Deployment and Dynamic Reconfiguration Planning For Distributed Software Systems

Naveed Arshad Dennis Heimbigner Alexander. L Wolf
Department of Computer Science
University of Colorado at Boulder
{arshad, dennis, alw} @ cs.colorado.edu

Abstract

Initial deployment and subsequent dynamic reconfiguration of a software system is difficult because of the interplay of many interdependent factors, including cost, time, application state, and system resources. As the size and complexity of software systems increases, procedures (manual or automated) that assume a static software architecture and environment are becoming untenable. We have developed a novel technique for carrying out the deployment and reconfiguration planning processes that leverages recent advances in the field of temporal planning. We describe a tool called Planit, which manages the deployment and reconfiguration of a software system utilizing a temporal planner. Given a model of the structure of a software system, the network upon which the system should be hosted, and a goal configuration, Planit will use the temporal planner to devise possible deployments of the system. Given information about changes in the state of the system, network and a revised goal, Planit will use the temporal planner to devise possible reconfigurations of the system. We present the results of a case study in which Planit is applied to a system consisting of various components that communicate across an application-level overlay network.

1. Introduction

Deployment and dynamic reconfiguration of software systems pose a tough challenge because of the architectural complexity of modern, distributed software systems. A significant body of research exists that addresses the optimization of the deployment and dynamic reconfiguration process. However, software systems continue to evolve in the direction of ever increasing complexity, and distributed systems are becoming the norm. In this environment, existing techniques are beginning to reach their limit. Many systems now require a system administrator to manage and evolve large scripts that control the system. Dynamic reconfiguration—the reconfiguration of a system while it is executing—only exacerbates the problem. This facility is required in large distributed systems where it may not be possible or economical to stop the entire system to allow modification to part of its hardware or software [14]. Clearly there is a need for new tools and techniques that automate the process of deployment and dynamic reconfiguration of software systems.

The dynamic reconfiguration process looks very much like the traditional control system model of "sense-plan-act". For software systems, sensing involves the monitoring of the system and its environment to detect problems such as machine or component failures. Planning involves the construction of a plan to return the software system to normal or near normal functionality. Acting is the execution of the steps as defined in the plan. Each step in the plan effects a state transition. The plan as a whole causes a transition from the present state of the system to a desired state. The problem is that there are number of ways in which this state transition can be performed. All of these ways have different time, cost, and resource usage implications. Finding the optimal plan is difficult when all of these variables are taken into account.

There has been a lot of work in the sensing and acting phases [3, 4, 6, 15, 16, 28] but much less work on the problem of finding the optimal techniques for planning a reconfiguration [3, 18]. The artificial intelligence (AI) community has been dealing with this kind of problem for a long time, where it arises in robot motion planning, intelligent manufacturing, and operations research. AI provide automated planners that avoid traditional state-space search mechanisms like breadth first, depth first, or best first. Instead they use heuristics and other techniques for searching the plan space. Recently these planners have become powerful enough to be used in real-world applications. Moreover, they now accept more powerful input specifications and are able to optimize time, cost, and resource constraints. There is a flavor of planner called *temporal planners* that is specifically geared towards time optimization.

In this paper we demonstrate the synergy between deployment, dynamic reconfiguration, and planning. Each planner requires a domain for the representation of the semantics of the possible transitions. Along with the domain, the planner requires a specification of the initial state and the goal state. We have developed an initial domain for the deployment and dynamic reconfiguration of software systems.

We have developed a tool, Planit, which can monitor a software system and obtain events indicating some kind of state change. Depending on the state of the system, Planit develops an initial state of the system. If it requires a change in configuration, it develops a desired state of the system based on its possible configuration rules. It writes the initial and desired state in a problem file and gives it to the planner. The planner computes a plan for the transition between the initial state and the desired state and returns a plan. Planit receives and interprets this plan and disseminates the new configuration to the system for execution.

Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)
1082-3409/03 $17.00 © 2003 IEEE

The technical part of the paper is organized as follows. Section 2 motivates the use of AI planning techniques for reconfiguration. Section 3 describes the domain of reconfiguration. Section 4 defines the reconfiguration process when using planning, and the operation of the underlying Planit system. Section 5 presents a case study and some performance measurements resulting from that study.

2. The Need for Planning

There are several reasons that motivate the use of planning for deployment and dynamic reconfiguration of software systems with the help of AI techniques.

The first reason is the dynamic nature of the environment for deployment and reconfiguration. Much of the prior work in reconfiguration (see Section 6) has implicitly assumed that the set of resources is fixed and available. For example, the set of machines is static and all of them are available; the set of deployed components is statically determined. In practice, this is seldom true, and so any process for deploying and reconfiguring a system must take such variability into account. Planning inherently has the ability to address these kinds of problems.

The second reason, which is driven in part by the first issue, is the large and complex nature of the search space that is involved in finding an optimal plan. Using traditional search-based techniques, such as depth-first search, can take a lot of time to find an optimal plan. Moreover, this is not search in the node space; this search is in the space of plans [25]. Therefore, without good heuristics the search process takes a long time to come up with an optimal plan.

The third reason is that in some situations an explicit contingency configuration is not given for a system. This happens when the system is trapped in an unanticipated state. The developers of the system may not have envisioned this unanticipated state, and the specific recovery plan for the system depends on that state. Many factors play a role in going into an unanticipated state. These factors include malicious external attacks, internal inconsistencies, failure of a critical resource, and many others. These factors at different times can lead to a partial failure of the system or can completely prevent the system from providing any kind of service. Bringing the system out from these unanticipated states is very difficult without any intelligence. Planners have the capability of finding the way out of unanticipated states provided the right set of inputs.

The fourth reason is to find a reconfiguration plan that is close to optimal in usage of time, cost, and resources. These factors sometimes conflict with each other, so the goal is to find the balance among them. Optimizing usage of resources is difficult without an intelligent technique. Here, time refers to the execution time of the plan. The time for finding the plan is also important and can affect the quality of the plan because the planner may produce a better plan when given more time. Coming up with an optimal plan that utilizes the resources, execution time and cost in the best possible way in the minimum time is not generally feasible with manual techniques.

These reasons are convincing enough to use more sophisticated techniques for automating the dynamic reconfiguration process. Finding the right plan for the reconfiguration process is the first step that we have taken in this direction. AI planners reduce the search space of finding the optimal plan by using different heuristics. The heuristic that is used by LPG [23], the planner that we use in this project, is called “Local Search on Planning Graphs”. Different planners [9, 23, 24] use other search heuristics to find an optimal plan.

3. The Domain of Reconfigurable Systems

In order to use planning technology to reconfigure software systems, we need to represent the structure and state of those systems using some kind of modeling language. This structural model is often referred to as the *architecture* of the system. The state model typically is represented by local predicates about individual elements of the architecture or global predicates about the architecture as a whole. These predicates allow us to specify both the current and desired states of the system.

We adopt a simplified version of the models used in Architecture Definition Languages (ADLs) [4, 5] 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. [21]. 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.

3.1 Component and Connector States

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.

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.

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.

4. Planning Activities

Our approach supports two related planning activities. First, planning is used for the initial deployment of the components and connectors on machines. Second, planning is used to support a form of replanning that occurs when a previously deployed system must be dynamically reconfigured due to a problem arising in the deployed system.

4.1 Planning Inputs

For both planning processes, the planner requires a number of inputs. These inputs are divided into three parts: the *domain*, the *initial* state, and the *goal* state.

The domain is relatively static. It specifies the following items.

1. Types of entities: in our case, this consists of components, connectors, and machines.
2. Entity Predicates/Facts: the predicates associated with entities (see the section marked “predicates” in Figure 3). 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.
3. Utilities: a variety of utility functions can be defined to simplify the specification (see the “functions” section of Figure 3). 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.
4. 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 3). 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 3) and post-conditions (“effects” in Figure 3). 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.

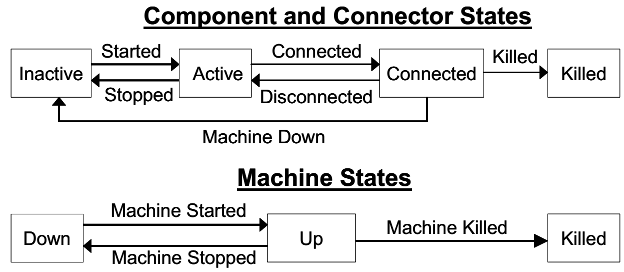

Figure 1. State Machine Diagrams for Components, Connectors, and Machines

The initial state represents the current state of the system (see the “init” section of Figure 4). 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 (see the “goal” section of Figure 4). It specifies predicates that represent constraints that must be satisfied in any plan constructed by the planner.

The last line of Figure 4 defines the *metric* that is to be used to evaluate the quality of a plan. In this case, the metric is minimal total execution time for the plan.

4.2 Explicit and Implicit Configurations

The initial predicates and the goal constraints are integral part of the configurations. They can be specified in two different ways: implicit and explicit configuration.

Implicit Configuration. The implicit configuration specifies a non-specific predicate about the system that needs to hold after the plan finishes. For example, it can be stated that component *A* must be connected, but without specifying exactly to what it is connected. This helps the system to specify partial information as a goal. In cases where the system does not have an explicit configuration of the system, it specifies the goal state in terms of the implicit configuration.

Explicit Configuration. In an explicit configuration the artifacts and their configurations are explicitly described as facts in the goal state. For example, it can be stated that component *A* is connected, and specifically that it is connected to connector *B*. Explicit configuration information can be specified in a number of ways, depending on the need the system. An explicit configuration typically requires the use of pairs of related predicates: *connected*-component and *component-is-connected* for example. The former predicate specifies that a specific component A is connected to a specific connector B and, hence, is an explicit configuration statement. The latter predicate specifies only that component A is connected to some (unspecified) connector. If *connected-component(A,B)* is true, then *component-is-connected(A)* must also be true.

4.3 Deployment Planning

Initial deployment is the process by which the system is deployed across the network for the very first time; the system is treated as having been not previously started anywhere in the network. We assume that all of the necessary files are accessible at every machine, so we are only concerned with the activation of the components and connectors on machines. Initial deployment takes a domain, an initial state, and a goal state as its inputs. The initial state in this case specifies the list of artifacts and facts about those artifacts, such as an indication of how much time it will take an artifact to start. The enumeration of artifacts contains a list of all the components, connectors, and machines for the system to be deployed. Each of these artifacts has its own time limitations, cost and resource constraints. For example, component *A* might have a start time of 11 seconds, a stop time of 5 seconds, and a connect time of 8 seconds.

The goal state specifies the normal operating state of the system in which all machines are up, all components and connectors are assigned to machines, and all components and connectors are connected. The goal state configuration can be given explicitly or implicitly. If no explicit configuration is given about a certain artifact, then Planit uses the implicit configuration by default.

4.4 Replanning

Once the system is deployed into a new configuration, problems may occur in the operation of the system: problems such as component or connector failure. In the event of a problem, the effect of that problem must be determined. For example, when a specific machine goes down, the effect is that all the components and connectors on that machine are killed. The analysis of the effects of a problem produces a new specification of the current state that reflects the fact that various components and connectors and machines are inactive or killed. Note that this analysis is carried out by Planit and not by the planner component.

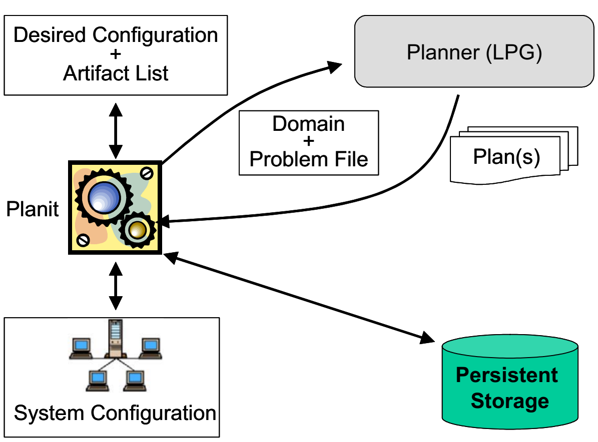

Figure 2. Planit Architecture

The next step is for Planit to construct a new goal state that indicates which failed components and connectors must be restarted and where. In some cases *contingency specifications* may exist for specific artifacts. A contingency specification indicates how and where to restart a specific component. Thus, there might be a specification that if component *A* is killed, then it should be restarted on machine *X*. If that is not possible, then restart it on machine *Y*. If that is not possible, then start it anywhere. For the artifacts with contingency configurations, the goal state includes an explicit configuration derived from the contingency specification. The artifacts that do not have any explicit configuration available are added to the goal state using an implicit configuration. If a component is still running, then all of the known facts about the component are listed in the new goal state explicitly.

At this point the planner is given the domain, the current state (as initial) and the new goal state. The planner is then charged with finding out the best possible plan to go from the initial state to the goal state. Note that we have effectively converted a replanning

```
(define (domain jtmc3)
(:requirements :typing :durative-actions :fluents :conditional-effects)
(:types connector component machine - object)                         } Types
(:predicates
        (connected-component ?c - component ?d - connector)
        (working-component ?c - component)
        (active ?d - connector)
        (dn-free ?d - connector)
        (interconnected ?d1 - connector ?d2 - connector)              Properties
        (ready ?d - connector)
        (killed ?d - connector)
        (up-machine ?m - machine)
        (at-machinec ?c - component ?m - machine)
        (at-machined ?d - connector ?m - machine)
        (component-is-connected ?c - component)
        (connector-started ?d - connector))

(:functions (start-time-component ?c - component)
        (interconnect-time ?dn1 - connector ?dn2 - connector)
        (connect-time-component ?c - component)                       Utility
        (start-dn-time ?d - connector)
        (machine-up-time ?m - machine)
        (machine-down-time ?m -machine)
        (available-connection ?m - machine)
        (local-connection-time ?m - machine)
        (remote-connection-time ?m - machine)
(:durative-action connect-component
 :parameters (?c - component ?d - connector ?m - machine)
 :duration (= ?duration (remote-connection-time ?m))
 :condition (and
        (at start (not (connected-component ?c ?d)))
        (over all (working-component ?c))
        (at start (not (killed ?d)))
        (at start (active ?d))                                        Actions
        (at start (ready ?d))
    )
 :effect (and
        (at end (connected-component ?c ?d))
        (at end (component-is-connected ?c))
        )       )
(:durative-action local-connection-component …)
(:durative-action start-component…)
(:durative-action start-connector…)
(:durative-action start-machine…)
(:durative-action dn-interconnection…)
```

Figure 3. Deployment and Reconfiguration Domain

Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)
1082-3409/03 $17.00 © 2003 IEEE

problem into a standard planning problem. This is possible because we have so much information about the current state of the system.

4.5 Planit Operation

Planit consists of three main packages. These packages serve different roles in its functionality. A high level figure of Planit and its interactions with the Planner and the system being configured is shown in figure 2. The heart of Planit is the Reconfiguration Manager. This Reconfiguration Manager serves as the mediator to the outside world. It is the interface between the running system and the Planner. It is assumed that sensors actively monitor the running system and report important events such as machine failures or component failures. These events are transmitted to the Reconfiguration Manager using the Siena publish/subscribe system [1].

Once an event is received by the Reconfiguration Manager, it delegates the event to a Problem Manager. The Problem Manager analyzes the event and evaluates the extent of damage to the system. The first part of the analysis determines the initial state part of the problem file. The second part of the analysis is responsible for checking the explicit and implicit configurations and determining the new goal state for repairing the damage. The Reconfiguration Manager then sends these new initial and goal states along with the domain file to the Planner. The Planner generates a series of increasingly better plans, where "better" is determined by some metric, such as number of steps or overall plan execution time. The Reconfiguration Manager checks for the best available plan that is generated within some time frame. The resulting plan is parsed, saved, and converted to a sequence of steps to be carried out on specific machines. This actual reconfiguration mechanism at each machine is not part of Planit.

Planit was developed on Sun OS 5.8 using an Ultra2/2200/512 with two 200Mhz CPUs and 512 MB RAM. The planner is LPG version 1.1 (Local search for Planning Graphs) [3]. Planit itself is implemented in Java 1.2. All the functionality has been tested for its compatibility with LPG. We believe Planit can be used with other temporal planners with few modifications.

5. A Case Study

In this section we demonstrate the operation of Planit using a specific domain and architecture for a system. We have selected a simple but concrete example to give an overview of all the tasks that Planit is able to perform. In this case study we demonstrate both the deployment process and the reconfiguration process. A subset of the domain developed for this case study is shown in Figure 3. The domain is written in PDDL [19] a widely used standard format for plan specification. The three artifact types used in this domain model are component, connector, and machine. Each type has its own independent operation and also operations that are dependent on other artifacts. For example, a component can be started, but it cannot be connected unless a connector is also started. For these dependencies each action has a set of preconditions and post-conditions. The action cannot be started unless the preconditions are met. The post-conditions can be assumed to be true after the action is performed. Sometimes there are preconditions that involve post-conditions of multiple artifacts.

Deployment. Figure 4 shows the initial state and goal state specifications for the initial deployment of our system. In the example, the goal state is an explicit configuration that asks the planner to go into this goal state and not any other state. The initial state plus the goal state are combined into a single problem file that is given to the planner along with a file containing the domain specification.

The planner, LPG in our case, constructs a plan as a sequence of steps, a subset of which is shown in figure 5. This figure only shows the deployment performed on machine2. The left-hand side shows the time for the execution of each action. In the middle, the name of the action and the artifacts involved are given. O the right-hand side, the time required for the execution of each action is given. It is also worth noting that there are actions that are carried out in parallel.

Dynamic Reconfiguration Process. If some part of the deployed system fails, then there is a need to carry out the dynamic reconfiguration process. Suppose there is a failure of machine2. We need to figure out what is the extent of the damage

```
(:init
        (= (start-time-component awacs0) 9.0)…
        (= (interconnect-time connector0 connector3) 6.0)…
        (= (interconnect-time connector3 connector0) 6.0)…
        (= (connect-time-component awacs0) 19.0)…
        (= (start-dn-time connector0) 30.0)…
        (= (machine-up-time machine0) 1.0)…
        (= (available-connection machine0) 15)…
        (= (local-connection-time machine0) 11.0)…
        (= (remote-connection-time machine0) 31.0)…
        )
(:goal
(and
        (connected-component awacs0 connector2)
        (connected-component groundRadar1 connector3)
        (connected-component satellite2 connector0)
        (connected-component positionFuselet3 connector1)
        (connected-component awacs4 connector2)
        (connected-component groundRadar5 connector3)
        (connected-component satellite6 connector0)
        (connected-component positionFuselet7 connector1)
)
)
(:metric minimize (total-time))
```

Figure 4. Problem File

```
0.001: (START-MACHINE MACHINE2)[1.000]
1.005: (START-CONNECTOR CONNECTOR2 MACHINE2)[30.000]
31.007:(START-COMPONENT AWACS4 MACHINE2)[5.000]
31.009:(DN-INTERCONNECTION CONNECTOR0
CONNECTOR2)[3.000]
31.010:(DN-INTERCONNECTION CONNECTOR2
CONNECTOR3)[3.000]
36.014:(START-COMPONENT AWACS0 MACHINE2)[9.000]
36.015:(LOCAL-CONNECTION-COMPONENT AWACS4
CONNECTOR2 MACHINE2)[11.000]
45.023:(LOCAL-CONNECTION-COMPONENT AWACS0
CONNECTOR2 MACHINE2)[11.000]
```

Figure 5. Plan for Explicit Configuration

(component-is-connected awacs0)
(component-is-connected awacs4)
(ready connector2)

Figure 6. Goal Description for Implicit Configuration

(START-COMPONENT AWACS4 MACHINE3)[5.000]
(START-COMPONENT AWACS0 MACHINE1)[9.000]
(LOCAL-CONNECTION-COMPONENT AWACS4 CONNECTOR3 MACHINE3)[11.000]
(START-CONNECTOR CONNECTOR2 MACHINE0)[30.000]
(LOCAL-CONNECTION-COMPONENT AWACS0 CONNECTOR1 MACHINE1)[11.000]

Figure 7. Plan for Implicit Configuration

to the whole system. This can be traced by checking the artifacts that are directly deployed on machine2 or connected to one of the artifacts on machine2. There are three artifacts, namely connector2, awacs0 and awacs2 deployed on this machine. These need to be restarted and reconnected. However, suppose the system does not know what to do when machine2 is down. In this case we use the implicit configuration to ask the planner to plan a way out of this state by providing another initial state and goal state. Figure 6 shows just the goal state specification. The rest of the facts in the problem file remain the same as described in the initial deployment process.

The resulting plan is shown in figure 7. The three artifacts are started and connected on other machines. The planner finds the best possible plan for the implicit goal state, while balancing the time, cost, and resource constraints.

Planit will take this plan and initiate the necessary steps to carry out the plan. This process repeats whenever there is a problem in the system. It asks Planit to develop a plan for going from a bad state to a good state. Planit checks the extent of damage and contingency configuration. It then develops the initial and goal state and develops a plan. Finally, it interprets and disseminates the information returned by the planner in the form of plans.

5.1 Experimental Results

In this section we give the results of the experiments that we have conducted using Planit and LPG. We have conducted experiments for the evaluation of two aspects of Planit: one for explicit reconfiguration and one for implicit reconfiguration.

Experimental Setup. Several system artifacts are used in our experiments. These artifacts can be broadly divided into component instances named AWACS, Ground Radar and Satellite. Connector and machine instances are also created. The experiments have been conducted on only the initial deployment of the system because, for the planner, this is the toughest task. We have performed five experiments. The experimental setup for these experiments is given in Table 1.

Table 1

Experiment No.	No. Components	No. Connectors	No. Machines
1	10	4	4
2	20	6	6
3	30	8	8
4	40	10	10
5	60	10	10

Results for Explicit Reconfiguration. The results in Table 2 show the plans that the planner was able to find given a 30-second period and a maximum of 5 plans. Time to find the best plan and the duration of the plan to go from the initial state to the goal state are also given. The execution time for the worst plan and the best plan are given for comparison.

One can see that in the case of explicit configuration, the planner has performed quite well. It is able to calculate at least one plan for all the experiments and in some cases the best and worst duration have significant differences among multiple plans.

Table 2

Experiment	No. of Plans Found in 30 Seconds	Time to Find Best Plan (seconds)	Duration of Best Plan (seconds)	Duration of Worst Plan (seconds)
1	5	12.39	67	83
2	4	18.64	66	137
3	3	27.95	100	144
4	2	23.00	76	84
5	1	17.93	138	N/A

Results for Implicit Reconfiguration. The results in Table 3 show the plans found using a 60-second window. Time to find the best plan and the duration of the best plan and the worst plan to go from the initial state to the goal state are also given.

The planner is able to calculate the results through Experiment 3. In the case of Experiments 4 and 5 the search space is so large that the planner is not able to calculate the plan in the specified time. We repeated Experiment 4, but with an unlimited time, and the planner was able to find a plan in 412 seconds as compared to the 60 seconds time limit for other experiments. We conclude that the increase in the number of artifacts can decrease the ability of the planner to find out the explicit reconfigurations in a small amount of time. However if one gives ample amount of time it will eventually finds a solution, provided a solution exists for the problem.

Table 3

Experiment	No of Plans Found in 60 seconds	Time to Find Best Plan (seconds)	Duration of Best Plan (seconds)	Duration of Worst Plan (seconds)
1	3	4.92	62	70
2	5	56.71	65	81
3	2	36.99	108	124
4	0	N/A	N/A	N/A
5	0	N/A	N/A	N/A

6. Related Work

The related work of this research can be seen from two perspectives. The first perspective is the techniques that are developed for solving the problem of deployment and of dynamic reconfiguration in software systems. The second perspective is research and use of planning to solve other real-world planning problems.

Surprisingly, and to the best of our knowledge, there has not been a direct use of AI planners in the solution of dynamic reconfiguration of software systems. Some authors have proposed planning for dynamic reconfiguration [3, 18]. However they do not use AI-style planners. The planning in these papers can be regarded as configuration scripts that trigger various parts of the script depending on the state of the system.

6.1 Deployment and Dynamic Reconfiguration

The deployment and dynamic reconfiguration problem has been the subject of much research. One of the very first research efforts in the area of dynamic reconfiguration of software systems was presented by Kramer and Magee [14]. They described several properties that a component requires in order to be reconfigured dynamically.

Agnew et al. [3] proposed a declarative approach. This approach makes the programmers responsible for writing the configuration changes in the form of a script. Another language, Gerel [18], was developed that takes a different perspective on the programming of reconfigurations. In this approach the language mechanism selects the configuration objects dynamically using their structural properties.

Research efforts have also applied workflow systems [11] to the dynamic reconfiguration problem, as well as agent-based approaches [12]. In these approaches, the components are divided into two categories: application components and management components. The interface between application components and management components provides appropriate methods to change the state of an application.

Another approach suggested, by Cook and Dage [13], uses multiple versions of the component running at the same time. They argue that in order to not break the present functionality of the system, multiple versions of the same components need to coexist together.

Research efforts have addressed the development of reconfiguration mechanisms on platforms like CORBA and J2EE [28, 20, 16]. Batista and Rodriguez [28] provide an approach that supports both program-based and *ad hoc* approaches to reconfiguration. Middleware has been used to provide dynamic configuration [16]. This approach uses the facilities of a flexible computing environment provided by object middleware such as CORBA, Java RMI, or DCOM. Dynamic reconfiguration approaches have been applied to the J2EE platform and Java-based software in general [16]. In these approaches the reconfiguration has been achieved by employing the power of Java to work across multiple platforms. AI Planning

The second perspective is the usage of AI planners and their usage in other fields. Planning can be viewed as a type of problem solving in which the agent uses beliefs about the actions and their consequences to search for a solution over the most abstract space of plans, rather than over a space of situations [25]. Planners have been developed to solve a range of problems in many different areas. There have been many research efforts that deal with temporal and resource planning [10, 17, 22, 24, 27]. These and other approaches attack the temporal planning problem through various ways. Some of these approaches include Graphplan (with extensions), model-checking techniques, hierarchical decomposition, heuristic strategies, and reasoning about temporal networks. These approaches are capable of planning with durative actions, temporally extended goals, temporal windows, and other features of time-critical planning domains [24].

The use of AI planning systems for solving real-world problems has significantly increased in recent years. The European Network of Excellence in AI Planning "PLANET" [8] identifies key areas where planning can be applied. These areas range from robot planning to intelligent manufacturing. PLANET has identified the various strengths and shortcomings of the AI planners. They have proposed areas of improvement for further research. Software deployment, however, does not appear to be one of their targets.

7. Future Work

There are many opportunities for future work.

Dynamic Architectures. Planit can be extended to accommodate the dynamic addition of new components. At this time Planit can only deal with a fixed initial set of components.

Configuration Scalability. To date we have not experimented with more then 120 artifacts. This number can be increased to show if the use of planners is viable for larger systems. Moreover, at this time we have only one domain file, both for deployment and for reconfiguration. Multiple domain files could be created that can capture the domain semantics in a better way and also reduce the search space of the planner.

Plan Execution. One important aspect of planning is immediate acting. This refers to the intertwining of the construction of the plan with the execution of the plan. At this time our planner does not provide such functionality. However, this facility is being added, and it will make the integration of planning and acting possible.

Plan Quality. There are very rudimentary measures for determining properties of a good state and what makes one state better than another state. There is a need for work in this area to determine better metrics that distinguish between a good state from a bad state or a very good state.

8. Conclusion

We have shown how to apply AI planning to the problem of deployment and dynamic configuration of software systems. Our approach supports both the initial deployment of a system as well as later reconfiguration to repair damage to that system. We have developed a system called Planit that manages the system being configured and incorporates a planner to support initial planning and replanning of the managed system.

9. References

1. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. ACM Transactions on Computer Systems, 19(3):332–383, August 2001.
2. 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.
3. 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.
4. D. Garlan, R. Monroe, and D. Wile. "Acme: An Architecture Description Interchange Language" P*roceedings of CASCON 97*, Toronto, Ontario, November 1997, pp. 169-183.
5. 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.
6. D.M. Heimbigner and A.L. Wolf. Post-Deployment Configuration Management. In Proceedings of the Sixth International Workshop on Software Configuration Management, number 1167 in Lecture Notes in Computer Science, pages 272–276. Springer-Verlag, 1996.
7. ECP-01 Planet Workshop on Automated Planning and Scheduling Technologies, 11 September 2001, Toledo, Spain (http://scom.hud.ac.uk/planet/ecp01_workshop/).
8. European Network of Excellence in AI Planning Web Site (http://planet.dfki.de/).
9. F. Bacchus and M. Ady, Planning with Resources and Concurrency: A Forward Chaining Approach, International Joint Conference on Artificial Intelligence (IJCAI-2001), pages 417-424, 2001.
10. J. Allen, 1991. "Planning as Temporal Reasoning". In Proc. Conf. on Knowledge Representation and Reasoning, 3-14.
11. J. Allen, and J. Koomen, 1983. Planning using a temporal world model. In Proc. 8th Intl Joint Conf. On Art. Intel., 741-747
12. J. Berghoff, O. Drobnik, A. Lingnau and C. Mönch. Agent-based configuration management of distributed applications. In Proceedings of the Third International Conference on Configurable Distributed Systems ICCDS '96, pages 52-59, Maryland, April 1996.
13. J. E. Cook and J. A. Dage. Highly Reliable Upgrading of Components. 21st International Conference on Software Engineering (ICSE99), Los Angeles, CA, May 1999
14. 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.
15. J. Magee and J. Kramer. Self Organizing Software Architectures. In Proceedings of the Second Inter-national Software Architecture Workshop, pages 35–38, October 1996.
16. J. Paulo, A. Almeida, M. Wegdam, L. Ferreira Pires and M. Sinderen. An approach to dynamic reconfiguration of distributed systems based on object-middleware. Proceedings of the 19th Brazilian Symposium on Computer Networks (SBRC 2001), Santa Catarina, Brazil, May 2001
17. J. Penberthy and D. Weld, 1994. Temporal planning with continuous change. In Proc. 12th National Conference. Artificial Intelligence.
18. M. Endler, J. Wei. Programming Generic Dynamic Reconfigurations for Distributed Applications, Proc. of the International Workshop on Configurable Distributed Systems London, pp. 68-79, IEE, March 92
19. M. Fox and D. Long. "The Third International Planning Competition: Temporal and Metric Planning". University of Durham, UK
20. M.J. Rutherford, K. Anderson, A. Carzaniga, D. Heimbigner, and A.L. Wolf, "Reconfiguration in the Enterprise JavaBean Component Model" In Proceedings of the IFIP/ACM Working Conference on Component Deployment, Berlin, 2002, pp. 67-81
21. 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.
22. N. Muscettola, 1994. HSTS: integrating planning and scheduling. "Intelligent Scheduling". Morgan Kaufmann
23. P. Doherty and J. Kvarnström, (2001). TALplanner: A Temporal Logic Based Planner. AI Magazine, Fall Issue, 2001
24. S. Edelkamp and M. Helmert The Model Checking Integrated Planning System AI-Magazine (AIMAG), Fall, 2001, pages 67-71
25. S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall 1995.
26. S. Shrivastava and S. Wheater, "Architectural Support for Dynamic Reconfiguration of Large Scale Distributed Applications" The 4th International Conference on Configurable Distributed Systems (CDS'98), Annapolis, Maryland, USA, May 4-6 1998
27. S. Vere. 1983. Planning in time: Windows and durations for activities and goals. IEEE Trans. Pattern Anal. And Machine Intel. 5:246-267
28. T. Batista and N. Rodriguez. Dynamic Reconfiguration of Component-Based Applications. In Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems, pages 32–39. IEEE Computer Society, June 2000.