F4Plan: An Approach to build Efficient Adaptation Plans*

Francoise André[2], Erwan Daubert[1], Grégory Nain[1], Brice Morin**[1], and Olivier Barais[2]

[1] INRIA, Centre Rennes - Bretagne Atlantique, Campus de Beaulieu, 35042 Rennes, France
{Erwan.Daubert | Gregory.Nain}@inria.fr — Brice.Morin@sintef.no
[2] University of Rennes1, IRISA, Campus de Beaulieu, 35042 Rennes, France
{Francoise.Andre | Olivier.Barais}@irisa.fr

Abstract. Today's society increasingly depends on software systems subject to varying environmental conditions imposing that they continuously adapt. A dynamic adaptation reconfigures a running system from a consistent state into another consistent state. To achieve this goal, a reconfiguration consists in executing a set of actions leading from source to target configuration. The planning of actions has often been neglected in adaptation mechanisms, leading to naive sequential schedules statically predefined. EnTiMid, a ubiquitous software system for assisted living, is one of these adapting systems using basic adaptation plan. This situation may cause problems when considering adaptations involving large set of actions and/or devices, particularly for distributed service-based applications. We propose a framework to ease the integration of different planning algorithms that produce more efficient adaptation plan than an ad-hoc algorithm.

1 Introduction

Large companies, banks, airports, buildings and even houses increasingly depend on software systems. These systems are continuously impacted by changes in their execution environment (infrastructural variation or a modification of the user requirements).

In this domain, EnTiMid is a home-automation software system to assist elderlies in their everyday life. Typically the system has to deal with dozens of devices, different needs per user or frequent changes in the ubiquitous environment. To address the combinatorial explosion of the number of potential configurations, engineers can develop such Dynamically Adaptive Systems (DASs) as Dynamic Software Product Lines (DSPLs) [5] by defining several variation points. Depending on the context, the system dynamically chooses suitable variants to realize those variation points. These variants may provide better quality of service, offer new services that did not make sense in the previous context, or discard some services that are no longer useful.

In previous work [9] using Aspect-Oriented Modeling (AOM) techniques, we can explicitly build a model of the configuration which is suitable for the current context, with no need for the designers to specify the whole set of configurations in extension. Using Model-Driven Engineering (MDE) techniques (model comparison) we were able to infer safe but sub-optimal migration plans to dynamically adapt the system. In particular, the simple heuristic we used tend to maximize the unavailability of services when some components have to be stopped and restarted to achieve a safe adaptation.

* The research leading to these results has received funding from the European Community's Seventh Framework Program FP7 under grant agreements 215412 (DiVA, http://www.ict-diva.eu/) and 215483 (S-Cube, http://www.s-cube-network.eu/).

** Now at SINTEF ICT, Oslo, Norway

In this paper we propose to improve adaptation systems by the use of planning algorithms to build more efficient reconfiguration plans than already existing ones. We illustrate this work on the EnTiMid system.

This paper is structured as follow. Section 2 starts with a presentation of EnTiMid and discusses our previous model-driven approach for designing and executing DAS. Then Section 3 details F4Plan, our proposal for designing and developing an efficient planning phase. Section 4 illustrated our approach on EnTiMid and demonstrates its benefits. Section 5 concludes this article and presents our future works.

2 Background

EnTiMid [10] is an ubiquitous software system built over an OSGi execution platform and developed in an assisted living context. The aim of this system, is to offer a level-sufficient abstraction of the devices in the home, making it possible for highlevel services to interact with physical devices (such as lights, heater or temperature sensors)

To address the problems of heterogeneity and dynamicity encountered in such an ubiquitous system, we proposed an Aspect-Oriented and Model-Based approach to tame the complexity of Dynamically Adaptive Systems (DAS) [8, 9]. The overall approach is illustrated in Figure 1.

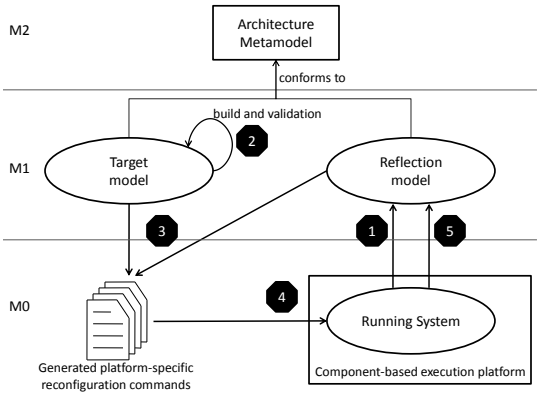

Fig. 1: Overview of our Approach to Dynamic Adaptation

The key idea is to keep an architectural model synchronized with the running system. This reflection model, which conforms to the architecture metamodel, is updated (step 1) when significant changes appears in the running system (addition/removal of components/bindings). It is important to note that the reflection model can only be modified according to runtime events. When a change has been made, a copy of this reflection model is used to build a target architectural model using model transformation or aspect model weaving. When a target model is derived, it is validated (step 2) using classic design-time validation techniques. This new model, if valid, represents the target configuration the running system should reach. Then, the adaptation plan to switch from the current to the target configuration is automatically generated. To do so, we first perform a model comparison between the source configuration (the reflection model) and the target configuration (step 3). This comparison produces an *ordered* set of adaptation actions. This safe sequence of actions is then submitted (step 4) to the running system in order to actually reconfigure it. Finally, the reflection model is automatically updated and becomes equivalent to the target model (step 5).

The sequence of atomic actions is ordered according to the following plan description: (1) components (that should be stopped) are stopped, according to the client/ server dependencies (clients are stopped before the servers), (2) bindings are removed, (3) components are removed, (4) attributes of already present components are updated, (5) components are added and their attributes are set, (6) bindings are added, (7) components are (re-)started, according to the client/server dependencies (servers are started before the clients).

In case of a large number of actions, this simple heuristic tends to maximize the unavailability of components, since the " Stop Component" actions are always executed at the beginning of the adaptation, and the "Start Component" actions are always executed at the end. To allow more efficient scheduling of actions we propose a new methodology based on general purpose planning algorithms. Our objective is to fit different possible needs concerning the planning phase. The next section concentrates on that proposal.

3 A generic approach for planning adaptation actions

3.1 Motivations

The MAPE[6] model defines four steps to do dynamic adaptation at runtime: the Monitoring, the Analysis, also called *Decision*, the Planning and the Execution. We define the semantic of these steps as follow. First, the Monitoring is used to detect changes inside the application or in its execution context (step 1 in Figure 1).When significant changes are detected, the Monitoring triggers the Analysis. This phase consists in deciding if adaptation is necessary to maintain the functionalities of the system. If adaptation is needed, the Decision also chooses the adaptation strategy that should be used (step 2). Once done, the Planning phase selects actions to execute and schedule them according to the chosen strategy (step 3). The execution of the selected actions is the last phase of the model (step 4). In this paper, we focus on the Planning phase.

As adaptation is performed at run-time, the time needed to actually perform the adaptation have to be minimized. Therefore, Planning is an important step of the MAPE model. It defines the actions necessary to properly apply the adaptation strategy, and orders the actions to ensure the consistency of the adaptation execution and minimize the time. Indeed some actions may be dependent of some others. For example, it is not possible to start component if its bindings are not already set and its attributes changed. At the opposite some actions can be independent, leading to a partial order between them (e.g.: two components can be started at the same time).

In our preliminary planning method described in the section 2, a static total order is defined on the different types of actions. This order can only lead to build a sequential schedule. Thus considering a distributed EnTiMid platform, whatever the number of components involved in an adaptation and their location, all the necessary "stop component" commands should be performed before to execute all the "remove binding" commands and so on.

Such an adaptation method consumes more time than needed because for example, the adaptation engine have to wait for all the "component stop" commands to be executed before launching the "component start" commands. Moreover, in a distributed and asynchronous system, synchronization operations between the different platforms should be added to enforce the sequentiality. Also, in this preliminary planning implementation it is not possible to add information or constraints on actions or on sequence of actions to give useful indications for the execution phase(e.g.: non-functional data as the execution time of an action or the among of resources used).

Research works on planning methods such as Artificial Intelligence planning, Motion planning or Control theory, have produced some algorithms that overcome these limita-

tions. In this paper, we propose an architecture for the planning phase to use, according to the needs, one of these algorithms in adaptation system for Service-Oriented Architecture. Most of times a planning phase using one of these algorithms is executed as follow.

It takes the strategy issued from the decision phase as input. This strategy consists of a *source configuration* and a *target configuration* (i.e. the current and the desired state of the system).

An *initial state* and a *goal state*, both given to the planning algorithm, are deduced from the strategy. These states are described in a language that depends on the planning algorithm used. The *domain* of actions, last input needed by the planning algorithm, represents all the possible actions.

In the following we describe our design proposition for the planning phase.

3.2 Our proposition: F4Plan (Framework for Planning)

As previously said, several planning algorithms exist, each one has its own characteristics. Therefore, to design an adaptation system, a planning algorithm has to be chosen among all existing ones according to the planning objectives. These objectives can be about minimizing the time spend in the planning phase and in that case choose a very simple planning algorithm, even if the resulting schedule is not the best one. In case there exists only one processor to execute the adaptation actions, it is not useful to select an algorithm that may exhibit some parallelism in the schedule. At the opposite it may be preferable to choose an algorithm that will spend some time to obtain the most efficient schedule if the adaptation actions are long and some of them may be executed simultaneously.

Each planning algorithm has its own dedicated language to express the initial/goal states and the domain of actions. In order to keep a coherent chain from the source and target configurations to the initial and goal states (i.e.: from the decision algorithm outputs to the planning algorithm inputs), a language translation is necessary.

We do not want to impose the choice of a specific decision algorithm nor a specific planning algorithm because this choice may depend on changing environmental constraints. For instance sometimes adaptation can concern only few elements, geographically closed, with the objective to quickly adapt. In that case, a simple planning/decision algorithm will be chosen. Sometimes the objective can be to perform proactive adaptation, involving a large set of distributed elements. The preference will then be for a more powerful planning/decision algorithm.

Thus, to ease the work of the final developer, we offer a set of translators from a description language to another. To face the combinatorial explosion of the number of translators, we propose to use a very common and powerful planning language, PDDL [4], as a pivot language.

At the end of the planning phase, the algorithm returns a schedule (a plan) that is used by the last phase of the MAPE model (the execution) to concretely realize the strategy.

An illustration of the benefits of using an efficient planning algorithm and a coherent chain of translation between decision and planning phases is given in section 4 to adapt the Ambient Assisted Living Application based on EnTiMid previously presented in 2).

4 Illustration

This section illustrates the use of F4Plan with a simple example into the EnTiMid system.

In the following, the changes operated on the EnTiMid platform located on the elderly person's house when the night comes will be depicted.

During the day the elderly person, in case of major problem, has to his/her disposal a remote control with a single button (the *SOS* component), that will send an emergency

message via SMS using the *SMS* component. In addition, to confirm the person that the SMS has been sent, a *Text-To-Speech* component emits a message. To connect these three components a *dispatcher1* component is involved to trigger in parallel the SMS sending and the vocal message emitting.

At night, when the person signals a problem, the house will be enlighten so that the person can realize what happened and get back its landmarks. As a consequence, lights must be manageable by the system in this configuration and N *light* components are added. These lights are bound with the remote control with the dispatcher1 already available into the platform. A component, here called *central lights command*, is also added to enable the elderly person to switch off the lights when everything get back to normal. A new dispatcher (called *dispatcher2*) is needed to connect lights with the central lights command.

The target configuration to reach is obtained using the mechanism described in section 2. In this scenario, a small subset of actions available to reconfigure the EnTiMid platform is considered. Only, **add component**, **add binding**, **start component** and **stop component** are used.

In this use case F4Plan uses a planning algorithm called GraphPlan [1]. This algorithm uses a PDDL subset called STRIPS [3] as input language.

Therefore, a translation between the decision output language, in our case the ART home made language and STRIPS is needed. Using our implementation, this translation is done with two translators automatically found, communicating through the PDDL pivot language. When the translation is done, the planning algorithm is executed.

At the end GraphPlan returns a partial ordered set of actions (Figure 2).

```
Step 1:
  Add Component "central lights command"
    // ... // Add Component "lightN"
    // Stop Component "dispatcher1"
Step 2:
  Add Binding "dispatcher1Tolight1"
    // ... // Add Binding "dispatcher2TolightN"
Step 3:
  Start Component "dispatcher1"
    // ... // Start Component "lightN"
```

Fig. 2: partial-order set of actions

To sum up, GraphPlan is well adapted for our Ambient Assisted Living use case, showing a substancial gain on the resulting plan reducing the number of steps (from 4N+6 to 3). Meanwhile, some other recent algorithms can provide the same kind of result with better performance.

The translation mechanism, associated with our proposition also implies a cost.

In all cases, the use of such not trivial planning algorithms has a cost. A tradeoff between this cost and benefit during the execution of plans needs to be done. Here we choose more complex planning algorithms to build more efficient plans and reduce their execution.

5 Conclusion

Nowadays, most software developments should consider the issue of their adaptation to the dynamism of the execution environments. However current solutions for adaptation are most often ad hoc and in consequence are not satisfying as long term solutions.

In this paper, we focus on the planning phase which has till now received little attention whereas it is an important phase especially in distributed environments. Most adap-

tation systems use a very simple ordering of actions even though many general purpose planning algorithms exist.

F4Plan allows the use of already developed planning algorithms. In that way, it is possible to take advantage of research works already done on planning methods such as Artificial Intelligence planning, Motion planning or Control theory. Such general purpose algorithms have already been applied in contexts close to our, in particular in applications for components deployment on computational Grids [7, 2]. Contrary to these approaches, our proposal does not impose a specific algorithm for the adaptation system and is self-adaptable, so that a choice between different planning algorithms, for example an algorithm looking for parallelism and a much simpler one, is always possible. So we are able to compute an efficient, coherent and valid plan to apply the decision strategy according to the constraints like time duration or resource consumption. The characteristics of the environment, in particular the distribution of all resources, either software or hardware, can also be taken in consideration to schedule and parallelize the actions.

F4Plan also provides an automatic way to translate the decision output into the planning input, offering several translators around a pivot language.

In our future work, we intend to study the distribution of the planning algorithms. This could be interesting to reduce the computation time of the schedule. Moreover in some cases, part of the schedule can be locally computed when it only involves local actions in a distributed environment.

Another subject of interest is the dynamic discovery of available adaptation actions. Indeed we currently use statically defined types of actions but in a large scale world of services, some actions can only be identified dynamically.

References

1. Avrim L. Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:1636–1642, 1995.
2. E. Deelman, G. Singh, M.H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
3. R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3/4):189–208, 1971.
4. Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
5. S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic Software Product Lines. *IEEE Computer*, 41(4), April 2008.
6. Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.
7. T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using AI planning techniques. In *Int'l. Parallel and Distributed Processing Symposium*, 2003.
8. Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg. Models@ Run.time to Support Dynamic Adaptation. *Computer*, 42(10):44–51, 2009.
9. Brice Morin, Olivier Barais, Grégory Nain, and Jean-Marc Jézéquel. Taming Dynamically Adaptive Systems with Models and Aspects. In *31st International Conference on Software Engineering (ICSE'09)*, Vancouver, Canada, May 2009.
10. Grégory Nain, Erwan Daubert, Olivier Barais, and Jean-Marc Jézéquel. Using MDE to Build a Schizofrenic Middleware for Home/Building Automation. In *In ServiceWave'08: Networked European Software & Services Initiative (NESSI) Conference*, Madrid, Spain, December 2008.