Taming Dynamically Adaptive Systems with Models and Aspects

Brice Morin, Olivier Barais, Grégory Nain, Jean-Marc Jézéquel

Taming Dynamically Adaptive Systems Using Models and Aspects*

Brice Morin[1], Olivier Barais[2], Grégory Nain[1] and Jean-Marc Jézéquel[1,2]
[1]INRIA, Centre Rennes - Bretagne Atlantique
[2]IRISA, Université de Rennes1
Campus de Beaulieu
35042 Rennes Cedex - FRANCE
{Brice.Morin | Gregory.Nain}@inria.fr
{Olivier.Barais | Jean-Marc.Jezequel}@irisa.fr

Abstract

Since software systems need to be continuously available under varying conditions, their ability to evolve at runtime is increasingly seen as one key issue. Modern programming frameworks already provide support for dynamic adaptations. However the high-variability of features in Dynamic Adaptive Systems (DAS) introduces an explosion of possible runtime system configurations (often called modes) and mode transitions. Designing these configurations and their transitions is tedious and error-prone, making the system feature evolution difficult. While Aspect-Oriented Modeling (AOM) was introduced to improve the modularity of software, this paper presents how an AOM approach can be used to tame the combinatorial explosion of DAS modes. Using AOM techniques, we derive a wide range of modes by weaving aspects into an explicit model reflecting the runtime system. We use these generated modes to automatically adapt the system. We validate our approach on an adaptive middleware for home-automation currently deployed in Rennes metropolis.

1 Introduction

Society's increasing dependence on software-intensive systems is driving the need for dependable, robust, continuously available adaptive systems. Such systems often propose many variability dimensions with many possible variants, leading to a wide number of possible configurations that is difficult to integrally check at design-time because of time and resource constraints. For example, associations and public institutions of the metropolis of Rennes are working together on a project which aims at allowing dependent people to stay at home as long as possible. Due to the large scale of the project, and the diversity of disabilities that have to be considered, the deployment context will be different for each equipped house. Furthermore each deployment context is going to continuously evolve along with the evolution of the person's disabilities.

This ability to evolve a system at runtime is one critical aspect of achieving continuously availability. Many popular programming frameworks such as OSGI [33] or Fractal [7] now provide support for dynamic adaptation through extension mechanism such as plugins or variability mechanism through introspection and reconfiguration API. However the high variability of crosscutting and non-crosscutting features in adaptive systems introduces an explosion of possible runtime system configurations (often called modes). When new features are introduced at deployment time (vs. design time), we also have to make sure that they do not lead the system into unwanted modes. Besides, due to the fact that features are often partially independent, the (implicit) state-machine representing the path between modes is highly connected, leading to a nearly quadratic explosion of transitions between modes [5, 35]. The inefficacy of the variability and extension mechanisms to tame the high-number of modes transitions might lead to several undesirable consequences related to Dynamically Adaptive System maintainability, including partial duplication of reconfiguration scripts or the non-cover of all the modes transition, etc.

Aspect-Oriented Modeling (AOM) was initially introduced to improve the modularity of software [11, 21], complementary to Model Driven Engineering (MDE)

*This work was partially funded by the DiVA FP7 European project (STREP) (See http://www.ict-diva.eu/) and the S-Cube European Network Of Excellence on Software Services and Systems (See http://www.s-cube-network.eu/).

to link models to the real world [14]. In [24], we have proposed a first approach that leverages AOM and MDE to manage variability at runtime. It relies on the notion of aspect models, that can be woven into an explicit model of the runtime configuration seating on top of the running system. Actual mode switches between runtime configurations are then triggered by reconfiguration scripts automatically generated based on the differences between the initial model and the newly woven one. The result of this approach is that modes becomes somehow implicit from the point of view of the system designer, and new modes can appear when new aspects are introduced during the life of the system. It is thus no longer possible to statically validate every accessible mode and each mode transition.

In this paper, we show how aspects can help designers determine interactions between dynamic variants and how runtime models can be used to validate new configurations on the fly, before committing them on the running system (making it easy to roll-back when a configuration is not valid). Indeed, once the system has been deployed, new variability dimensions and variants that have not been foreseen may appear while the system is running and cannot be stopped. In this case, it is very useful to validate configurations on the fly before actually adapting the running system.

The reminder of this paper is organized as follows. Section 2 describes the general process for managing dynamic variability. Section 3 presents how we leverage AOM and MDE techniques and tools to determine interactions and validate configurations on the fly. Section 4 outlines how our approach was validated in a home-automation context deployed in the metropolis of Rennes. Section 5 discusses related work and section 6 concludes by presenting a set of open research problems based on our experience.

2 Process Overview

This Section presents our approach for managing variability at runtime [24]. The overall process is described in Figure 1.

2.1 Maintaining a High-Level Representation of the Running System

Maintaining a model at runtime [4] representing the running system (Figure 1, step 1) allows us to reason on the model and manipulate the model independently from the running system. Using high-level abstractions, we can discard all the platform-specific runtime information we do not need and reason more easily and more efficiently in the next steps of the process.

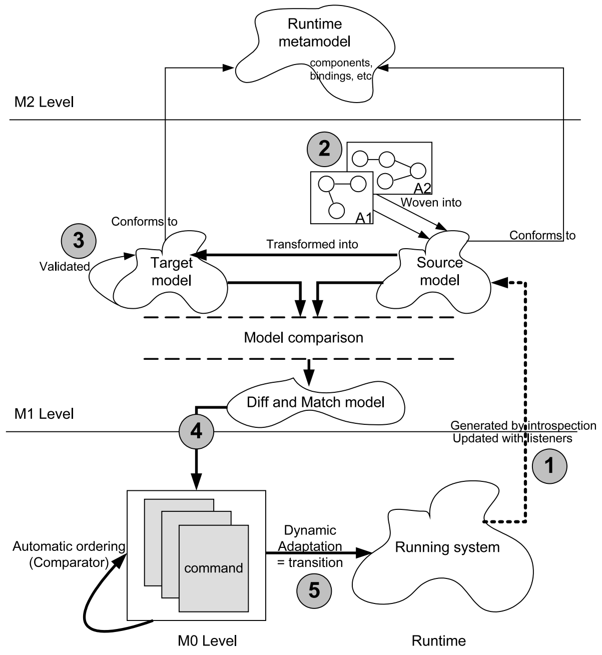


Figure 1. MDE and AOM for Dynamic Adaptation

Recent component-based middleware platforms like OSGi [33] propose introspection APIs that allows discovering the architecture of a running system. We use these APIs to collect and format relevant information in the form of a platform-independent and high-level model. In the case of large systems, using introspection in order to generate a reference model from scratch may be time-consuming especially when only small changes appear. To tackle this issue, we observe the architectural reconfigurations appearing in the running system in order to update the model. This limits the flow of data manipulated in the system. Moreover, recent middleware platforms already propose this kind of observers or propose mechanisms to easily implement such kind of observers [6, 7]. Note that the changes that may affect the source model are not directly reflected to the running system.

2.2 On-demand Construction of Configurations with Aspects

In order to manage variability and avoid the combinatorial explosion of artifacts needed to support this variability, we propose to focus on variation points and variants instead of focusing on whole configurations. A variability dimension is a particular concern that may be realized in different ways. We use aspects to represent the different variants of a variation point. Using Aspect-Oriented weavers, whole configurations can be built on-demand by selecting a set of aspects as illustrated in Figure 1, step 2. In practice, we use SmartAdapters [18, 25] an Aspect-Oriented Modeling (AOM) tool for weaving aspects at a model level. However, the approach presented in this paper is not dependent from SmartAdapters and other AOM tools like MATA [13] could also be used. Components present in all the configurations constitute a base model where aspects are woven.

SmartAdapters has formerly been applied to Java programs and UML class diagrams [18]. More recently, we have generalized this approach to any domain-specific modeling language [23]. This allows us to leverage the notion of aspect for runtime models representing at a high level of abstraction the architecture of a system at runtime. SmartAdapters automatically generates an extensible Aspect-Oriented Modeling framework specific to our metamodel.

In SmartAdapters, an aspect is composed of three parts: *i)* an advice model, representing what we want to weave, *ii)* a pointcut model, representing where we want to weave the aspect and *iii)* weaving directives specifying how to weave the advice model at the join points matching the pointcut model. The advice model is a model fragment representing a given concern. In our case, it represents a pre-assemby of components that may not be fully specified. The pointcut model is also a model fragment that is parameterized by roles (See [31]), equivalent to wildcards in AspectJ pointcuts [15, 16]. Both the advice model and the pointcut model are defined using the concrete syntax of the domain. Finally, weaving directives specify how to integrate the advice model into the target model, using a generated domain-specific action language [23].

We (optionally) extend each aspect with a context describing when to trigger the weaving of aspects. A context is a slice of the environment describing when the aspect is useful and its impact on QoS properties. For example, a buffering aspect can *optimize* the bandwidth of the network if the system has enough free memory (*e.g.* > 512 Mb) and if the bandwidth is saturated (*e.g.* $> 90\%$). Aspects with a context are chosen according to the execution context and the QoS properties to optimize [12]. Aspects with no context can be manually triggered by the user.

Figure 2 illustrates an internalization (`I18N` for short) aspect. The pilot of the case study is currently deployed in Rennes. Rennes is an international city hosting many students from different countries where lots of different languages are spoken. Within a single day, people from different countries may transit in the home. Systematically translating all the information in all the possible languages may cause an information overhead that could make information difficult to catch. This is why internationalization should be handled dynamically.

In order to design this aspect, we leverage the ability of SmartAdapters to integrate variability into aspects [18]. Indeed, each language (`EN`, `FR`, `DE` in Figure 2) is considered as a variant. The behavior of the advice can be described as follows. The `I18N` interface provides a set of methods responsible for translating a pre-defined set of labels. It also provides a look-up method `get(String myString): String` that returns, if possible, the translation of `myString` for a given language. In a component (*e.g.*, `FR`), if the look-up method cannot translate a word, the component ask the dispatcher to find a translation for this word. The dispatcher will ask the components according to a predefined policy (*e.g.*, `EN` first if available). If the word exists in the local database, it is translated (*e.g.*, from English to French) thanks to the translator that sends a request to a website dedicated to translation. Note that in the advice, all the components and bindings are unique (depicted with a '1' in the Figure). This means that even if there exist multiple join points and/or even if the aspect is applied several times, these elements will only be woven once in the base model. In the composition protocol, the bindings are not unique and will be introduced for all the identified join points. The pointcut simply identifies any component that requires the `I18N` interface, with no more assumption. The weaving process has 2 steps: matching (or join point detection) [31] and composition. In our example, the matching step detects all the components that require the `I18N` interface. Then, the composition step binds the `I18N` server interface provided by the advice to the client interfaces of the join points. When an aspect is being composed, the inverse composition protocol is automatically generated. It allows us to easily unweave an aspect when it is not longer adapted to the context.

After some aspects have been woven into the source model, the target model we obtain is validated, as illustrated in Figure 1, step 3. We will present in more details this validation step in Section 3.

2.3 On-the-fly Generation of Reconfiguration Scripts

As mentioned by Zhang and Cheng in [35], if there exists N possible configurations, this may lead to N(N-1) possible transitions. If N is large, it rapidly becomes difficult to handle these transitions by hand.

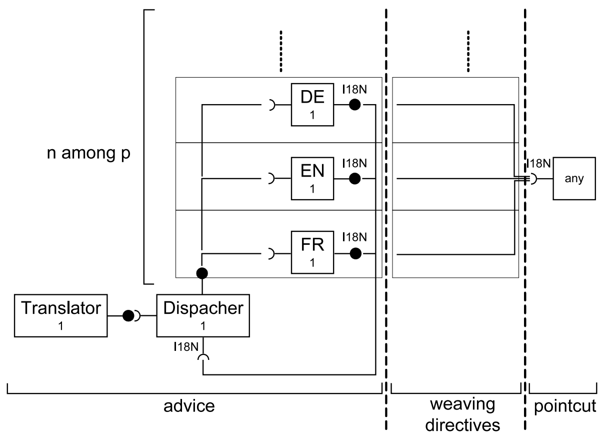


Figure 2. I18N aspect

In a traditional model-driven development, models are refined and transformed step by step down to source code. If some changes appears in the requirements, it is possible to propagate these changes to the models and regenerate the source code in order to rapidly propose a new version of the system. In the context of adaptive systems, it is not possible to adopt this schema. Indeed, such systems should offer continuous services and cannot be stopped, regenerated and restarted. The system should keep executing while being reconfigured from one configuration to another.

Once a target model (representing the system we want to reach) is created and validated, it is compared with the source model (representing the actual architecture of the running system). In the current implementation of our tool, we use EMF Compare[1] in order to compare models. It produces a diff and a match model that specifies the differences and the similarities between the source and the target models, as illustrated in Figure 1, step 4. The comparison engine is generic, so it is possible to compare any kind of models. The algorithm is quite similar to the algorithm proposed by Nejati *et al.* in the context of statechart specifications [28]. It considers the properties of each model element as well as its neighbors in order to compute a similarity degree. Note that it is possible to customize the comparison engine to consider the specificity of a given domain metamodel. However, the generic engine provides sensible results for our metamodel describing runtime architecture, with no customization.

Then, we automatically analyse both diff and match models to obtain the relevant changes between the source model and the target model *e.g.*, addition/removal of components/bindings, changes of attribute values, etc. However, it is not possible to adapt the running system during this analysis. For example, if the model comparison detects a component removal before a binding removal, directly adapting the system would lead to a dangling binding that might not be allowed by the underlying execution platform.

In order to tackle this issue, we reify each significant modification as a reconfiguration command during the analysis (Figure 1, step 4). Each command implements an atomic platform-specific reconfiguration (adding and/or removing bindings and/or components, etc.) and declares a priority. We first stop the components that have to be stopped and then remove bindings before removing components. We add components before adding bindings and finally restart the components that should be restarted. When the analysis of the model comparison is achieved, we execute the ordered sequence of commands to actually adapt the running system in a safe way (Figure 1, step 5). This set of commands is the transition that transforms the source system into the target system. Depending on the execution platform we use (*e.g.*, OSGi, Fractal or OpenCOM) a factory will instantiate the corresponding commands.

3 Validating Target Configurations

In Section 2, we showed how we can obtain configurations by weaving some aspects into a base configuration. Using aspect models, instead of directly adapting the running system using low-level reconfiguration scripts [10] allows us to reason more easily and help designers in identifying interactions between aspects [13]. More details about aspect interaction detection are given in Section 3.1.

Then, we showed how to generate reconfiguration scripts to make the running system evolve from a source configuration (the current configuration), to a target configuration. However, in the context of adaptive systems, we should ensure that the target configuration we want to reach makes sense. This is why we do not directly reflect the changes appearing in the source model to the running system. When the number of configurations is limited, for example in the case of critical embedded systems, it is possible to validate at design time all the possible configurations [35]. However, in larger-scale adaptive systems, this systematic validation may become too time and resource consuming to be realistic. Moreover, once the system has been deployed, new variation points that have not been foreseen may appear while the system is running and cannot be stopped. In this case, it is very useful to validate configurations on the fly before actually adapting the

[1]See http://www.eclipse.org/modeling/emft/

running system. This is detailed in Section 3.2.

3.1 Detecting Aspect Interactions

In Section 2 we introduced an internationalization aspect. Most of the aspects of our case study (see Section 4) and most of the components of the base system (for example, GUI) also needed this aspect. Weaving the I18N aspect before the other aspects may cause some important messages not to be translated. Using techniques like Critical Pair Analysis (CPA) allows us to detect interaction between aspects [13]. Basically, if the pointcut of an aspect *A1* can be matched in the advice of another aspect *A2* it means that *A2* introduces some join points for *A1*. In other words, *A1* should be woven after *A2* in order to be able to consider newly introduced join points.

For example, let us introduce a second aspect responsible for preventing devices deployed in the house (lights, electric shutters, etc) to be damaged due to too many successive transitional regimes. This aspect, called **event filter**, will ignore all the antagonist actions that appears in a too short period. All the events sent to the unstable device controller by the device proxy are derived into the event filter component. These events are cached during a given period that depends on the type of the device. Events are delegated to the device if no antagonist events appeared during the cache period. These filters cannot be systematically deployed as they make devices less reactive in standard conditions. This is why filters should be dynamically and locally deployed and undeployed. Every canceled actions has to be logged and displayed in a language the user understand, using the I18N interface. As the event filter aspect introduces a component that requires the I18N interface, it introduces a new join point for the internationalization aspect. Consequently the I18N aspect should be woven after the event filter aspect.

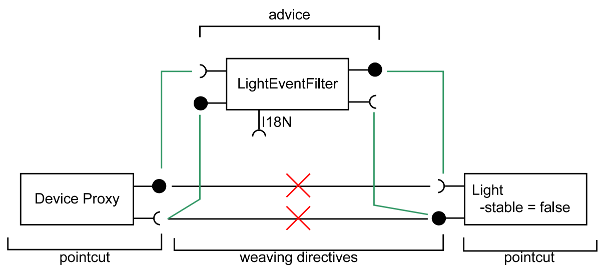


Figure 3. Event Filter aspect

This kind of basic analysis can help designers in identifying interactions that cannot easily be detected when directly working with low-level reconfiguration scripts. However, CPA has limitations. For example, this kind analysis is not associative. If no interaction exists between *A1* and *A2* and no interaction exists between *A1* and *A3*, nothing ensures that there exists no interaction in the triplet {*A1*, *A2*, *A3*}, depending on the weaving order. Determining the interactions that may exist in all the possible subsets of aspects is very complex as the number of combinations grows rapidly.

3.2 Validating Target Configurations

As explained in the introduction of this section, it is not always possible to check all the possible configurations of an adaptive system *a priori*, for time and resource issues. Moreover, the apparition of unforeseen adaptation while the system is already deployed makes it impossible to perform all the validation process at runtime. However, in the context of high-insurance adaptive systems [20] any configuration we wanted to reach should be validated.

In order to validate target configurations (Figure 1, step 3), we propose to define some invariants on the metamodel we use to represent runtime architecture and check these invariants for every constructed (by aspect weaving) target configuration. These invariants are expressed as Kermeta [26] meta-aspects that are woven into the metamodel we are using for representing runtime architecture. Kermeta meta-aspects can be used to refine existing meta-classes by integrating contracts (pre/post-conditions, invariants), attributes and references, operations and super-classes. The invariant illustrated in Figure 4 specifies that all the client and non optional ports defined in the component type of the component should be bound. In other words, it detects if mandatory bindings are missing. It uses an OCL-like syntax[2] which provides high-level operators for navigating and querying models: *select*, *exist* and *forall* in our example. Another invariant checks that the server interface of a binding is a sub-type of the client interface. Currently, six invariants are woven into our metamodel. Note that end-user can define more specific invariants to ensure the validity of the configurations.

Invariant checking, as well as the steps related to aspect weaving and aspect interaction detection, can be performed on a tiers system, independent from the running system itself. Indeed, all the models (metamodel, configurations and aspects) can be serialized in XML and transmitted to other systems.

[2]Kermeta now allows to define constraints using the real OCL syntax

```
aspect class Component {
  inv optionalClientPortBound is do
    self.type.ports.select{p |
        not p.isOptional and
        p.role == PortRole.CLIENT
    }.forAll{p |
        self.binding.exists{b |
          b.client == p
        }
    }
  end
}
```

Figure 4. Checking mandatory bindings

Another possible solution to validate target configuration before actually adapting the running system would be to simulate models. This can be done by describing the behavior of each configurations, for example using state machines or Petri nets [35]. Then, it would be possible to use Kermeta [26] to execute these models and perform the simulation and detect deadlocks, for example. However, in order to manage the explosion of variants, this behavior should be associated to each aspect and should be composable in order to obtain the behavior of whole configurations. SmartAdapters is well adapted to compose structural aspects, in class or component diagrams. However, to consider the semantic of behavioral models, it has to be customized by hand. Another solution would be to use a specific aspect weaver dedicated to the composition of behavioral models. Such an approach is presented in Section 5 and its integration with our approach is discussed in perspectives.

If the target configuration we want to reach is valid, then the process continues, as illustrated in Figure 1, step 4, in order to actually adapt the running system. If the target configuration is not valid, then our rollback mechanism simply consists in discarding this target configuration and do not submit it to the following steps of the process. Indeed, as the modification on the model are not directly reflected to the running system, we do not have to cancel platform-level reconfigurations. An error report is automatically raised by Kermeta [26], specifying which invariants are violated by the target configuration. This helps the system or the user in understanding why the configuration is not valid. For example, if we consider that the I18N client port is mandatory, then we are able to detect the cases where the I18N has been woven before (or not at all) other aspects, without using the preliminary critical pair analysis. Indeed, the invariant illustrated in Figure 4 detects missing mandatory bindings. This case is illustrated in Figure 5 that shows a fragment of the base model where the event filter has been woven and the I18N is not woven at all.

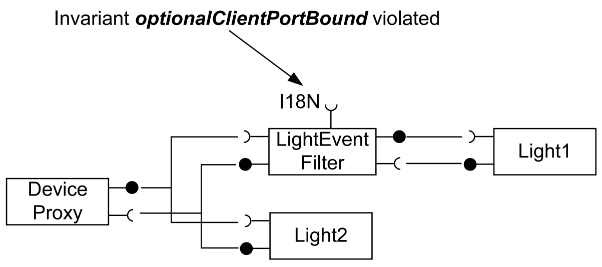


Figure 5. Invariant violated

4 Application to Home-Automation

4.1 EnTiMid to help people to stay at home

Industrials, associations and public institutions of the metropolis of Rennes, are working together on a project which aims to allow dependent people to stay at home as long as possible. Due to the large scale of the project, and the diversity of disabilities that have to be considered, the deployment context will be different for each equipped house. The technologies used will vary, in order to compensate handicaps or because a technology is already installed, and people do not want it to be removed. Moreover, the system installed in these houses will have to provide a remote access to the devices of the house, and transmit all the necessary information from the sensors of the house to a control center where information will be treated. Those access and transmissions can be realized through various ways (Internet, POTS, SMS) and the medium used will vary according to the availabilities.

An abstraction layer over all these devices has been developed in the form of a multi-facet middleware called EntiMid [27]. Based on an OSGi platform [33], EnTiMid is composed of different components (called bundles), that can be dynamically added, removed, started or stopped, with no need to restart the entire system. Each of them can offer or require services to/from other ones, but such services can disappear at any time. One of those services, identified as *TechProvider*, aims at translating the information caught from a device network protocol, into EnTiMid standard event messages, and vice versa. By this way, high level services can manage devices through unified messages, whatever the underlying technology is. EntiMid allows engineers to prescribe the most adapted technology, with no regard to the communication technology it uses. Then high level services can be developed to offer different kind of services to inhabitants and health professionals. For example, automatic energy, heating, access or light management to ease the everyday life; remote control and alert transmissions, to allow professionals to intervene on the house, in a short time, according to the information collected.

A show apartment will soon be available, and EnTiMid will be deployed in order to test its functionalities with devices installed by industrials. Those real conditions will have for consequence an identification of new needs of development and variability.

4.2 Designing Variability Dimensions

We now introduce and justify the need for dynamic variability in our case study. Each variability dimension is represented by a set of aspects designed with SmartAdapters.

Device Management. Physical devices are managed by EnTiMid. Most of the devices are installed when the system is deployed. However, new physical devices may be installed after the initial deployment and consequently, they should be managed dynamically while maintaining the functionalities offered by already deployed devices. In our case study, we have to manage 6 lights, 4 heaters, 3 mixing valves (controlling water temperature) and 3 electric shutters. Moreover, depending on the evolution of the patient, devices should be managed in different ways. For example, in the case the patient becomes visually impaired, the power of the lights should be increased by 10%. Another focus can be the evolution of a mobility handicap. In a first stage, the patient can move alone in the house, allowing him to manually close the shutters. Then the evolution of the disease makes it difficult for the patient to get out of his bed. A remote control will then be offered to this person to ease his everyday life, and the control system of the house have to adapt to this situation.

Each low-level protocol (KNX[3], X10, X2D[4], etc) manages devices in an ad-hoc way. Consequently, we would have to define a variability dimension for each protocol. For the sake of clarity, we present one generic variability dimension that harmonizes the concepts present in each low-level protocol. It is illustrated in Figure 6. In order to represent this variability dimension, we leverage the ability of SMARTADAPTERS to integrate variability into aspects [18]. Each type of device controller (Light, Heater, Shutter and Mixing Valve) is a variant. Each type of component can be instantiated several times. Each device also offers an interface for loading pre-defined scenarios. All the device controllers are connected to a device proxy that receives messages from the EnTiMid platform and dispatches these messages to the appropriate device.

[3]http://www.knx.org/

[4]http://www.english.deltadore.com

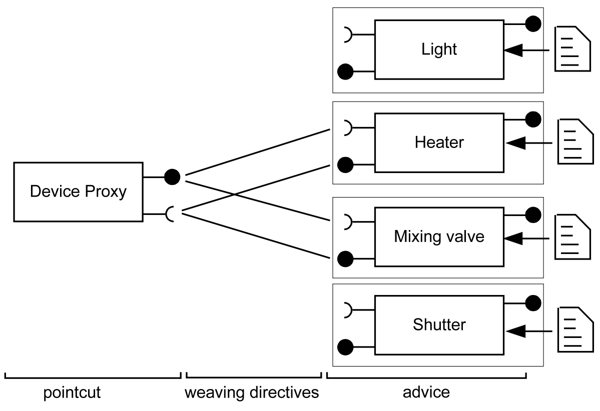
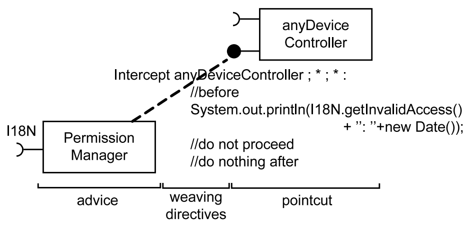


Figure 6. Device Management aspect

Permission Management. All the physical devices in the house are supposed to be potentially controlled by EnTiMid. However, doctors can choose for each device whether it is controlled by the system or by the patient, according to the degree of autonomy of the patient. If the device has no permission manager, the patient can interact with the device. With a permission manager, the patient cannot interact with the device that only follows a pre-defined scenario.

The permission management aspect illustrated in Figure 7. In order to control the access from the user, an aspectual component [29] intercepts, using an AspectJ-like pointcut, every call to the services of the controller, log each attempt and does not proceed. The device will simply execute its pre-defined scenario without being interrupted.



```
Intercept anyDeviceController ; * ; * :
        //before
        System.out.println(I18N.getInvalidAccess()
                                  + ": "+new Date());
        //do not proceed
        //do nothing after
```

Figure 7. Permission Management aspect

Two other aspects (internationalization and event filter) have already been introduced in previous sections. Table 1 summarizes the number of aspects we really need for each variability dimension to implement our case study.

Dimensions	Aspects	Aspect variants
Device Mgmt	3 (KNX, X10, X2D)	4 (1 per device type)
Permission Mgmt	1	0
Event Filter	3	4
I18N	1	10
Total	8	18

Table 1. Number of aspects per variability dimension

4.3 Constructing a Configuration by Aspect Weaving

We are now going to propose to illustrate the weaving process with some of the aspects we have identified in the previous sub-section into our motivating example (Figure 8). The top of the figure illustrates a snippet of the base configuration. It allows to access low-level devices using two high-level protocols: UPnP[5] and DPWS[6], via the EnTiMid component. In the bottom part of the figure three aspects have be woven: a filter aspect that filters the events send to Light1 ; two permission managers that restrict the heater and the shutter to their pre-defined scenarios. Finally, the internationalization aspect is woven. The interaction between the aspects can be observed in the example: the event filter and the permission manager aspects uses the internationalization aspects.

With the five variability dimensions we have defined earlier in this section, we can obtain a wide range of possible configurations. If we consider the three aspects that directly impact devices (device management, permission management and event filter), we obtain five possible modes for each device, as shown in Table 2, where 0 indicate that the aspect is not active for the device. As the devices are managed independently, this leads to $5^{16} \approx 15 \cdot 10^{10}$ possible configurations. This number is even greater if we consider the internationalization aspect.

If we consider the $15 \cdot 10^{10}$ possible configurations, we obtain approximately $225 \cdot 10^{20}$ possible transitions from one configuration to another. The configurations are obtained on-demand: the weaving of some aspects can be triggered by hand depending on the choices of a human operator (doctor or technician), while some other aspects (*e.g.*, event filter) are triggered by the context. Before adapting the system, all the invariants

[5]http://www.upnp.org/

[6]http://en.wikipedia.org/wiki/Devices_Profile_for_Web_Services

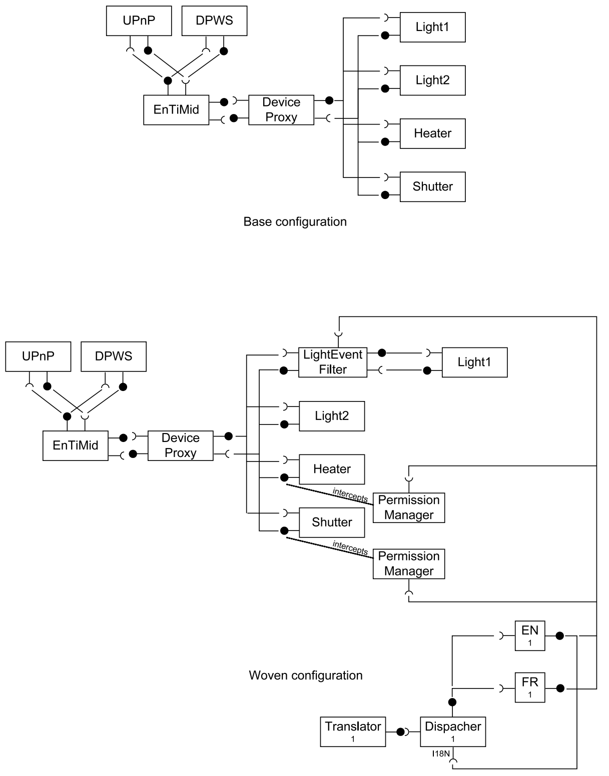


Figure 8. Base and woven configurations

Device Mgmt.	Permission Mgmt.	Event Filter
0	0	0
1	0	0
1	0	1
1	1	0
1	1	1

Table 2. Five Operating Modes per Device

defined in the metamodel are checked on the woven target model. This allows us to determine whether the target model is well-formed or not. If the configuration is valid, the transition toward the target model is automatically computed using model comparison. If we consider the internationalization aspect, we should multiply the number of configurations by 2^{10} as we should handle at least one language among 10. Table 3 summarizes the number of dimensions we have defined, the total number of aspects we need, the number of configurations we obtain on-demand by aspect weaving and the number of transitions we generate on-demand when moving from one configuration to another.

Dimensions	Aspects	Configurations	Transitions
4	8	$> 15 \cdot 10^{13}$	$> 225 \cdot 10^{26}$

Table 3. Number of configurations and transitions managed by aspects

5 Related Work

In [35], Zhang and Cheng propose to adopt a model-driven approach for designing and validating dynamically adaptive software systems. This approach focus on the behavior of adaptive systems whereas we mainly focus on the architecture of running systems. In [35], the behavior is modeled with state-based diagrams like Petri nets. Zhang and Cheng define an adaptive system as a set of simple adaptive systems. A simple adaptive system is defined by three entities: a source system, a target system and transitions responsible for moving the source system to the target system. All the source systems, target systems and transitions should be explicitly modeled, leading to an explosion of artifacts needed to manage an adaptive system. However, this exhaustive representation allows validating intensively the system at design time. Finally, using code generation, adaptive programs are derived from the models. In our approach, configurations are constructed on demand by selecting and weaving a set of aspects. Once composed, it is possible to check the target configurations we want to reach. Then, the transitions needed to adapt the system is automatically generated using model comparison.

In [10], David *et al.* present SAFRAN (Self-Adaptive FRActal compoNents) an Aspect-Oriented approach for implementing self-adaptive system on the Fractal [7] platform. In order to adapt the system, they define adaptation policies, separately from the business logic, which follow this pattern: **when** <*event*> **if** <*condition*> **do** <*action*> where actions are low-level reconfiguration scripts. Batista *et al.* [3] propose the same kind of approach for the OpenCOM [6] execution platform. These approaches do not propose an explicit representation of the target configurations. Consequently, it is not possible to easily visualize the system nor to perform validation, simulation before actual adaptation. In our approach, configurations are obtained by weaving aspects into a model representing the current system. Woven configurations are checked against invariants. We then generate all the adaptation logic needed to adapt the system while these approach have to specify low-level reconfiguration scripts. Moreover, our approach is not specific to a given execution platform. Finally, script-based approach do no offer support for easily determining interactions between reconfiguration scripts while Aspect-Orientation provides some mechanisms [13].

Ensuring software correctness is an important issue and this is amplified when dealing with software variation . Correctness is even more important in Dynamically adaptive System where variation is handled at runtime. This issue has been addressed by the software product lines community [30, 32]. One of the main concerns for correctness in product lines is about the methods to be used in order to limit the number of tests to be performed for a family of products. Two issues are especially considered: the increase of work for the programmer and the time spent to perform them [8, 19]. These contributions mainly introduce formal methods in order to exploit the commonalities of a software family in order to achieve these issues. They rely on SAT solver [9] or more generally on model-checking [17] techniques in order to verify those tests. In our case, the main difference is the fact that verifications can only be done at runtime. New aspect can be designed and integrated, consequently unanticipated evolution can occur. Using MDE techniques allows software developer to apply his aspect on an abstraction of its runtime system to check its correctness.

In [24], we present a first approach that combines AOM and MDE in order to manage variability at runtime. In this paper, we show how aspects can help designer in determining interactions between dynamic variants and how models allows to validate new configurations independently from the running system and easily roll-back when a configuration is not valid.

In [34], Wolfinger *et al.* demonstrate the benefits of integrating Software Product Line techniques to manage the runtime reconfiguration and adaptation mechanisms on the .NET platform. Automatic runtime adaptations are attained by using the knowledge documented in variability models. As many authors [1, 2, 22] advocate that aspect-oriented software development (AOSD) is an effective technique to support feature variability, this approach is close to ours. Automatic runtime adaptations are attained by using the knowledge documented in variability models. However, they do not propose to do a preview of the running system at the model level to check its correctness.

In the domain of Aspect-Oriented Modeling, Nejati *et al.* [28] propose an approach for matching and merging statechart specifications. This approach would be useful for extending our approach with behavior, as mentioned in Section 3.2. If we describe the behavior of our aspects it would possible to merge this behavior with the base model in order to obtain the complete behavior. Describing the behavior of our aspects mainly consists in modeling the behavior of the interfaces of each component and compose these behavioral models when components are assembled. Once we obtain the global behavior, it is possible to reuse the concepts proposed by Zhang and Cheng [35] for validating the behavior of adaptive systems.

6 Conclusion

In this paper, we have presented our approach for managing the complexity of dynamically adaptive systems. This approach combines aspect-oriented and model-driven techniques in order to limit the number of artifacts needed to realize dynamic variability. Our aspect model weaver allows us to construct configurations on-demand by selecting, by hand or according to predefined conditions, a set of aspects. Using the woven configuration, it is possible to validate this configuration before actually adapting the running system. Using aspects instead of low-level reconfiguration scripts allows us to detect some interactions that can provide assistance when selecting the set of aspects to be woven. Then, target configurations obtained after aspect weaving are checked with respect to the invariant we have defined into our metamodel. If a target configuration is not valid, the roll-back mechanism simply consists in not submitting this target configuration to the sub-sequent steps of the adaptation process. If the configuration is valid, we generate the adaptation logic using model comparison. This allows us to automatically determine a safe transition to make the system evolve from a its current configuration to the target configuration.

In future works, we plan to extend our approach following different axis. Currently, we describe our systems according to their runtime architecture (components, bindings, etc). We will also consider the behavior of dynamically adaptive systems. This can be realized if we can modularize and compose the behavior of components. Thus, it would be possible to decompose the system behavior into aspects, as we do for the architecture. We plan to reuse the approach we have presented in the related work section to consider the behavior. Another axis is about the validation of target configurations. Currently, we ensure that the target configurations ensure the invariants defined in our metamodel. With the definition of the behavior, we would be able to perform simulation in order to detect some deadlocks, for example.

References

[1] V. Alves, P. Matos Jr, L. Cole, A. Vasconcelos, P. Borba, and G. Ramalho. Extracting and Evolving Code in Product Lines with Aspect-Oriented Programming. *Transactions on Aspect-Oriented Software Development IV*, 4640/2007, 2007.

[2] S. Apel, T. Leich, and G. Saake. Aspectual mixin layers: aspects and features in concert. In *ICSE '06: 28th International Conference on Software Engineering*, pages 122–131, Shanghai, China, 2006. ACM.

[3] T. Batista, A. Joolia, and G. Coulson. Managing Dynamic Reconfiguration in Component-Based Systems. In *EWSA'05: 2nd European Workshop on Software Architecture*, pages 1–17, Pisa, Italy, 2005.

[4] N. Bencomo, G. Blair, and R. France. Models@run.time (at MoDELS) workshops. www.comp.lancs.ac.uk/ bencomo/MRT/.

[5] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair. Genie: Supporting the Model Driven Development of Reflective, Component-based Adaptive Systems. In *ICSE'08: Formal Research Demonstrations Track*, Leipzig, Germany, 2008.

[6] G. Blair, G. Coulson, J. Ueyama, K. Lee, and A. Joolia. Opencom v2: A component model for building systems software. In *IASTED Software Engineering and Applications*, USA, 2004.

[7] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J. Stefani. The FRACTAL Component Model and its Support in Java. *Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, 2006.

[8] M. Cohen, M. Dwyer, and J. Shi. Coverage and Adequacy in Software Product Line Testing. In *ROSATEA '06: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 53–63, Portland, Maine, 2006. ACM.

[9] K. Czarnecki and K. Pietroszek. Verifying Feature-Based Model Templates Against Well-Formedness OCL Constraints. In *GPCE'06: 6th Int. Conf. on Generative Programming and Component Engineering*, pages 211–220, Portland, Oregon, USA, 2006. ACM.

[10] P. David and T. Ledoux. An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components. In *SC'06: 5th Int. Symposium on Software Composition*, volume 4089 of *Lecture Notes in Computer Science*, pages 82–97, Vienna, Austria, 2006.

[11] E. Figueiredo, N. Cacho, C. SantAnna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Filho, and F. Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *ICSE'08: 30th International Conference on Software Engineering*, pages 261–270, Leipzig, Germany, may 2008. ACM.

[12] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J.-M. Jézéquel. Modeling and Validating Dynamic Adaptation. In *3rd International Workshop on Models@Runtime (MODELS'08)*, Toulouse, France, oct 2008.

[13] P. Jayaraman, J. Whittle, A. Elkhodary, and H. Gomaa. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis. In *MoDELS'07: 10th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, Nashville USA, Oct 2007.

[14] J.-M. Jézéquel. Model Driven Design and Aspect Weaving. *SoSyM'08: Journal of Software and Systems Modeling*, 7(2):to appear, march 2008.

[15] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An Overview of AspectJ. In *ECOOP'01: Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.

[16] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *ECOOP'97: Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Berlin, Heidelberg, and New York, 1997. Springer-Verlag.

[17] T. Kishi, N. Noda, and T. Katayama. Design Verification for Product Line Development. In *SPLC'05: 9th International Software Product Lines Conference*, volume LNCS 3714, pages 150–161. Springer, 2005.

[18] P. Lahire, B. Morin, G. Vanwormhoudt, A. Gaignard, O. Barais, and J. M. Jézéquel. Introducing Variability into Aspect-Oriented Modeling Approaches. In *MoDELS'07: 10th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, Nashville USA, Oct. 2007.

[19] M. Mannion and J. Cámara. Theorem Proving for Product Line Model Verification. In *Software Product-Family Engineering, 5th International Workshop, PFE 2003*, volume LNCS 3014, pages 211–224. Springer, 2003.

[20] P. McKinley, B. H. C. Cheng, C. Ofria, D. Knoester, B. Beckmann, and H. Goldsby. Harnessing digital evolution. *Computer*, 41(1):54–63, 2008.

[21] M. Mezini and K. Ostermann. Variability Management with Feature-Oriented Programming and Aspects. *SIGSOFT Software Engineering Notes*, 29(6):127–136, 2004.

[22] M. Mezini and K. Ostermann. Variability management with feature-oriented programming and aspects. In *SIGSOFT'04/FSE-12: 12th ACM SIGSOFT international symposium on Foundations of Software Engineering*, pages 127–136, Newport Beach, CA, USA, 2004. ACM.

[23] B. Morin, O. Barais, J. M. Jézéquel, and R. Ramos. Towards a Generic Aspect-Oriented Modeling Framework. In *3rd Int. ECOOP'07 Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD*, Berlin, Germany, August 2007.

[24] B. Morin, F. Fleurey, N. Bencomo, J.-M. Jézéquel, A. Solberg, V. Dehlen, and G. Blair. An Aspect-Oriented and Model-Driven Approach for Managing Dynamic Variability. In *MoDELS'08: 11th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, Toulouse, France, October 2008.

[25] B. Morin, G. Vanwormhoudt, P. Lahire, A. Gaignard, O. Barais, and J.-M. Jzquel. Managing Variability Complexity in Aspect-Oriented Modeling. In *MoDELS'08: 11th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, Toulouse, France, October 2008.

[26] P. Muller, F. Fleurey, and J. M. Jézéquel. Weaving Executability into Object-Oriented Meta-languages. In *MoDELS'05: 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, Montego Bay, Jamaica, Oct 2005. Springer. Kermeta is available at: http://www.kermeta.org/.

[27] G. Nain, E. Daubert, O. Barais, and J. M. Jézéquel. Using MDE to Build a Schizonfrenic Middleware for Home/Building Automation. In *ServiceWave'08: Networked European Software & Services Initiative (NESSI) Conference*, Madrid, Spain, december 2008.

[28] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and Merging of Statecharts Specifications. In *ICSE'07: 29th International Conference on Software Engineering*, pages 54–64, Minneapolis, MN, USA, 2007. IEEE Computer Society.

[29] N. Pessemier, L. Seinturier, T. Coupaye, and L. Duchien. A Component-Based and Aspect-Oriented Model for Software Evolution. In *IJCAT'07: International Journal of Computer Applications in Technology, Special Issue on Concern-Oriented Software Evolution*, volume 4089 of *Lecture Notes in Computer Science*, page 259273, Vienna, Austria, mar 2006. Springer-Verlag.

[30] K. Pohl and A. Metzger. Software Product Line Testing. *Commun. ACM*, 49(12):78–81, 2006.

[31] R. Ramos, O. Barais, and J. M. Jézéquel. Matching Model Snippets. In *MoDELS'07: 10th Int. Conf. on Model Driven Engineering Languages and Systems*, page 15, Nashville USA, Oct. 2007.

[32] M. Svahnberg and J. Bosch. Issues Concerning Variability in Software Product Lines. In *International Workshop on Software Architectures for Product Families*, volume LNCS 1951, pages 146–157, Spain, 2001. Springer.

[33] The OSGi Alliance. OSGi Service Platform Core Specification, Release 4.1, May 2007. http://www.osgi.org/Specifications/.

[34] R. Wolfinger, S. Reiter, D. Dhungana, P.Grunbacher, and H. Prahofer. Supporting runtime system adaptation through product line engineering and plug-in techniques. In *ICCBSS'08: 7th Int. Conf. on Composition-Based Software Systems*, pages 21 – 30, 2008.

[35] J. Zhang and B. H. C. Cheng. Model-based Development of Dynamically Adaptive Software. In *ICSE'06: 28th International Conference on Software Engineering*, pages 371–380, Shanghai, China, 2006. ACM Press.