



## Dynamic Variability in complex, Adaptive systems

Deliverable reference: <b>D2.1</b>	Date: 29 March 2010	Responsible partner: SINTEF
Title:  <h3>D2.1 Transformation Framework</h3>		
Editor(s): Vegard Dehlen, SINTEF Franck Fleurey, SINTEF		Approved by: Arnor Solberg Classification: Public
<p>Abstract / Executive summary:</p> <p>Constructing and executing distributed systems that can automatically adapt to the dynamic changes of the environment are highly complex tasks. Non-trivial challenges include provisioning of efficient design time and run time representations, system validation to ensure safe adaptation of interdependent components, and handling of possible combinatorial explosions of adaptive system artefacts such as configurations, variant dependencies and adaptation rules. These are all challenges where current approaches offer only partial solutions. Furthermore, existing technologies are typically only provided at the implementation level which makes them complex to use. This deliverable describes the first version of the model transformation framework in Work Package 2. As stated in the DoW the objective of WP2 is to develop a domain-specific language for adaptive system specifications, and a composition and transformation framework. The first phase of the project has mainly focused on the former part (the domain specific language for adaptive system specifications) and is described in this deliverable. The composition and transformation framework part of WP2 will be the focus in the second phase of DiVA. This deliverable consists of three parts. The main part is the tools and metamodels developed so far. The second part is video tutorials showing the usage of these tools, while the third part is this document. This document presents the domain-specific modelling language for adaptive system specification and the tools that support specification, model checking and design time simulation of the adaptation model of adaptive systems. The proposed approach combines aspect-oriented and model-driven principles to cope with the combinatorial explosion and provide model level representations of variants, context variables and adaptation rules. An open source Eclipse-based editor has been developed to support system specification according to the meta-model. Based on the implemented adaptation meta-model, model checking and simulation facilities are provided to prepare for safe adaptation. At runtime the adaptation models are used to drive the adaptation. The approach is validated through case studies.</p>		
<p>DiVA IS A STREP FUNDED UNDER CONTRACT 215412 TO THE SEVENTH FRAMEWORK PROGRAMME, THEME 1.2: SERVICE AND SOFTWARE ARCHITECTURES, INFRASTRUCTURES AND ENGINEERING</p>		<p>INTERNATIONAL DOCUMENT AVAILABLY: PARTNER + REPORT NUMBER ISBN</p>



## Table of Contents

<b>D2.1 Transformation Framework .....</b>	<b>1</b>
<b>Table of Contents .....</b>	<b>2</b>
<b>Version History .....</b>	<b>3</b>
<b>Contributing partners.....</b>	<b>4</b>
<b>List of Figures.....</b>	<b>5</b>
<b>1 Introduction .....</b>	<b>6</b>
<b>2 Relevance to other work packages .....</b>	<b>7</b>
<b>3 Related Work .....</b>	<b>8</b>
<b>4 Conceptual Model.....</b>	<b>9</b>
<b>5 DSML framework for adaptive systems: initial approach.....</b>	<b>10</b>
5.1 META-MODEL FOR VARIABILITY AND ADAPTATION .....	10
5.1.1 <i>Modelling variability</i> .....	11
5.1.2 <i>Modelling the context</i> .....	12
5.1.3 <i>Modelling adaptation</i> .....	12
5.1.4 <i>Modelling constraints</i> .....	13
5.2 SIMULATION AND VALIDATION .....	13
5.2.1 <i>Simulation Model and Implementation</i> .....	13
5.2.2 <i>Simulation output</i> .....	14
5.2.3 <i>Constraint checking and rule termination</i> .....	14
5.3 RESULTS AND LIMITATIONS .....	15
<b>6 DSML framework for adaptive systems: 2<sup>nd</sup> approach.....</b>	<b>16</b>
6.1 ILLUSTRATIVE EXAMPLE: A SEMI-AUTONOMOUS EXPLORATION ROBOT.....	17
6.2 ADAPTATION META-MODEL.....	17
6.3 MODELLING THE EXPLORATION ROBOT .....	19
<b>7 Simulation and validation of the adaptation model .....</b>	<b>22</b>
<b>8 Case studies and initial results.....</b>	<b>26</b>
8.1 CUSTOMER RELATIONSHIP MANAGEMENT (CRM) SYSTEM .....	26
8.2 CASE STUDIES INITIAL RESULTS.....	29
<b>9 Conclusions.....</b>	<b>29</b>
<b>10 Future work.....</b>	<b>30</b>
<b>References.....</b>	<b>31</b>

## Version History

Version	Description	Date	Who
0.1	Initial outline for kick-off-meeting	March 10 2008	Vegard Dehlen and Franck Fleurey
0.6	Added theory parts	August 20 2009	Vegard Dehlen
0.8	Cleaned up text, fixed references, added text	August 26 2009	Vegard Dehlen
1.0	Last fixes	August 31 2009	Vegard Dehlen
1.1	Added section on the ECA approach Added section about the CAS case study End of the Intro Edited		Franck Fleurey
1.2	First internal review	October 19 2009	Brice Morin
1.5	Updates according to review	November 17 2009	Vegard Dehlen
1.6	Updates according to review	November 19 2009	Arnor Solberg
1.7	Updates according to review	March 27 2009	Franck Fleurey

## Contributing partners

 **SINTEF**  
Forskingsveien 1  
0314 Oslo  
Norway

**Vegard Dehlen**  
Vegard.Dehlen@sintef.no

**Arnor Solberg**  
Arnor.Solberg@sintef.no

**Franck Fleurey**  
Franck.Fleurey@sintef.no

## List of Figures

FIGURE 1 OVERVIEW OF THE PROPOSED APPROACH .....	9
FIGURE 2 - META-MODEL FOR VARIABILITY AND ADAPTATION.....	11
FIGURE 3 - VARIABILITY IN THE SERVICE DISCOVERY APPLICATION.....	12
FIGURE 4 - CONTEXT OF THE SERVICE DISCOVERY APPLICATION.....	12
FIGURE 5 - ADAPTATION RULES FOR THE FUNCTIONALITIES OF THE SDA.....	13
FIGURE 6 - INVARIANTS OF THE SDA.....	13
FIGURE 7 - SIMULATION MODEL.....	14
FIGURE 8 - EXCERPT OF THE SIMULATION GRAPH FOR THE SDA.....	15
FIGURE 9 ADAPTATION META-MODEL .....	16
FIGURE 10 SCREENSHOT OF THE ADAPTATION MODEL EDITOR. THE EDITOR IS COMPOSED OF SIX TABS FOR EDITING CONTEXT, VARIABILITY, PROPERTIES, IMPACTS ON PROPERTIES AND PRIORITY RULES. ....	18
FIGURE 11 MODEL OF THE CONTEXT OF THE ROBOT.....	19
FIGURE 12 MODEL OF THE VARIABILITY AND CONSTRAINTS IN THE ROBOT .....	20
FIGURE 13 PROPERTIES OF THE ROBOT.....	21
FIGURE 14 IMPACT OF THE VARIANTS ON THE PROPERTIES OF THE ROBOT .....	21
FIGURE 15 ADAPTATION RULES OF THE ROBOT .....	22
FIGURE 16 IMPLEMENTATION OF THE ADAPTATION SIMULATOR .....	23
FIGURE 17 SIMULATION OUTPUT FOR A SINGLE CONTEXT. ....	25
FIGURE 18 SIMULATION OUTPUT FOR A SIMPLE CONTEXT EVOLUTION SCENARIO .....	26
FIGURE 19 CHARACTERISTICS OF THE ADAPTATION MODEL FOR FOUR CASE STUDIES.....	29
FIGURE 20 LINKS BETWEEN STAGES OF DEVELOPMENT FOR A DIVA SYSTEM.....	31

## 1 Introduction

Context-aware software systems that can automatically adapt to changes in their environments play increasingly vital roles in society's infrastructures. The demand for Dynamic Adaptive Systems (DAS) appears in many domains, ranging from crisis management systems such as disaster or power management, to entertainment and business applications such as mobile interactive gaming, tourist guiding and business collaborations applications. However, constructing and executing DAS are complicated. A main challenge is to cope with the variability that can lead to explosion of several adaptive system artefacts. The set of possible configurations of an adaptive system is typically specified by identifying variation points, which represents points in the software where variability may occur. Having variability at each variation point implies a combinatorial explosion of configurations and quadratic explosion of possible configuration transitions, which again can cause possible explosion of variant dependencies and adaptation rules. This makes it difficult to provide consistent adaptation rules and to convey optimized configurations for the particular context. To cope with DAS complexities we need proper modelling and validation techniques all along the development cycle. In WP2 we have developed design time tools to model and validate the adaptation model. The validation includes to check that the adaptation model is according to the adaptation metamodel and do not contradict any of the constraints specified in the adaptation model. Furthermore, we have developed tools to perform design time simulations of the adaptation model. The validation tools developed in WP2 essentially represents the semantics of the domain specific modelling language (DSML), thus, we see those as part of the DSML provided in WP2. The reasoning and validation framework of WP 4 will provide further validations and testing of the adaptive system (e.g., taking time constraints into account).

Current approaches for specifying the adaptation logic of adaptive systems rely on the direct use of language or platform mechanisms such as reflection, dynamic loading of code or architecture reconfigurations to build and execute DAS. Most modern languages and middleware platforms include these kinds of low-level mechanisms to support runtime adaptation. However, using such techniques, the adaptation is captured in low-level platform specific scripts and tightly coupled with the application code. The development of these scripts typically comes very late in the development cycle, is particularly error-prone, and the resulting system is brittle to any change in the platform or in the application.

To overcome these problems, the state of the art has recently evolved to support variability and adaptation modelling, and also to make use of models at runtime to drive and monitor runtime adaptation [2][3][6][7][8]. Two main families of formalisms have been proposed in order to capture adaptation policies. The most common one is based on event-condition-action (ECA) rules directly relating environment events to reconfiguration actions (e.g., [6][8]). These approaches benefit from using well-known policy definition formalisms, they can be implemented very efficiently and allow early simulation and verification. These approaches are very well suited for small to medium scale<sup>1</sup> context-aware systems (e.g., many embedded systems). However, they have scalability problems related to the management and validation of large sets of possibly interacting rules when context and variability spaces grow. To cope with the scalability issue, optimization based approaches have been proposed (e.g., [3][7]). These approaches do not explicitly capture the adaptation rules, instead they use utility functions to capture high level goals such as for example "optimizing the performance". The utility is evaluated at runtime for all potential configurations to choose the optimal one. These more abstract adaptation policy expressions solve the scalability problem related to specifying

---

<sup>1</sup> To better qualify scalability here: *small-scale* implies that the complete set of possible configuration can be enumerated by the developer, *medium-scale* implies that the complete set can be processed by a computer, *large-scale* implies that the set of possible configuration is too large to be enumerated at all.

adaptation policies. However, the problem with these approaches is a costly runtime adaptation process since the system has to solve a complex optimization problem for every adaptation, and weaker support for early validation.

This deliverable reports on the progress made with respect to adaptation modelling in WP2. At this point, the project has gone through two main iterations for modelling adaptation. The first one focussed on modelling variability in the system and its environment and relied on simple event-condition-action rules described in [2]. This approach has been implemented and used to carry out a couple of academic case studies. Experience from these studies allowed validating the use of simple constructs for modelling the variability in the system and its environment but the simple ECA rules to link the variability with the context proved difficult to manage in the context of large systems, even while avoiding the need to refer to actual configurations by using aspect oriented techniques to encapsulate variability. In the second iteration we have overcome this limitation by replacing the ECA rules with a combination of constraints and optimization goals. The proposed approach intends to combine the strength of rule-based and optimization-based techniques in order to offer a solution scalable to complex systems while providing abstraction, efficiency and early verification and validation capabilities. The idea of the approach is to combine local adaptation rules and property-based adaptation goals.

The approach provides a Domain Specific Modelling Language (DSML) for capturing context information, system variability, constraints and adaptation policies. The DSML allow for design-time model-checking and simulation of the adaptation models. Moreover, platform-specific adaptation logic can be generated from the models. The approach has been implemented in Eclipse and is integrated in a complete model-driven approach for DAS development. The scalability of the approach is evaluated on two industrial case studies. Initial results show that the proposed formalism is able to cope with large-scale adaptive systems.

This document is structured as follows: Section 2 shows relevance to other work packages in DiVA while Section 3 covers related work. Section 4 presents our conceptual model while Section 5 and 6 respectively describe the first and second iteration in the design of the adaptation DSML. Section 7 details the semantics of the language and shows how early validation can be performed through model-checking and simulation. Section 8 presents the results of four case studies. Section 9 presents concluding remarks while Section 10 covers future work.

## 2 Relevance to other work packages

Work package 2 has strong relationships to the other technical work packages since it provides the metamodels that integrates the different DiVA technologies:

- *WP1-WP2*: WP1 deals with the requirements and analysis phase. The final step of this phase is to transform its output into the format of the DSML created in WP2 (described in the following sections). See D1.2 for further details.
- *WP2-WP3*: WP3 provides the ART metamodel for runtime composition and reconfiguration. The work packages are connected through the aspect models, where variants are modelled as aspects and later realized into OSGi components. WP3 has provided the runtime structure for the aspect and component models, while WP2 will build tools on top of this to further integrate the tool chain. See D3.2 for further details.
- *WP2-WP4*: The adaptation policies specified in the DSML of WP2 is used for runtime reasoning in WP4. In addition, WP2 has specified the semantics of the DSML allowing simulation of instanced models, which is used by WP4 for testing purposes. See D4.2 for further details.

### 3 Related Work

There are several recent state of the art reviews in the area of adaptive application modelling and execution, e.g., [1][12][13]. [1] especially focuses on surveying adaptive system construction and execution approaches that are based on model-driven engineering techniques and aspect-oriented techniques, [12] focuses on surveying middleware based self adaptation approaches and related model based approaches supporting adaptive system design, [13] especially focuses on surveying adaptation in a distributed service oriented environment. While general approaches for adaptive system development and execution are contextually relevant for the work presented in this deliverable, we narrow the scope in this related work section and compare our work with existing techniques for expression of adaptation policies. This is appropriate since the presented DSML is not a complete environment for adaptive system construction and execution, instead our DSML could be an alternative for modelling the adaptation logic in these broader scoped approaches. In general there are two families of approaches that have been defined for capturing adaptation policies: i) approaches based on explicit event-condition-action (ECA) rules and ii) approaches based on the definition of utility functions to be optimized. The two-level formalism proposed in this deliverable combines the strengths of these two approaches with respect to efficiency, scalability and verification capabilities.

Most existing approaches are based on using an ECA type of rules to formalize adaptation policies [6][8][10]. For example, in [10] the adaptation rules are triggered by context events and express system reconfigurations. In [6] the rules use guards and the actions details how the reconfiguration should be performed at the platform level. The approach presented in [8] uses event-condition-action rules and has a specific focus on conflict resolution and negotiation between interacting adaptive systems. An overview of these techniques can be found in [9].

The main strengths of ECA approaches are twofold; i) the readability and elegance of each individual rules, and ii) the efficiency with which the rules can be processed. At runtime, rules are matched and applied to adapt the system configuration. On the other hand, the main limitations of these techniques are related to scalability and validation. Managing a large set of possibly interacting adaptation rules rapidly becomes difficult. Validation becomes a major issue: how to ensure that the set of rules will yield the best possible configuration for every possible context of the application.

To overcome the validation problem, [5] proposes to capture adaptation policies early in the development cycle using temporal logic. The proposed formalism is an extended version of linear temporal logic which includes adaptation specific operators. Formal validation and verification technique associated with this approach are detailed in [11]. In [4] the authors propose to represent the adaptation policy under the form of a state-transition system in which the states correspond to the system configurations and the transitions correspond to the adaptations between these configurations. This technique makes adaptation policies easy to understand but can only be applied to systems with a very limited number of configurations and possible adaptations. In [2] an equivalent state-transition model is derived through design-time simulation of condition-action rules. The state-transition model is used for validation and verification purposes. The approach is easier to use but still requires the enumeration of all possible configurations and adaptations of the system which limits its applicability for large systems.

The second family of techniques consists in viewing adaptation as an optimization problem. The adaptation policies are expressed as high-level goals to achieve and at runtime the configuration of the system is optimized with respect to these goals [3][7][8]. The proposed approach uses parameterization and compositional adaptation. Each component type describes the properties it needs and the properties it offers, while their implementations are responsible for describing how these properties are computed. Moreover, each component implementation has to describe

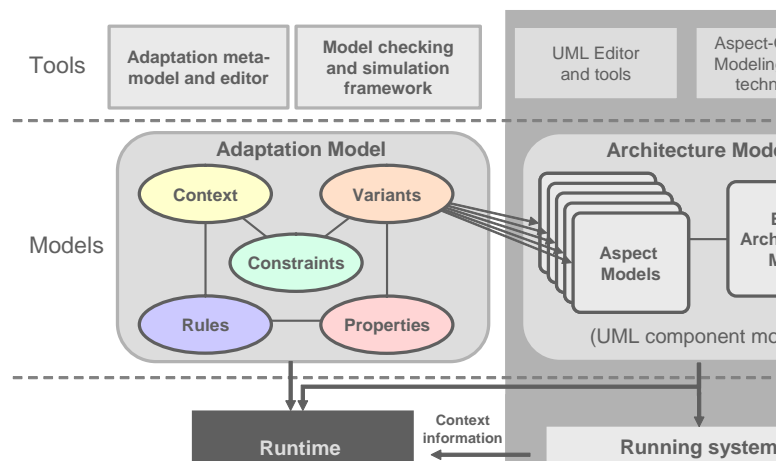
a utility function. These utility functions describe whether a given component implementation is useful in a particular context.

The main benefit of optimization-based approaches is the abstraction they provide through properties in order to allow to expressing much simpler adaptation rules. In addition, utility functions are an efficient way to determine how well suited a configuration is, depending on the context. However, specifying these functions may not be easy for designers and may require several iterations in order to adjust. Also, while the approach does not explicitly describe all the possible configurations of the system a priori, the runtime reasoning has to calculate utility values for all of them, thus encountering scalability and efficiency issues.

The approach proposed in this deliverable is a compromise between rule-based approaches like [2] and optimization-based approaches like [3] which enable for design-time validation techniques such as defined in [11]. This makes the proposed approach suitable for large-scale dynamic adaptive system by mastering the combinatorial explosion of the number of contexts and configurations.

### 4 Conceptual Model

Figure 1 presents the conceptual model of the approach. The conceptual model includes two main parts illustrated in the figure by the vertical division: the right hand side shows the architecture of the system and base technologies, the left hand side presents the adaptation layer managing the dynamic variability. The model is further divided into three horizontal layers. The mid (*Models*) layer includes the adaptation model which drives the specification, validation and execution of the adaptive system, the right hand side shows the architecture models of the system. The top layer presents the tools and base technologies that are used to build these models at design time. Finally the bottom layer presents how these models are processed at runtime (work of WP3).



**Figure 1** Overview of the proposed approach

For the architecture models, the idea of the proposed approach is to use existing technologies. The models are built in UML2 which allow using all the UML2 tools at design time and deploy the system on existing compatible middleware. To represent the variability in the architecture

model, the idea is to use Aspect-Oriented Modelling techniques: the optional functionalities are modelled as aspects which can be woven or not in the application [14][20].

The adaptation model capture the dynamic variability information, i.e., which functionality should be used depending on the context. To achieve that, the adaptation model needs to capture the variability in the system, the environment of the system and link them together. The objective of the proposed approach is to allow modelling adaptation and validate the adaptation logic early in the development cycle, thus, appropriate tool support is provided to model the adaptation and validate it at design time. At runtime, the model is processed by a runtime adaptation framework which is connected to the running application. The runtime adaptation framework receives context information as an input from the running application and triggers adaptations when required [14]. Details of the run time model weaver, the causal connection and our usage of models at run time, is presented in deliverable D3.2a from WP3.

The following sections detail two approaches which have been investigated in order to capture and validate the adaptation models. These approaches use the same representation for context elements and variability but differ by the way the adaptation rules are captured. The first one uses ECA rules while the second one uses a combination of hard-constraints and QoS goals.

## 5 DSML framework for adaptive systems: First iteration

This section presents the first iteration over the DiVA adaptation DSML and is illustrated on a simple Service Discovery Application (SDA). The SDA was specified and exploited in DiVA to describe the principles of the DSML approach before the DiVA case study specifications were ready. Furthermore, since the SDA is a very small and simple example it was suitable for the early development and experimentation of the approach. In Section 8 we provide results on our DSML approach using the DiVA CAS case study.

The SDA is a solution to tackle heterogeneity of service discovery protocols are presented in [15]. The solution allows an application to adapt to different service discovery protocols and needs during execution. The service discovery platform can take different roles that individual protocols could assume:

- User Agent (*UA*) to discover services on behalf of clients,
- Service Agent (*SA*) to advertise services, and,
- Directory Agent (*DA*) to support a service directory.

Depending on the required functionality, participating nodes might be required to support 1, 2, or the 3 roles at any time. A second variability dimension is the specific service discovery protocols to use, such as ALLIA, GSD, SSD, and SLP. Each service discovery protocol follows its own rules. As a result, in order to get two different agents understanding each other, they need to use the same protocol. These decisions have to be performed during execution.

The next sub-section presents an overview of the adaptation meta-model and the following sub-sections detail how it is instantiated for the SDA example.

### 5.1 Meta-model for variability and adaptation

As detailed in the previous section the adaptation model includes four different aspects: *variants*, *adaptation rules*, *dependencies* and *context*. Additionally, links to the architecture models and concepts for rules and expressions are supplied. The meta-model is shown in Figure 2. As can be seen from the figure, colours are used to differentiate between the categories. The colours indicate the following:

- Grey – base and aspect architecture models;
- Orange – variability information;
- Purple – adaptation rules;
- Red/pink – dependencies, formulated as constraints;
- Yellow – context information;
- Blue – expressions.

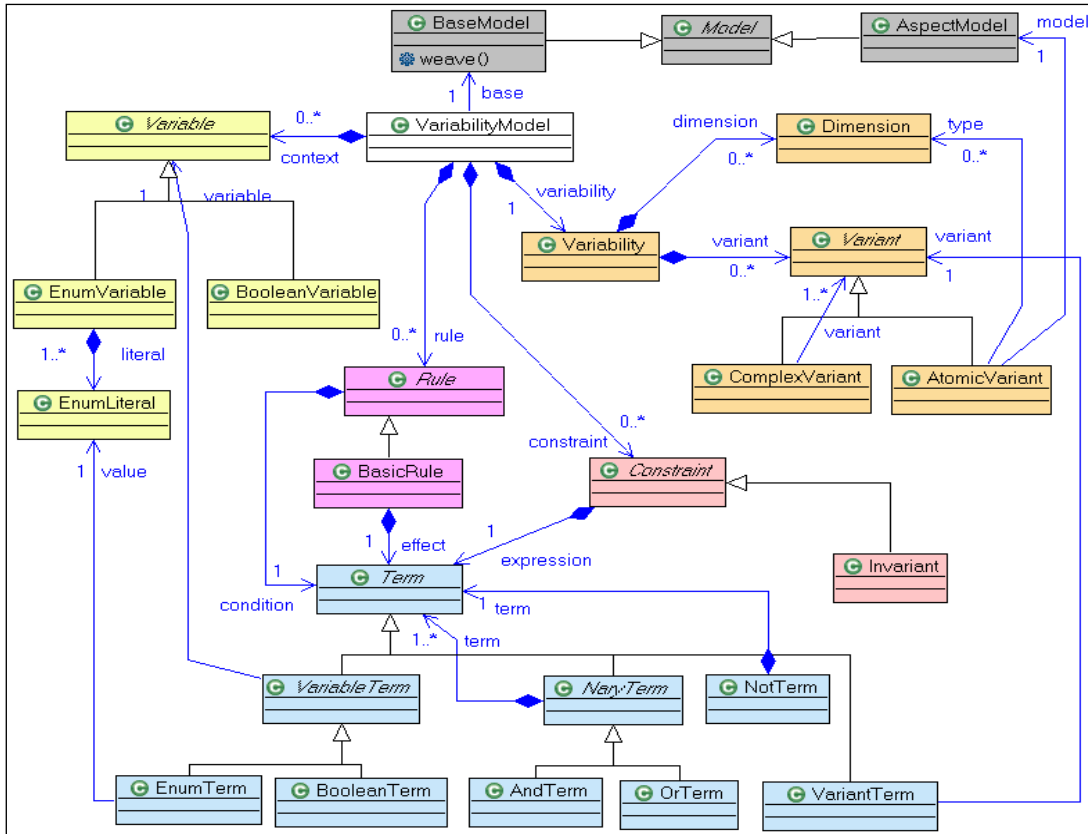


Figure 2 - Meta-model for variability and adaptation

The following shows how the meta-model is instantiated for the SDA. To make the example readable we use a textual concrete syntax. This concrete syntax is processed by our prototype tool in order to build the adaptation model.

### 5.1.1 Modelling variability

Figure 3 shows a model of the variability information in our service discovery example, located in the section identified by the `#variability` keyword. We start by defining two variability dimensions: one for functional variability and another for different discovery protocols that the application can use. A variability dimension can best be described as a category of variants, while a variant is an aspect or concern that is described outside of the base model and may vary to produce adaptation. So far, we have specialized variants into atomic variants and complex variants. The latter is used to express a collection of several variants, thus forming a partial or full configuration. This concept was added because we encountered in our example that some combinations of variants can be foreseen during the requirements phase. As an example, the Discovery Agent functionality corresponds to having both the User Agent and the Service Agent functionalities. DA is thus defined as a complex variant referring to UA and SA.

```
#variability /* Variability of the application */  
  
dimension Functionality : UA, SA  
variant DA : UA, SA  
  
dimension DiscoveryProtocol : ALLIA, SLP
```

Figure 3 - Variability in the Service Discovery Application

### 5.1.2 Modelling the context

Information about the context and sensors are delimited by the *#context* keyword. The meta-model supports two types of context variables: Booleans and enumerations.

The context model, as shown in Figure 4, starts with defining a variable for whether or not the device is running low on battery and, similarly, if the application has been elected as a Discovery Agent. Next, we have defined an enumeration that holds different roles. The application has to act as one of these roles at all time. Finally, there are two variables that tell which protocols are required, which can be one or many.

```
#context /* Context of the system */  
  
boolean LowBatt // Battery is low  
// Node has been elected Discovery Agent  
boolean ElectedDA  
  
// Node is required to act either as  
// User Agent or as Service Agent  
enum SrvReq : UA, SA  
  
// Node is require to use one or  
// more of the following prototcols  
boolean ALLIAReq  
boolean SLPReq
```

Figure 4 - Context of the Service Discovery Application

### 5.1.3 Modelling adaptation

Once the variability and context have been modelled, the adaptation rules can be specified. The adaptation rules link the context variables and the variants in order to specify the configuration to use with respect to a particular context. Currently, adaptation is based on simple *condition-action* rules. The *condition* part is a Boolean expression based on the context information, while the *action* is a change in the configuration of variants.

```
/* Adaptation rules for functionalities */  
  
rule BecomeDA : // Becomes a DA  
  condition ElectedDA and not LowBatt and not DA  
  effect DA  
  
rule StopDA : // Stop being a DA  
  condition (LowBatt or not ElectedDA) and DA  
  effect not DA  
  
rule BecomeUA : // Become a User Agent  
  condition SrvReq=UA and not UA  
  effect UA and not SA  
  
rule BecomeSA : // Become a Service Agent  
  condition SrvReq=SA and not SA  
  effect not UA and SA
```

**Figure 5 - Adaptation rules for the functionalities of the SDA**

Figure 5 depicts the adaptation rules for the variants in the functionality category. The first rule is called “*BecomeDA*”, which is triggered when an application is elected as a discovery agent. If the device also has sufficient batteries and it is not a discovery agent already, the adaptation will proceed and the application will assume the role of a discovery agent.

#### 5.1.4 Modelling constraints

Finally, Figure 6 shows the dependencies between variants. These are currently modelled as constraints, more specifically invariants. For example, the first invariant states that the application must use at least one functionality variant. If it does not, an error message will be produced by the tool.

```
invariant AtLeastOneFunctionality : UA or SA  
invariant NotDAWithLowBatt : not (LowBatt and DA)  
invariant AtLeastOneProtocol : ALLIA or SLP  
invariant NoSLPWithLowBatt : not (SLP and LowBatt)
```

**Figure 6 - Invariants of the SDA**

## 5.2 Simulation and Validation

The main benefit of using a model to describe adaptation is that it enables to process this model at design-time in order to validate it [16]. This section defines the semantics of the DSML meta-model defined in the previous section how it has been implemented for design-time simulation and the automated verification of invariants.

### 5.2.1 Simulation Model and Implementation

The goal of the simulation is to build a model of the valid potential configurations and adaptations of the application. To do that, the simulation starts from an initial configuration and applies the adaptation rules to move to a new configuration. Figure 7 presents the simulation model. According to this model, a simulation is composed of a set of configurations and a set of adaptations between these configurations. Each configuration refers to a set of variants and a set of variable terms. The variants correspond to the aspect to be woven in order to build this configuration. The Variable terms define the state of the context variables for this configuration. An adaptation links a source configuration with a target configuration. An adaptation is triggered by a context event and refers to one or more adaptation rules. The context event is a change in the values of one or more context variables.

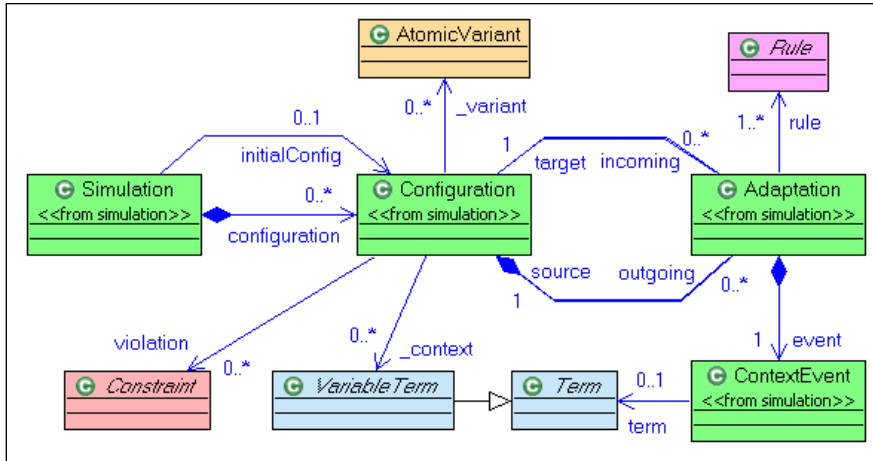


Figure 7 - Simulation model

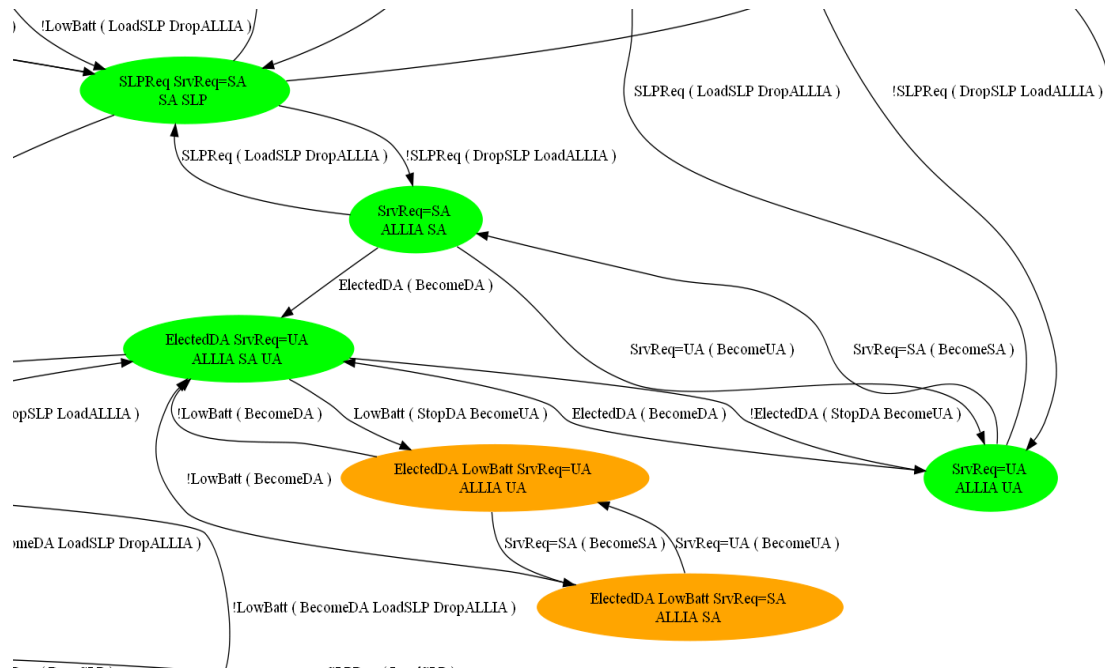
Based on this simulation model, a prototype simulator has been implemented using the Kermet platform [17]. The simulator starts from an initial configuration and for each variation of the context variables it evaluates the guards of the adaptation rules. If the guard of an adaptation rule is true in the new context then this rule must be applied and the guards of all the rules are evaluated again. Adaptation rules are applied until none of their guards evaluates to true.

### 5.2.2 Simulation output

The output of a simulation can be rendered as a graph in which each node is a configuration and each edge is an adaptation. Figure 8 shows an excerpt of the simulation graph for the service discovery application. The complete simulation graph for this example contains 24 configurations obtained by aspect weaving and 70 adaptations. In the label of each node, the first line corresponds to the values of the context variables and the second line to the set of aspects that should be used to create the corresponding configuration. Each edge in the graph corresponds to an adaptation to a change of one context variable. The label of the edges starts with the context variable change and details the set of adaptation rules that were applied. In the graph presented in Figure 8 the configurations have been coloured in order to easily visualize the battery level. Configurations for which the battery is high are displayed in green and configurations with low battery are displayed in orange.

### 5.2.3 Constraint checking and rule termination

The main benefit of the simulation model is to allow for validating the adaptation rules at design-time. As shown in the previous section the adaptation graph can be visualized. Colours can be used in order to highlight specific properties. This allows for a manual validation of the specified rules. In addition, the simulation process can identify live-locks and sink states in the adaptation graph and allow automatically verifying invariants on the system.



**Figure 8 - Excerpt of the simulation graph for the SDA**

Sink states in the simulation graph correspond to cases where some adaptation rules lead to a configuration from which the system cannot adapt anymore. In a design, this could be done voluntarily but in most cases this is due to some incorrect or missing adaptation rules. Live-locks correspond to cases where the system bounces between several configurations while the context is not changing. This situation always reveals an error in the adaptation rules. The simulator can identify live-locks while it computes the simulation graph. For a single change of the context, no adaptation rule should be able to apply twice. Indeed, if after applying a rule (and possibly some others), if the same rule can apply again then the rule could be applied an indefinite number of times. When this situation is detected by the simulator, it reports an error in the rules and provides the configuration in which the problem occurs and the sequence of rules which is looping.

The meta-model presented in Section 3 allows defining invariants on the system. These invariants are checked by the simulator on all the configurations that are created during the simulation. Any violation of these invariants reveals an error in the adaptation model.

### 5.3 Results and limitations

Experiments with the proposed DSML have shown that modelling the variability in the system with Dimensions and Variants and modelling the variability in the environment with Boolean and enumerated Variable is reasonably strait-forward. In addition, the event-condition-action model gives good results in terms of ease-of-use and performances for small to medium sized systems. However, we identified two specific issues when trying to apply this DSML to large-scale systems:

Firstly, in their current form, the number of adaptation rules can quickly grow as the number of context elements and variants increase. Our main goal is to tackle the issue of an explosive growth in the number of configurations and the artefacts to be used in their construction. However, we do not want to move the complexity associated into the rules as a consequence. Consequently, as a step towards improving our adaptation rules, we aim to express the *rules* using semantics. In that sense, the rule should be of the form “*choose a set of variants with properties that match the current context*”. The above embraces a more declarative approach. Although, sometimes we still might want to allow rules on variant configurations since pre-

defined full or partial configurations might be extracted or derived from the requirements straightforwardly, as was the case in our variability model.

Secondly, our current simulation prototype enumerates all the configurations and adaptations between them. While this is very useful and works well while the number of configurations is manageable, this approach has the typical model-checking scalability issues when the number of configuration and adaptation grows. Several techniques can be combined in order to keep the simulation space manageable, for example, adding constraints on the context, considering subsets of variability dimensions or using heuristics to limit the depth of simulations.

The next section presents the second iteration which introduces a combination of hard-constraints and optimization goals to tackle these issues.

## 6 DSML framework for adaptive systems: Second iteration

The role of the adaptation model from Figure 1 is to formalize how and when a system should adapt. The adaptation model thus has to capture the variability in the system, the variability in the context of the system and rules to link changes in the context of the system with the configuration to be used. We argue that the innovations provided by this new version of the DSML as presented in this Section is: i) better management of adaptive system complexities through model based abstractions and by providing expression of high level adaptation rules, ii) scalability by providing mechanisms for coping with the possible explosion of adaptive system artefacts such as configurations, variant dependencies and adaptation rules, and thereby, enabling efficient identification of the optimal configuration for a particular context situation, iii) formal validation and simulation of the adaptation logic, and iv) ease of use through means of a model based specification language and supporting tools.

The following sub-sections present the DSML meta-model of the proposed formalism for adaptation modelling and show how this DSML is instantiated for modelling a *mapping robot example*. Again we are presenting the DSML using a simpler example than those provided in the DiVA case studies, both because we wanted to apply our ideas on a concrete example before the DiVA case studies were properly specified and because of its suitability in the experimentation and prototyping phase.

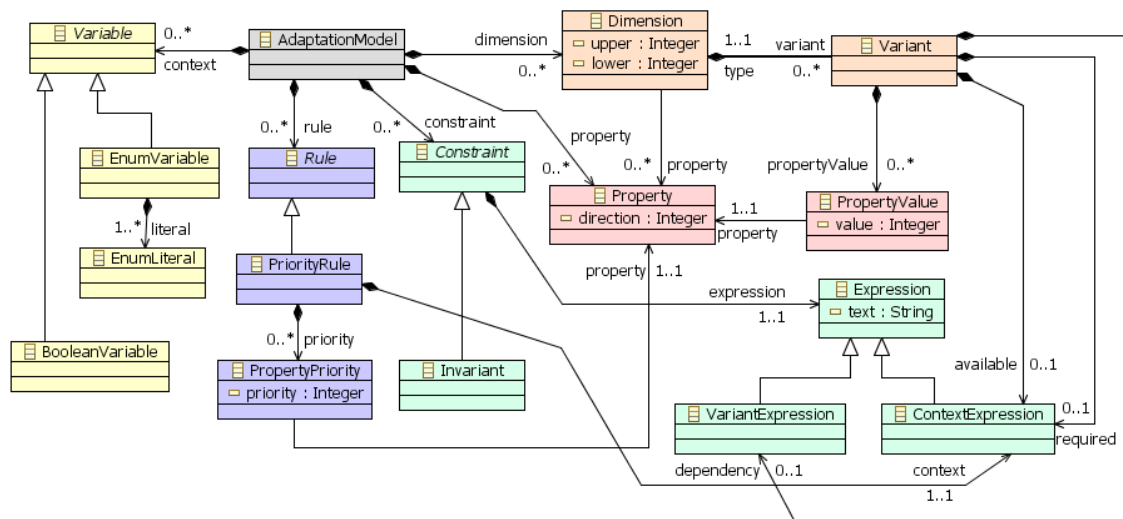


Figure 9 adaptation meta-model

## 6.1 Illustrative example: a semi-autonomous exploration robot

The mapping robot system is a semi-autonomous exploration robot which builds a map of an unknown environment when in motion. The robot is connected to a central system which collects the topographic data and can give directions to the robot. The robot has 3 main modes: i) idle, ii) going to a specific location or iii) exploring autonomously. It is equipped with three different sensors which can be used alternatively for routing and for drawing the map: i) *Camera*, which provides the most detailed map, ii) *Infrared sensors*, which can work without light sources and use limited resources, and iii) *Ultrasonic sensors*, which consumes limited resources while providing good routing capabilities.

While being drawn, the map is either stored locally in the robot's memory with periodic transmissions or directly streamed to a server. To allow for transmissions as well as for receiving commands the robot is equipped with Bluetooth and GPRS networking capabilities. The robot can employ three different routing strategies: i) *local routing strategy* that uses the sensors to navigate, ii) *map routing strategy* that uses pre-knowledge of the terrain, and iii) *external routing strategy* that involves interactions with a central computer or an operator. Furthermore, to build the map when moving, the robot can either use a simple or detailed map drawing strategy.

Depending on its environment, i.e., on its mode, on the terrain conditions or on the resources available, the robot has to dynamically adapt in order to optimize the map building and to use appropriate sensors and algorithms.

## 6.2 Adaptation meta-model

Figure 9 presents an overview of the proposed adaptation meta-model. The *AdaptationModel* is the root container, and an adaptation model instance contains five different elements which correspond to the five properties of class *AdaptationModel*:

- **Dimension/Variant:** A Dimension corresponds to a variation point in the application and is associated with one or more alternative variants which can be used for this variation point. Basically, a variant is a piece of functionality that can be included or not in the system. For example, a networking dimension can contain various alternative networking mechanisms such as Bluetooth, WIFI or GPRS. Any combination of these variants can be used at a particular point in time. In practice, each variant has a link to one or more aspects which describe generic functionality and how this piece of functionality can be instantiated and composed into a particular application.
- **Context Variable:** The total set of context variables describes the environment of concern of the application. Thus, the context variables that are relevant for the runtime adaptation should be modelled. The values of the context variables specify the current context at a particular point in time, and value changes may trigger an adaptation. Examples of context variables are network type, network bandwidth, noise and remaining battery
- **Constraint:** The Constraint concept is used for specifying particular dependency rules, such as a dependency between a context variable and a particular variant. Constraints are used both to describe dependencies among variants and dependencies between context variables and variants. Constraints are expressed using simple first order logic expressions. Constraints can for example express that a Bluetooth networking variant can only be used if some Bluetooth signal is available in the environment.
- **Property:** A set of properties of the system together with the impact of using each variant on these properties. For example: a typical property for embedded systems is the power consumption. If the level of power consumption is specified for each variant, the power consumption of alternative system's configurations can be compared so that configuration with lower power consumption can be chosen.

- **Property Rules:** A set of adaptation rules which link the context to the properties which should be optimized. These rules allow expressing high-level adaptation requirements such as for example “minimize power consumption when the battery level is low” or “optimize the system response time when a user is active”. These rules are expressed using expressions to describe in which context they apply and a set of priorities to associate to the properties of the system.

All expressions are first order logic expressions which can include variant terms and context terms. A variant term has a reference to a variant and is true if the variant has been selected in the current configuration of the system. A context term has a reference to a context variable that holds the value of this context element. The class *ContextExpression* corresponds to an expression which includes only context terms, the class *VariantExpression* correspond to an expression which contains only variant terms and the class *Expression* corresponds to an arbitrary expression. Examples of such expression are presented in the next section.

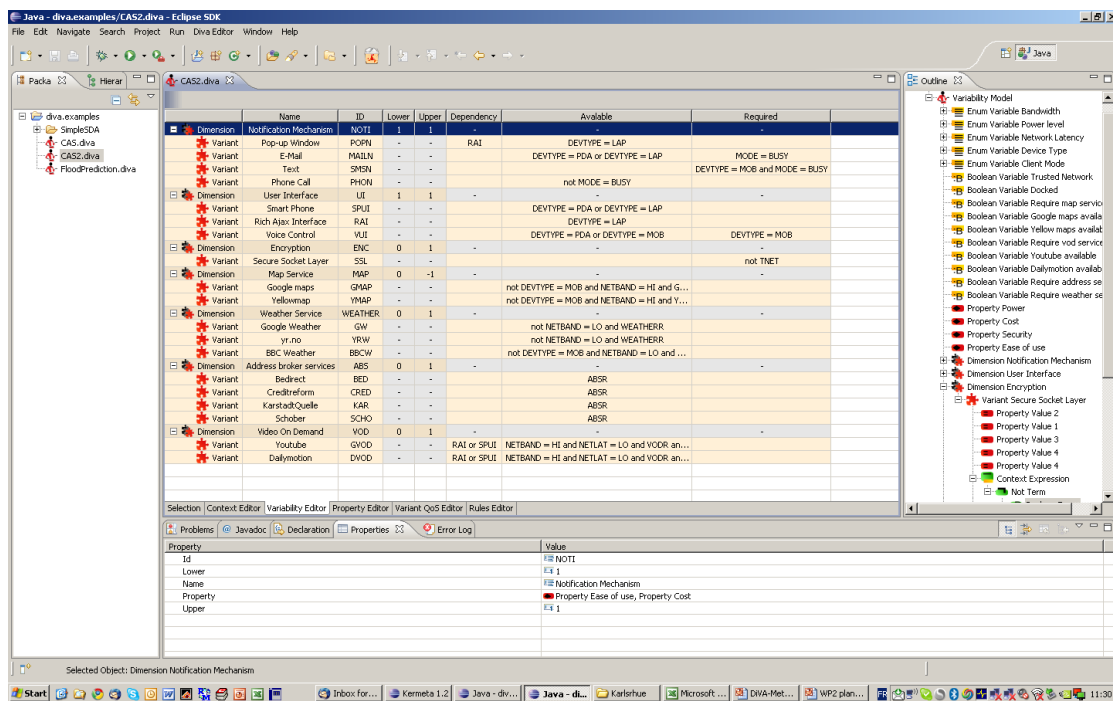


Figure 10 Screenshot of the adaptation model editor. The editor is composed of six tabs for editing context, variability, properties, impacts on properties and priority rules.

Conceptually, the adaptation meta-model can be divided in two: the variants, context variables and constraints on the one hand and the properties and rule on the other hand. The variant, context and constraints formalize the hard constraints on the adaptation. They narrow the search space for the optimal configuration for example by specifying dependencies or exclude possible configurations that are erroneous or do not make sense, however, in general they do not point to the optimal configuration. The properties and rules specify what properties should be optimized and then allow ranking the potential configuration in order to choose the best. Thus, the key idea to cope with the explosion of adaptation rules is to; first, specify hard constraints, and second, express implicit adaptation rules in terms of what properties of the system should be optimized (and not explicit rules specifying which exact variants should be used in a particular context). This allows easy reasoning for the designer who does not have to explicitly point at each and every variant to be used. These mechanisms also make the adaptation model easier to build compared to our earlier presented approach [2], while it is still efficient to process. In [2], the

variants and context information are linked by guard/action rules which express which variant should be used depending on context information and current configuration. This approach gave good results for small scale adaptive application since the adaptation rules explicitly points to the variants that should be used. A simple evaluation of the rules provides the optimal configuration for a given context. However, this approach does not scale since the number of adaptation rules may explode (due to the number exponential growth of the possible transitions between configurations), Moreover, writing a large set of guard/action rules is proven difficult and error-prone, since the interactions between a large set of rule is very difficult to master.

### Adaptation model editor

To specify adaptation models we have defined a concrete syntax and implemented an editor for it. The concrete syntax for the adaptation model is table based: the different elements of the adaptation model are presented in tables where the values of the attributes of the objects can be edited. Figure 10 presents a screenshot of the editor. The editor is implemented using the Eclipse Modelling Framework. The following section shows how the adaptation model of the mapping robot was built, it includes screenshots of the different tables of the editor.

## 6.3 Modelling the exploration robot

This section walks through the mapping robot example in order to detail the elements of the adaptation meta-model.

Figure 11 presents how the context of the mapping robot was modelled. Only the elements of the context which are relevant for runtime adaptation have to be modelled here. We have identified 5 such context variables. The robot has a mode which depends on what it has been instructed to do. The robot has 3 main modes: idle, exploration and going to a target zone. The two next variables “Light” and “Bluetooth Signal Available” corresponds to characteristics of the physical environment of the robot. The two last variables “Low Memory” and “Low Battery” corresponds to resources of the robot itself.

	Name	ID	Values
Enum	Mode	Mode	{IDLE, EXPLORE, GOTO}
Literal	IDLE	IDLE	-
Literal	EXPLORE	EXPLORE	-
Literal	GOTO	GOTO	-
Boolean	Light	LIGHT	-
Boolean	Bluetooth Signal Available	BTSig	-
Boolean	Low Memory	LowMem	-
Boolean	Low Battery	LowBatt	-

Figure 11 Model of the context of the robot

In general, the context variables can correspond to any stimuli of the system that should be taken into account for runtime adaptation. It can include user interactions, interaction with other systems or sub-systems as well as data coming from sensors. The adaptation meta-model supports Boolean and enumerated variables. These variables are defined at design time so the objective of these variables is to remain general and independent from future implementation design decisions. For example, in the case of the robot, the *LowMem* variable mean that the system is running out of memory but at this point the actual amount of memory the robot will have is not defined. At runtime, probes have to be implanted in the system to compute the actual values of these variables based on the implementation of the system.

Figure 12 presents the table containing the variation points of the robot and the alternative variants which can be used. Each variation point is modelled by a dimension. A dimension has a name and ID, a multiplicity and contains variants. The multiplicity of a dimension corresponds to the minimal and maximal number of variant which can be used simultaneously in a single configuration of the system. For example, the mapping robot has 3 alternative routing strategies and maximum one of these strategies (0..1) can be applied for a particular robot configuration. The variants represent the variability of the respective variation points/dimensions. Each variant include a name and ID, dependencies with other variants and constraints on when it can or should be used with respect to the context. Let's consider for example the "External Routing" routing strategy. This routing strategy involves asking a central computed for a route and then following instructions. To be able to use this functionality communication with the central computer needs to be possible either through a Bluetooth network or via GPRS. This constraint is modelled in the dependency column: The expression "GPRS or BT" expresses the fact that the variant can only be used together with the GPRS or BT variants. The Available and Required expressions correspond to contexts in which the variant respectively can or must be used. For example, all the routing strategies should only be used when the robot is in "GOTO" mode as it is the only mode which requires routing capabilities.


















	Name	ID	Lower	Upper	Dependency	Available	Required
[-] 	Dimension	Routing	RTG	0	1	-	-
	Variant	Local Routing	LR	-	-	Mode = GOTO	-
	Variant	Map Based Routing	MBR	-	-	not LowMem and Mode = GOTO	-
	Variant	External Routing	ER	-	-	GPRS or BT	Mode = GOTO
[-] 	Dimension	Networking	NET	0	1	-	-
	Variant	Bluetooth	BT	-	-	BTSig	-
	Variant	GPRS	GPRS	-	-	not BTSig	-
[-] 	Dimension	Sensors	SEN	0	1	-	-
	Variant	Camera	CS	-	-	LIGHT and not Mode = IDLE	-
	Variant	Infrared	IRS	-	-	not Mode = IDLE	-
	Variant	Ultrasonic	USS	-	-	not Mode = IDLE	-
[-] 	Dimension	Map Building Strategy	MBS	1	1	-	-
	Variant	Simple Map	SM	-	-	-	-
	Variant	Detailed Map	DM	-	-	BUFFER or BT	-
[-] 	Dimension	Data Transmission	DATA	1	1	-	-
	Variant	Streaming	STREAM	-	-	BT or GPRS	LowMem
	Variant	Buffering	BUFFER	-	-	not LowMem	-

Figure 12 Model of the variability and constraints in the robot

The proposed approach allows modelling three types of constraints: the dependency, availability and requirement of variants. This solution was chosen over having a global set of arbitrary constraints in order to impose for all constraints to be attached to a variant. This makes the constraints much easier to write and understand since they are local and specific.

At this point, the context variables, variants and constraints have been modelled. These elements formalize the variability, the relevant context information and hard constraints between context and variants. These constraints define all the valid combination of variants which can be used in a given context. However they do not specify what combination of variant should be chosen in a particular context. The idea of the proposed approach is to avoid having to specify the complete adaptation model with these low level constraints but to use rules expressed at a higher level of abstraction. This is the role of the properties, impacts and adaptation policies presented in the following. They complement the adaptation model with enough information for choosing the best configuration for each context.

	Name	ID	Direction
Property	Power Consumption	PWR	0
Property	Network usage	NETUSE	0
Property	Map Detail	MAP	1
Property	Routing accuracy	ROUTE	1
Property	Data Latency		0

Figure 13 Properties of the robot

Figure 13 presents the 5 properties that were defined for the mapping robot. These properties correspond to functional or extra-functional properties of the system which should be optimized while the system adapts. Each property has a name and ID and a direction. The direction specifies if the property value should be minimized (0) or maximized (1). This direction is fixed in the model for each property as we have not found any case where we would like to vary the direction dynamically. For the mapping robot the directions are that we want to minimize “Power Consumption”, “Network Usage” and “Data Latency” and we want to maximize the “Routing accuracy” and the “Map Detail”.

The properties are the abstraction that will be used to express the adaptation policy. Instead of linking the context directly to the variants to be used, in the proposed approach the context is linked to the properties which should be optimized in this context. To choose the appropriate variants, their impact on these properties has to be modelled. This is done using qualitative impact values.

	Power Consumption	Network usage	Map Detail	Routing accuracy	Data Latency
Routing (RTG)	true	false	false	true	false
Local Routing (LR)	Low	-	-	Medium	-
Map Based Routing (MBR)	High	-	-	Medium	-
External Routing (ER)	Medium	-	-	High	-
Networking (NET)	true	false	false	false	false
Bluetooth (BT)	High	-	-	-	-
GPRS (GPRS)	Medium	-	-	-	-
Sensors (SEN)	true	false	true	true	false
Camera (CS)	High	-	High	High	-
Infrared (IRS)	Low	-	Medium	Low	-
Ultrasonic (USS)	Low	-	Low	Medium	-
Map Building Strategy (MBS)	true	true	true	false	false
Simple Map (SM)	Low	Low	Low	-	-
Detailed Map (DM)	High	High	High	-	-
Data Transmission (DATA)	true	false	false	false	true
Streaming (STREAM)	High	-	-	-	Low
Buffering (BUFFER)	Low	-	-	-	High

Figure 14 Impact of the variants on the properties of the robot

Figure 14 shows the editor that supports the specification of the impact each variant has on the properties. The rows of this table correspond to the dimensions and variants defined earlier and the columns corresponds to the properties of the system. For each dimension the value “true” specifies that this dimension has an impact on the corresponding property. In this case, for each variant a qualitative appreciation of its impact on the property has to be specified. In the example of the mapping robot only the values Low, Medium and High have been used. If we consider for example the “Routing” dimension, the model specifies that the routing strategy impacts the power consumption and the routing accuracy. For each routing strategy variant

values for this impact is provided: The local routing has low power consumption but only a medium routing accuracy while the external routing has medium power consumption but a high accuracy. This table is the base of the adaptation optimization since it will be used to make different trade-offs depending on the context.








	Name	ID	Guard	Power Consumption	Network usage	Map Detail	Routing accuracy	Data Latency
 Rule	IDLE Mode	NM	Mode = IDLE	High	High	Low	Low	N/A
 Rule	Exploration mode	PM	Mode = EXPLORE	N/A	N/A	High	Low	Low
 Rule	Go to mode	FM	Mode = GOTO	N/A	N/A	Low	High	Medium
 Rule	Battery is Low	LB	LowBatt	High	N/A	N/A	N/A	N/A
 Rule	Battery is OK	LB	not LowBatt	Low	N/A	N/A	N/A	N/A
 Rule	BT available	BT	BTSig	N/A	Low	N/A	N/A	N/A
 Rule	no BT	GP	not BTSig	N/A	High	N/A	N/A	N/A

Figure 15 Optimization Goals of the robot

Finally, Figure 15 presents the optimization goals for the mapping robot. These goals are expressed by “Priority Rules” which capture what properties of the system matters depending on the context. For example rules 4 and 5 corresponds to the battery level. Rule “Battery is low” specifies that if the battery is low optimizing the power consumption of the robot has a high priority. Conversely, rule 5 specifies that when the battery is ok optimizing the power consumption is a secondary concern.

On this example the guard for every rule is a single context variable but in practice it does not have to be the case. Any arbitrary context expression can be used. If several rules match a given context simple strategies such as using the maximum value for each property are used to combine them.

The main characteristic of the proposed approach is that the rules express the adaptation policy in terms of the properties of the system to optimize and not directly in terms of the variants to use. This reduces significantly the number and complexity of the adaptation rules since the number of rules is correlated to the number of properties and not the number of variants.

The adaptation model for the mapping robot is now complete. For a particular context, the context variables, variants and constraints define a set of valid configurations. The properties, impacts and priority rules allow comparing these candidate configurations in order to choose the best suited. The following section details how this is achieved.

## 7 Simulation and validation of the adaptation model

The tables presented in the previous section present the complete adaptation model defined for the mapping robot. This section discusses the semantics of the adaptation model and describes the tools that were developed in order to simulate and verify it.

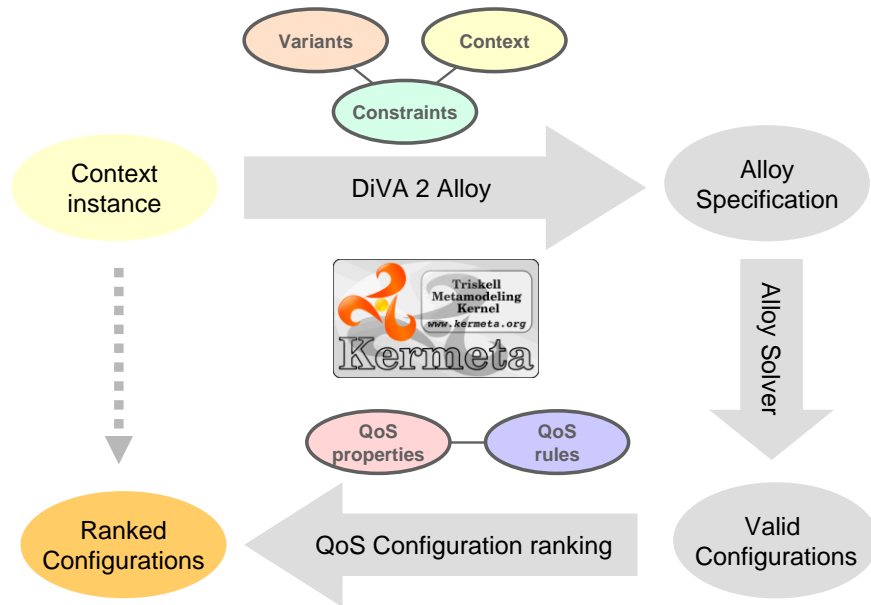
### Semantics and implementation of the adaptation model

Conceptually, the adaptation model is separated in two parts. On the one hand the context variables, the variants and the hard constraints and on the other hand the properties and priority rules. From a given context, processing the adaptation model has to yield the best suited configuration for the system in that context. In the proposed approach this is done in two steps:

1. The hard constraints are processed in order to enumerate candidate configurations for the system.

2. The priority rules are used to compute a score and rank the candidate configuration.

After the second step, the candidate configuration which has the best score is chosen and the system can be adapted.



**Figure 16** Implementation of the adaptation simulator

Figure 16 presents an overview of the implementation of the adaptation simulator. The simulator starts with a set of values for the context variables and outputs a set of ranked configuration which can be used in that context. The first step is to solve the constraints to find valid configurations. This is done by deriving an Alloy<sup>2</sup> specification from the adaptation meta-model and using constraint solving capability of the Alloy framework to output a set of valid configurations. The valid configurations can then be ranked according to their properties. The transformation to an Alloy specification and the computation of configuration scores are implemented within the Kermeta<sup>3</sup> environment. The ranking of configurations is done in four steps:

1. Compute the values of each property for each configuration. The value of a property  $p$  for a configuration  $C$  is computed by summing the contributions of the variants it contains. If we denote  $p(C)$  the value of property  $p$  for configuration  $C$ ,  $d_p$  the direction of property  $p$  and  $p(v)$  the impact of variant  $v$  on property  $p$ , then:

$$\forall C = \{v_1, \dots, v_n\}, p(C) = \sum_{i=1}^n (-1)^{d_p} p(v_i) \quad (1)$$

<sup>2</sup> Alloy is a structural modelling language based on first-order logic, for expressing complex structural constraints and behaviour. The Alloy Analyzer is a constraint solver that provides fully automatic simulation and checking. See <http://alloy.mit.edu/community/node/310>

<sup>3</sup> Kermeta is a metaprogramming environment based on an object-oriented DSL for metamodel engineering and is based on Eclipse. See <http://www.kermeta.org/>

2. Compute the priority  $w$  associated to each property  $p$ . This is done by evaluating the guards of all adaptation rules and combining the priorities provided by the rules  $R_{true}$  which guard is true. Let  $w(p)$  be the priority of property  $p$  and  $w(r, p)$  the priority of property  $p$  in rule  $r$ ,  $w(p)$  is:

$$w(p) = \max_{r \in R_{true}}(w(r, p)) \quad (2)$$

3. Compute a score  $S$  for each configuration  $C$ . This is done by summing the values of all properties for each configuration using weights corresponding to property priorities. If we denote  $S(C)$  the score of configuration  $C$ :

$$S(C) = \sum_p K^{w(p)} p(C) \quad (3)$$

Where  $K$  is a constant greater than 1. The constant  $K$  corresponds to a weight associated to priorities.  $K=5$  means that 5 contributions with a “Low” priority adds up to the same score as 1 contribution with a “Medium” priority. In our experiments  $K$  has been arbitrarily fixed to 5.

4. Rank the configuration according to their scores  $S$ .

The ranking process involves summing property values and priorities but in the model these elements are defined as qualitative values (such as “Low”, “Medium” and “High” in the robot example). In practice simple strategies are applied to transform these qualitative values to integers:

- For the impact of variants the values can be {N/A, Very Low, Low, Medium, High, Very High} which is mapped to {0, 1, 2, 3, 4, 5}.
- For priorities the values can be {N/A, Low, Medium and High} and they are mapped to {0, 1, 2, 3}

We have kept the computation of the score very simple and based on integer arithmetic since so far our experiments do not seem to require more advanced computation. In the literature more advanced mechanisms such as fuzzy-logic have been defined in order to handle qualitative values consistently. As future work we will investigate further if the proposed approach can benefit from such mechanisms.

### Simulation of the mapping robot adaptation model

The implementation of the meta-model allows simulating the adaptation model. Provided with a set of values for the context variables, the simulator outputs the ranking of valid configuration together with their scores. The interface of the simulator is currently text based.

```
(BTSig LIGHT LowMem Mode=EXPLORE)
BT SM STREAM      (SCORE = 36)
BT DM STREAM      (SCORE = 82)
BT SM STREAM USS  (SCORE = 87)
BT IRS SM STREAM  (SCORE = 111)
BT DM STREAM USS  (SCORE = 133)
BT CS SM STREAM   (SCORE = 136)
BT DM IRS STREAM  (SCORE = 157)
BT CS DM STREAM   (SCORE = 182)
-> | Bluetooth | Camera | Detailed Map | Streaming |
```

Figure 17 Simulation output for a single context.

Figure 17 shows the output of one simulation for the mapping robot. The first line correspond to the context of the system, it is the input of the simulation. The variables which do not appear (such as “Low Battery”) have the value false. For this simulation, the robot is in exploration mode, Bluetooth signal is available, the memory is low and there is light in the area. Based on the hard constraints of the adaptation model, only 8 configurations are valid in this context. The scores of these configuration range from 36 to 182. Based on these score, the best configuration to use according to the adaptation model is Bluetooth network, Camera sensor, detail mapping strategy and data streaming to the central computer. This intuitively corresponds to what we expected in such a context.

### Validation of the adaptation model

Like any specification or implementation task, the specification of the adaptation model can be error-prone. Before assuming that an adaptation model is correct, it needs to be properly validated. The benefit of modelling adaptation aside of the system using a dedicated language is that this model can be validated before it is integrated with the rest of the application. Two types of validation can be carried out:

- The verification of invariant properties
- The simulation of adaptation scenarios

The verification of invariant properties allows validating the constraints defined in the adaptation model. The modeller can express invariants using both context variables and variants and check that these constraints hold in all reachable configurations of the system. If a constraint does not hold the constraint solver can enumerate the configuration which violates the invariant.

For the mapping robot we might for example express that GPRS network should never be used when Bluetooth could be used:

*Invariant: not BTSig and GPRS*

To check such invariants, they are translated to the Alloy specification and just like for the simulation, the alloy constraint solver is applied and yields the potentially valid configurations which violate the invariant. In the case of the example no violation is found (which is quite trivial when looking at the availability constraints of the adaptation model).

- (BTSig LIGHT Mode=IDLE) → | **Buffering** | **Simple Map** |
- 1) Robot is switched to goto mode  
(BTSig LIGHT Mode=GOTO) → | **Bluetooth** | **Camera** | **External Routing** | **Simple Map** | **Streaming** |
  - 2) The robot is in the dark  
(BTSig Mode=GOTO) → | **Bluetooth** | **External Routing** | **Simple Map** | **Streaming** | **Ultrasonic** |
  - 3) Robot is switched to exploration mode  
(BTSig Mode=EXPLORE) → | **Buffering** | **Detailed Map** | **Infrared** |
  - 4) The internal available memory runs low  
(BTSig LowMem Mode=EXPLORE) → | **Bluetooth** | **Detailed Map** | **Infrared** | **Streaming** |
  - 5) The Bluetooth signal is lost  
(LowMem Mode=EXPLORE) → | **GPRS** | **Infrared** | **Simple Map** | **Streaming** |
  - 6) The robot gets to a light area  
(LIGHT LowMem Mode=EXPLORE) → | **Camera** | **GPRS** | **Simple Map** | **Streaming** |
  - 7) The Bluetooth signal comes back  
(BTSig LIGHT LowMem Mode=EXPLORE) → | **Bluetooth** | **Camera** | **Detailed Map** | **Streaming** |
  - 8) The robot has some free memory  
(BTSig LIGHT Mode=EXPLORE) → | **Buffering** | **Camera** | **Detailed Map** |
  - 9) The robot is running out of batteries  
(BTSig LIGHT LowBatt Mode=EXPLORE) → | **Buffering** | **Infrared** | **Simple Map** |
  - 10) Robot is switched to goto mode  
(BTSig LIGHT LowBatt Mode=GOTO) → | **Buffering** | **Local Routing** | **Simple Map** | **Ultrasonic** |
  - 11) Robot is back to IDLE mode  
(BTSig LIGHT LowBatt Mode=IDLE) → | **Buffering** | **Simple Map** |

Figure 18 Simulation output for a simple context evolution scenario

Checking for properties is a good way of validating the constraints present in the adaptation model. For the verification of the properties impact and adaptation rules, applying simulation on typical adaptation scenario is a complementary way of catching unexpected behaviours of the adaptation model. Because the total number of contexts for an adaptive application is huge (it grows exponentially with the number of context variables), in general it cannot be exhaustively simulated but it can be tested with representative context evolution scenarios. The way such representative scenarios can be chosen is out of the scope of the deliverable and part of our ongoing research based on software testing techniques.

Figure 18 presents the simulation of an adaptation scenario for the mapping robot. Each step of the scenario corresponds to a change in the context of the system.

## 8 Case studies and initial results

The usability and scalability of the proposed approach has been evaluated on a set of academic examples (two of them presented above) and on two industrial scenarios in the context of the DiVA project. The academic examples include the mapping robot presented in this deliverable and a flood prediction system developed at Lancaster University. The industrial cases are an airport crisis management system (proposed by Thales) and a Customer Relationship Management (CRM) system (proposed by CAS Software AG). In this section we present the CRM case study

### 8.1 Customer Relationship Management (CRM) system

Fig. 1 presents part of the context model for the system. In total, 14 variables were defined for the D-CRM system. As shown on the figure, the variables capture various aspects of the context: resources availability (Power Level and Low Bandwidth), devices (Device), system modes (Calendar Mode), information related to user interactions (Map Service Requested) or service availability (Google maps available).

	Name	ID	Values	
+	Enum	Power level	POW	{LO, HI}
	Boolean	Low Bandwidth	LBW	-
+	Enum	Device	DEVTYPE	{MOB, PDA, LAP}
+	Enum	Calendar Mode	MODE	{OFF, BUSY, DRIV, ON, FREE}
	Boolean	Map service Requested	MAPR	-
	Boolean	Google maps available	GMAPA	-
	Boolean	Yahoo maps available	YMAPA	-

Fig. 1. Model of the context of D-CRM System

As an example, in the D-CRM system, the “*Low Bandwidth*” variable means that the system is running out of bandwidth. At this point the actual bandwidth the system will need and the low bandwidth threshold are not defined. At runtime, probes have to be implanted in the system to compute the actual values of these variables based on the implementation of the system.

Fig. 2 presents the table containing the variation points of the system and the alternative variants which can be used. Each variation point is modelled by a dimension. The system has 4 alternative notification strategies and one and only one of these strategies (multiplicity [1..1]) has to be used for a particular configuration of the system.

	Name	ID	Lower	Upper	dependency	
[-]	Dimension	Notification Mechanism	NOTI	1	1	-
	Variant	Pop-up Window	POPN	-	-	RAI and TCC
	Variant	E-Mail	MAILN	-	-	
	Variant	Text	SMSN	-	-	TCC
	Variant	Phone Call	PHON	-	-	VCC
[-]	Dimension	Communication Channel	CC	0	-1	-
	Variant	Text CC	TCC	-	-	
	Variant	Voice CC	VCC	-	-	
[-]	Dimension	Ranking	RK	1	1	-

Fig. 2. Model of the variability in the D-CRM System

Fig. 3 presents the set of hard adaptation constraints defined for the D-CRM System. As a reminder, the approach supports two different types of constraints. The Available and Required expressions correspond to contexts in which the variant respectively can or must be used. For example, the pop-up window notification mechanism is only available if the device used by the user is a laptop (DEVTYPE=LAP in the “available” column). The phone call notification mechanism is required if the user is driving (MODE=DRIV in the required column).

	Name	ID	available	required
[-]	Dimension	Notification Mechanism	NOTI	-
	Variant	Pop-up Window	POPN	DEVTYPE = LAP
	Variant	E-Mail	MAILN	DEVTYPE = PDA or DEVTYPE = LAP
	Variant	Text	SMSN	MODE = BUSY and not DEVTYPE=MOB
	Variant	Phone Call	PHON	(DEVTYPE = MOB and MODE = BUSY)
	Variant	Phone Call	PHON	not (MODE = BUSY)
	Variant	Phone Call	PHON	MODE=DRIV
[-]	Dimension	Communication Channel	CC	-
	Variant	Text CC	TCC	not MODE=DRIV
	Variant	Voice CC	VCC	not MODE=BUSY
[-]	Dimension	Ranking	RK	-

Fig. 3. Model of the adaptation constraints in the D-CRM System

At this point, the context variables, variants and constraints for the D-CRM system have been modelled. In practice, simulation has been used at this point to make sure that the constraints conform to the case study requirements. Even if the approach only relies on local constraints (attached to variants) experience shows that the use of simulation is required to check the coherence of the set of constraints. Only once we were satisfied with the hard-constraints we started modelling the Properties and adaptation goals for the D-CRM system.

	Name	ID	Direction
Property	Power Consumption	POW	0
Property	Cost	COST	0
Property	Usability	USE	1
Property	Performances	PERF	1
Property	Proximity Search	PROX	1
Property	Security	SECU	1
Property	Disturbance	DIST	0

Fig. 4. Properties of the D-CRM System

Fig. 4 presents the 7 properties that were defined for the D-CRM System. These properties correspond to functional or extra-functional properties of the system which should be optimized while the system adapts. Each property has a name and ID and a direction. The direction specifies if the property value should be minimized (0) or maximized (1). This direction is fixed in the model for each property as we have not found any case where we would like to vary the direction dynamically. For the D-CRM system the directions are that we want to minimize “Power Consumption”, “Cost” and “Disturbance” and we want to maximize the “Usability”, the “Performances”, the “Proximity Search” capabilities and the “Security”.

	Power Consumption	Cost	Usability	Performances	Proximity Search	Security	Disturbance
Notification Mechanism (NOTI)	false	true	true	false	false	false	true
Pop-up Window (POPN)	-	-	High	-	-	-	Medium
E-Mail (MAILN)	-	-	Medium	-	-	-	Low
Text (SMSN)	-	Low	Low	-	-	-	Low
Phone Call (PHON)	-	Medium	Low	-	-	-	High
Communication Channel (CC)	false	false	false	true	false	false	true
Text CC (TCC)	-	-	-	High	-	-	Low
Voice CC (VCC)	-	-	-	Medium	-	-	High
Ranking (RK)	false	false	false	true	false	false	true
Default (DRANK)	-	-	-	Low	-	-	Medium
Reduced (RRANK)	-	-	-	High	-	-	Low
Security (SEC)	true	false	false	true	false	true	false
Weak security (WSEC)	Low	-	-	High	-	Low	-
Strong security (SSEC)	Medium	-	-	Low	-	High	-

Fig. 5. Impact of the variants on the properties of the D-CRM System

Fig. 5 shows the editor that supports the specification of the impact each variant has on the properties. In the example of D-CRM only the values Low, Medium and High have been used. If we consider for example the “Notification Mechanism” dimension, the model specifies that the notification strategy impacts the cost, usability and disturbance of the system. For each notification mechanism variant values for this impact are provided: The pop-up notification has high usability but is more disturbing for the user than an email. This table is the base of the adaptation optimization since it will be used to make different trade-offs depending on the context.

Name	context	Power ...	Cost	Usability	Performances	Disturbance
On mobile	DEVTYPE = MOB	High	-	Low	-	-
On PDA	DEVTYPE = PDA	Medium	-	Medium	-	-
On Laptop	DEVTYPE = LAP	-	-	High	-	-
General	MODE = ON or MODE = BUSY or MODE = DRIV	-	Medium	-	High	-
Voice Preferred	MODE = DRIV	-	-	-	-	Low
Text Preferred	MODE = BUSY	-	-	-	-	High
Sparetime	MODE=FREE and (DEVTYPE = LAP or DEVTYPE = PDA)	-	High	-	Low	Low
Offline	MODE = OFF	-	-	-	-	High

Fig. 6. Adaptation rules of the D-CRM System

Finally, Fig. 6 presents the adaptation rules for the D-CRM System. These rules are “Priority Rules”: they capture what properties of the system matters depending on the context. For

example rules 5 and 6 corresponds to the specific situation in which the user is driving or busy in a meeting. In these two situations, the system should adjust to different levels of disturbance. The disturbance property of the system is related to the availability of the user. When driving, the user is available but the system should adopt a voice communication channel. On the other hand, when the user is in a meeting, the system should adapt a strategy which only uses text notification for important events. On this example the guard for most of the rules are single context variable but in practice it does not have to be the case. Any arbitrary context expression can be used. If several rules match a given context simple strategies such as using the maximum value for each property are used to combine them.

The adaptation model is now complete. In practice, simulation has been used intensively in order to check and incrementally adjust the adaptation model of the D-CRM system.

## 8.2 Case studies initial results

Figure 19 presents some characteristics of the adaptation models that have been modelled using the DSML environment presented in this deliverable for the four case studies. For each adaptation model we have counted the number of elements which have to be explicitly modelled and computed the number of actual contexts and configurations the system has to consider. The number of context implies all possible combinations of values for the context variables. The number of configurations corresponds to all valid combination of variants according to the multiplicities defined on the variability dimensions. The results show that comparing the academic examples and the industrial scenarios, there is an explosion of the number of possible contexts and configurations (e.g., 884,736 contexts and 1,474,560 configurations for the airport crisis management system). However, there is no explosion in terms of the size of the adaptation model, the factor is only between 2 or 3 for the number of context variables, variants and constraints. The differences in terms of the number of properties and rules are minor. More case studies are needed to draw any definitive conclusions, but the current results indicate that the proposed approach do scale to handle realistic sized industrial cases.

	# Variables	# Context	# Variants	# Config.	# Const.	# Prop.	# Rules
Mapping Robot	5	48	12	192	13	5	7
Flood Prediction	5	48	9	112	9	3	5
Airport Crisis	18	884736	27	1474560	33	8	8
CRM	15	98304	20	92160	25	4	7

Figure 19 Characteristics of the adaptation model for four case studies

The second element that needed to be validated is the ability of the simulator and model checking capability to scale properly. Early results indicate that acceptable simulation times can be achieved. For example, in the case of the CRM system, simulating the adaptation model for a particular context only takes a few seconds. This simulation includes the transformation to an Alloy specification, the resolution of constraints, the evaluation of properties priorities and the computation of configuration scores. Overall, the approach seems to be well suited for the applications which were considered.

## 9 Conclusions

In this deliverable we propose a modelling language and associated tools for capturing and validating runtime adaptation early in the development cycle. The proposed approach allows expressing high-level adaptation rules based on properties of the adaptive system. Simulation and model-checking capabilities have been implemented to allow for the validation of the adaptation model. The approach has been validated on several academic case studies and two preliminary implementations of the DiVA case studies.

The proposed approach has four main benefits.

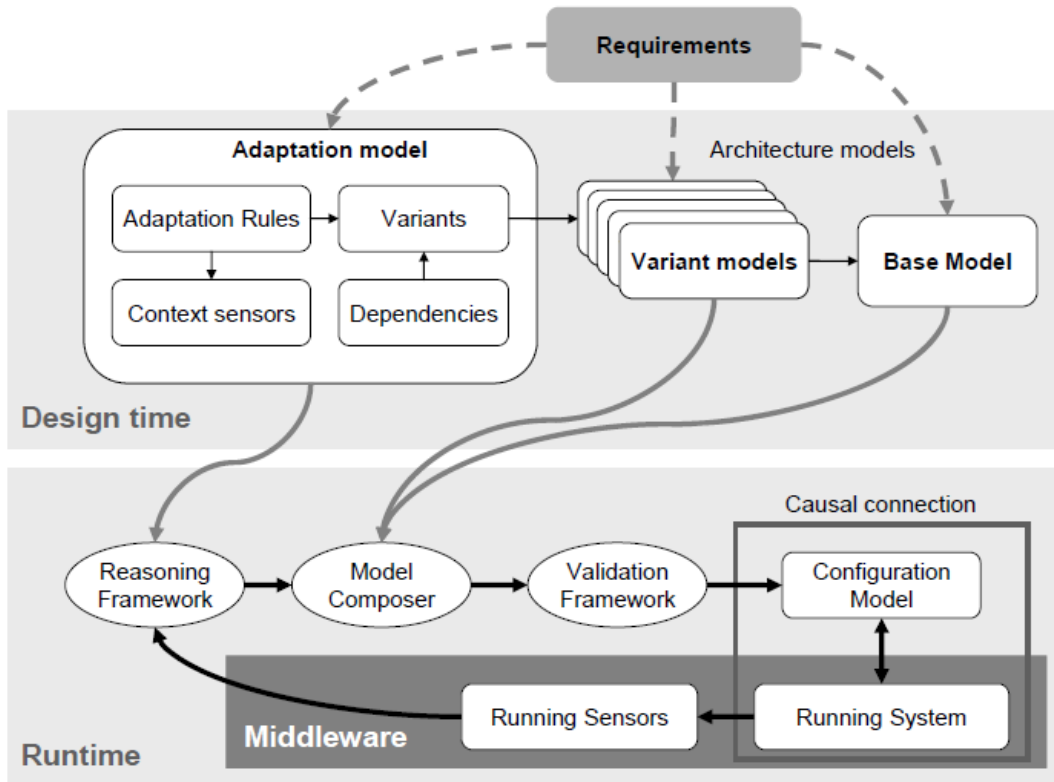
1. Firstly, it copes with the explosion of the number of contexts and configurations by using property-based policies. The case studies show that the number of elements in the adaptation model only grows linearly when the number of contexts and configuration grow exponentially.
2. Secondly, it allows for early verification and validation. The proposed approach allows statically simulating runtime adaptation at design-time in order to model-check properties on it or to test it on context evolution scenarios.
3. Thirdly, it permits the automated generation of the adaptation logic. To implement the adaptation the adaptation is processed directly by a generic runtime adaptation framework in order to drive the architecture adaptations in the running system [18][19].
4. And fourthly, it provides separate specification of the adaptation logic at the model level, abstracting complexity and avoiding adaptation logic and system logic tangling.

## 10 Future work

This deliverable presents our work on an adaptation meta-model that supports specification, simulation and execution of adaptive systems. Future work will be focused around evolution of the preliminary technology and on further integration with the other work packages.

Firstly, as to the preliminary work presented in this deliverable, based on additional studies future work will include refining how the variants and context variables are modelled. We will investigate the possibility of using well defined formalisms such as feature diagrams to better organize variants. For the context modelling, we will investigate the introduction of some structuring mechanism (such as classes for instance). Both these evolutions might have an impact on the adaptation meta-model but will not change the two-stage philosophy of the approach.

Secondly, since the different work packages of DiVA have delivered preliminary technology, WP2 will in the next phase of the project start the work on providing mappings and transformations to connect the different stages of the DiVA development process – all the way from requirements to runtime models. For the WP1-WP3 part this will entail work on transformations from WP1 models to the DSML defined in this deliverable. For the WP2-WP3 part it means working on the aspect models connecting the DSML with the runtime components. This will require close cooperation with work packages 1 and 3 in particular to ensure that DiVA is providing a holistic and consistent approach to developing adaptive systems. See Figure 20 for how the phases of requirements (WP1), design (WP2) and runtime (WP3 and WP4) are linked. The technical part of this work will be a part of the second deliverable on the transformation framework, i.e. D2.2, while the methodological part will be deliverable D2.3.



**Figure 20** Links between stages of development for a DiVA system

## References

- [1] Deliverable D3.1: "Survey and evaluation of approaches for runtime variability management", part of FP7 project DiVA, EU FP7 STREP, contract 215412, 2008.
- [2] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, J.M. Jézéquel. Modeling and Validating Dynamic Adaptation. In the Models@run.time at MODELS 2008, Toulouse, France.
- [3] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven (2006). Using architecture models for runtime adaptability. *Software IEEE* 23(2), 62–70.
- [4] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair (2008). Genie: Supporting the model driven development of reflective, component-based adaptive systems. In ICSE 2008 - Formal Research Demonstrations Track.
- [5] J. Zhang and B. H. C. Cheng. Model-based Development of Dynamically Adaptive Software. In proceedings of the ICSE '06 Conference, New York, NY, USA, pp. 371–380.
- [6] P. David and T. Ledoux (2006). Safe Dynamic Reconfigurations of Fractal Architectures with FScript. In Proceeding of Fractal CBSE Workshop, ECOOP'06, Nantes, France.
- [7] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. Using product line techniques to build adaptive systems. In proceedings of SPLC'06, Washington, DC, USA, pp. 141–150.
- [8] J. O. Kephart and R. Das, "Achieving Self-Management via Utility Functions", In *IEEE Internet Computing* 11(1):40–48. January/February 2007.

- [9] J. Keeney, V. Cahill, M. Haahr, "Techniques for Dynamic Adaptation of Mobile Services", in *The Handbook of Mobile Middleware*, Auerbach, ISBN: 0849338336.
- [10] J. Keeney and V. Cahill, "Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework", *Proceedings of Policy 2003*, Lake Como, Italy, 2003, pp. 3--14, jun, IEEE, ISBN 0-7695-1933-4, TCD-CS-2003-19
- [11] Zhang, J. and Cheng, B. H. 2005. Specifying adaptation semantics. In *Proceedings of the 2005 Workshop on Architecting Dependable Systems* (St. Louis, Missouri, May 17 - 17, 2005). WADS '05. ACM, New York, NY, 1-7.
- [12] Romain Rouvoy Deliverable D1.1 Requirements of mechanisms and planning algorithms for self-adaptation. MUSIC, FP6 Integrated project, Contract no 035166 , October 2007. <http://www.ist-music.eu/MUSIC/results/music-deliverables>.
- [13] ALIVE Deliverable D2.1, State of the Art. ALIVE project FP7 project number FP7-215890. <http://www.ist-alive.eu/>.
- [14] 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. *MODELS'08: 11th International Conference on Model Driven Engineering Languages and Systems*, France, 2008.
- [15] C.A. Flores-Cortés, G. Blair, and P. Grace. An Adaptive Middleware to Overcome Service Discovery Heterogeneity in Mobile Ad-hoc Environments. *IEEE Dist. Systems Online*, 2007.
- [16] Ji Zhang and Betty H.C. Cheng. Model-based development of dynamically adaptive software. In *International Conference on Software Engineering (ICSE'06)*, China, 2006.
- [17] Pierre A. Muller, Franck Fleurey, and Jean M. Jézéquel. Weaving executability into object-oriented meta-languages. In Lionel C. Briand and Clay Williams, editors, *Proceedings of the 8th International on MoDELS*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278, Montego Bay, Jamaica, October 2005. Springer.
- [18] Deliverable D3.2: "Reference architecture, first version", part of FP7 project DiVA, EU FP7 STREP, contract 215412, 2009.
- [19] B. Morin, O. Barais, J-M. Jézéquel, F. Fleurey and A. Solberg. [Models@run.time](#) to support dynamic adaptation. *IEEE Computer special issue on models@run.time* , vol. 42, no. 10, October 2009.
- [20] B.Morin, O. Barais, G. Nain and J-M. Jézéquel. Taming Dynamically Adaptive Systems using models and aspects. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering (ICSE)*, Pages 122-132, 2009.