A Classification of Dynamic Reconfiguration in Component and Connector Architecture Description Languages

Arvid Butting[1], Robert Heim[1], Oliver Kautz[1], Jan Oliver Ringert[2], Bernhard Rumpe[1], Andreas Wortmann[1]
[1]Software Engineering, RWTH Aachen, Aachen, Germany, http://www.se-rwth.de/
[2]School of Computer Science, Tel Aviv University, Tel Aviv, Israel, http://cs.tau.ac.il

***Abstract*—Architecture description languages (ADLs) facilitate model-driven engineering by fostering reuse of component models. Some of the over 120 ADLs contributed by academia and industry feature dynamic architecture reconfiguration and the underlying mechanisms vary significantly. When considering employing an ADL supporting dynamic reconfiguration it is challenging to keep track of the possibilities. We conducted a literature study investigating the different reconfiguration mechanisms of component & connector (C&C) ADLs. To this effect, we started with the 120 ADLs studied in [29], reduced these to C&C ADLs, investigated their reconfiguration mechanisms, and classified these along six dimensions. The findings unravel the state of dynamically reconfigurable C&C ADLs and support developers considering employing one in choosing the most suitable language.**

I. Introduction

Component & connector (C&C) architecture description languages [29], [32] combine the benefits of component-based software engineering with model-driven engineering (MDE) to abstract from the accidental complexities [19] and notational noise [54] of general-purpose programming languages (GPLs). They employ abstract component models to describe software architectures as hierarchies of connected components.

We adopt the notion of C&C ADLs as described in [32], where components encapsulate the functionality of the system within well-defined stable interfaces and connectors enable component interaction. These concepts abstract over technical language details of C&C ADLs. In many ADLs the configuration of C&C architectures is fixed at design time. The environment or the current goal of the system might however change during runtime and require dynamic adaptation of the system [45] to a new configuration that may only include a subset of already existing components and their interconnections or may introduce new components and connectors.

To support dynamic adaptation a modeled C&C architecture either has to adapt its configuration at runtime or it must encode adaptation in the behaviors of the related components. This encoding introduces implicit dependencies between components and forfeits abstraction of behavior paramount to C&C models. It thus imposes co-evolution constraints on different levels of abstraction and across components. Dynamic reconfiguration mechanisms and their formulation in ADLs help to mitigate these problems by formalizing adaptation as structural reconfiguration. This allows components to maintain encapsulation and abstraction of functionality.

Different C&C ADLs have suggested different reconfiguration mechanisms currently lacking detailed classification. On the one hand, this creates challenges for engineers in selecting ADLs with the right reconfiguration mechanisms. On the other hand, a classification might help ADL creators to design appropriate reconfiguration mechanism. Our goal is to identify the modeling dimensions for dynamic reconfiguration in C&C ADLs. We therefore investigate dynamic reconfiguration in C&C ADLs and develop a classification of C&C ADL reconfiguration mechanisms. Our contribution consists of (1) a study of dynamic reconfiguration in C&C ADLs, and (2) a classification of C&C ADLs along different dimensions of dynamic reconfiguration.

Sec. II gives an example to demonstrate benefits of dynamic reconfiguration, before Sec. III presents concepts of dynamic reconfiguration in C&C ADLs. Afterwards, Sec. IV discusses our study and Sec. V compares it to related work. Finally, Sec. VI concludes.

II. Example

As motivating example, we consider different C&C model configurations of a shift controller for an automatic transmission system for cars, as modeled in [23]. In this example, the car's clutch can adopt the six positions Park, Reverse, Neutral, Drive, Sport, and Manual that influence when to shift gears. Each of these positions is reflected in the software architecture by an equivalent transmission operating mode (TOM). In C&C software architectures, each shifting behavior would typically be modeled as an individual component. With dynamic reconfiguration, the architecture can adapt at run time. To this end, reconfiguration modifies parts of the architecture, for example, by redefining the connections between components. There are different approaches to realizing dynamic reconfiguration, *e.g.,* stating different configurations of activated components and connectors or exchanging connectors and instantiating or deleting subcomponents.

This research has partly received funding from the German Federal Ministry for Education and Research under grant no. 01IS16043P. The responsibility for the content of this publication is with the authors.

[BHK+17] A. Butting, R. Heim, O. Kautz, J. O. Ringert, B. Rumpe, A. Wortmann:
A Classification of Dynamic Reconfiguration in Component and Connector Architecture Description Languages.
In: Proceedings of MODELS 2017 Satellite Event: Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium, Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations co-located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017), 2017.
www.se-rwth.de/publications/

In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 75–84. ACM, 2010.

[52] Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The Koala Component Model for Consumer Electronics Software. *IEEE Computer*, 2000.

[53] Rainer Weinreich and Georg Buchgeher. Paving the Road for Formally Defined Architecture Description in Software Development. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 2337–2343. ACM, 2010.

[54] David S. Wile. Supporting the DSL Spectrum. *Computing and Information Technology*, 2001.

[55] Qing Wu and Ying Li. ScudADL: An Architecture Description Language for Adaptive Middleware in Ubiquitous Computing Environments. In *ISECS International Colloquium on Computing, Communication, Control, and Management*, 2009.

[56] Qian Zhang. Visual Software Architecture Description Based on Design Space. In *International Conference on Quality Software*, 2008.