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.

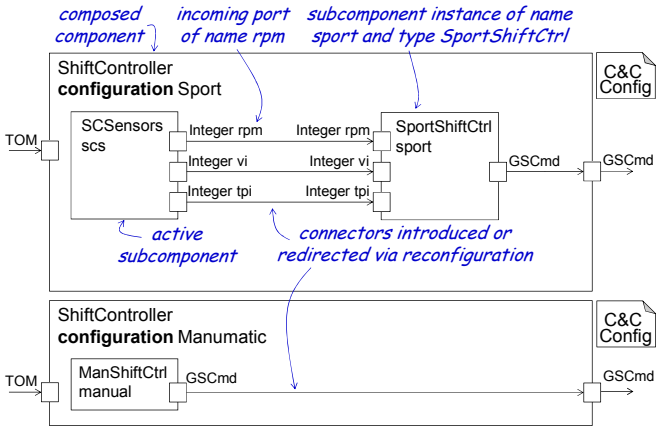

Fig. 1. Three configurations of the component `ShiftController`. Unconnected subcomponents are omitted.

Fig. 1 illustrates dynamic reconfiguration of the composed component `ShiftController` by depicting two exemplary configurations. The top depicts the configuration `Sport`, where the subcomponent `SportShiftCtrl` is employed to shift the gears according to several values measured by the `SCSensors` component. In this configuration, two subcomponents are active and four connectors are involved. The bottom depicts the `Manumatic` configuration, which has one active subcomponent `ManShiftCtrl` that reads the position of the clutch and changes gears accordingly. The subcomponents that are not used in the current configurations are either unconnected, deactivated (*e.g.,* to save energy), or removed – depending on the realization of dynamic reconfiguration. `TOM` messages received by the `ShiftController` component are used to change between different configurations, *e.g.,* based on an automaton with a state for each configuration.

With explicit dynamic reconfiguration mechanisms, the connectors between software components become exchangeable or modifiable at runtime, which enables to specify different operating modes of software architectures. Some ADLs also support dynamic instantiation and removal of subcomponents. In many ADLs, the number of possible configurations is unrestricted, yielding great flexibility. In others, configurations are made explicit and controlled, yielding the possibility of describing flexible software architectures for dynamic distributed systems or system networks, while still supporting (automatic) formal analyses on these systems. The latter is crucial for modeling reliable and safe software architectures.

III. Dynamic Reconfiguration in C&C ADLs

Modeling C&C architectures using ADLs [31], [32] abstracts from the notational noise [54] and accidental complexities [19] of GPLs. ADLs enable modeling complex systems as interacting, encapsulated components with stable communication interfaces. Connectors enable components to interact with each other via their interfaces. Many ADLs yield additional features, such as communication constraints, non-functional properties, component behavior modeling techniques, or modeling elements for domain-specific aspects. Different domains successfully adopted ADLs, such as avionics [18], automotive [9], cloud systems [36], and robotics [46].

We consider an ADL as dynamic if it supports modeling dynamically reconfigurable architectures, *i.e.,* it allows at least to model dynamic establishment or removal of connectors. The importance of dynamically reconfigurable architectures has long been recognized [27] and it has been implemented for multiple ADLs. Nonetheless, there are many ADLs that support static architectures only, such as ArchFace [51], ALI [6], C3 [3], COSA [49], DiaSpec [13], Gestalt [47],LISA [53], MontiArc [11], Palladio [7], UNICON-2 [16], or xADL [26]. Many of these ADLs focus on and excel in different concerns, such as expressiveness (EAST-ADL [15]), domain-specificness (DiaSpec [13]), or extensibility (xADL [26]). Where available, rules for dynamic reconfiguration are usually expressed by designated modeling elements of the ADL. However, there is no consensus on how to describe dynamic reconfiguration in architectural models.

We propose a classification of dynamic reconfiguration mechanisms. Our starting point are 120 architecture languages found in [29]. From this, we excluded ADLs that are not based on C&C models (such as P++ [48]) and component models with GPL implementations only (such as the k-Component model [17], which is realized in C++ instead of an explicit ADL). Afterwards, we excluded all ADLs for which no peer-reviewed publications were available (such as DADL [34]) and all ADLs based on formal approaches for which we did not find an implementation (such as TADL [35]). This left 23 ADLs that we examined regarding their capabilities and mechanisms for dynamic reconfiguration.

A. A Classification of Dynamic Reconfiguration

During examining the ADL's publications, we identified six dimensions for the classification of modeling C&C ADL dynamic reconfiguration mechanisms.

Restricted vs. Open Reconfiguration: With restricted reconfiguration, the number of possible configurations is finite at design time. For instance, AADL [18], AutoFocus [5], [4], MontiArcAutomaton [23], or PRISMA [40] enable dynamic reconfiguration in a restricted fashion. Here, composed components can change between a finite number of configurations (called "modes"). Switching between modes is also restricted: specific transition govern under which circumstances a component may change its configuration. Dynamic Wright [2], for instance, enables to model dynamic reconfiguration between a finite number of configurations predefined at design time in response to the occurrence of special control events [2]. With open reconfiguration, components may dynamically change to configurations at runtime that have not been predefined at design-time (*cf.* π-ADL [38], LEDA [12], PiLar [14], and ArchJava [1]) or may change between an unrestricted number of configurations predefined at design time [28]. While restricted reconfiguration lacks the flexibility of open

reconfiguration, it may increase model comprehensibility and facilitate static analysis.

Imperative vs. Declarative Specification: In imperative reconfiguration mechanisms, the reconfiguration is defined programmatically, whereas declarative specifications typically describe configurations. Imperative specifications exist in many ADLs: ArchJava [1] allows to instantiate and connect components similar to objects in Java. The π-ADL [38], LEDA [12], and PiLar [14] include control structures and operators for modeling dynamic reconfiguration such as the instantiation and removal of components and connectors. Dynamic Wright [2] provides special actions (new, del, attach, detach) to instantiate or remove architectural elements. The architecture modification language of C2 SADL [30] enables to specify sequences of operations for performing dynamic reconfigurations. In Fractal [10], components may contain various controllers, which are explicit language elements to add and remove subcomponents as well as connectors. A controller's implementation, *e.g.,* in Java [10], defines the reconfiguration mechanism. Declarative specifications appear in AADL [18], AutoFocus [4], Darwin [28], and Koala [52]. In the AADL [18] and AutoFocus [4], dynamic architectures are declaratively described by predefined modes. In Koala [52], switches are parts of component configurations. Darwin [28] supports to declaratively specify special services for component instantiation. To some extent, reconfiguration can built upon both imperative and declarative mechanisms. ACME/Plastik [25], *e.g.,* provides imperative operators for the instantiation of components, but additionally supports to declaratively specify dependencies of the new component to further architectural elements. When the component is instantiated, these elements are also instantiated.

Programmed vs. Ad-Hoc Reconfiguration: With programmed reconfiguration, the architecture model is aware of reconfiguration possibilities, whereas with ad-hoc reconfiguration, it is not. In programmed reconfiguration, reconfiguration conditions and effects are specified at design time. At runtime, the reconfiguration is applied when the conditions for reconfiguration are met. Programmed reconfiguration can be restricted (through specification of modes) or open (through imperative reconfiguration specification). Ad-hoc reconfiguration introduces greater flexibility, but the reconfiguration options are invisible in the architecture models. Combining ad-hoc reconfiguration with open reconfiguration can have the form of external scripts selecting from predefined configuration modes at architecture runtime. The ADLs AADL [18] and ArchJava [1], for example, support programmed reconfiguration through modes and imperative programs, respectively. ACME/Plastik [25], C2 SADL [30], and Fractal [10] support ad-hoc reconfigurations. In ACME/Plastik [25], for example, ad-hoc reconfiguration is possible by executing reconfiguration scripts that operate on a runtime API. While ad-hoc reconfiguration allows to simulate unforeseen architectural changes, *e.g.,* for testing the robustness of an architecture at runtime, it complicates analysis and evolution.

Dynamic Component Instantiation & Removal: Many dynamic ADLs also allow to instantiate or remove components at runtime. In ACME/Plastik [25], so-called actions can remove and create connectors and components. ArchJava [1] embeds architectural elements in Java and hence allows to instantiate objects corresponding to special component classes similarly to ordinary Java objects. Although dynamic removal of components cannot be explicitly modeled with ArchJava, component instances are garbage collected, when they cannot be referenced anymore. C2 SADL [30] supports ad-hoc instantiation and removal of components. The controller concept of Fractal [10] enables to dynamically instantiate and remove components. The π-ADL [38], LEDA [12], and PiLar [14] provide language constructs for the specification of dynamic architectures, such as the instantiation, removal, or movement of components. Dynamic Wright [2] supports modeling instantiation and removal of components in response to the occurrence of special control events. MontiArcAutomaton [23] supports modeling dynamic component instantiation and removal on mode changes. Darwin [28] only support modeling dynamic instantiation but not dynamic removal of components. ExSAVN [56], AVDL [44], and AOSEPADL [21] do not support dynamic component instantiation or removal. While a reconfiguration mechanism with dynamic component instantiation and removal is more expressive than a mechanism relying on establishing and removing only connectors at runtime, the former complicates formal analyses on the architecture.

Instructed vs. Triggered Reconfiguration: With instructed reconfiguration, the environment can intentionally instruct a component's reconfiguration. If an ADL supports modeling the application of a reconfiguration in response to the occurrence of a dedicated reconfiguration event only, we denote the reconfiguration mechanisms as *instructed* (the environment instructs the component to reconfigure). The nature of these events depends on the expressiveness of the respective ADL and can range from receiving messages of specific types to components providing explicit reconfiguration services. In contrast, if a component reconfigures based on internal conditions over events that have to be satisfied (*e.g.,* reconfigure automatically if the incoming messages indicate a high load) we denote the reconfiguration mechanism as *triggered*. The ADLs AVDL [44], Darwin [28], Dynamic Wright [2], and OOADL [50] support instructed reconfiguration only. With ACME/Plastik [25], ArchJava [1], AutoFocus [5], [4], Fractal [10], MontiArcAutomaton [23], LEDA [12], and PiLar [14], for instance, triggered reconfiguration is possible as well. Triggered approaches are more powerful than instructed approaches as they can emulate instructed reconfiguration. Triggered reconfiguration, however, reduces complexity of specifying configuration changes at the cost of flexibility.

Self-Direction: With self-directed approaches, the configuration of a component cannot be changed by its environment at

will, instead only the component itself knows when and how to reconfigure. With self-direction, reconfiguration is encapsulated into the composed component [24], [50], but can be made accessible through its interface, *e.g.,* through dedicated reconfiguration messages. This can be either instructed or triggered. MontiArcAutomaton is a representative of a dynamic, mode-based ADL supporting self-directed reconfiguration only, as modes of a component are only accessible in the scope of the component itself [23]. In Darwin [28], a component may provide special services that dynamically change the configuration of the component. ACME/Plastik [25] supports imperative specification of programmed reconfiguration. Its language elements (e.g., on, detach, remove, dependencies, active property) only allow for self-directed reconfiguration. Similarly, Dynamic Wright [2] only allows self-directed reconfigurations by means of special actions (new, del, attach, detach) for dynamically modifying component configurations. In AADL [18], modes of subcomponents can be mapped to modes of their enclosing components. The subcomponents then switch their modes according to the mode transitions of their enclosing components. PiLar [14] is an imperative ADL providing reflective commands that enable components to retrieve any other system part that can be manipulated afterwards using the language's dynamic commands. In Fractal [10], components may be reconfigured from their environments via the external interfaces of their membranes and from their subcomponents via their membranes' internal interfaces. The π-ADL [38] and LEDA [12] include the mobility aspects of the π-calculus [33]. Therefore, neither AADL, nor PiLar, Fractal, π-ADL, or LEDA are self-directed. Self-direction retains the encapsulation of component behavior as reusable black boxes. While it can reduce the flexibility of reconfiguring, non self-directed reconfiguration challenges system composition when the reconfiguration makes assumptions about the internals of (sub-)components.

B. Summary and Observations on Dynamic Reconfiguration

Table I summarizes our findings for the 23 identified ADLs with unique dynamic reconfiguration mechanisms. The ADLs are classified according to the identified dimensions. The columns of the table indicate whether the reconfiguration mechanism of each ADL is restricted, is imperative (Imper.) or declarative (Decl.), is programmed or ad-hoc, allows for dynamic component instantiation or removal, is triggered, and whether it is self-directed. The symbols ✓, ×, **?** denote that a concept is supported, not supported, or that support is unknown based on the available literature.

The majority of ADLs supporting dynamic reconfiguration support open reconfiguration, enabling changing the architecture in ways unforeseen at design time (for instance by loading and applying change models). All ADLs, but C2 [30] support programmed reconfiguration, whereas C2 is the single ADL supporting only ad-hoc reconfiguration. Greater flexibility is supported by ACME/Plastik [25], Fractal [10], and MAE [43], which support both reconfiguration mechanisms. Where component instantiation is supported, component removal is usually supported as well. Only Darwin [28] does not support removal despite supporting instantiation. Whether this is due to the semantic challenges of removal or the lack of a driving use case is not disclosed. It is furthermore somewhat surprising that only half of the ADLs support triggered reconfiguration, as instructed mechanisms are substantially less flexible.

While support for instructed or triggered mechanisms and support for component instantiation or removal are spread uniformly over the 20 years of relevant papers, there seems to be a trend towards programmed, declarative specification of reconfiguration in the last decade. We assume to obtain better model checking support, as these specifications are often fixed at design time. Detailed investigation of the complexities of the related reconfiguration specification mechanisms might give insights into this.

IV. Discussion

We conducted a literature study based on the 120 ADLs presented in [29]. The authors performed a systematic search using multiple databases and used the resulting list of ADLs to investigate the industrial requirements on ADLs. Limiting research to these 120 ADLs can be considered a threat to the design of our study. Future work includes conducting a systematic mapping study on C&C ADLs to identify ADLs potentially missing in [29]. Nevertheless, we are confident that our findings are valid for the majority of C&C ADLs. As we only considered publications available in English, there might, however, be publications on C&C ADLs with novel reconfiguration mechanisms not accessible to us. Additional threats arise from including publications using other than established terminology (*cf.* [32]) to describe ADLs. To avoid preventing relevant publications or including irrelevant publications, ambiguous publications were reviewed by at least three of the authors and their inclusion was discussed based on these reviews. To prevent the threat of classification fatigue, it was performed in sessions of at most one hour broken followed by breaks of at least 15 minutes.

V. Related Work

Our study depends of the notion of component & connector architecture description languages as introduced in [32], where the authors investigated the properties of components and connectors in nine ADLs. In that study, dynamic reconfiguration is not investigated systematically. Moreover, the presented study leverages the data collected for and presented in [29]. In that study, the authors investigated the requirements on architecture languages from an industrial perspective. They do not classify reconfiguration features of the identified architecture languages. The authors of [22] investigate the properties relevant to system-of-systems engineering of four ADLs (including UML [37] and SysML [20]), but do not study their different dynamic reconfiguration mechanisms. In [39], the authors investigate six first-generation [31] ADLs and find that at least Darwin [28] and Rapide [27] support dynamic reconfiguration. They do not detail or compare the mechanisms.

ADL	Restricted Reconf.	Specification Style	Programmed Reconf.	Ad-hoc Reconf.	Component Instantiation	Component Removal	Triggered Reconf.	Self-directed Reconf.
AADL [18]	✓	Decl.	✓	**?**	×	×	×	×
ACME/Plastik [25]	×	Imper.	✓	✓	✓	✓	✓	✓
AOSEPADL [21]	✓	Decl.	✓	×	×	×	×	✓
ArchJava [1]	×	Imper.	✓	**?**	✓	✓	✓	✓
AutoFocus [4], [5]	✓	Decl.	✓	**?**	×	×	✓	✓
AVDL [44]	**?**	Decl.	✓	×	×	×	×	×
C2 SADL [30]	×	Imper.	×	✓	✓	✓	×	×
Darwin [28]	×	Decl.	✓	**?**	✓	×	×	✓
Dynamic Wright [2]	✓	Imper.	✓	×	✓	✓	×	✓
ExSAVN [56]	**?**	**?**	✓	×	×	×	**?**	**?**
Fractal [10]	×	Imper.	✓	✓	✓	✓	✓	×
KADL [41]	**?**	Decl.	✓	×	✓	✓	✓	**?**
Koala [52]	✓	Decl.	✓	**?**	×	×	✓	×
LEDA [12]	×	Imper.	✓	×	✓	✓	✓	×
MAE [43]	×	Decl.	✓	✓	✓	✓	×	✓
MontiArcAutomaton [42]	✓	Decl.	✓	×	✓	✓	✓	✓
OOADL [50]	×	Imper.	✓	×	×	×	×	✓
π-ADL [38]	×	Imper.	✓	**?**	✓	✓	✓	×
PiLar [14]	×	Imper.	✓	**?**	✓	✓	✓	×
PRISMA [40]	×	Imper.	✓	×	×	×	✓	✓
Rapide [27]	×	Decl.	✓	×	✓	✓	✓	✓
Scud-ADL [55]	×	**?**	✓	×	✓	✓	**?**	×
SOADL [24]	×	Decl.	✓	×	✓	✓	×	✓

TABLE I
CLASSIFICATION OF DYNAMIC C&C ADLS ACCORDING TO THE SIX IDENTIFIED RECONFIGURATION DIMENSIONS.

Dynamic reconfiguration is related to self-configuration and self-management of self-adaptive software [45]. A survey of dynamic architecture specifications for self-management [8] investigated 14 specification mechanisms and produced a taxonomy comprising the initiation and selection of reconfiguration, available reconfiguration operations, and reconfiguration management. The survey's notion of dynamic reconfiguration agrees with the notion introduced in this work. The survey [8] includes the formal basis that reconfiguration mechanisms rely on (*i.e.,* graph-based, process algebra based, logic-based, and other approaches). Our classification includes more ADLs and abstracts from theoretical backgrounds. The survey [8] defines an ADL's reconfiguration mechanisms to be centralized if dynamic reconfiguration is solely handled by specialized components. Centralized approaches as defined in [8] are not self-directed (cf. Sec. III) as in these approaches specialized components manage the reconfiguration of other components. An ADL solely supports restricted reconfiguration (cf. Sec. III) if it solely supports pre-defined selection as defined in [8].

VI. CONCLUSION

We examined C&C ADLs and identified six dimensions of dynamic reconfiguration. Slightly less than half of the ADLs under investigation allow dynamic reconfiguration between predefined configurations. The others allow reconfiguration between configurations not predefined at design time. In most examined ADLs that allow reconfiguration between predefined configurations, only, the specification style for describing reconfiguration mechanisms is declarative. However, there are also ADLs providing a declarative specification style in combination with arbitrary reconfiguration. All but one ADL support programmed reconfiguration, whereas the minority supports ad-hoc reconfiguration. In all but one ADL where dynamic component instantiation is possible, dynamic component removal is also possible. Nearly half of the examined ADLs support modeling self-directed programmed reconfiguration, only. Surprisingly, only half of the ADLs support triggered reconfiguration. Finally, there appears to be no general consensus on modeling dynamic reconfiguration in C&C ADLs.

REFERENCES

[1] Jonathan Aldrich, Craig Chambers, and David Notkin. ArchJava: Connecting Software Architecture to Implementation. In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, 2002.

[2] Robert Allen, Rémi Douence, and David Garlan. Specifying and Analyzing Dynamic Software Architectures. In *International Conference on Fundamental Approaches to Software Engineering (FASE'98)*, 1998.

[3] Abdelkrim Amirat and Mourad Oussalah. C3: A Metamodel for Architecture Description Language Based on First-Order Connector Types. In *11th International Conference on Enterprise Information Systems (ICEIS 2009)*, pages 76–81, 2009.

[4] Vincent Aravantinos, Sebastian Voss, Sabine Teufl, Florian Hölzl, and Bernhard Schätz. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. In *Joint proceedings of ACES-MB 2015 – Model-based Architecting of Cyber-physical and Embedded Systems and WUCOR 2015 – UML Consistency Rules*, 2015.

[5] AutoFocus 3 Website. http://af3.fortiss.org/. Accessed: 2016-01-18.

[6] R. Bashroush, I. Spence, P. Kilpatrick, T. J. Brown, W. Gilani, and M. Fritzsche. ALI: An Extensible Architecture Description Language for Industrial Applications. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, pages 297–304, March 2008.

[7] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-Based Performance Prediction with the Palladio Component Model. In *Proceedings of the 6th International Workshop on Software and Performance*, 2007.

[8] Jeremy S. Bradbury, James R. Cordy, Juergen Dingel, and Michel Wermelinger. A Survey of Self-management in Dynamic Software Architecture Specifications. In *Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems*, 2004.

[9] Manfred Broy, Franz Huber, and Bernhard Schätz. AutoFOCUS – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik-Forschung und Entwicklung*, 1999.

[10] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL Component Model and Its Support in Java. *Software - Practice and Experience*, 2006.

[11] Arvid Butting, Arne Haber, Lars Hermerschmidt, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language. In *Modelling Foundations and Applications (ECMFA'17), Held as Part of STAF 2017*, pages 53–70. Springer International Publishing, 2017.

[12] Carlos Canal, Ernesto Pimentel, and José M. Troya. Specification and Refinement of Dynamic Software Architectures. In *Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, 1999.

[13] Damien Cassou, Pierrick Koch, and Serge Stinckwich. Using the DiaSpec design language and compiler to develop robotics systems. In *Proceedings of the Second International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob)*, 2011.

[14] Carlos E. Cuesta, Pablo de la Fuente, Manuel Barrio-Solórzano, and M. Encarnación Gutiérrez Beato. An "abstract process" approach to algebraic dynamic architecture description. *The Journal of Logic and Algebraic Programming*, 2005.

[15] Vincent Debruyne, Françoise Simonot-Lion, and Yvon Trinquet. EAST-ADL - An Architecture Description Language. In *Architecture Description Languages*. Springer, 2005.

[16] Robert DeLine. Toward User-Defined Element Types and Architectural Styles. In *Proceedings of the 2nd International Software Architecture Workshop*, 1996.

[17] Jim Dowling, Vinny Cahill, and Siobhàn Clarke. Dynamic Software Evolution and The K-Component Model. In *Workshop on Software Evolution, OOPSLA*, volume 2001, 2001.

[18] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[19] Robert France and Bernhard Rumpe. Model-Driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering 2007 at ICSE.*, 2007.

[20] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, 2014.

[21] Zhitao Fu, Tong Li, and Yan Hu. An Approach to Aspect-Oriented Software Evolution Process Architecture. In *Intelligent Computation Technology and Automation, 2009. ICICTA '09. Second International Conference on*, 2009.

[22] Milena Guessi, Everton Cavalcante, and Lucas B. R. Oliveira. Characterizing Architecture Description Languages for Software-Intensive Systems-of-Systems. In *Proceedings of the third international workshop on software engineering for systems-of-systems*, 2015.

[23] Robert Heim, Oliver Kautz, Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. Retrofitting Controlled Dynamic Reconfiguration into the Architecture Description Language MontiArcAutomaton. In *Software Architecture - 10th European Conference (ECSA'16)*, volume 9839 of *LNCS*, pages 175–182, Copenhagen, Denmark, December 2016. Springer.

[24] Xiangyang Jia, Shi Ying, Honghua Cao, and Donghui Xie. A New Architecture Description Language for Service-Oriented Architecure. In *Sixth International Conference on Grid and Cooperative Computing*, 2007.

[25] Ackbar Joolia, Thais Batista, Geoff Coulson, and Antonio T.A. Gomes. Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform. In *5th Working IEEE/IFIP Conference on Software Architecture, 2005. WICSA 2005.*, 2005.

[26] Rohit Khare, Michael Guntersdorfer, Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. xADL: Enabling Architecture-Centric Tool Integration with XML. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001.

[27] David C. Luckham and James Vera. An Event-Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, 1995.

[28] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying Distributed Software Architectures. In *Proceedings of the 5th European Software Engineering Conference*, 1995.

[29] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. What Industry Needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering*, 2013.

[30] Nenad Medvidovic. ADLs and Dynamic Architecture Changes. In *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, 1996.

[31] Nenad Medvidovic, Eric M. Dashofy, and Richard N. Taylor. Moving architectural description from under the technology lamppost. *Information and Software Technology*, 2007.

[32] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 2000.

[33] Robin Milner. *Communicating and Mobile Systems: The π-calculus*. USA: Cambridge University Press, 1999.

[34] Jelena Mirkovic, Ted Faber, Paul Hsieh, Ganesan Malaiyandisamy, and Rashi Malaviya. DADL: Distributed Application Description Language. *USC/ISI Technical Report# ISI-TR-664*, 2010.

[35] Mubarak Mohammad and Vangalur Alagar. A Formal Approach for the Specification and Verification of Trustworthy Component-Based Systems. *Journal of Systems and Software*, 84(1), 2011.

[36] Antonio Navarro Pérez and Bernhard Rumpe. Modeling Cloud Architectures as Interactive Systems. In *Proceedings of the 2nd International Workshop on Model-Driven Engineering for High Performance and Cloud Computing*, 2013.

[37] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.3 (10-05-05), 2010. http://www.omg.org/spec/UML/2.3/Superstructure/PDF/ [Online; accessed 2015-12-17].

[38] Flavio Oquendo. π-ADL: an Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM SIGSOFT Software Engineering Notes*, 2004.

[39] M. Ozkaya and C. Kloukinas. Are We There Yet? Analyzing Architecture Description Languages for Formal Analysis, Usability, and Realizability. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 2013.

[40] Jennifer Pérez, Nour Ali, Jose A Carsí, and Isidro Ramos. Designing software architectures with an aspect-oriented architecture description language. In *International Symposium on Component-Based Software Engineering*, 2006.

[41] Pascal Poizat and Jean-Claude Royer. A Formal Architectural Description Language based on Symbolic Transition Systems and Modal Logic. *Journal of Universal Computer Science*, 2006.

[42] Jan Oliver Ringert, Alexander Roth, Bernhard Rumpe, and Andreas Wortmann. Language and Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems. *Journal of Software Engineering for Robotics (JOSER)*, 2015.

[43] Roshanak Roshandel, André Van Der Hoek, Marija Mikic-Rakic, and Nenad Medvidovic. Mae—a system model and environment for managing architectural evolution. *ACM Trans. Softw. Eng. Methodol.*, 13(2):240–276, April 2004.

[44] Jungwoo Ryoo and Hossein Saiedian. AVDL: A highly adaptable architecture view description language. *Journal of Systems and Software*, 2006.

[45] Mazeiar Salehie and Ladan Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2009.

[46] Christian Schlegel, Andreas Steck, and Alex Lotz. Model-Driven Software Development in Robotics: Communication Patterns as Key for a Robotics Component Model. In *Introduction to Modern Robotics*. iConcept Press, 2011.

[47] Robert W Schwanke, Veronika A Strack, and Thomas Werthmann-Auzinger. Industrial Software Architecture with Gestalt. In *Proceedings of the 8th International Workshop on Software Specification and Design*, page 176. IEEE Computer Society, 1996.

[48] Vivek Singhal and Don S. Batory. P++: a Language for Software System Generators. Technical report, University of Texas at Austin, Department of Computer Sciences, 1993.

[49] Adel Smeda, Adel Alti, and Abbdellah Boukerram. An Environment for Describing Software Systems. *WSEAS Transactions on Computers*, 2009.

[50] Jeffrey J P Tsai and Kuang Xu. Architecture Specification of Multimedia Software Systems. In *IEEE International Conference on Multimedia Computing and Systems, 1999*, 1999.

[51] Naoyasu Ubayashi, Jun Nomura, and Tetsuo Tamai. Archface: A Contract Place Where Architectural Design and Code Meet Together.

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.