Towards a Systematic Approach for Software Synthesis

Hamid Bagheri and Kevin Sullivan

University of Virginia,
151 Engineer's Way,
Charlottesville, VA 22903 USA
{hb2j, sullivan}@virginia.edu

Abstract. Development of software-intensive systems nowadays rely extremely on middleware platforms as a major building block especially to handle the distribution issues. This dependency has become even more crucial in the distributed embedded systems environment. As such, the architectural choices of such systems are being driven by middleware platforms. However, diversity and high frequency of evolution in middleware platforms lead to architectural models becoming obsolete relatively rapidly, which is in distinct contrast to the resistance nature of software architecture to frequent change. We believe that the key to this is to abstract away from architectural platforms and their induced architectural styles to more abstract representation of applications. In recent work we have shown that architecture-independent application models, developed using modern model-based development (MBD) techniques, can be mapped to application architectures in a variety of architectural styles. Although the work provided an important proof of concept, the styles, or architectural spaces, to which application models were being mapped were simple, idealized styles. Di Nitto and Rosenblum recognized that middleware and similar platforms induce defacto architectural styles. In this paper, we discuss some of the related issues we are addressing in our research towards a systematic approach for software synthesis.

1 Introduction

Software-intensive systems are continuously growing in size and complexity. In recent years, they have ever more migrated from the traditional, localized setting to highly distributed, and embedded environments. While software engineering researchers and practitioners have recognized software architecture as a promising means of managing the complexity of software systems in general [17, 15], other studies have shown its significant role in developing distributed embedded systems [16, 11].

Distributed embedded systems, furthermore, rely extremely on middleware as a major building block to handle the distribution issues [8]. However, because of the pervasiveness of middleware platforms, the architectural choices are being driven by such platforms and since they are both changing rapidly and are very diverse, the architecture of most of software-intensive systems and distributed embedded systems, in particular, are accidental nowadays [5]. This is in distinct contrast to the way that software architecture is designated to be, i.e. software architecture typically comprises the early decisions made about a system, and is consequently very difficult to change [17]. As such, there is a pressing need to understand how to make architectural changes much more readily.

We believe that the key to this is to abstract away from architectural styles and architectural platforms to more abstract representation of applications. In recent work [2] we demonstrated the feasibility of separating and combining formal representations of application properties and architectural styles, respectively. In doing so, we defined style-specific *architectural mappings* that relate style-independent application models to architectural models in given styles.

We have continued studying the notion of architectural mappings and the ways in which they can be defined and exploited in system development. In this paper we discuss some of the issues we are addressing in our work.

2 Previous Work

Our earlier work [2] suggests that the concept of *application type*, parallel to the notion of architectural style, is important, and that it is possible to separate, and combine formal representations of, application contents and architectural styles, respectively. To that end, we formulate the mapping problem as one of finding satisfying solutions to a specification that combines an application model of a given application type, with an architectural style specification, and with rules for mapping application models of the given type to architectural models in the given style. We have implemented such mappings using Alloy as a language and satisfaction engine [9].

In view of the increasing platform diversity and complexity of software-intensive systems, model-based development (MBD) approach has become a viable means to address system-integration issues in the early phases of development. In recent work [3] we showed that software architectural styles can serve as analogs to choices of platforms in model-based development, and that the concept of application type leads naturally to an abstract, user-friendly approach to application modeling. That is, the proposed separation of concerns supports a model-based development and tools approach to architectural-style-independent application modeling, and architecture synthesis with style as a separate design variable. More precisely, by providing a prototype tool, *Monarch* [1], we illustrated how an approach giving as inputs the formal specifications of application descriptions and architectural styles can be implemented in a computationally effective manner by being placed within the formal framework of MBD.

These work provided a proof of concept of the feasibility of the proposed formal architectural mappings in an automated way. However, it suffers from some shortcomings especially with respect to the pervasiveness of middleware in system development. In the next section, we discuss some of the ongoing issues we are addressing in our work.

3 Proposed Work

3.1 Middleware-induced architectural styles

Middleware infrastructures are emerging to be used extensively as a major building block in facilitating system development especially in the large-scale distributed systems. Notwithstanding the several categories of middleware platforms, there are numerous middleware infrastructures from which to choose such as TAO [13], Aura [16], PolyORB [18] and even Enterprise JavaBeans (EJB) [6].

An approach that may be commonly used and could be ineffective and counterproductive in practice is that a middleware is chosen first with respect to its provided services and in turn leads to an unnecessary impact over the system's architecture. In contrast, deferring middleware decisions has several advantages such as separation of concerns and promoting level of abstraction in the early phase of software design [12, 17]. Furthermore, a middleware decision is not independent of the system's architecture. As such, decisions made during the development of the system's architecture may limit the decision space of the middleware that will be used to implement the system.

Problems can arise when the architectural styles chosen for the application conflict with the assumptions of the chosen middleware. Blair et al. [4] argue that the architectural models can be used in systematic synthesis of middleware configurations. Particularly, it would be helpful to consider structural and behavioral constraints implied by middleware infrastructures as architectural styles [12]. Formal definition of these styles will allow architects to exploit these styles in a way that avoids unintentional mismatch between the required application's properties and the constraints imposed by the middleware-induced architectural styles.

Although a number of approaches explored to separate and relate middleware infrastructures and architectural styles induced by them in various domains (e.g. embedded systems [11], web-based systems [7]) insufficient progress has been made on mapping architecture-independent application models into the modern and practical, middleware-induced architectural styles and in turn, into the realized architecture implementations. We envisage an approach that is based on model-based development to mapping architecture-independent application models, considered as platform-independent models, to the realized architecture implementations in conformance with the architectural styles that are induced by middleware platforms and other complex and practical application frameworks. This approach can be used to automate the derivation of the architectural models (Platform-specific models) from the application models (Platform-independent model) that refines application types.

3.2 Code Generation

The architectural styles so derived promise benefits for both development and maintenance. However, formal specifications often lack bindings to implementation-level constructs. Thereby, it is particularly difficult to verify the fidelity of the developed software system with respect to the formally generated architectural model. To use generated architectural models and stylistic guidelines extracted from the middleware platforms in an effective manner, they should be provided with support for their implementation [14]. Implementing architectural models further is an issue of considerable importance that relates design decisions to implementation elements that realize those decisions [17], which in turn, leads to a gap between the architectural concepts from one side and the constructs of the target programming language from the other side.

There are various kind of tools intended for supporting the implementation of considerable part of code on varying programming languages. However, to our best knowledge neither of them pay enough attention to the key role that architectural styles can play in filling the implementation gap. The lack of flexibility on the subject of the architectural styles is a significant limitation of current approaches to code generation from architectural models [10]. That is, the architect is forced to develop models in a specific architectural style supported by a given approach, rather than a suitable style chosen by the architect.

In this regard, architectural frameworks are emerged to support specific architectural styles. In concrete terms, an architectural framework is a software technologies built upon the functionalities provided by the programming language and the operating system that provides services with respect to supported architectural styles [17]. Architectural frameworks are practical technologies that facilitate the system's development in conformance to specific architectural styles. They are considered as a significant strategy for bridging the gap between architectural models and their associated implemented technologies. We investigate the extensions of our work to include subsequent mappings for synthesis of executable code from formally derived architectural models on the basis of architectural styles for a wide variety of such frameworks that support architectural styles to which the architectural models conform, which in turn returns the responsibility for stylistic decisions to the architect.

4 Conclusion

In this paper we have discussed the role of architectural mappings in synthesis of software implementations from abstract application models. We have also touched upon a number of issues we are exploring in our study of architectural mappings. Consequently, we believe that architectural mappings represent a promising approach to addressing the challenges of software-intensive systems and especially of the embedded systems, and will continue to be a focus of our ongoing research in this domain.

References

1. Monarch tool suite. `http://monarch.cs.virginia.edu/`.
2. H. Bagheri, Y. Song, and K. Sullivan. Architectural style as an independent variable. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE'10)*, 2010.

3. H. Bagheri and K. Sullivan. Monarch: Model-based development of software architectures. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (Models'10)*, 2010.
4. G. S. Blair, L. Blair, V. Issarny, P. Tuma, and A. Zarras. The role of software architecture in constraining adaptation incomponent-based middleware platforms. In *IFIP/ACM International Conference on Distributed systems platforms*, pages 164–184, New York, United States, 2000. Springer-Verlag New York, Inc.
5. G. Booch. The accidental architecture. *IEEE Software*, 23(3):9—11, 2006.
6. L. DeMichiel and M. Keith. Enterprise JavaBeans specification documentation, 2006.
7. S. Giesecke and J. Bornhold. Style-based architectural analysis for migrating a web-based regional trade information system. In *First International Workshop on Web Maintenance and Reengineering (WMR 2006) in conj. with CSMR 2006*, volume 193, pages 15—23, Bari, Italy, 2006. CEUR Workshop Proceedings.
8. V. Issarny, M. Caporuscio, and N. Georgantas. A perspective on the future of middleware-based software engineering. In *2007 Future of Software Engineering*, pages 244–258. IEEE Computer Society, 2007.
9. D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.
10. S. Malek. Effective realization of software architectural styles with aspects. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 313–316, 2008.
11. S. Malek, M. Mikic-Rakic, and N. Medvidovic. A Style-Aware architectural middleware for Resource-Constrained, distributed systems. *IEEE Trans. Softw. Eng.*, 31(3):256–272, 2005.
12. E. D. Nitto and D. Rosenblum. Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In *Proceedings of the 21st international conference on Software engineering*, pages 13—22, Los Angeles, California, United States, 1999. ACM.
13. D. Schmidt and C. Cleeland. Applying patterns to develop extensible ORB middleware. *Communications Magazine, IEEE*, 37(4):54—63, 1999.
14. M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Trans. Softw. Eng.*, 21(4):314–335, 1995.
15. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
16. J. P. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. *In Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29—43, 2002.
17. R. N. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.
18. T. Vergnaud, J. Hugues, L. Pautet, and F. Kordon. PolyORB: a schizophrenic middleware to build versatile reliable distributed applications. In *Reliable Software Technologies - Ada-Europe 2004*, pages 106–119. 2004.