntax. Most of the planners available today (e.g., LPG[13], MIPS[9]) use PDDL. One reason for using PDDL is to also to experiment with different planners so that we can compare the performance of the planners for the dynamic reconfiguration planning problems.

AML is the modeling language for a system called ASPEN (Automated Planning and Scheduling Environment). It uses C like syntax. ASPEN is used in many NASA spacecrafts for planning the reconfigurations in these spacecrafts. ASPEN also has to capability of replanning in case of failure of some part of the plan. Because in dynamic reconfiguration the plans can fail at certain time, it is a good idea to see how replanning increase the automation of the dynamic reconfiguration process. Before we look at the modeling aspects of planning specification languages we first look at some other attempts to model the dynamic reconfiguration in component based systems in the 'Related Work' section.

2. Related Work

The area of modeling systems for dynamic reconfiguration is an area of significant research. Conic [17] was an environment of designing distributed systems that separated the idea of system structure from functional behavior. Darwin [18] a successor of Conic was a language that gave the flexibility of representing static and dynamic structures that evolve during the lifetime of the system. POLYLITH [20] provided the adaptation capability in heterogeneous environments. Durra [16] describes the system as a set of components that have alternate configurations for runtime and their respective configuration transitions. Manifold [21] is an attempt to use coordination languages for reconfiguration. It represented the reconfiguration as control or event driven. Agnew et al. used a declarative reconfiguration language for the dynamic reconfiguration of component-based systems [2].

3. The Synergy between Dynamic Reconfiguration and Planning

Although, dynamic reconfiguration can be used whenever a change is needed in the system, however, dynamic reconfiguration is specially required when the system is attacked from outside. Often, malicious attacks can take the system into unanticipated states. These unanticipated states can result from the loss of critical components and this loss could be catastrophic in some cases. The task of dynamic reconfiguration at this time needs to restore the system and its functionality [25]. However, doing this involves many constraints that if done manually could take a lot of time. Planning can be used to overcome some of the problems associated with these constraints.

Critical parts of the system must be back up in a certain period of time. In planning one can specify the temporal goals and also the criticality of the components. Therefore, given these constraints the planner's job is to come up with a plan that starts the critical parts of the system as soon as possible. Moreover, resources e.g. disk space, bandwidth, machines could be scarce at the time of dynamic reconfiguration. Therefore, system cannot use the whole set of resources because they may be used somewhere else. However, in planning one can specify the availability of resources and ask the planner to optimize the set of resources over the course of planning. The priority of whether the backing up of the critical parts of the system has precedence or the minimization of resources is the priority can also be specified in planning in terms of metrics. The planner then produces plans that satisfy the given metrics.

Malicious stacks often leave the system in an unhealthy state. There are situations where system cannot go back to its original state, because of the catastrophic failure of some of the components. Moreover, there may not be a predefined configuration of the system present based on the available set of components. In this kind of situation planning is helpful because it can find an optimum state based on the present set of constraints and availability of the components in the system.

Component based systems are constructed in hierarchical way as components can be built from other components. In hierarchical systems the desire is to specify high-level goals and the system should be able to figure out low-level actions that achieve the goal. According to Brown et al. 40 percent of the root cause of the outage problems comes from operator errors [5]. These outage problems occur because the operators have to take care of the system at a number of levels. This is a similar problem with spacecraft mission operations. The reconfiguration of a spacecraft is achieved by specifying high-level goals and the planner in the spacecraft finds the acceptable set of low level commands to achieve the high level goals. Therefore, we can use planning to solve the similar problem in CBS.

Planning could be used to above the problem of restoring the system after an attack. However, the semantics of the system are needed to be encoded in a planning language before we can make use of the advantages planning can give to us. Planning languages have their own strength and weaknesses. A single planning language cannot solve all the problems mentioned above. In this paper we will model dynamic reconfiguration of component-based systems in two languages to evaluate their applicability towards component-based systems.

4. PDDL (Planning Domain and Definition Language)

PDDL is the language of the International Planning Competition (IPC) [10]. It emerged as a research language but now it is being used in a variety of disciplines for planning. European Network of Excellence in AI Planning (PLANET) [22] has identified a number of areas where planning could be applied. These areas range from intelligent manufacturing to robot motion planning. PDDL is mostly used in these areas as the language for planning. There are a number of planners that use PDDL [9,13].

A PDDL specification consists of two parts. There is domains specification that is fixed and common to all problems in that domain. The second part is a set of problem specifications that specify specific initial and goal states. The domain encodes all the semantics of the planning. These semantics include type of entities being used in planning, entity predicates, utility functions and actions. Following is a brief description of these parts.

- Types of entities: in our case, this consists of components, connectors, and machines.
- Entity Predicates/Facts: the predicates associated with entities (see the section marked “pproperties” in Figure 2). An example might be “at-machine”, which is a predicate that relates a component (or connector) to the machine to which that component is assigned. The domain actually specifies simple predicates, which are n-ary relations. These can be combined using logical operators into more complex predicates. As with Prolog, instances of these n-ary relations can be asserted as facts, and a state is effectively a set of asserted facts. Predicates are also referred to as constraints.
- Utilities: a variety of utility functions can be defined to simplify the specification (see the “functions” section of Figure 2). An example might be “local-connection-time”, which computes the time to connect a component given that the component and the connector are on the same machine.
- Actions: the actions are the steps that can be included in a plan to change the state of the system (see “durative-action” items in Figure 2). The output plan will consist of a sequence of these actions. An example is “start-component”, which causes the state of a component to become “active”. Actions have preconditions (“at start” in Figure 2) and post-conditions (“effects” in Figure 2). The post-conditions can add, modify, or remove facts from the on-going state that is tracked by the planner during plan construction. The actions are called “durative” because they have an assigned execution time that is used in calculating the total plan time.

The second part of PDDL is the problem. A problem is variable and consists of initial state and a goal state. The initial state represents the current state of the system. This section defines the known entities (components, connectors, machines) and asserts initial facts about those entities. The goal state represents the desired state of our system. It specifies predicates that represent constraints that must be satisfied in any plan constructed by the planner. Finally, the metric, that is to be used to evaluate the quality of a plan. In our case the metric is mostly the minimal total execution time for the plan.

5. AML (Aspen Modeling Language)

AML (Aspen Modeling Language) is the language of a system from JPL (Jet Propulsion Library), called ASPEN. ASPEN is basically targeted for the spacecraft mission operations. The mission operators specify high level goals in AML and the job of ASPEN is to produce optimized sequence of low level operations that are actually executed [7].

AML has a C-like syntax. The central data structure in AML is an 'activity'. Activities can be decomposed into further sub activities. There are seven core model classes in AML. The modeling like in PDDL is separate from the actual problem. However, because ASPEN works with iterative repair, the activities specified in the model file can be added, deleted, moved etc to resolve the conflicts in a plan. A brief description of each class is given below.

- Activity: An activity represents an action in the plan. An activity has a start time, end time and duration. Activities can be divided into sub activates by using decompositions. Activities can use resources. Activates can also change the state of a state variable. Activates can also dependent on other activities through constraints. (see 'Activites' in Figure 3).
- Resources: A resource represents a variable over the lifetime of the system. A resource could be depletable, undepletable, atomic or concurrent. (see 'Resources' in Figure 3).
- States: A state represents value of a discrete variable over time. Activities can change the state variable. (see 'states' in Figure 3).
- Parameters: A parameter is an abstracted value of a range. There are four types available in AML i.e. integer, boolean, float and string.
- Parameter Dependencies/Functions: Dependencies amongst the parameters of the system could also be defined. The dependencies represent a parameter constraint network. The planner has to satisfy all the dependencies in a valid plan in AML.
- Temporal Constraints: Temporal Constraints maintain the partial ordering of activities. Partial ordering of activities are necessary to specify the ordering of actions in the plan. (see 'constraint' in activity StartComponent in Figure 3)
- Preferences: Preferences are the metrics given in AML. Preferences are used to calculate the score of plans. There are five basic preference functions available in AML. These are local activity variable, activity/goal count, resource/state variable, resource/state change count and state duration.

Apart from the model, the initialization of a plan is also needed for ASPEN to make a plan. The initialization consists of the goals and the initialization of any resources or parameters. The goals can be specified as mandatory or optional.

After having described the basic syntax of these two planning specification languages, we are going to describe a basic architecture of a component-based system. This description is followed by the discussion of modeling this architecture in the planning specification language.

6. The Domain of Reconfigurable Systems

We adopt a simplified version of the models used in Architecture Definition Languages (ADLs) [11,12] as the basis for our specification. Our model differs from ADLs in that it also includes information about the structure and state of the environment. In our case, that environment is the set of machines onto which a software system is deployed. For our purposes, then, the model consists of three kinds of entities: *components*, *connectors,* and *machines*.

Components contain the logic of the system. A component is any entity that one can manage. An instance of a component can only exist at one machine at one time. Components need to be connected to a connector in order to communicate with other components. Some of the operations that can be performed on the components are starting a component, stopping a component, and connecting a component.

Connectors provide communication links. Each connector instance exists on one machine. The connector can be linked to other connectors for communication. A connector can be thought of as a weak form of the connectors described by Mehta et al. [19]. The connector needs to be connected to another connector before it

can accept connections from the component. The connector has almost the same operations as a component, except that it has an interconnect operation that links it with other connectors.

Machines are places where components and connectors are deployed. A machine may have a resource constraint that controls the number of components and connectors that may be assigned to that machine. The operations that can be performed on the machine are *start* and *stop*. Note that we do not explicitly model inter-machine connections. We assume that all the machines are connected to a network and that any two machines can communicate using, for example, TCP/IP. We assume that a connector deployed on a machine will use the inter-machine communication channels as the substrate for the connector's communication activities.

Components, connectors, and machines all have associated state machines that define what states they can achieve and in what order. These states are shown in Figure 1. Components and connectors have essentially the same set of states, so they are unified in the figure.

Component and Connector States: A component (or connector) starts in the *inactive* state. It can transition to the *active* state and to the *connected* state, which indicates that the component/connector has been connected to some existing connector. The component/connector can also reach a *killed* state, which indicates that it has failed. A killed component/connector cannot be restarted; rather, a new instance must be created and started.

Machine States: Machines have a somewhat simpler state machine. They can be *down*, *up*, or *killed*. Components and connectors cannot be assigned to a machine unless it is in the up state.

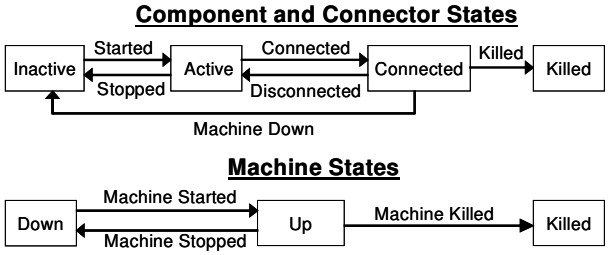

Fig. 1. State Machine Diagram for Components, Connectors and Machines

7. Modeling the Dynamic Reconfiguration Domain in PDDL

In this section we will describe our experiences with modeling in PDDL. We will model five aspects of dynamic reconfiguration of component based systems.

Initial and Target Configuration

PDDL has a notion of initial state and goal state. Our goal is to change the configuration of system from an initial configuration to a target configuration. Therefore our initial state is the initial configuration and our goal state is the target configuration. In different situations the initial configuration and goal configuration are different. For example there are two kinds of possible initial configuration. The first is when the components are being deployed. In this case the initial state could be just a set of machines, components and connectors. However, when there is reconfiguration of a running system, the initial state is the present configuration of the system. Likewise, the goal state could be also of a different set of target configurations. This could be divided broadly into two kinds of goal states: Explicit Configuration and Implicit Configuration. The explicit configuration is just another configuration in which all the topology of the connections is clearly described for example component A needs to connect to connector 3 and component B needs to be connected to connector 1. In this case the planner's job is to find an optimal set of machines on which to deploy the set of connectors and components. In implicit configuration the exact topology cannot be stated. This happens when there is an attack on the system and it leaves the system in a state where neither an explicit target configuration is available nor the system can go back to the original configuration where it was before the attack. In this case PDDL provides a very powerful tool to specify an implicit goal state. This goal state could be just a property component that needs to be true after the planning for example component A must be connected. In this case we have not

specified where component A should connect. This is the job of the planner to find out a set of actions that lead the component A to a connected state. This feature is also helpful for an initial deployment of components when we do not want to specify and explicit goal configuration.

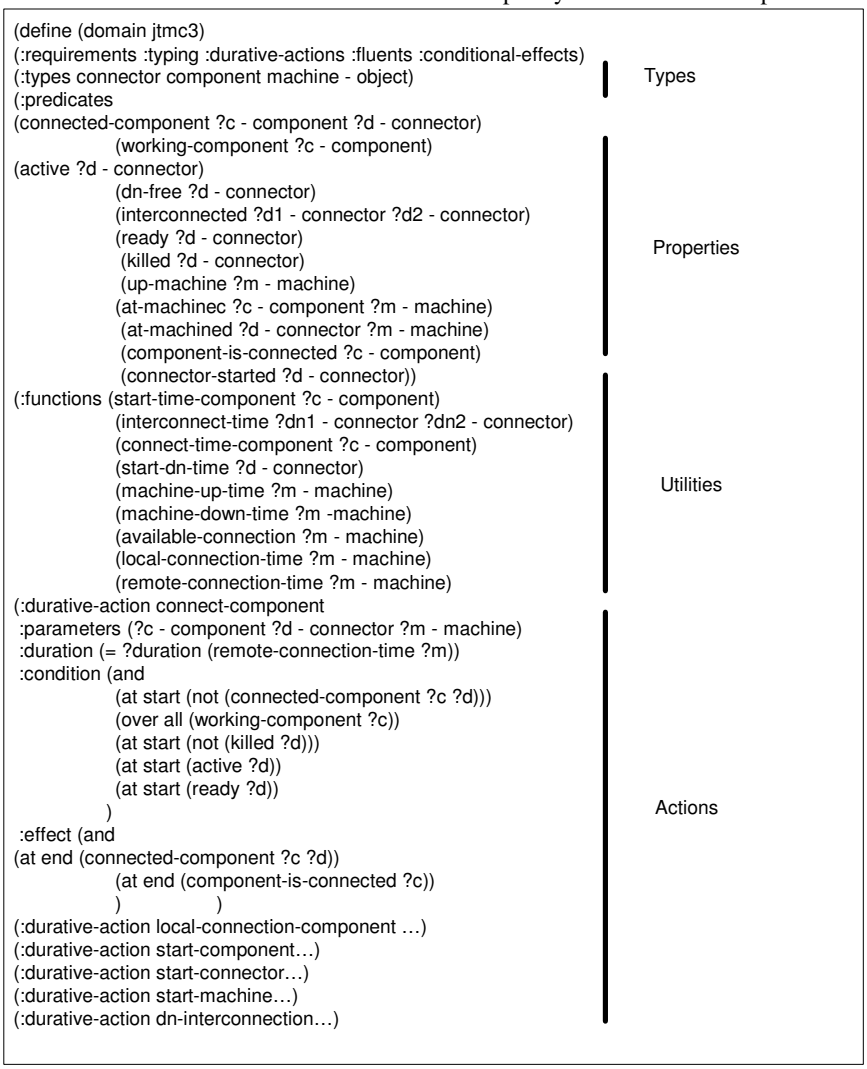

Fig. 2. Modeling of Dynamic Reconfiguration Domain in PDDL

Actions

An action is the main semantic of PDDL. An action takes parameters. These parameters consist of the entities that are involved somehow in the execution of this action. It may be possible that an actual entity may not be modified at all but its properties are necessary to modify the state of another entity. An action consists of precondition and postcondition. A precondition is essentially the state of the system under which an action can take place. The determination of the precondition is through the properties of the entities of the system. These properties are specified through predicates. Only an action can change the value of a predicate.

Dependency Modeling

The precondition and postcondition essentially determines the dependencies between the artifacts of the system. In component based system one needs to specify a number of types of dependencies [15]. However, all of these dependencies cannot be modeled in PDDL in a straightforward way. Therefore, while modeling dependencies in PDDL one has to be careful about dependencies that are only required in the reconfiguration planning. In our example domain these dependencies are the dependency of a component and connector can only be deployed on a machine if the machine has the available capacity. Another dependency is that a component cannot communicate unless it is connected with a connector.

Resource Modeling

As discussed before the reconfiguration process is constrained by the number of resources. However, in our domain we have only one resource and that is the capacity of the machine. This resource is modeled with a predicate associated with machine entity. However, other resources could also be modeled through predicates and associated with their specific types.

State Modeling

The planning specification language models the state of the system at each step in a plan. Therefore, it is necessary that the state of the system be modeled through some kind of mechanism. In this case the state is represented by the value of the predicates of the objects in the system. However, as PDDL is limited in its syntax representation, not all the variables in the state should be modeled. Only the predicates determine the state of the system from a reconfiguration point of view must be modeled.

8. Modeling Dynamic Reconfiguration Planning in AML

We will now try to model the same dynamic reconfiguration domain in AML. We will again model them in the five aspects we have described before in the previous section.

Initial and Goal Configurations

The basic data structure in AML is an activity. An activity can be thought of a strong form of an action in PDDL, however, they are both very different in terms of the features they provide for modeling. There is no direct way to specify an initial state or initial configuration in AML. However, a goal state can be defined by specifying a set of activities as mandatory goals. This is different from the goal state of PDDL where the predicates determine the goal state. The planner for AML has to satisfy all the decompositions and constraints with other activities in order to satisfy a mandatory goal. There could be also optional goals that can be specified. The optional goals are satisfied only when the mandatory goals are satisfied first. Then if there is time and resources available the optional goals are satisfied.

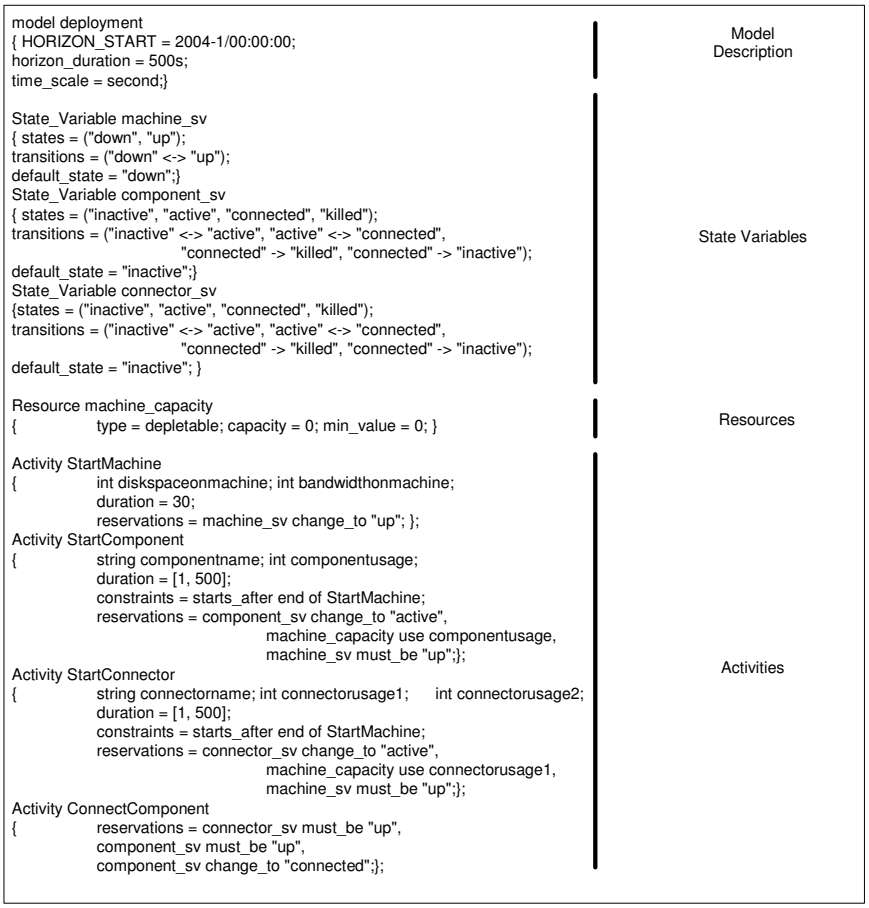

Fig. 3. Modeling of Dynamic Reconfiguration Domain in AML

Actions

The modeling of actions in AML is performed through activities. As described before an activity has parameters, decompositions, reservations and constraints. However, lets try to model our reconfiguration domain in terms of activities in AML. AML provides a very powerful way of representing a hierarchy amongst the entities to be planned. However, there is no apparent hierarchy in this domain. Therefore, decompositions are not very useful for us in modeling the domain.

The modeling of the relative ordering of activities in AML can be performed in two ways. The first way is to constrain the activity by providing the relative ordering of the activities that needs to be performed before the execution of the present activity. For example in figure 3 the activity StartComponent can not be started unless the activity StartMachine is performed. The second way is to have a state variable associated with the activity that flags the completion of an activity by changing the state of the artifact i.e. machine. For example in figure 3 we have a state variable called machine_sv that has two possible states. The default state is “down” and the StartMachine activity changes the state variable to ”up”. The StartComponent has a reservation that unless machine_sv is up it cannot be started. However, while performing experiments with actual ASPEN system, both of the above ways have shortcomings when it comes to actual planning.

The problem with the first way is to that we need to specify as many “start machine” operations in the domain file as the number of machine. This creates a really big domain file even if we have a small number of machines. On the other hand the StartComponent needs to be defined for the machine it is intended to deploy. This is in

essence equivalent to specifying an actual plan. The second way also has problems because there is only one state variable associated with each machine. In case we have more then one machine then we need to specify a state variable for each machine. Again this is really verbose and requires a really huge domain specification from the modeling perspective.

Resource Modeling

Unlike in PDDL, AML provides separate notion for specifying a resource. However, a problem with modeling resources in AML is the separate notion of states and resources. In PDDL we were able to associate state with a resource. The state changes as the actions were being performed on the resource. However, in AML only actions and resources can be combined or actions and states can be combined. For example in our domain a machine is a resource, however, it can have a state through the predicates and performing actions like StartMachine etc can change the state. This way of modeling is not possible in AML. Resources are a completely separate notion in AML and they cannot have a state. The lack of this flexibility reduces our ability to model a component based system in AML for planning.

State Modeling

The state modeling of AML is performed through state variables. The problem with state variables is that only one instance of a state variable can be initialized. Unlike in PDDL where each object that is initialized in the problem file has states associated with it through predicates. Therefore, in order to model the state a new state variable must be declared for each entity.

Dependency Modeling

The only type of direct dependency modeling performed in AML is parameter dependencies. However, parameter dependencies are not really useful in our domain because the dependencies here are either resource dependencies or action dependencies (that are discussed under Actions).

9. Conclusion and Future Work

The modeling of dynamic reconfiguration in PDDL and AML has given us mixed results. On one hand PDDL has modeled almost the entire state diagram in fig 1, AML has some basic shortcomings in modeling of our domain. However, it does not mean that AML does not have any features that we cannot use. Our domain was basically a flat domain where no sub activities are being performed but if we take this abstraction to a higher level i.e. systems of systems then AML is really useful for building a hierarchical domain. Moreover, AML is also good for scheduling as opposed to planning. Although, PDDL has modeled the dynamic reconfiguration domain really well, the building of such a robust domain for dynamic reconfiguration is really tough. A small mistake in PDDL domain specification can lead to many mistakes in the actual plan. Moreover, in PDDL there is no notion of a hierarchy. Although, there has been attempts to add a hierarchical notion in PDDL [1,4] it has not been the part of the official specification of PDDL.

We have seen in this discussion that PDDL is good for modeling the dynamic reconfiguration of component based systems. On the other hand AML is good for modeling the hierarchical notions and scheduling. Our future work will focus on integrating the strengths of these two modeling tools.

References

1. G. Armano, G. Cherchi, and E. Vargiu, "An Extension to PDDL for Hierarchical Planning". Workshop on PDDL, International Conference on Planning and Scheduling (ICAPS'03), Trento (Italy), June, 2003
2. B. Agnew, C. R. Hofmeister, J. Purtilo. Planning for change: A reconfiguration language for distributed systems. Distributed Systems Engineering, Sept. 1994, vol.1, (no.5):313-22.
3. N. Arshad, D. Heimbigner, A. Wolf, "Deployment and Dynamic Reconfiguration Planning for Distributed Software Systems". ICTAI 2003 pages 39-46.
4. A. Botea, M. Müller, and J. Schaeffer. "Extending PDDL for Hierarchical Planning and Topological Abstraction", Workshop on PDDL, International Conference on Planning and Scheduling (ICAPS'03), Trento (Italy), June, 2003
5. A. B. Brown and D. A. Patterson, "To Err Is Human," Proceedings of the First Workshop on Evaluating and Architecting System Dependability (EASY '01), Goeteborg, Sweden (July 2001).
6. M. R. Barbacci, C.B. Weinstock, D.L. Doubledry, M.J. Gardner, and R.W. Lichota. "Durra: a structure description language for developing distributed applications". IEEE Software Engineering Journal, pages 83-94, 1993.
7. S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D. Tran, "ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling," SpaceOps 2000, Toulouse, France, June 2000
8. X. Chen and M. Simons, "A Component Framework for Dynamic Reconfiguration of Distributed Systems", In Proceedings of the IFIP/ACM Working Conference on Component Deployment, pp 82-96, Springer-Verlag 2002.
9. S. Edelkamp and M. Helmert The Model Checking Integrated Planning System AI-Magazine (AIMAG), Fall, 2001, pages 67-71
10. M. Fox and D. Long. "The Third International Planning Competition: Temporal and Metric Planning". In Preprints of the Sixth International Conference on AI Planning and Scheduling, Toulouse, France, pp 115-118.
11. D. Garlan, R. Monroe, and D. Wile. "Acme: An Architecture Description Interchange Language" *Proceedings of CASCON 97*, Toronto, Ontario, November 1997, pp. 169-183.
12. D. Garlan, R. T. Monroe, D Wile "Acme: Architectural Description of Component-Based Systems" *Foundations of Component-Based Systems*, Gary T. Leavens and Murali Sitaraman (Eds), Cambridge University Press, 2000, pp. 47-68.
13. A. Gerevini, I. Serina, "LPG: a Planner based on Planning Graphs with Action Costs", in Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02), AAAI Press, pp. 13-22, 2002.
14. M. Hiltunen, "Configuration Management for Highly Customizable Services", Proceedings of the 4th International Conference on Configurable Distributed Systems" (ICDCS'98), Annapolis, Maryland, May, 1998.
15. A. Keller and G. Kar, "Dynamic Dependencies in Application Service Managemet," in 2000 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, 2000.
16. J. Kramer and J. Magee. "Dynamic Configuration for Distributed Systems," IEEE Transactions on Software Engineering, Vol. SE-11 No. 4, April 1985, pp. 424-436.
17. J Kramer, J. Magee, and A. Finkelstein. "A constructive approach to the design of distributed systems". In 10th Int. Conf on Distributed Computing Systems, May 1990.
18. J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. "Specifying Distributed Software Architectures". In Proceedings of the 5th European Software Engineering Conference, volume 989 of Lecture Notes in Computer Science, pages 137-153. Springer-Verlag, September 1995.
19. N. Mehta, N. Medvidovic and S. Phadke, Towards a Taxonomy of Software Connectors, Technical Report, Center for Software Engineering, University of Southern California,USC-CSE-99-529, 1999.
20. J. Purtilo, "The Polylith Software Bus," ACM TOPLAS, January 1994.
21. G. A. Papadopuolos, F. Arbab, "Dynamic Reconfiguration in Coordination Languages". HPCN Europe 2000: 197-206
22. European Network of Excellence in AI Planning (PLANET) Web Site (http://planet.dfki.de/). (Accessed on Jan 13th, 2004)
23. E. Rechtin and M. Maier, The Art of System Architecting, CRC Press, 1997
24. X. Wu, "A Component-Based Architecture for Building and Managing Global Information Systems", A position paper presented at International Workshop on Component-Based Software Engineering 1998.
25. A. L. Wolf, D. Heimbigner, J. Knight, P. Devanbu, M. Gertz, and A. Carzaniga. "Bend, Don't Break: Using Reconfiguration to Achieve Survivability". Third Information Survivability Workshop, October 2000.