Evolving Adaptable Systems: Potential and Challenges

Klaus Schmid[1], Holger Eichelberger[1], Ursula Goltz[2], Malte Lochau[2]
[1]University of Hildesheim, Institute of Computer Science, Software Systems Engineering
[2]TU Braunschweig, Institute for Programming and Reactive System
Email: {schmid|eichelberger}@sse.uni-hildesheim.de, {goltz|lochau}@ips.cs.tu-bs.de

1 Motivation

Future software systems will be even longer in service than today (which often is already measured in decades). During operation, a system must change in many ways, e.g., due to environment changes, functional enhancements, etc. While adaptive software systems already cope with changing environments, existing approaches are insufficient to address an open-ended world where unpredictable changes may happen that cannot be foreseen. Here, we discuss how existing work in (dynamic) software product lines, evolution of product lines (e.g. [1]) and adaptive systems (e.g. [2]) can be combined to improve evolvability of adaptive systems.

Before we discuss the evolution of adaptive systems, we first outline the concept of variability and its relationship to adaptive systems. A key concept of software product lines (SPL) is the notion of variability [4]. A variability model defines the differences among valid products as well as interdependencies among different characteristics of a SPL. While the differentiation among the products is typically made at development time, more recently approaches, referred to as dynamic software product lines (DSPL) [5], target variation at runtime.

While evolution of SPL can be seen as an orthogonal dimension to variability engineering, variability models can be used to capture and organize evolution variants. As we explained in [1], this requires extensions to existing variability modeling approaches. These extensions are required to support the potential openness of evolutionary changes.

The term *self-adaptive system* is typically used for systems that are capable of performing changes of their internal or external behavior or structure. This is often done to counter changes in the environment, external threats, or other operational challenges like compensating system faults [3].

An adaptive system consists of the following major parts / subsystems (cf. Figure 1):

- *Monitor:* collects data on the current situation (internally and externally to the system) as a basis for decisions on changes to the system.
- A*daptivity model:* describes the adaptations and constraints on them (i.e., possible modifications and the effect on a benefit function). This is the basis for planning adaptations. Here, we address only approaches with an explicit or implicit control.
- A*daptivity manager:* interprets the adaptivity model and determines adaptations. It consists of the adaptivity model and a planning component.
- *Adaptivity executor:* realizes the adaptation plan, i.e., performs the actual adaptations.
- *Capability implementation*s realize the actual functionality of the system. Adaptation modifies parameters or interconnections among them.

If we compare adaptive systems and DSPLs, we see that DSPLs are a class of adaptive systems. The (runtime) variability model in DSPLs corresponds to the adaptivity model in adaptive systems. It should be noted, however, that in adaptivity models sometimes not only the capabilities of a system are represented, but the adaptation behavior of the system is directly encoded (removing the need for a planning component).

However, the kind of information in an adaptivity model can be more complex than in a typical variability model. While variability models are usually logic-based, adaptivity models can as well contain complex metrics and trade-offs. What is common in both cases is that the scope of the variability model / adaptivity at any point in time determines the range of operations of the system. We call this also closed adaptivity as the total space of adaptations is (implicitly) predefined.

This leads to a major shortcoming of typical approaches to adaptivity. They are able to handle changes in the environment of a system at runtime, but only within a predefined scope. Long-living systems need to be capable of extending their adaptation space, as no matter how thorough the analysis and construction of the adaptation space is done, there will always be adaptations which have not been expected. Such an evolution of the adaptation space, i.e. of the adaptation model and capability implementations is currently not well supported.

In the next section, we discuss a basic approach for opening the space of adaptivity of adaptive systems to provide the basis for their evolution.

2 Modularization of Models for Evolving Adaptable Systems

What does it mean to evolve an adaptive system? As discussed, we need to open the space of potential adaptations. This requires modifications based on the structure of adaptive systems given above:

- The *adaptivity model* of the system must be expanded; potential adaptations, their interdependencies and monitored events must be added.
- The *capability implementations* within the system must be augmented. This may include both the addition of new functionality and the modification of existing functionality.
- Ideally, the *adaptivity manager* can remain the same. This requires a significant level of conceptual generality of its implementation.
- *Monitor* and *Executor* can ideally also remain the same if they are generic enough.

This leads us to different forms of evolvability requirements for each of these parts:

- The evolution of capability implementation is today already well supported by means of concepts like modularization, information hiding, etc. Thus, we can decouple individual capability realizations rather well. Existing plug-in architectures already support even live-updates of systems with new capabilities.
- Existing approaches to modeling adaptivity are rather monolithic in nature. We do not know of any specific approach that explicitly addresses evolvability concerns.
- The adaptivity manager is in general a single component, which may contain subcomponents (e.g., for planning, for monitoring, etc.). As we know from software architectures this makes it rather difficult to evolve. This would be less of a problem, if a generic adaptivity manager could be developed so that all system-specific aspects could be relegated to the adaptivity model.

In the field of software architectures we learned that decomposition into components, decoupling, and well defined interfaces are key for easy evolution of software systems. If we transfer these principles to the situation of adaptive systems, the question is: how to realize these concepts there?

Any new introduced capabilities are strongly related (coupled) with modifications of the adaptivity model. To minimize coupling and establish well defined interfaces, we must package these capabilities with related modifications to the adaptivity model. This is analogous to packaging variability model fragments with relevant functionality for supporting the independent evolution of SPL [1].

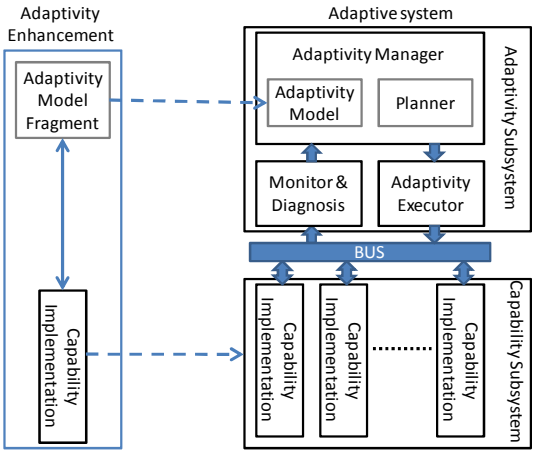

Figure 1: Adaptive system with adaptivity enhancement

3 Challenges

What challenges must be overcome to achieve a modularization as outlined above?

- Adaptivity Manager: What is the ultimate power of an adaptivity manager? Is there a criterion similar to Turing-completeness in the context of adaptivity, so we can be sure that we need not add capabilities to the adaptivity manager itself?
- Adaptivity Modeling Languages (AML): What support for evolution is provided by existing AMLs? How can we express modularization in them, and how can we support the decoupling by language means? Can we develop complete AMLs, i.e. ALMs being sufficiently powerful that the language itself need not be augmented?
- Capability Implementation: How can we clearly separate individual capability implementations, so that they can be introduced at a later time and how do we deal with their interactions?

And finally, how do we put this all together?

Our proposal is to derive adaptivity bundles, similar to the feature bundles described in [1]. These would contain the added capabilities, along with the declarative adaptivity model fragment (cf. Figure 1). This would also require the management of dependencies among the various adaptivity model fragments. This kind of dependency management can be easily realized using established SPL techniques like variability management [6].

References

[1] Schmid, K.: Verteilte Evolution von Produktlinien: Herausforderungen und ein Lösungsansatz, 1. Workshop Design For Future - Langlebige Softwaresysteme (L2S2), FZI Karlsruhe, 2009.

[2] Steiner, J.; Goltz, U.;Maaß, J.: Dynamische Verteilung von Steuerungskomponenten unter Erhalt von Echtzeiteigenschaften, 6. Paderborner Workshop Entwurf mechatronischer Systeme, 2009.

[3] Cheng, Betty H.C. et al.: Software Engineering for Self-Adaptive Systems: A Research Roadmap (Draft Version), Dagstuhl Seminar Proc. 08031, 2008.

[4] van der Linden, F.; Schmid, K.; Rommes, E.; Product Lines in Action, Springer Verlag, 2007.

[5] Hallsteinsen, S.; Hinchey, M.; Park, S.; Schmid, K.; Dynamic Software Product Lines; IEEE Computer, Vol. 41, No. 4, pp. 93-95, 2008.

[6] Schmid, K.; John, J; A Customizable Approach to Full Lifecycle Variability Management; Science of Computer Programming, Vol. 53, No. 3, pp. 259-284, 2004.