Using Models@Runtime for Designing Adaptive Robotics Software: an Experience Report[⋆]

Juan F. Inglés-Romero[1], Cristina Vicente-Chicote[1], Brice Morin[2], and Olivier Barais[2]

[1] Dept. of Information and Communication Technologies
Technical University of Cartagena
Edif. Antigones (ETSIT), 30202 Cartagena, Spain
`juanfran.ingles@upct.es, cristina.vicente@upct.es`

[2] INRIA, Centre Rennes - Bretagne Atlantique
and IRISA, University of Rennes 1
Campus de Beaulieu, 35042 Rennes Cedex, France
`Brice.Morin@inria.fr, barais@irisa.fr`

Abstract. Robotic systems are becoming increasingly complex, as their tasks and working environments become ever richer. As a result, there is an urgent need to provide robots with self-awareness and self-adaptation capabilities that allow them to autonomously deal, among other things, with software and hardware failures, changes in the environment, or interactions with other systems. The use of high-level models that can be adapted at run-time by the robot itself, promises to significantly boost the applicability and performance of robotic systems. This paper reports our experience in applying the DiVA model-driven adaptive approach to a robotics case study, describing its benefits and limitations for robotics.

1 Introduction

With increased flexibility and ease of use, robots are at the dawn of a new era, turning them into ubiquitous helpers to improve our quality of life by delivering efficient services in our homes, offices, and public places. In order to achieve such flexibility, the management of uncertainties will be a key component of success [17]. Enabling robots to manage the different sources of uncertainty they must deal with (e.g., changes in the environment, altered requirements, software and hardware failures, etc.) requires providing them with self-awareness and self-adaptation capabilities [2]. This implies enabling robots to build and dynamically adapt models of themselves and their environments.

The Strategic Research Agenda (SRA) [17], delivered one year ago by the European Robotics Technology Platform, defines *adaptation* as *a change to the process or the method of execution performed by the system itself, generally at*

[⋆] This work has been partially funded by the EXPLORE project (Spanish MICINN, TIN2009-08572,http://www.dsie.upct.es/proyectos/web_explore/) and the DiVA project (EU FP7 STREP, contract 215412, http://www.ict-diva.eu/)

runtime. Adaptation may involve cognitive decision making and can take place over both short and long timescales, affecting any level of the system. According to the SRA, future robots, and later groups of robots, will adapt their hardware and software to changes of the environment, work piece, and processes.

Among the mid- and long-term challenges related to adaptation (spanning dimensions such as control, learning, modeling, etc.), the SRA highlights the need for *more automatic (or semi-automatic) use of models for different purposes, including [...] adaptation and reconfiguration*. In this vein, the Model-Driven Engineering (MDE) paradigm promisses to bring great benefits to robotics [3].

In a context different from robotics, the DiVA Project[3] proposes to leverage models both at design-time and runtime (models@runtime) to support the dynamic adaptation of complex software systems. This paper reports our experience in applying the DiVA approach to a robotics case study, describing its benefits and limitations for robotics.

The rest of the paper is organized as follows: Section 2 surveys related work; Section 3 briefly introduces models@runtime in the context of the DiVA Project; Section 4 describes our experience in applying the DiVA model-driven adaptive approach to a robotics case study; Section 5 reports the lessons learned and open challenges; and, Section 6 concludes and presents some future research lines.

2 Related Work

Since the late 90s, great research efforts have been made in self-adaptive and autonomic software development. As a result, some interesting high-level reference models and frameworkshave been developed [14,9]. In addition, these efforts have also resulted in modern execution platforms, such as Fractal [7], OSGi[4] or SCA[5], which provide APIs for software introspection and reconfiguration. These platforms currently exhibit some limitations as, for instance, they do not allow to preview the effects of a reconfiguration until it is actually executed, or to simulate *what-if* scenarios in order to evaluate different possible configurations *a priori*. Moreover, in the case of complex adaptive systems, a large number of low level reconfiguration scripts (calls to the reconfiguration API) need to be manually coded, making the process cumbersome and error prone.

Putting the focus on the robotics domain, some interesting results have been achieved by the bio-inspired and cognitive system communities on low-level robot behavior adaptations based, e.g., on genetic algorithm mutations [10]. However, in order to deal with the increasingly growing complexity of real-word robotic systems and working environments, higher-level adaptation mechanisms need to be developed. In this vein, it becomes necessary to shift the focus from low-level self-adaptive algorithms to higher-level self-adaptive software components and component-based architectures [9]. Furthermore, the envisaged adoption of MDE by the robotics community promises not only to help raising the level of

[3] http://www.ict-diva.eu/
[4] http://www.osgi.org
[5] http://www.eclipse.org/stp/sca/

abstraction at which robotics systems are designed, but also enable their self-adaptation using models@runtime.

Although there exist plenty of robotics-specific software architeture styles and frameworks, commonly supported by platform-specific (and hardly interoperable) middlewares [15], most of them currently lack of support for model-driven robotics software development and self-adaptation [11]. Among the few model-driven tool-chains for robotics software development, it is worth highlighting Smartsoft [18] and V3CMM [6], both enabling component-based platform-independent design modelling and platform-specific code generation by means of model transformations. However, to date, neither Smartsoft nor V3CMM support runtime software adaptation (V3CMM only supports structural and behavioural variability modelling at desing-time). Conversely, the work presented in [8], addresses robotics software runtime adaptation at an architectural level although, having not adopted a MDE approach, it strongly dependes on the Prism-MW[6] specific middleware platform.

Finally, it is worth highlighting the very interesting initiative started by the BRICS project[7], funded by the 7^{th} EU Framework Program, where both MDE and robotic system adaptation (to achieve robust autonomy) play a key role, although these two goals do not appear explicitly related in the proposal.

3 An Overview of DiVA

The idea of "models@runtime" is to leverage models both at design-time and runtime to monitor, dynamically adapt or evolve software systems. A dedicated workshop[8] is held at MODELS since 2006.

In the context of the DiVA project [16,5], we leverage models@runtime to support the design and the execution of Dynamic Software Product Lines (DSPL) [12]. At design-time, we describe four facets of a DSPL, that are then leveraged at runtime to drive the dynamic adaptation process:

– **Variability**: describes the different features of the system, and their natures (options, alternatives, etc)
– **Environment/Context**: describes the relevant aspects of the context we want to monitor (environment), as well as the current context.
– **Reasoning**: describes when the system should adapt. It consists in defining which features (from the variability model) to select, depending on the current context, using the appropriate formalisms.
– **Architecture**: describes the configuration of the running system in terms of architectural concepts.

The role of these models is to formalize how and when a system should adapt. Thus, adaptation models capture the variability in the system and in its context, and link changes in the latter with configurations of the former.

[6] http://csse.usc.edu/ softarch/Prism/
[7] http://www.best-of-robotics.org/
[8] http://www.comp.lancs.ac.uk/~bencomo/MRT/

It is important to note that designers do not specify the whole possible set of architecture configurations in extension. Instead, each feature of the variability model is refined into an aspect model that can be easily woven into a base model (which contains components and bindings that should be present in all configurations). This way, the system is designed in intention, and configurations are explicitly built when needed. When a configuration is built (by aspect weaving) it is first validated by checking some invariants. Then if the configuration is valid, we rely on a model comparison (between the current configuration and the newly produced one) to infer a safe migration path that is actually executed to adapt the running system. This prevent designers from writing low-level and error-prone reconfiguration scrips. Interested readers are referred to [16,5] for more details about the use Aspect-Oriented Modeling in DiVA.

4 Applying Models@Runtime to a Robotics Case Study

4.1 Case Study Description

To illustrate how DiVA can address adaptation in robotics, we present a simple case study developed using its current version. This experience will allow us to later discuss the advantages and drawbacks of DiVA in the context of robotics.

The case study takes place in a room, containing a number of obstacles, where two commercial robots (e-pucks [9]) are initially placed at arbitrary positions. One of them plays the role of *Victim*, while the other plays the role of *Rescuer*.

The goal of the *Victim* is to help the *Rescuer* find it as soon as possible. To achieve this, it indicates its position using an acoustic or light signal. The *Victim* uses the Bluetooth to communicate both changes in its signaling policy and its current state, which can be: *i*) OK, *ii*) Wounded, or *iii*) KO. The *Victim* is equipped with infrared (IR) proximity sensors to detect close objects, which it can avoid adopting different strategies, namely:*i*) surround the obstacle, or *ii*) change the movement direction. The first action has a greater impact on energy consumption. Additionally, the *Victim* can adopt different strategies to improve its visibility, namely: *i*) run randomly, *ii*) walk randomly, or *iii*) stay still, in descending order in terms of visibility and resource consumption.

The goal of the *Rescuer* is to find the *Victim* in the shortest time possible with the available resources. It is equipped with three sensors for this purpose, each one having a different precision and consumption: *i*) camera, *ii*) microphones (which can identify the direction of sound), and *iii*) IR proximity sensors. It can receive Bluetooth communications from the *Victim*, allowing it to select the most appropriate sensor and strategy to find it. The *Rescuer* uses the same obstacle avoidance strategies as the *Victim*. Both robots are equipped with sensors to measure environmental light and noise, and their battery level.

Robots are expected to dynamically adapt their behaviour depending on their context (i.e., their role, battery level, light conditions, etc.) in order to achieve their goals using the most appropriate sensors and strategies.

[9] http://www.e-puck.org

4.2 Modeling Dynamic Variability and Adaptation

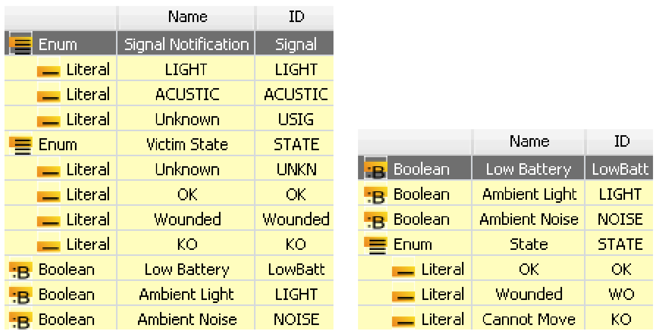
As described in Section 3, DiVA considers four facets of a DSPL: *Variability*, *Environment/Context*, *Reasoning*, and *Architecture*. Next, we present the adaptation models developed for the case study using the DiVA Eclipse-based editors.

Fig. 1 shows how the context of both robots is modeled. In both cases, three boolean contex variables capture the changes in the environmental light and noise, and in the robot battery. The *Rescuer* includes two additional context variables: *Signal Notification* and *Victim State*, which capture the changes in the *Victim*'s signaling policy and state. Note that, whenever the *Rescuer* has no information about the *Victim*, these variables are set to the `UNKNOWN` value. For the *Victim*, only one additional variable is modelled: *State*, which is set by one of the *Victim*'s internal components according, e.g., to the time it has been lost.

In robotics, we can think of three sources of contextual information having impact in robot software adaptation: *i*) the environment, *ii*) the robot internal state and resources, and *iii*) the perception of (and, eventually, the communication and collaboration with) other systems, either robotics or not.

Fig. 2 shows the variability model including the dependencies among the variants and the adaptation constraints. The variants represent different possible realizations of a variability dimension. For instance, there are three variants for the *Search Strategy* dimension, one for each of the strategies the *Rescuer* can use to find the *Victim*. As the caridinality of this dimension is set to [1..1], one (and only one) of the strategies needs to be selected for each particular configuration of the *Rescuer*. The *Detailed* strategy involves the highest search accuracy, and thus requires the camera and microphones variants (see Dependency column). The Available and Required expressions correspond to contexts in which the variant respectively can or must be used. For example, given the importance of energy consumption, it only makes sense to consider the *Detailed* strategy when the battery of the robot is not low.

The next step is to model the properties relevant for the system, i.e., the functional and extra-functional properties that need to be optimized. Each property has a name and a direction, the later specifying if it should be minimized (0)



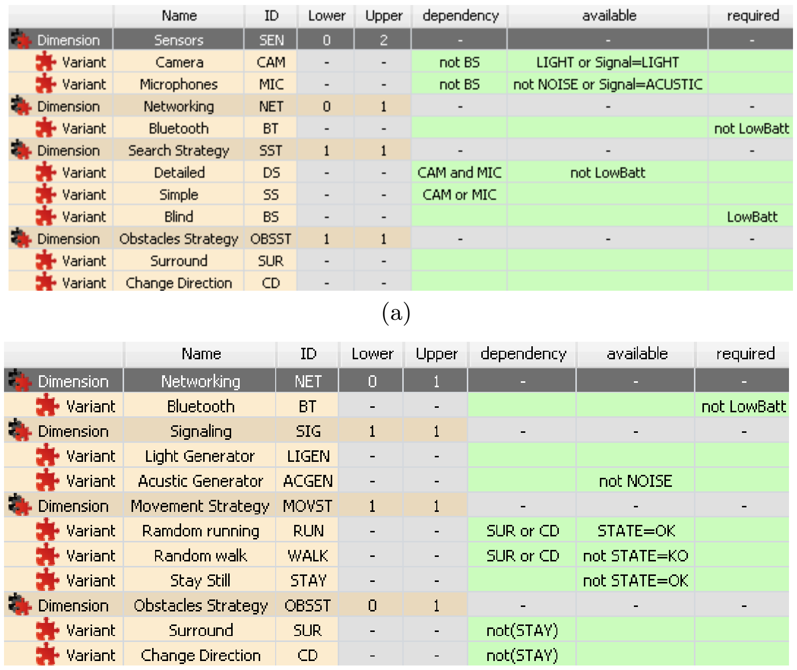
	Name	ID
Enum	Signal Notification	Signal
Literal	LIGHT	LIGHT
Literal	ACUSTIC	ACUSTIC
Literal	Unknown	USIG
Enum	Victim State	STATE
Literal	Unknown	UNKN
Literal	OK	OK
Literal	Wounded	Wounded
Literal	KO	KO
Boolean	Low Battery	LowBatt
Boolean	Ambient Light	LIGHT
Boolean	Ambient Noise	NOISE

(a)

	Name	ID
Boolean	Low Battery	LowBatt
Boolean	Ambient Light	LIGHT
Boolean	Ambient Noise	NOISE
Enum	State	STATE
Literal	OK	OK
Literal	Wounded	WO
Literal	Cannot Move	KO

(b)

Fig. 1. Context model for the robot playing the role of (a) *Rescuer* and (b) *Victim*.



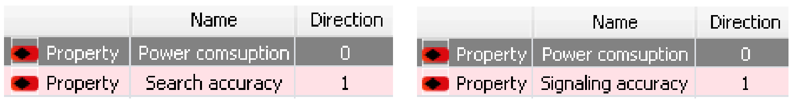
	Name	ID	Lower	Upper	dependency	available	required
Dimension	Sensors	SEN	0	2	-	-	-
Variant	Camera	CAM	-	-	not BS	LIGHT or Signal=LIGHT	
Variant	Microphones	MIC	-	-	not BS	not NOISE or Signal=ACUSTIC	
Dimension	Networking	NET	0	1	-	-	-
Variant	Bluetooth	BT	-	-			not LowBatt
Dimension	Search Strategy	SST	1	1	-	-	-
Variant	Detailed	DS	-	-	CAM and MIC	not LowBatt	
Variant	Simple	SS	-	-	CAM or MIC		
Variant	Blind	BS	-	-			LowBatt
Dimension	Obstacles Strategy	OBSST	1	1	-	-	-
Variant	Surround	SUR	-	-			
Variant	Change Direction	CD	-	-			

(a)

	Name	ID	Lower	Upper	dependency	available	required
Dimension	Networking	NET	0	1	-	-	-
Variant	Bluetooth	BT	-	-			not LowBatt
Dimension	Signaling	SIG	1	1	-	-	-
Variant	Light Generator	LIGEN	-	-			
Variant	Acustic Generator	ACGEN	-	-		not NOISE	
Dimension	Movement Strategy	MOVST	1	1	-	-	-
Variant	Ramdom running	RUN	-	-	SUR or CD	STATE=OK	
Variant	Random walk	WALK	-	-	SUR or CD	not STATE=KO	
Variant	Stay Still	STAY	-	-		not STATE=OK	
Dimension	Obstacles Strategy	OBSST	0	1	-	-	-
Variant	Surround	SUR	-	-	not(STAY)		
Variant	Change Direction	CD	-	-	not(STAY)		

(b)

Fig. 2. Model of the variability and constraints for (a) *Rescuer* and (b) *Victim*.



	Name	Direction
Property	Power comsuption	0
Property	Search accuracy	1

(a)

	Name	Direction
Property	Power comsuption	0
Property	Signaling accuracy	1

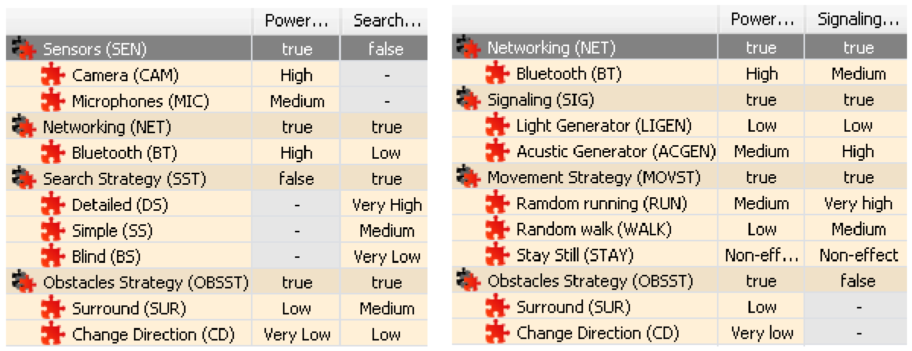
(b)

Fig. 3. Selected properties for (a) *Rescuer* and (b) *Victim*.

or maximized (1). As shown in Fig. 3, we have selected to minimize the power consumption, and maximize the search and signaling accuracy.

Fig. 4 shows the impact of variants (rows) on each property (columns). When a dimension has an impact on a certain property, for each of its variant a qualitative appreciation of this impact has to be specified. For example, the *Signaling* dimension affects both the power consumption and the signaling accuracy. In particular, the *Light Generator* has a low power consumption and signaling accuracy, while the *Acoustic Generator* has a medium power consumption and a high accuracy. This table is the base to make different trade-offs among the variants and to select the optimal configuration for the actual context.

Finally, Fig. 5 shows the robot adaptation rules. These are Priority Rules, i.e., they capture the relevant system properties depending on the context. For example, the rule *Battery is low* specifies that when the battery is low, optimiz-



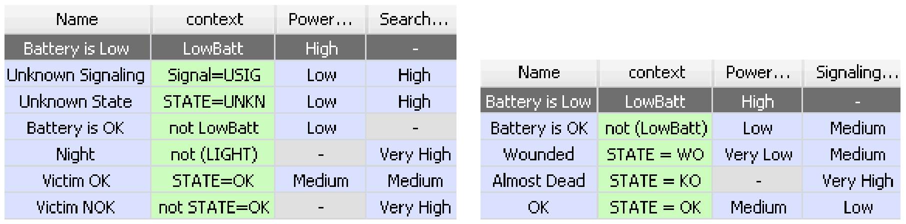
	Power…	Search…
Sensors (SEN)	true	false
Camera (CAM)	High	-
Microphones (MIC)	Medium	-
Networking (NET)	true	true
Bluetooth (BT)	High	Low
Search Strategy (SST)	false	true
Detailed (DS)	-	Very High
Simple (SS)	-	Medium
Blind (BS)	-	Very Low
Obstacles Strategy (OBSST)	true	true
Surround (SUR)	Low	Medium
Change Direction (CD)	Very Low	Low

(a)

	Power…	Signaling…
Networking (NET)	true	true
Bluetooth (BT)	High	Medium
Signaling (SIG)	true	true
Light Generator (LIGEN)	Low	Low
Acustic Generator (ACGEN)	Medium	High
Movement Strategy (MOVST)	true	true
Ramdom running (RUN)	Medium	Very high
Random walk (WALK)	Low	Medium
Stay Still (STAY)	Non-eff…	Non-effect
Obstacles Strategy (OBSST)	true	false
Surround (SUR)	Low	-
Change Direction (CD)	Very low	-

(b)

Fig. 4. Impact of variants on robot properties for (a) *Rescuer* and (b) *Victim*.



Name	context	Power…	Search…
Battery is Low	LowBatt	High	-
Unknown Signaling	Signal=USIG	Low	High
Unknown State	STATE=UNKN	Low	High
Battery is OK	not LowBatt	Low	-
Night	not (LIGHT)	-	Very High
Victim OK	STATE=OK	Medium	Medium
Victim NOK	not STATE=OK	-	Very High

(a)

Name	context	Power…	Signaling…
Battery is Low	LowBatt	High	-
Battery is OK	not (LowBatt)	Low	Medium
Wounded	STATE = WO	Very Low	Medium
Almost Dead	STATE = KO	-	Very High
OK	STATE = OK	Medium	Low

(b)

Fig. 5. Robot adaptation rules for (a) *Rescuer* and (b) *Victim*.

ing power consumption has a high priority. Conversely, the *Battery is ok* rule specifies that this is a secondary concern when the battery is OK.

It is worth noting that DiVA allows the simulation of the previous adaptation models provided the user inputs a sequence of contexts (set of values for the context variables). For each context, the simulator calculates and shows (it must be said that not in a very friendly and readable way) the best possible architecture configuration. This facility, has allowed us to perform multiple tests on the models developed as part this work, although we decided not include the screenshots showing the results for the sake of simplicity and for the space limitations.

4.3 Runtime Architecture to Support Dynamic Variability

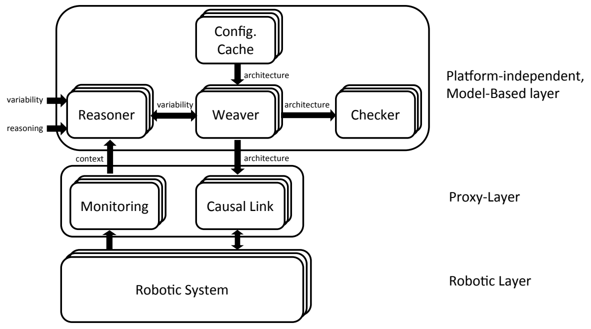
In order to support actual runtime adaptation in robotics, we can consider the three-layer architecture developed as part of the DiVA project (see Fig. 6(a)). In the platform-independent model-based layer, the components produce and consume models, i.e., when the context model is updated, the *Reasoner* calculates a derived variability model accordingly. For all the features included in the variability model, the *Weaver* composes the corresponding architectural model, which then is checked and submitted to the proxy layer. The Proxy layer is responsible for bridging the gap between design-time models and the runtime. The

Causal Link component receives the architectural model and reconfigures the architecture being executed by the robot. Additionally, the *Monitoring* component observes runtime events generated at runtime by the probes in order to create and update the context model. Finally, the Robotic layer basically contains all the specific components of the robot (e.g. see Fig. 6(b)).

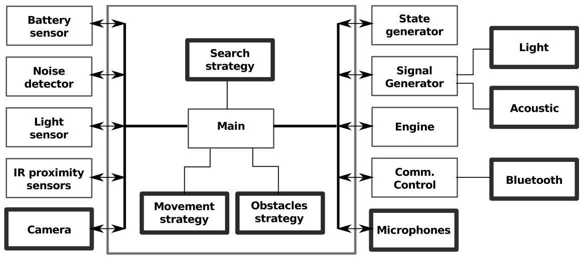
5 New Challenges for Runtime Adaptation in Robotics

Models@Runtime have demostrated promissing results for dynamically adapting business system architectures. However, the experience reported in this paper shows that several issues remain in this approach when applied to robotics. In this section, we present a set of problems and challenges that we feel that are of particular importance and that need to be addressed in the next stage of Model-Driven Engineering for robotics. These challenges are the result of the lessons learned from the experience described in Section 4.

- *Cooperative adaptation engines.* In DiVA, as in most adaptation approaches, it is not easy to design several adaptation engines that need to cooperate. In robotics, this is a fundamental requirement as robots often need to cooperate with other systems (e.g., other robots). Thus, designers should be able to model all adaptation engines and their relations.



(a)



(b)

Fig. 6. (a) Runtime architecture to support dynamic variability in robotics. (b) Case study robot component. Thick lines denote some variability degree.

– *Multi-layer adaptation engines.* Robots might belong to teams and, eventually, to other higher-order communities. Thus, they might need to adapt themselves according to both individual and global adaptation strategies. The current versions of DiVA can not manage layered engines in which the adaptation can be driven both locally and a globally.
– *Model the impact of context dimensions during simulation.* To obtain a correct simulation, it would be interesting to model the impact of selecting certain context dimensions. For instance, if the Bluetooth variant is deactivated, it should not be possible to update the context (e.g., the state of other robots). However, during the simulation step provided by DiVA, all the context variables can be modified at any time, even if the variant controlling the update of these variables is deactivated. This is a major drawback for simulation. In fact, designers need to modify the context model themselves to obtain a correct simulation at each adaptation step.
– *Context sharing.* DiVA allows the description of the relevant aspects to be monitored (environment) as well as the current context. However, it does not support context sharing, sometimes necessary in robotics (e.g., to allow cooperating robots to share their local maps to obtain a global one).
– *Context uncertainty management.* If we add the possibility of sharing context models, we also need to manage the confidence of the shared information. Thus, it is important to know how safe or acurate is the information shared by each robot, as it can (willingly or not) provide others with useless or even dangerous information. Coupling context information with fuzzy logic could simplify the design of context models [4].
– *Using Model@runtime for implementing not only the system architecture but also its components.* Current approaches, such as DiVA, mainly model the system architecture. As a consequence, the adaptation engine can only work at that level (adding or removing components or bindings among them, changing the value of component attributes, etc.) To support other kinds of adaptation, it can be interesting to use models@runtime for component implementation [13]. This would allow us to also adapt some parts of the component implementation.

These six challenges are not exhaustive. They just aim to describe some of the new requirements that need to be managed to make models@runtime usable in the context of adaptive robotic systems.

6 Conclusions and Future Work

This paper reports our experience of using the DiVA model-driven approach to design and implement an adaptive robotic system. Both the benefits and the drawbacks of such an approach have been described. This paper makes the following claims that, in our opinion, are worth being discussed at the workshop:

– As a community, we need to take the next step and adopt the perspective that robotics systems are software intensive systems and their architecture has to

be properly modeled. As stated in [1] software architecture is, *fundamentally, a composition of architectural design decisions* (dimensions in DiVA). These design decisions (that fix some variation points) should be represented as first-class entities in the software architecture and it should be possible to add, remove and change architectural design decisions against limited effort. For that, a "models@runtime" approach like DiVA provides a first answer.
– Models should be used both at design- and at run-time. There is a clear benefit of modeling adaptive robotic systems, as the same models can be used to simulate the robot behavior in a particular context, and then directly executed by the robot at run-time.
– Robotic system are adaptive systems. They should be resource- and context-aware. As systems with limited resources, robots cannot always be confident in the context and cannot always collect the same level of information.
– Robotic systems should be designed to support cooperation with other systems. In that direction, model-driven engineering for robotics research should share some knowledge and design principles with model-driven engineering for Systems of Systems.

For the future, we plan to address some of the challenges identified in Section 5. In particular, in the short- and mid-term, we will work in two main directions: extending the DiVA framework so it can manage several adaptation engines, and supporting a better context representation.

References

1. J. Bosch. Software architecture: The next step. In Flávio Oquendo, Brian Warboys, and Ronald Morrison, editors, *EWSA*, volume 3047 of *Lecture Notes in Computer Science*, pages 194–199. Springer, 2004.
2. D. Brugali and A. Shakhimardanov. Component-Based Robotic Engineering. Part II: Models and Systems. *IEEE Robotics and Automation Magazine*, 17(1):100–112, 2010.
3. H. Bruyninckx. Robotics software: the future should be open. *IEEE Robotics and Automation Magazine*, 15(1):9–11, 2008.
4. F. Chauvel, O. Barais, I. Borne, and J.M. Jézéquel. Composition of qualitative adaptation policies. In *23rd IEEE/ACM International Conference on Automated Software Engineering - ASE'08*, L'Aquila, Italy, sep 2008. Short paper.
5. B. Morin et al. Models@Run.time to Support Dynamic Adaptation. *IEEE Computer*, 42(10):44–51, 2009.
6. D. Alonso et al. V3CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development. *Journal of Software Engineering for Robotics*, 1(1):3–17, 2010.
7. E. Bruneton et al. The FRACTAL Component Model and its Support in Java. *Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, 2006.
8. G. Edwards et al. Architecture-driven self-adaptation and self-management in robotics systems. *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 142–151, 2009.

9. P. Oreizy et al. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.
10. D. Floreano and L. Keller. Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection. *PLoS Biol*, 8:e1000292, 2010.
11. J. Georgas and R. Taylor. Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In *SEAMS 2008*, pages 105–112, 2008.
12. S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic Software Product Lines. *IEEE Computer*, 41(4), April 2008.
13. E. Hoefig, P.H. Deussen, and H. Coskun. Statechart Interpretation on Resource Constrained Platforms: a Performance Analysis. In *4th International Workshop on Models@Run.Time (at MODELS'09)*, Denver, Colorado, USA, Oct 2009.
14. J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Workshop on the Future of Software Engineering (FOSE 2007)*, pages 259–268.
15. N. Mohamed, J. Al-Jaroodi, and I. Jawhar. Middleware for robotics: A survey. In *IEEE International Conference on Robotics, Automation, and Mechatronics (RAM 2008)*, pages 736–742, 2008.
16. B. Morin, O. Barais, G. Nain, and J. M. Jézéquel. Taming Dynamically Adaptive Systems with Models and Aspects. In *ICSE'09: 31st International Conference on Software Engineering*, Vancouver, Canada, May 2009.
17. European Robotics Technology Platform(EUROP). Robotic visions to 2020 and beyond - the strategic research agenda for robotics in europe. appendix: Technology roadmaps. `http://www.robotics-platform.eu/sra`, 2009.
18. C. Schlegel. Communication patterns as key towards component interoperability. In *Software Engineering for Experimental Robotics*, volume 30, pages 183–210. Springer-Verlag, Berlin, Heidelberg, 2007.