A Model-Driven Approach for Developing Self-Adaptive Pervasive Systems*

Carlos Cetina, Pau Giner, Joan Fons and Vicente Pelechano

Research Center on Software Production Methods
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{ccetina,pginer,jjfons,pele}@pros.upv.es

Abstract. Adaptive systems are generally difficult to implement, and their quality depends much on the designer experience or creativity. This paper presents a model driven approach to develop adaptive systems by means of run-time models. Our approach applies techniques from the Software Product Lines (SPLs) to address the different requirements of evolution and involution scenarios in Pervasive Systems. Finally, we show how models drive the adaptation in order to dynamically change the system architecture.

1 Introduction

Pervasive computing is defined as a technology that "weaves itself into the fabric of everyday life until it is indistinguishable from it" [1]. To be successful, the pervasive computing functioning should be transparent to the user. Such transparency is achievable only if the software frees users from having to repair and reconfigure the system when faults or changes occur in the environment.

Pervasive systems are highly dynamic and fault-prone since their components are liable to appear and disappear at any time. On the one hand, new kinds of entities (devices such as media players, light sensors or fire alarms) can be incorporated to the system. When a new resource is added to the system, the pervasive system should adapt itself to take advantage of the new capabilities introduced by this resource. On the other hand, existing entities may fail or be disconnected from the system for a variety of reasons: hardware faults, OS errors, software bugs, network faults, etc. When some resource is removed, the system should adapt itself in order to offer its services in an alternative way to reduce the impact of the resource loss.

In a previous work [2], a methodology based on SPLs principles was defined to cope with adaptivity of Pervasive Systems. This approach is based on the reuse of the knowledge from the design of SPLs to support adaptivity in the resulting systems. By means of model transformations, the SPL knowledge is systematically reused at run-time.

* This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and cofinanced by FEDER.

The present work is focused on providing a model-based support for some adaptation scenarios very common in Pervasive Systems (evolution and involution scenarios). These scenarios have different requirements regarding adaptation, and the way in which models are handled at run-time should consider those particular requirements.

The contribution of this work is twofold. On the one hand, a model-based approach is introduced in order to organize the knowledge required for adaptation according to the specific demands (support for involution and evolution scenarios) of pervasive systems. In this way, adaptation knowledge is separated from the structure of the system and different adaptation mechanisms can be offered depending on how much critical the adaptation is. On the other hand, we present how models drive the system adaptation within the context of each scenario.

The remainder of the paper is structured as follows. Section 2 gives an overview of the different models used to structure the knowledge about the system. Section 3 defines the architecture proposed for the kind of system this work is dealing with. Section 4 defines for different scenarios how to achieve the adaptation of a system that follows the introduced architecture using the presented models. Section 5 discusses related work. Finally, Section 6 presents some conclusions to the paper.

2 The Models for System Adaptation

The present work considers system adaptation as a reaction to a change in system resources. Therefore, two different kind of scenarios are considered: *evolution scenarios* (a resource is added) and *involution scenarios* (a resource is removed).

Evolution and involution scenarios have different requirements regarding adaptation. On the one hand, in involution scenarios time becomes critical since these scenarios are normally related to failure-recovery. For example, if an alarm system fails in a smart home, an alternative system (e.g., house lights blinking) should be used immediately as a backup. On the other hand, evolution scenarios are not so demanding in this aspect, and even the opinion of the final users can be considered (interacting with the users or considering their preferences) to provide a better adaptation according to user needs.

In order to enable system adaptation, it is required a knowledge about (1) the current state of the system and (2) the possible ways of changing it. In the present work, models are used for both purposes, being queried at run-time to perform the adaptation. On the one hand, models are used for capturing the state of system components and the communication channels among them. On the other hand, the space of possible system changes is captured by means of feature modeling techniques.

The consequences of a change in the system (e.g., enabling the security feature) can be obtained by reasoning over a model that captures all the possible system features and their dependencies. This is acceptable for evolution scenarios where the system is being upgraded. However, since involution scenarios require an immediate adaptation, information required for the system reaction is precalculated in models. In this way, the effort of reasoning over the different possibilities of adaptation is avoided. More detail about the different models involved in the proposal is provided below.

- **Feature model.** Feature Modeling is a technique to specify the variants of a system in terms of features [3]. In feature models, features are hierarchically linked in a tree-like structure and are optionally connected by cross-tree constraints. This model describes the possible system functionality and its dependencies in a coarse-grained manner. The impact of activating a feature is captured in this model. For example, if a security system is installed, other features depending on a security system such as presence simulation can be activated.
- **Realization model.** Realization model defines relationships between features and components of the architecture. Atlas Model Weaving has been extended [4] to define the *default* and *alternative* relationships. In this way, some components of the architecture are labeled as default or alternative components for supporting a certain feature. In case of failure of a component, this model permits to quickly find an alternative to replace it.
- **Component model.** This model represents the different components that conform the system. Component state is captured in the model. This model is in synchronization with the system since components make use of this model to store and retrieve their state.
- **Structural model.** This model represents the communication channels established between the different components of the system. Since this work deals with highly dynamic systems, the connection among components change quite often. This model reflects both, the possible connections and the ones that are in use in the current state of the system.

The introduced models have been structured in this way in order to decouple system adaptation (Feature and Realization models) from system description (Component and Structure models). In addition, precalculated information to better support involution scenarios is isolated in a model (Realization model). Next sections describe the use of these models to achieve system adaptation.

3 The Model-Based Adaptation Approach

To perform adaptation, our approach is based upon a framework for adaptive systems proposed in [5] by analyzing common terminology and synergy between different approaches. This framework introduces the roles of (1) triggers which specify the event or condition that causes the need of adaptation; (2) adaptation actions which realize the actual adaptation; and (3) adaptation rules that define which triggers cause which adaptation actions. In our approach, these rules are driven by run-time models to modify the system architecture using adaptation actions.

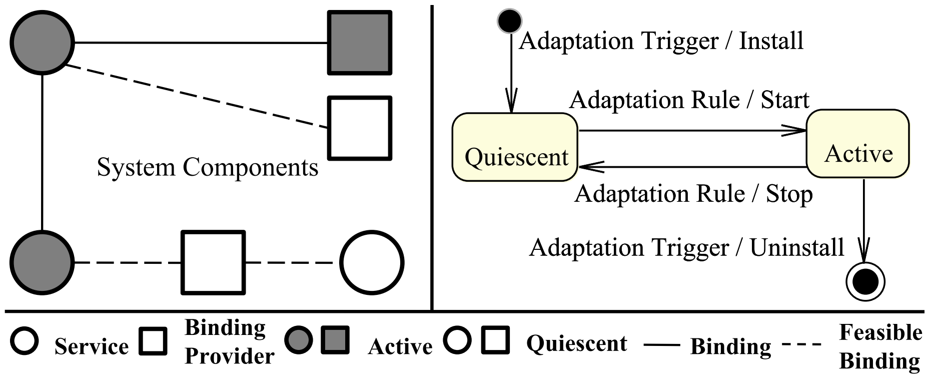

Fig. 1. The adaptation architecture.

3.1 The System Architecture

In order to allow a flexible adaptation process, we have considered an architecture based on communication channels (called bindings). This architecture for the final system allows an easy reconfiguration process since communication channels can be established dynamically between the components that form the system (see left of Fig. 1). These components are classified in Service and Binding Providers as follows:

– **Service.** A *Service* coordinates the interaction between resources to accomplish specific tasks (these resources can be hardware or software systems);
– **Binding Provider.** A *Binding provider* (BP) is a resource adapter that handles the issues of dealing with heterogeneous technologies. The BP provides a level of indirection between Services and resources. Resource operations interact with the environment (sensors and actuators) and provide functionality from external software systems. Services coordinate these resource operations to offer high-level functionality. If the resource operations do not match the Service expectations, then a BP is used to adapt these operations. Hence, the BPs decouple Services from resource operations.

For example, in a smart home a security service is composed of several resources such as presence sensors, movement detectors, sirens, contact detectors, SMS senders, silent alarms and so on. The security service coordinates the behaviour of all these resources.

3.2 Adaptation Actions

The system architecture has to be modified as a result of the dynamic adaptation. Old components must be dynamically replaced by new components while the system is executing. The adaptation actions are in charge of this dynamic reconfiguration. These actions deal directly with the system components by means of the following operations: Component State-Shift and Component Binding.

1. **Component State-Shift** Kramer and Magee [6, 7] described how in an adaptive system, a component needs to transit from an active (operational state) to a quiescent (idle) state in order to perform the system adaptation. We have applied this approach to our systems by means of the OSGI framework [8]. The OSGI Framework defines a component life cycle where components can be dynamically installed, started, stopped, and uninstalled (see right of Fig. 1). On the one hand, *Triggers* are in charge of perform the install/uninstall operations. For example, when a resource fails or a new resource is installed in the system. On the other hand, *Adaptation Rules* are in charge of perform the start/stop operations. For example, when a Binding Provider must be activated to handle a new resource.
2. **Component Binding** Once a component transits to an active state, it needs bindings with other components. These bindings are implemented by using the OSGI Wire Class (an OSGI Wire is an implementation of the publish-subscribe pattern oriented to dynamic systems). The OSGI Wires establish communication channels between components to send messages one another.

Adaptation actions provide the basics operations to dynamically change the system architecture. Adaptation rules orchestrate the execution of these actions by means of the run-time models. The next section details how the adaptations rules queries the models in order to apply the adaptation actions.

4 Adaptation Rules

In a nutshell, an adaptation rule is in charge of (1) handling the adaptation triggers, (2) gathering the necessary knowledge from the run-time models and (3) applying the adaptation actions.

As we state above, evolution and involution scenarios present different requirements. In involution scenarios the system must provide an autonomic response in a reduced amount of time. While in evolution scenarios, the system does not present the same time requirement and even the user might assist the adaptation. To fulfil theses requirements, we have defined two kinds of adaptation rules taking into account the type of scenario.

4.1 Adaptation in Evolution Scenario

When a component is plugged-in, first the adaptation rule queries the feature model for which new features could potentially be activated. Then the user confirms the features activation. Furthermore, activating new features can fulfil other feature constraints which might be enabled. Therefore, each time the user confirms a feature activation, the adaptation rule queries the feature model for new features. Finally, the Component and Structure Models drive the adaptation actions in order to dynamically reconfigure the system architecture and support the new features. The steps to perform this adaptation (see Fig. 2) are detailed next:

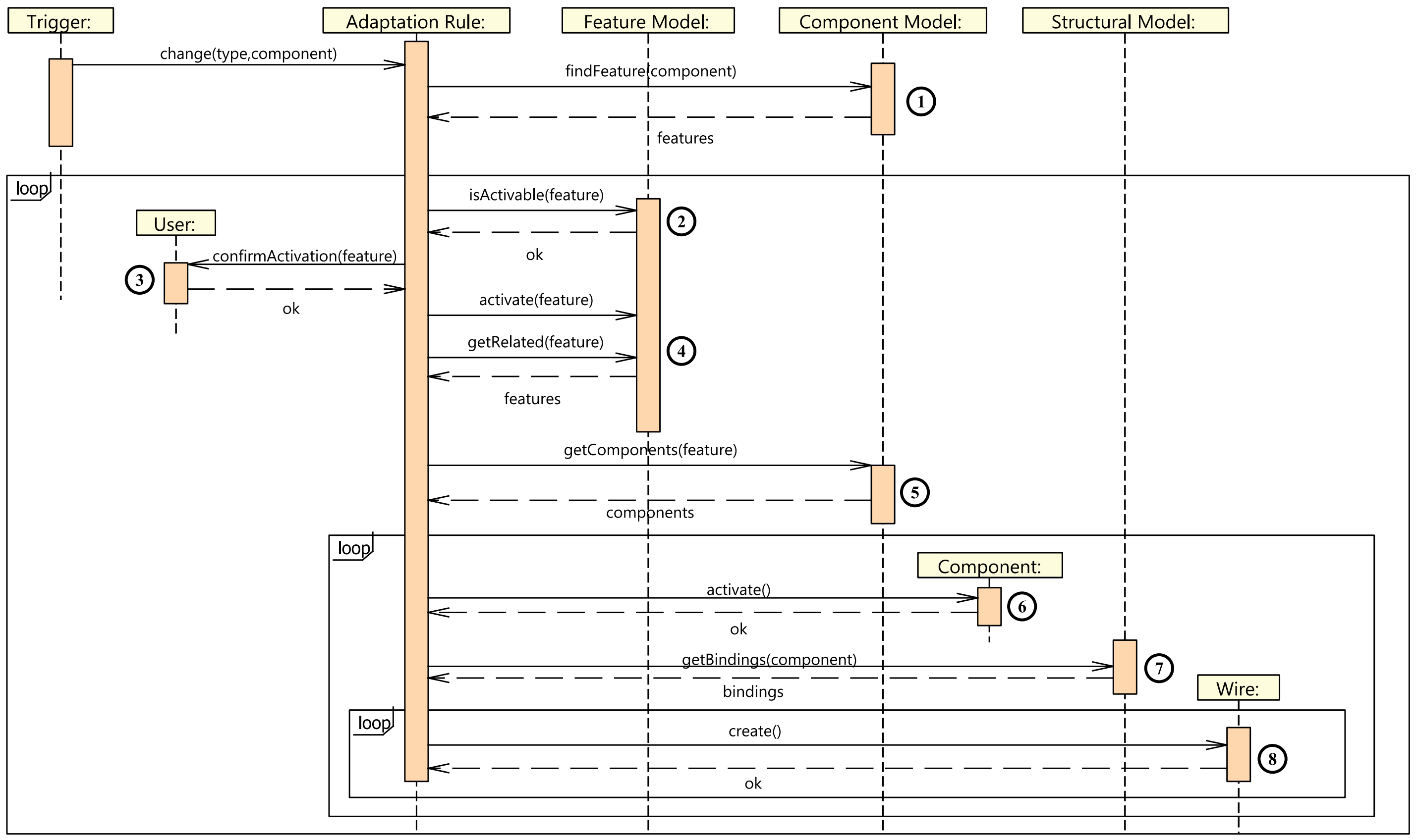

Fig. 2. Adaptation process for evolution scenarios.

1. By means of the Component model, the adaptation rule identifies those features which are related to the trigger component. With these features, the rule creates an ordered set called the *evolution set*. For each one of the features, the rule performs the following steps, 2 to 5.
2. The rule checks the possibility of feature activation. This information is in the Feature Model, specifically it depends on the requires, excludes and mandatory relationships between features. If all these constraints are fulfilled, then the feature can be activated.
3. Once the rule checks the feature activation, it asks the user for confirmation by means of a dialog in the user interface. The message shows the name of the feature and a description stored in he Feature Model. The message also provides three options to the user: "Yes", "Remind me later" and "No".
4. Activating a new feature can fulfil other feature constraints. In this step, the rule checks for new activable features. The rule adds these new features to the *evolution set*.
5. In terms of the platform, activating a feature implies performing *adaptation actions* to system components. In this step, the rule queries the Component model for the feature components. For each one of these components, the rule performs the following steps, 6 and 7.
6. The rule applies the *State-Shift* action to the component. Therefore, the component transits from a quiescent state to an active state.
7. To connect the new active component with the rest of the system, the rule queries the Structural model for the component bindings.
8. Finally, the rule applies the *Binding* action to create the communication channels between the components.

Due to space constraints, the sequence diagrams in this section represent only the general case for adaptation. Diagrams consider only affirmative responses, lacking alternative behaviour.

In our experience applying this approach to the smart home domain [2], we have notice that the time response delay comes mainly from these factors: feature dependency resolution (steps 2 and 4) and user confirmation (step 3). How much time the user takes to confirm cannot be foreseen, and dependency resolution is more time consuming than other simpler queries (for example, step 7). However, we consider that installing new resources in the system is not as critical as handling resource failures. Thus, in evolution scenarios we offer an advance system response (dependency resolution and user participation) although this response takes extra time.

4.2 Adaptation in Involution Scenario

Involution scenarios are triggered by the removal of a resource. A fast adaptation of the system is required to minimize the impact of the lost resource. In order to offer a good response time, adaptation is automatic (not requiring user intervention) and resource alternatives are precalculated in a model (the realization model). In this way, the latency of asking the user is avoided and the effort of reasoning with the feature model (e.g., looking for dependencies) is also reduced.

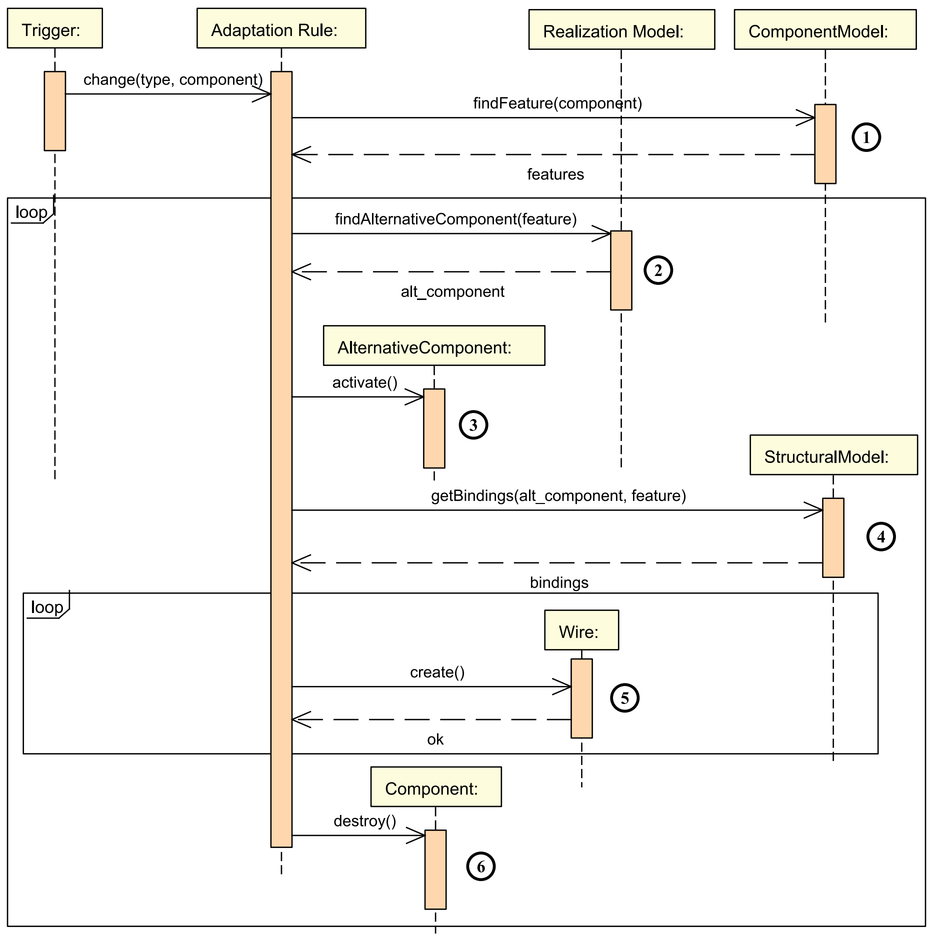

Fig. 3. Adaptation process for involution scenarios.

In Fig. 3, the adaptation process for an involution scenario is illustrated. Given the removal of a component, the affected feature is obtained and an alternative component for this feature can be directly retrieved from the Realization Model. More detail about the process is given below:

1. When a change is produced in the system, the affected features are obtained in the same way as in the evolution scenario. The following steps are performed for each feature.
2. The rule queries the Realization model to obtain a component that can replace the affected one for a given feature. Since this information is expressed explicitly in this model, queries are straightforward.
3. Once the rule has found an alternative component (initially in the quiescent state) it is activated.

4. The alternative component may require communication with other components. This information is obtained from the Structural model.
5. For each of the required bindings, a wire is created to establish the necessary communication channel between components.
6. Finally, the affected component is destroyed. This implies the removal of inactive wires. The destruction of this component is deferred until the end of the adaptation process, since the priority in involution scenarios is to offer the new services immediately.

The adaptation rule for involution reduces the delays commented for the evolution scenarios. On the one hand, model queries are simplified. Reasoning over a feature model is a time-consuming activity and termination becomes difficult to guarantee [9]. On the other hand, the user does not participate in the process, which is a requirement for the autonomic behavior required by this kind of scenarios.

5 Related Work

Hallsteinsen et al present the Madam approach [10] for adaptive systems. This approach builds systems as component based systems families with the variability modeled explicitly as part of the family architecture. Madam uses property annotations on components to describe their Quality of Service. For example a Video Streaming component may have properties such as start up time, jitter and frame drop. At run-time, the adaptation is performed using these properties and a utility function for selecting the component that best fits the current context.

Trinidad et al [11] present a process to automatically build a component model from a feature model based on the assumption that a feature can be modeled as a component. By means of augmenting the system with a feature model and a model reasoner, this approach enables systems to dynamically changing its features at run-time.

Both Hallsteinsen and Trinidad apply SPL techniques to develop adaptive systems. However, their approaches do not take into account the differences between evolution and involution scenarios. Therefore, they do not exploit the specific scenario requirements.

Zhang et al. [12] introduce a method for constructing and verifying adaptation models using Petri nets. They address directly the verifications of the adaptation models by means of visual inspection and automated analysis. On the other hand, our approach is focused on reuse at run-time the variability modeling of SPLs. However, our approach can benefit from SPL reasoners in order to check system properties [9]. Finally, Zhang's approach separates the adaptation specification and non adaptation specifications as our approach does. However, our approach introduce precalculated adaptations in order to achieve a faster response in involution scenarios.

6 Conclusions

In this paper, we provide support for adaptation in pervasive systems by means of run-time models. Our approach focusses on addressing the differences between evolution (a resource is added) and involution (a resource is removed) scenarios. In involution scenarios, we use models with precalculated knowledge in order to provide an autonomic response in a reduced amount of time. While in evolution scenarios, we offer an advanced system response (feature dependency resolution and user participation) because we consider that installing new resources in the system is not as critical as handling resource failures. Finally, we showed how models drive the system adaptation within the context of each scenario.

References

1. Weiser, M.: The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev. (3) (1999) 94–104
2. Cetina, C., Fons, J., Pelechano, V.: Applying Software Product Lines to Build Autonomic Pervasive Systems. 12th International Software Product Line Conference, SPLC 2008. (2008)
3. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. In: Proceedings of the Third Software Product Line Conference 2004, Springer, LNCS 3154 (2004) 266–282
4. Fabro, M.D.D., Bzivin, J., Valduriez, P.: Weaving models with the eclipse amw plugin. In: Eclipse Modeling Symposium. (2006)
5. Bencomo, N., Blair, G., France, R.: Model-driven software adaptation report on the workshop m-adapt at ecoop 2007. Object-Oriented Technology. ECOOP 2007 Workshop Reader (2008) 132–141 Springer, LNCS.
6. Kramer, J., Magee, J.: The evolving philosophers problem: dynamic change management. Software Engineering, IEEE Transactions on Software Engineering (1990) 1293–1306
7. Kramer, J., Magee, J.: Analysing dynamic change in software architectures: a case study. Configurable Distributed Systems, 1998. Proceedings. Fourth International Conference on Configurable Distributed Architecture (1998) 91–100
8. Marples, D., Kriens, P.: The open services gateway initiative: an introductory overview. Communications Magazine, IEEE (12) (Dec 2001) 110–114
9. Benavides, D., Segura, S., Trinidad, P., Ruiz-Corts, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems. (2007)
10. Hallsteinsen, S., Stav, E., Solberg, A., Floch, J.: Using product line techniques to build adaptive systems. Software Product Line Conference, 2006 10th International (Aug. 2006) 21–24
11. Trinidad, P., , Ruiz-Cortés, A., na, J.P.: Mapping feature models onto component models to build dynamic software product lines. International Workshop on Dynamic Software Product Line (2007)
12. Zhang, J., Cheng, B.: Model-based development of dynamically adaptive software. In: ICSE '06: Proceedings of the 28th international conference on Software engineering, New York, NY, USA, ACM (2006) 371–380