Towards Flexible Evolution of Dynamically Adaptive Systems

Gilles Perrouin*, Brice Morin†, Franck Chauvel†, Franck Fleurey†,
Jacques Klein‡, Yves Le Traon‡, Olivier Barais§ and Jean-Marc Jézéquel§
* *PRECISE, University of Namur, Belgium*
gilles.perrouin@fundp.ac.be
† *SINTEF IKT, OSLO, Norway*
{Brice.Morin,Franck.Chauvel,Franck.Fleurey}@sintef.no
‡ *SnT - University of Luxembourg, Luxembourg*
{jacques.klein,yves.letraon}@uni.lu
§ *IRISA, University of Rennes 1, France*
{jezequel,barais}@irisa.fr

***Abstract*—Modern software systems need to be continuously available under varying conditions. Their ability to dynamically adapt to their execution context is thus increasingly seen as a key to their success. Recently, many approaches were proposed to design and support the execution of Dynamically Adaptive Systems (DAS). However, the ability of a DAS to evolve is limited to the addition, update or removal of adaptation rules or reconfiguration scripts. These artifacts are very specific to the control loop managing such a DAS and runtime evolution of the DAS requirements may affect other parts of the DAS. In this paper, we argue to evolve all parts of the loop. We suggest leveraging recent advances in model-driven techniques to offer an approach that supports the evolution of both systems and their adaptation capabilities. The basic idea is to consider the control loop itself as an adaptive system.**

***Keywords*-Dynamically Adaptive Systems; Evolution; Models@Run.Time**

I. Introduction

We are now living in *societies of digital systems* [1] where various devices interact in an unpredictably changing environment offering services to humans ranging from mobile music experience to assistance in crises management. To support such digital societies, we can recourse to *Dynamically Adaptive Systems* (DAS). DAS can be seen as open distributed systems that have the faculty to adapt themselves to the ongoing circumstances and find a way to continue accomplishing their functionalities. Engineering such systems is complex since they combine several adaptation mechanisms that ensure that functional and non functional-properties, such as security and performance, are satisfied at any time.

Two main reasoning paradigms have been explored for DAS: *Event-Condition-Action* (ECA) rules and goal optimization. ECA-based techniques [2] directly link the features of the system to specific context fragments through a set of rules. With a rather low level of abstraction, they provide an efficient runtime processing, but offer a limited scalability for dealing with the full specification of a complex adaptive system [3]. At a high-level of abstraction goal-optimization approaches [4], [5], [6] propose to describe which (QoS) properties should be optimized in which contexts. The impact of each feature on these properties guides the selection of suitable configurations. These approaches offer a high level of abstraction and better scalability but the real-time adaptation can be penalized by the execution of an optimization algorithm for each adaptation.

While DAS offer reconfiguration possibilities, combining distinct reasoning approaches is hardly supported. This lack of flexibility prevents proper support for unexpected events and hinders evolution.

For example, let us consider a Dynamic Customer Relationship Management (D-CRM) system. The key objective of the D-CRM is to provide accurate client-related information depending on the context. When the user is working in his office, the system can notify him by e-mail, via a rich Web-based client. He can also access critical resources because he is connected to a trusted network. When the user is on the go, notifications are filtered retaining only client-related or critical ones using a goal-based reasoner. Yet, if the user is on the go and still would like to access to the internal network, the use of a VPN is necessary. Such a VPN has two alternative communication modes (IPSec or SSL), which can be easily modeled via ECA rules. Yet, no ECA reasoner is available in the system and since the choice of a given communication mode is not driven by QoS properties, tweaking these rules into goals, though possible, is rather counter-intuitive and inefficient.

Moreover, since current approaches for developing and executing DAS rely on hand-written *ad hoc* control loops, which cannot change at runtime. For example, control usually handles only one adaptation paradigm (ECA or Goals) or propose a hardwired combination of the two [3]. This lack of flexibility in the control loops is a major lock for the runtime evolution of adaptive systems.

In this paper, we present an innovative approach, which allows dynamically adapting not only the system but also both its adaptation mechanisms and its adaptation policies.

Exploiting the state of the art on ECA-based and Goal-based techniques, our idea is to define a dynamically reconfigurable adaptation loop. The dynamic reconfiguration of this adaptation loop is achieved by employing similar adaptation techniques as the ones used to adapt the system itself. This "bootstrapping" of the adaptation loop allows evolving DAS in a much more flexible manner.

The paper is organized as follows: Section II presents some important concepts exploited by our proposal. This proposal is described in Section III. Section IV wraps up.

II. Background

To face evolution problems arising in our digital societies, we rely on two important lines of research regarding DAS engineering: *control loops* [7], [8] and *software architecture* [9].

A. Modelling Control Loops

Control loops are the crucial to DAS engineering since they describe the process by which DAS adapt to their environment. Cheng *et al* [10], [7] claim that the loop should be explicitly and externally modeled to enable designers to think about control loops. This is the key to flexible evolution. Models@Run.Time [11] allows control loop models to be updated and enacted dynamically in the running system. To reason on variations of the environment, work carried out by the Dynamic Software Product Line (DSPL) community is particularly relevant [12]. In a DSPL, feature diagrams [13] represent the *variability space* [14] at the functional level *i.e.*, the variability of the adaptive system. We can also use feature models to describe the variability of the control loop itself, which manages the adaptive system. This offers a high-level support to reason about the control loop.

B. Software Architecture

Software architecture plays a major role in system's evolution and DAS are no exception to this rule [15]. One the one hand, architectural patterns have been proposed [9], [16] to separate the running system from its adaptation and reasoning mechanisms. For example, Kramer and Magee [9] propose a layered architecture in which a "goal management" layer is responsible for deriving plans from high-level goals, such plans are managed and transformed into actions in the "change management" layer. Finally the "component control" executes these actions and reports on the status of the new derived architecture. On the other hand, architecture research has permitted the development of adaptive middelware based on reflection mechanisms [16], [17], [18]. Using model-driven techniques, such as code generation, it is possible to project the models to different platforms, such as Fractal [18] or OSGi [1].

[1] http://www.osgi.org/Main/HomePage

III. Dynamic Adaptation of the MAPE Loop

Our claim is that we can flexibly evolve a DAS by dynamically adapting the components of its control loop. Such a *meta-adaptation* mechanism is able to extend the reasoning abilities of a DAS and thus make it resilient to a larger number of unexpected situations. In our work, we focus on the the well-known *Monitoring-Analysis-Planning-Execution* loop (MAPE) [8] on which we illustrate the main elements of a DAS before considering its dynamic adaptation.

A. A DAS MAPE Loop

1) Monitoring: The way monitoring is performed is specific to each business application and depends on the types of targeted property. Monitoring can handle physical resource information (battery, CPU, memory, etc), and/or the user behavior (how the user uses the application, what are her preferences, *etc*), and/or QoS properties (response time, availability, *etc*). There exist different techniques to bridge the gap between runtime events and context models: Complex Event Processing (CEP) [19], Fuzzy Logic [20], *etc*.

2) Analysis: The analysis activity determines which functionalities to use in which context. ECA are *if-then* rules that link such functional features to context fragments. A goal-based reasoner is responsible for finding a selection of features that offers the best trade-off between the goals of the system, which may evolve depending on the context. Finally, users can specify their preferences, by selecting the features they need. As illustrated in Section I, the choice of an adequate reasoner is context-dependent.

Once decided upon, features need to be combined in a configuration. This can be done through the use of structural or behavioral aspect weavers [11], [21], [22] or simply chosen amongst pre-validated configurations in a repository. Finally, it is important to ensure that these configurations are consistent, before going further in the adaptation process [23]. Depending on the structural/behavioral nature of the configurations several options may be offered.

3) Plan Variability & Execute Variability: The planning activity consists in obtaining a safe reconfiguration script [23] able to drive the transition from the actual system to the target system and providing roll-back mechanisms if the target configuration is not possible to reach. There exist different planning strategies and tools (such as PDDL [24]) in order to optimize the order of the reconfiguration commands, that could be integrated here. Execution consists in applying the (optimal) sequence of reconfigurations commands, either directly or in a transactional way, enabling to return to the last consistent configuration.

B. Adaptation Logic of the MAPE Loop

The MAPE loop can be implemented as a component-based system, so that a MAPE loop can potentially be

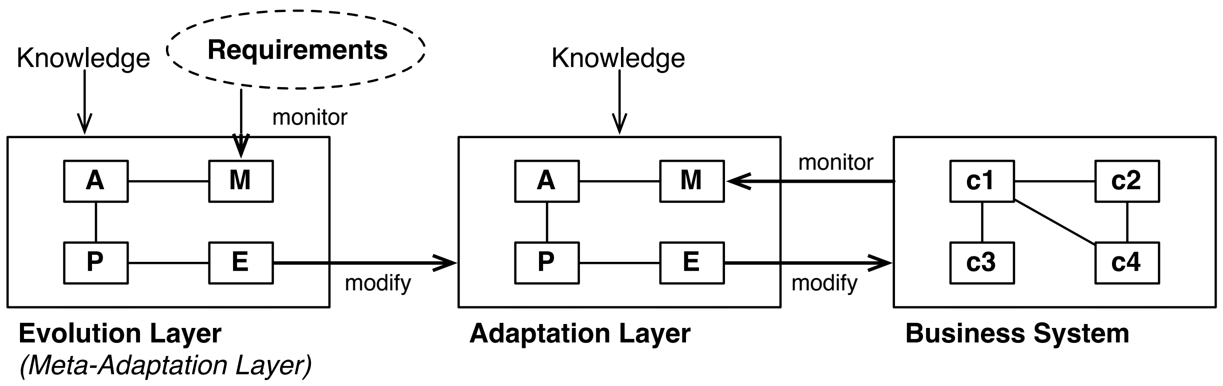

Figure 1. An Overview of the Proposed Approach: Dynamic Adaptation of the Adaptation Loop to Support Evolution

reconfigured at runtime. In other words, we can use a MAPE loop to manage another MAPE loop, as illustrated by Figure 1.

This basically defines a 3-layered architecture:

1) Business Layer (right): This layer describes the architecture of the business application. This layer is managed by the adaptation layer, which can dynamically reconfigure the business architecture (addition/removal of components/bindings) depending on the monitored context.

2) Adaptation Layer (middle): This layer is responsible for managing the business layer. It basically consists of a MAPE loop we usually found in most adaptive systems [8]. This MAPE loop is fed with knowledge (mainly variability and reasoning models) to determine when and how to adapt the business architecture. The novelty is that this loop is dynamically adaptive and managed by the evolution layer.

3) Evolution (Meta-Adaptation) Layer (left): This layer is responsible for managing the adaptation layer *i.e.*, to reconfigure the MAPE loop that manages the business system. This layer is also a MAPE loop, but it does not evolve at runtime[2]. It monitors the requirements of the business architecture. More precisely, it observes the knowledge of the adaptation layer and can decide to dynamically reconfigure the adaptation layer to make it to take full advantage of this new knowledge.

The dynamic reconfiguration of the adaptation loop is mainly driven by the evolution of the models (knowledge) describing the different facets of the business system. Since the evolution of models can be monitored similarly to the evolution of the execution context [19], the information necessary to drive the logic defining the subsequent evolutions of the adaptation layer can be easily obtained. In the following, we describe how to define such a logic with respect to the evolution of the different models.

- **Variability Model.** An evolution of the variability model implies that some functionalities may not be available anymore or new functionalities have been added. In the latter case, this has an impact on building configurations, *e.g.*, it is not possible anymore to rely uniquely on the selector repository since the available configurations will not provide these new functionalities. We therefore need to deploy an aspect weaver and a runtime checker to derive new configurations containing novel functionalities and check their validity at runtime. For example, the checker will ensure that the necessary components realizing new functionalities are accessible: either installed in the platform or via an URL referring to a remote component repository.
- **Reasoning Model.** An evolution of the reasoning model implies to ensure that all the adaptation logic can be handled by the reasoners. If goals are introduced and only the ECA reasoner is present we need to deploy the goal-based reasoner as well. If some reasoning paradigms are removed (*e.g.*, no more ECA rules) the corresponding reasoners have to be removed accordingly to optimize performance and resource consumption.
- **Context Model.** The evolution of the context model has to be performed in accordance with the variability model. if there is a new dimension of the environment to observe then the corresponding monitoring components has to be made available and shown in the variability model.

The advantage of working at the evolution layer is that by manipulating abstractions of the adaptation process (variability, reasoning and context) the number of elements to consider is kept small and we can consider simple evolution scenarios based on ECA-rules. These scenarios can be also fully determined and checked at design-time.

IV. Conclusion

Adaptive systems have the ability to adapt to their execution context according to planned rules or goals. These systems are often managed by a control loop, which observes the system and its context, reasons and determines if and how to dynamically adapt the system. Current approaches

[2]Additional meta-adaptation layers can be defined [15].

for developing and executing adaptive systems offer no or very limited support for the evolution of such systems. In particular, the loop controlling the adaptive system is static: it cannot dynamically adapt to leverage new kind of knowledge in a safe way.

In this paper, we sketched a vision to address adaptive system evolution. By considering a MAPE loop as a model-based adaptive system, we can combine reasoning strategies (goals or ECA) to cope with new requirements, optimize DAS to use the most adapted reasoner or easily integrate new dimensions of the environment. Component-based DAS approaches already own the basic building blocks on which such a vision can be implemented. The first item on our agenda is to support our vision and to assess its benefits on concrete case studies.

Acknowledgment

This work was partially funded by: the Walloon Region under the NAPLES project, the Belgian Science Policy Office's MoVES project, BNB and FNRS, Luxembourg's FNR MITER and SINTEF MODERATES projects.

References

[1] A. Carzaniga, G. Denaro, M. Pezze, J. Estublier, and A. L. Wolf, "Toward deeply adaptive societies of digital systems," in *ICSE COMPANION*. IEEE, 2009, pp. 331–334.

[2] P. David and T. Ledoux, "Safe dynamic reconfigurations of fractal architectures with fscript," in *5th Fractal Workshop at ECOOP*, vol. 4067, 2006.

[3] F. Fleurey and A. Solberg, "A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems," in *MoDELS*, 2009, pp. 606–621.

[4] H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hughes, "Goal-based modeling of dynamically adaptive system requirements," in *ECBS*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 36–45.

[5] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *RE*. IEEE Computer Society, 2001, pp. 249–262.

[6] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: A framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 7–16.

[7] S. Cheng, D. Garlan, and B. Schmerl, "Making self-adaptation an engineering reality," in *SELF-STAR*. Springer, 2004, pp. 158–173.

[8] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[9] J. Kramer and J. Magee, "A rigorous architectural approach to adaptive software engineering," *J. Comput. Sci. Technol.*, vol. 24, no. 2, pp. 183–188, 2009.

[10] B. Cheng and *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1–26.

[11] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@ Run.time to Support Dynamic Adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[12] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, 2008.

[13] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[14] G. Perrouin, F. Chauvel, J. DeAntoni, and J.-M. Jézéquel, "Modeling the variability space of self-adaptive applications," in *DSPL@SPLC*, Limerick, Ireland, Sep. 2008, pp. 15–22.

[15] G. Edwards, J. Garcia, H. Tajalli, D. Popescu, N. Medvidovic, G. Sukhatme, and B. Petrus, "Architecture-driven self-adaptation and self-management in robotics systems," in *SEAMS workshop@ICSE*. IEEE, 2009, pp. 142–151.

[16] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[17] N. Bencomo, G. Blair, C. Flores, and P. Sawyer, "Reflective component-based technologies to support dynamic variability," in *VaMoS*, Essen, Germany, 2008.

[18] E. Bruneton, T. Coupaye, M. Leclercq, V. Quema, and J. Stefani, "The Fractal Component Model and its Support in Java," *Software Practice and Experience*, vol. 36, no. 11-12, pp. 1257–1284, 2006.

[19] B. Morin, T. Ledoux, M. Ben Hassine, F. Chauvel, O. Barais, and J.-M. Jézéquel, "Unifying Runtime Adaptation and Design Evolution," in *Conference on Computer and Information Technology*, Xiamen, China, Oct 2009.

[20] F. Chauvel, O. Barais, I. Borne, and J.-M. Jézéquel, "Composition of qualitative adaptation policies," *ASE*, pp. 455–458, 2008.

[21] J. Kienzle, W. Al Abed, and J. Klein, "Aspect-oriented multi-view modeling," in *AOSD*. New York, NY, USA: ACM, 2009, pp. 87–98.

[22] C. Parra, X. Blanc, A. Cleve, and L. Duchien, "Unifying design and runtime software adaptation using aspect models," *Science of Computer Programming*, 2011.

[23] B. Morin, O. Barais, G. Nain, and J. Jézéquel, "Taming Dynamically Adaptive Systems with Models and Aspects," in *ICSE*, Vancouver, Canada, May 2009.

[24] M. Ghallab, ENSICA, C. K. Isi, K. Golden, S. Penberthy, D. E. Smith, Y. Sun, D. Weld, and D. Mcdermott, "The planning domain definition language," Yale CVC, Tech. Rep., 1998.