A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems*

Franck Fleurey and Arnor Solberg

SINTEF, Oslo, Norway
{Franck.Fleurey,Arnor.Solberg}@sintef.no

Abstract. 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 scalable solutions to cope with the possible combinatorial explosions of adaptive system artifacts such as configurations, variant dependencies and adaptation rules. These are all challenges where current approaches offer only partial solutions. Furthermore, in current approaches the adaptation logic is typically specified at the code level, tightly coupled with the main system functionality, making it hard to control and maintain. This paper presents a domain specific modeling language (DSML) allowing specification of the adaptation logic at the model level, and separation of the adaptation logic from the main system functionality. It supports model-checking and design-time simulation for early validation of adaptation policies. The model level specifications are used to generate the adaptation logic. The DSML also provides indirection mechanisms to cope with combinatorial explosions of adaptive system artifacts. The proposed approach has been implemented and validated through case studies.

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 artifacts. 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

* This work was partially funded by the DiVA project (EU FP7 STREP, contract 215412). See http://www.ict-diva.eu/

A. Schürr and B. Selic (Eds.): MODELS 2009, LNCS 5795, pp. 606–621, 2009.

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 proper modeling and validation techniques all along the development cycle are needed.

Current approaches 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. 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 modeling, and also to make use of models at runtime to drive and monitor runtime adaptation [3][4][7][10][11]. Two main families of formalisms have been proposed in order to capture adaptation policies. The most common one is based on event-guard-action rules relating environment events to reconfiguration actions (e.g., [7][11]). 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[1] context-aware systems (e.g., many embedded systems). However, they have scalability problems related to the management and validation of large sets of rules when context and variability spaces grow. To cope with the scalability issue, optimization based approaches have been proposed (e.g., [4][10]). 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 possible configurations to choose the optimal one. These more abstract adaptation policy expressions solve the scalability problem related to specifying 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.

The contribution of this paper is an approach that combines the strength of rule-based and optimization-based techniques in order to offer a solution scalable to highly-adaptive 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 Modeling Language 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 form the models. The approach has been implemented in Eclipse and is

[1] 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.

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.

The paper is structured as follows: Section 2 presents an illustrative example. Section 3 presents the abstract syntax of the DSML and its application using the illustrative example. Section 4 details the semantics of the language and shows how early validation can be performed through model-checking and simulation. Section 5 briefly describes the implementation and usage of the proposed approach. Section 6 presents the results of four case studies. Section 7 compares the proposed formalism with existing techniques. And finally, Section 8 presents concluding remarks and future work.

2 Illustrative Example: A Semi-autonomous Exploration Robot

To illustrate our approach we use a simple example of a mapping robot. The 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.

3 Modeling Dynamic Variability and Adaptation

The role of the adaptation model 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. In the following we first present the abstract syntax of the proposed formalism and then an actual adaptation model for the mapping robot are build using a corresponding concrete syntax. An Eclipse based editor is implemented for specifying the adaptation models.

3.1 Overview of the DSML Abstract Syntax

Fig. 1 presents the main concepts of the abstract syntax of the proposed DSML for adaptation modeling. The proposed languages can be logically divided in three parts: it provides i) simple mechanisms to model the variability in the DAS, ii) simple mechanisms to model the context of the DAS and iii) an innovative combination of hard constraints and property optimization policies to model adaptation. We have chosen simple mechanisms to model variability and context to provide ease of use. Moreover, these mechanisms have been sufficient for modeling the current set of case studies (see section 5).

The system variability is modeled using *Dimension*, *Variant* and *VariantConstraint*s. A dimension typically corresponds to a variation point in the system and the variants correspond to the alternatives for this variation point. The multiplicity on the dimension (*upper* and *lower* properties of class *Dimension*) specifies how many of the variants can be included in order to build a valid configuration. Dimensions and variants can be easily represented as a feature diagram. Arbitrary dependency constraints between variants belonging to different dimensions can be expressed by attaching dependency constrains to the variants. The application of such constraints is elaborated further when describing the modeling of the robot example in the following subsections.

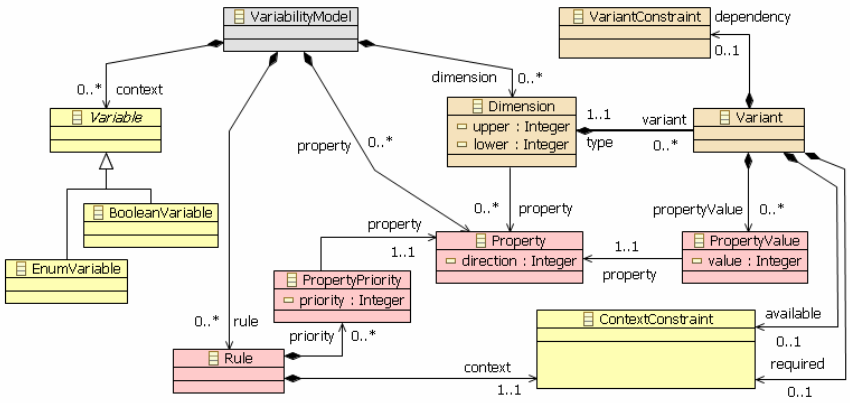

Fig. 1. Excerpt of the adaptation DSML abstract syntax

The context of the system is modeled using a set of variables (*Variable* and its subclasses on the diagram of Fig. 1). The objective of these context variables is to capture the elements of the environment which can implicate a system adaptation. To keep simplicity and remain at a high-level of abstraction, the proposed approach captures all the context information in either Boolean or Enumerated variables. If the configuration of a system depends on continuous value such as for example the amount of available memory, these will be abstracted into discrete ranges such as {LOW, MEDIUM, HIGH}. This kind of abstraction has the benefit of decoupling the adaptation model from the actual values and thresholds for a particular domain or application. Thus, the actual values might be unknown at design-time, and they may be adjusted at deployment time or even at runtime.

The most important part of the DSML is to support the specification of the adaptation logic, in essence, the relations between the context and the variability of the system. As discussed in the introduction, existing approaches are either based on event-guard-action rules or on global optimization of some utility function associated to the system configurations. The approach proposed in this paper is a combination of constraints and property optimization which intent to provide a scalable solution in order to handle large-scale adaptive systems, and at the same time enable early simulation and validation (detailed comparisons with existing approaches are presented in section 7).

To determine the adaptation logic the DSML enable firstly to specify constraints associated to variants. In Fig. 1 these constraints corresponds to the two compositions named *available* and *required* between *Variant* and *ContextConstraint*. These constraints are local to each variant and specify in which context the variant can (a*vailable*) or must (*required*) be used. The constraint can be any first order logic expression combining context variables. Concrete examples are provided with the robot example in the next subsection. In practice, variant constraints allow reducing the set of configuration suitable for a particular context. However, in general constraints cannot point to the specific configuration which should be used. The idea of the proposed approach is to enable specification of adequate local hard constraints in order to break the combinatorial explosion of the potential number of configurations to be considered, then, a general property-based set of adaptation rules is specified to come up with the best suitable configuration for the particular context.

Therefore, secondly, the DSML enable expressing property-based rules. This includes associating a set of properties of concern for the adaptive system (*Property* in Fig. 1). These properties of concern are qualities of the system which should be optimized through the runtime adaptation, for example, the performance or the power consumption. A direction is associated with each property to determine if the property value should be minimized or maximized. Typically, performance should be maximized while power consumption should be minimized. Once the properties are defined, the DSML facilitate specification of the impact implicated by each variant on these properties (*PropertyValue* in Fig. 1). For each variant, a qualitative impact value can be defined. In practice an integer is used to represent the impact value but this integer just encode qualitative values such as {0: no impact, 1: low, 2: medium, 3: high}. These impact values allow comparing the values of the properties for alternative configuration in order to choose the best suited one.

Finally, the DSML support specification of priority rules (*Rule* and *PropertyPriority* in Fig. 1). These rules allow linking the context with the particular set of properties which should be optimized. Each rule is related to a context to determine in which context the rule applies and it specifies a set of property priorities for the particular context. Basically these rules allow specifying which properties should be optimized depending on the context. The main benefit of this approach is that the general adaptation policy is captured at a high-level of abstraction. An example of rule at this level might be “if the battery runs low, the power consumption should be prioritized over performances”. By computing the specified impact each variant has with respect to the properties, these rules can be used to evaluate alternative configurations and to choose the most suited.

The next sub-section details how the proposed adaptation language is applied to the robot example. Then section 4 comes back to the details of the semantics of the language and presents how adaptation can be simulated.

3.2 Step by Step Modeling of the Mapping Robot

This section walks through the modeling of the adaptation logic of the mapping robot example to show an application of the DSML and elaborate further on its elements. We have implemented an Eclipse based editor to support the modeling.

Fig. 2 presents how the context of the mapping robot was modeled using the Eclipse based modeling editor. We have identified 5 context variables for the system. The Mode variable captures what the robot has been instructed to do. The robot has 3 main modes: IDLE, EXPLORE and GOTO. The two next variables "Light" and "Bluetooth Signal Available" specifies characteristics of the physical environment of the robot. Finally, the two last variables "Low Memory" and "Low Battery" specifies 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	Blutooth Signal Available	BTSig	-
Boolean	Low Memory	LowMem	-
Boolean	Low Battery	LowBatt	-

Fig. 2. 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 includes user interactions, interaction with other systems or sub-systems as well as data coming from sensors. For example, in the case of the robot, the *LowMem* variable is used to determine whether the system is running out of memory. When modeling, the actual amount of memory the robot will have is not necessarily known. At runtime, probes have to be implanted in the system to compute the actual values of these variables and an actual threshold for *LowMem* need to be set.

	Name	ID	Lower	Upper	Dependency	Avalable	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	

Fig. 3. Model of the variability and constraints in the robot

Fig. 3 presents the variability of the mapping robot and the dependency and adaptation constraints. For example, the robot has 3 alternative routing strategies (see the *routing dimension* and its three *variants*). A maximum of one of these strategies (specified with the *lower* and *upper* multiplicity [0..1]) can be used for a particular configuration of the robot. The *External Routing* strategy involves requesting a central computer for a route, and then to follow the instructions. To be able to use *External Routing,* communication with the central computer is required either through a Bluetooth network or via GPRS. This hard constraint is modeled in the dependency column: The expression *GPRS or BT* expresses the fact that the variant can only be used together with the GPRS or Bluetooth variants. The *Available* and *Required* expressions correspond to contexts in which the variant respectively can or must be used. For example, it only makes sense to consider a routing strategy when the robot is in *GOTO* mode as it is the only mode which requires routing capabilities.

	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

Fig. 4. Properties of concern of the robot

At this point, the context variables, variants and adaptation constraints have been modeled. Next we model the properties of concern. Fig. 4 presents the 5 properties of concern identified for the mapping robot. These properties correspond to functional or extra-functional properties of the system which should be optimized through adaptations. Each property has a name and ID and a direction. The direction specifies if the property value should be minimized (0) or maximized (1). For the robot the directions specify that we want to minimize *Power Consumption*, *Network Usage* and *Data Latency* and we want to maximize the *Routing accuracy* and the *Map Detail*.

	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

Fig. 5. Impact of the variants on the properties of the robot

Fig. 5 shows the specification of the impact each variant has on the properties of concern. 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 are 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 to make different trade-offs and to find the optimal configuration for the actual context.

Finally, Fig. 6 presents the adaptation rules specified for the mapping robot. These rules are *Priority Rules*: they 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 for the mapping Robot.

	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

Fig. 6. Adaptation rules of the robot

In this example the guard for every rule is a single context variable, however, the DSML allow arbitrary context expressions. If several rules match a given context simple strategies such as using the maximum value for each property are used to combine them.

4 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.

4.1 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

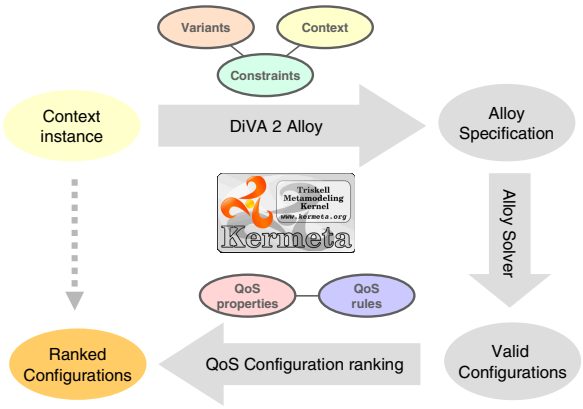

Fig. 7. Implementation of the adaptation simulator

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.

To do early testing and validation an adaptation simulator is implemented. Fig. 7 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 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 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:

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

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:

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

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

$$S(C) = \sum_{p} K^{w(p)} p(C) \tag{3}$$

Where K is a constant greater than 1 (fixed to 5 in our experiments). The constant K corresponds to a weigh 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.

4. Rank the configuration according to their scores S.

The ranking process involves summing property values and priorities, however, in the model these elements are defined as qualitative values (such as "Low", "Medium" and "High" in the robot example). For the simulation, 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 advance 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.

4.2 Simulation of the Mapping Robot Adaptation Model

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

Fig. 8 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 includes the following variants: Bluetooth network,

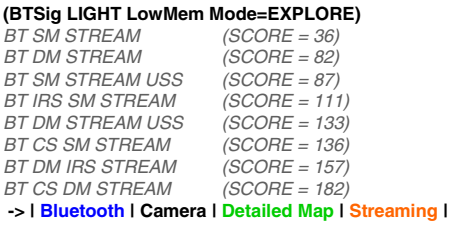

Fig. 8. Simulation output for a single context

Camera sensor, detail mapping strategy and data streaming to the central computer. This intuitively corresponds to what we expected in such a context.

4.3 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. Besides the fact that it provide separation of concern, the benefit of modeling the adaptation logic separately from the main system functionality is that this model can be validated before it is integrated with the rest of the application. Two types of validations can be carried out: the verification of invariant properties and the simulation of adaptation scenarios.

The verification of invariant properties allows validating the constraints defined in the adaptation model. The modeler 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).

Checking for properties is a good way of validating the constraints present in the adaptation model. For the verification of the impact with respect to the properties of concern and the adaptation rules, applying simulation on typical adaptation scenario is a complementary way of catching unexpected behaviors 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 complete simulations taking all context into consideration cannot be performed. However, the adaptation can be tested with representative context evolution scenarios. The way such representative scenarios can be chosen is out of the scope of the paper and part of our ongoing research based on software testing techniques.

Fig. 9 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.

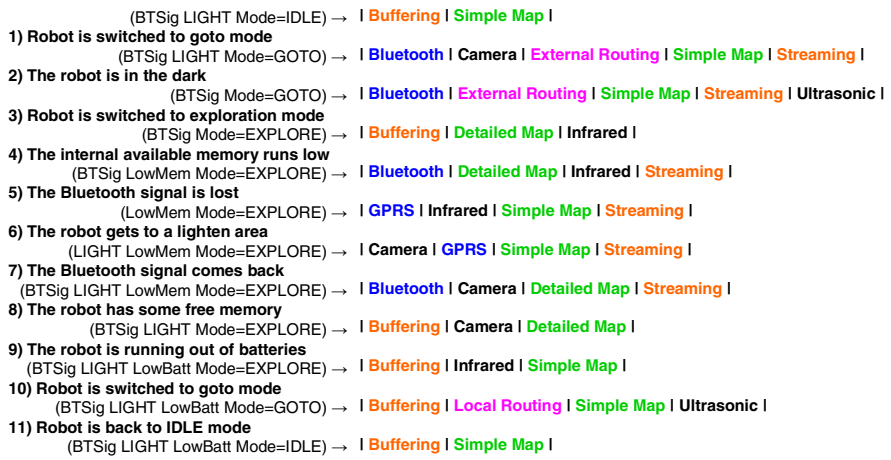

Fig. 9. Simulation output for a simple context evolution scenario

5 Case Studies and Initial Results

A complete environment for the presented DSML to support both modeling and validation has been developed. This environment includes an editor, a simulator and validation tools. The environment has been built using the Eclipse-Modeling Framework (EMF) and the Kermeta platform for semantics and simulation support. The editor is a table-based editor which allows editing all the aspects of the adaptation model (see the screenshot figures in Section 3). The tools are developed as open-source and are available from the DiVA project web page http://www.ict-diva.eu/.

In the context of the DiVA project, the adaptation DSML presented in this paper is introduced as the core of a platform for development and execution of adaptive systems. At runtime, the DiVA approach relies on AOM dynamic weaving techniques to adapt the running system. A detailed description of how MDE and AOM techniques are combined for that matter can be found in [2].

The usability and scalability of the proposed approach has been evaluated on a set of academic examples and on two industrial scenarios in the context of the DiVA project. The academic examples include the mapping robot presented in this paper and a flood prediction system developed at Lancaster University. The industrial cases are an airport crisis management system and a Customer Relationship Management (CRM) system.

Fig. 10 presents some characteristics of the adaptation models that have been modeled using the DSML environment presented in this paper for the four case studies. For each adaptation model we have counted the number of elements which have to be modeled 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., 884736 contexts and 1474560 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 need 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
Maping 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

Fig. 10. 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. We are currently in the process of applying the approach to other domains such as a real-time video processing application and to even larger industrial scenarios. The initial results of these are promising, however, they are not yet fully completed.

6 Related Work

There are several recent state of the art reviews in the area of adaptive application modeling and execution, e.g., [1][16][17]. [1] focuses especially on surveying adaptive system construction and execution approaches that are based on model-driven engineering techniques and aspect-oriented techniques, [16] focuses on surveying middleware based self adaptation approaches and related model based approaches supporting adaptive system design, [17] focuses especially 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 paper, 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 modeling 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 paper has been built in an attempt to combine 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 [7][11][14]. For example, in [14] the adaptation rules are triggered by context events and express system reconfigurations. In [7] the rules use guards and the actions details how the reconfiguration should be performed at the platform level. The approach presented in [11] 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 [13].

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 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, [15] 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 [6]. In [5] the authors proposes 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 [3] 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 [4][10][11]. 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 paper is a compromise between rule-based approaches like [3] and optimization-based approaches like [4] which enable for design-time validation techniques such as defined in [6]. This makes the proposed approach a good trade-off for large-scale dynamic adaptive system by mastering the combinatorial explosion of the number of contexts and configurations.

7 Conclusion and Future Work

In this paper we proposed a modeling 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 industrial scenarios.

The proposed approach has four main benefits. Firstly, it copes with the explosion of the number of contexts and configuration by using property-based policies. The case studies show that the number of element in the adaptation model only grows linearly when the number of contexts and configuration grow exponentially. 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. 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. And fourthly, it provides separate specification of the adaptation logic at the model level, abstracting complexity and avoiding adaptation logic and system logic tangling.

Based on additional studies, future work will include refining how the variants and context variables are modeled. We will investigate the possibility of using well defined formalisms such as feature diagrams to better organize variants. For the context modeling, 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. We will also investigate alternative simulation semantics. For instance, the computation of configurations scores could rely on fuzzy-logic instead of integer arithmetic.

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] Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.-M., Solberg, A., Dehlen, V., Blair, G.: An aspect-oriented and model-driven approach for managing dynamic variability. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 782–796. Springer, Heidelberg (2008)

[3] Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jézéquel, J.M.: Modeling and Validating Dynamic Adaptation. In: The Models@run.time at MODELS 2008, Toulouse, France (2008)

[4] Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using architecture models for runtime adaptability. Software IEEE 23(2), 62–70 (2006)
[5] Bencomo, N., Grace, P., Flores, C., Hughes, D., Blair, G.: Genie: Supporting the model driven development of reflective, component-based adaptive systems. In: ICSE 2008 - Formal Research Demonstrations Track (2008)
[6] Zhang, J., Cheng, B.H.C.: Model-based Development of Dynamically Adaptive Software. In: Proceedings of the ICSE 2006 Conference, New York, NY, USA, pp. 371–380 (2006)
[7] David, P., Ledoux, T.: Safe Dynamic Reconfigurations of Fractal Architectures with FScript. In: Proceeding of Fractal CBSE Workshop, ECOOP 2006, Nantes, France (2006)
[8] Pessemier, N., Seinturier, L., Coupaye, T., Duchien, L.: A Safe Aspect-Oriented Programming Support for Component-Oriented Programming. In: WCOP 2006@ECOOP, vol. 2006–11 of Technical Report, Nantes, France. Karlsruhe University (2006)
[9] Soria, C.C., Pérez, J., Cars, J.A.: Dynamic Adaptation of Aspect-Oriented Components. In: Schmidt, H.W., Crnković, I., Heineman, G.T., Stafford, J.A. (eds.) CBSE 2007. LNCS, vol. 4608, pp. 49–65. Springer, Heidelberg (2007)
[10] Hallsteinsen, S., Stav, E., Solberg, A., Floch, J.: Using product line techniques to build adaptive systems. In: Proceedings of SPLC 2006, Washington, DC, USA, pp. 141–150 (2006)
[11] Kephart, J.O., Das, R.: Achieving Self-Management via Utility Functions. IEEE Internet Computing 11(1), 40–48 (2007)
[12] Capra, L., Emmerich, W., Mascolo, C.: CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. IEEE TSE 29(10), 929–945 (2003)
[13] Keeney, J., Cahill, V., Haahr, M.: Techniques for Dynamic Adaptation of Mobile Services. In: The Handbook of Mobile Middleware, Auerbach, ISBN: 0849338336
[14] Keeney, J., Cahill, V.: Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework. In: Proceedings of Policy 2003, Lake Como, Italy, pp. 3–14. IEEE, Los Alamitos (1933)
[15] Zhang, J., Cheng, B.H.: Specifying adaptation semantics. In: Proceedings of the, Workshop on Architecting Dependable Systems. WADS 2005, St. Louis, Missouri, May 17, pp. 1–7. ACM, New York (2005)
[16] Romain Rouvoy Deliverable D1.1 Requirements of mechanisms and planning algorithms for self-adaptation. MUISIC, FP6 Integrated project, Contract no 035166 (October 2007), http://www.ist-music.eu/MUSIC/results/music-deliverables
[17] ALIVE Deliverable D2.1State of the Art. ALIVE project FP7 project number FP7-215890, http://www.ist-alive.eu/