Submitted for publication, 2003.

AcmeStudio: Supporting Style-Centered Architecture Development.

Bradley Schmerl and David Garlan
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
{schmerl,garlan}@cs.cmu.edu

Abstract

Software architectural modeling is crucial to the development of high-quality software. Software engineering tool support is required for this activity, so that models can be developed, viewed, analyzed, and refined to implementations. This support needs to be provided in a flexible and extensible manner so that the tools can fit into a company's process and can use particular, perhaps company-defined, domain-specific architectural styles. Early architecture development environments were restricted to supporting particular architectural styles, and were not easy to tailor to other styles or domains. In this research demonstration, we describe AcmeStudio, a style-neutral architecture development environment that can (1) allow architects to tailor the environment for particular domains; (2) be used as a framework for integrating domain-specific architectural analyses; and (3) can be extended to integrate with other tools that are used in a company's software development process. We illustrate the main features of AcmeStudio and provide some insight into how we have tailored it to real-world domains specific to NASA and Ford Motor Company.

1. Introduction

Developing a software architectural model is seen as a crucial step for producing high quality software. In addition to allowing designers to focus on high-level abstractions such as computational components and their interactions, an architectural model is suitable for analyses that can prevent errors from propagating to later phases of software development. Such analyses include performance analysis [8], simulation [5], and protocol analysis [1].

One of the major difficulties in providing tool support for architectural design and analysis is the need to tailor those capabilities to the application domain. While some analyses may be generally applicable across many domains, typically the more significant forms of analysis take advantage of the particular kind of system built within an organization. For example, representational and analytic needs of a web services domain, which may be concerned with performance and throughput, will be quite different than those for an embedded controller domain, which may be concerned with schedulability and resource allocation.

One possible solution is to create many specialized environments – one for each domain. Indeed, during the first decade of interest in architecture description languages and tools we saw the introduction of dozens of notations and analytical methods, each specialized for some particular family of systems. For example, C2 [10] was restricted to layered event-based systems, and MetaH [11] focused on architectures for embedded avionics control systems.

Constructing a new tool from scratch for each domain incurs a prohibitive cost. On the other hand, it is not desirable to require architects to use tools that are not suitable to their domains. Furthermore, integrating tools to take advantage of their respective benefits is also extremely difficult.

Over the past decade we have been experimenting with another approach. The key idea is to provide a generic infrastructure that can be specialized to particular domains. Specifically, we capture the dimensions of variability and representation for architectural design as an *architectural style*. Each style defines the vocabulary of elements that can be used in the domain, rules specifying how these elements may be assembled, and how they should be depicted in architectural diagrams. Each style may also be coupled with appropriate analysis tools.

In this research demonstration we summarize the lessons we have learned in carrying out such an approach. Using our experience with four generations of architectural design tools we outline what we have come to understand as the essential ingredients for developing successful architectural design tools. These lessons are embodied in our architecture design tool called AcmeStudio. As we will show, one of the most important criteria is to be able to integrate such tools with an organization's existing processes, artifacts, analyses, and assets. We will illustrate these lessons by exhibiting two significant case studies. The first, for Ford Motor Company, involved tailoring AcmeStudio to integrate with Ford's automotive design approach [9]. The second, for NASA Jet Propulsion Laboratory (JPL), sees AcmeStudio act as an architectural tool

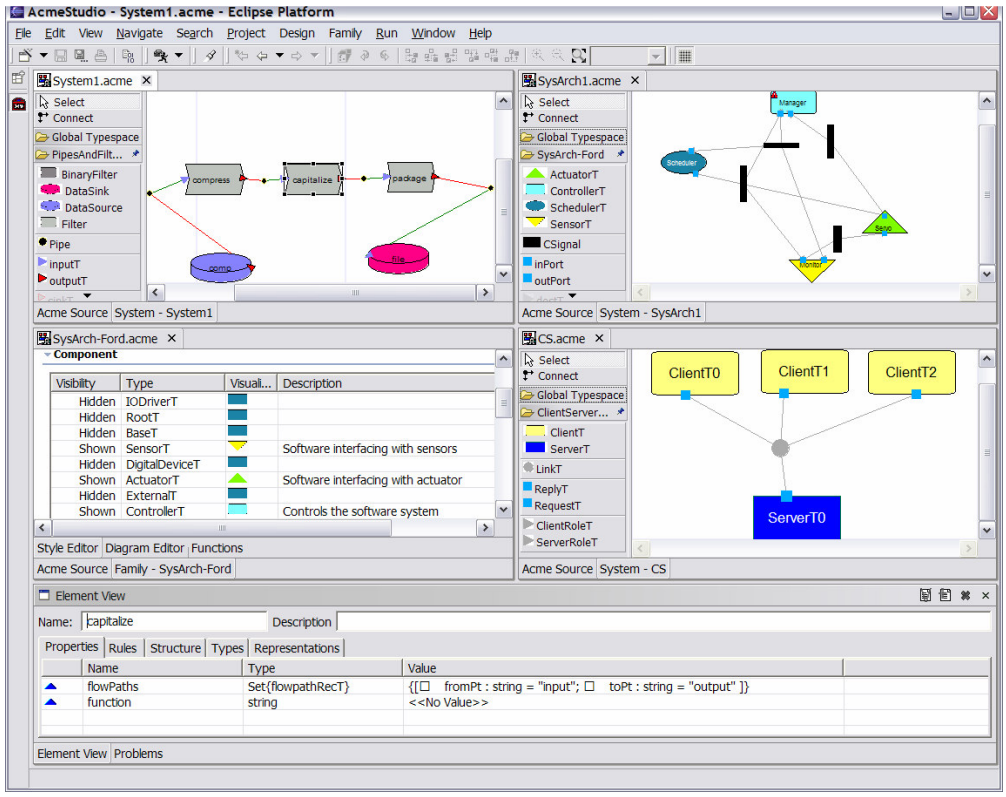

Figure 1. Using AcmeStudio for architectural design.

for fairly complex architectural style [7], and is being tailored to integrate with their requirements and code generation tools.

2. A Tour of AcmeStudio

AcmeStudio is an ADE that supports the Acme architecture description language [4]. Acme is a style neutral ADL that allows architects to define and combine architectural styles, and has well-defined semantics for mixing styles and checking a model's conformance to styles. AcmeStudio is written as a plugin to IBM's Eclipse IDE framework.[1]

In this section we briefly describe how AcmeStudio is used to develop architectural models, and how AcmeStudio can be tailored for different architectural styles.

2.1. Developing Architectural Models

Figure 1 shows a session with AcmeStudio. In this session, three architectural models are being described, and one architectural style is being developed. This figure illustrates AcmeStudio tailored to different styles. On the left of each architecture window is a type palette, which displays the available vocabulary for a particular domain. For example, in the top-left window is an architectural model in a pipe-filter style; the palette allows an architect to drag pipes, filters, data stores, and their associated interfaces into the diagram to create the diagram. The style used for this model is different from the other two depicted in the top and bottom right windows. These models are in the Ford System Architecture automotive style and a client-server style, respectively.

The bottom-most window shows selected elements from the models and styles in more detail, displaying their properties, rules, substructure, and typing. This window also allows the user to enter values for properties, define rules, etc.

2.2. Developing Architectural Styles

A key original aspect of AcmeStudio is the ability to easily and quickly tailor it to specific architectural styles. AcmeStudio achieves this primarily through two means:

1. Providing editors to define Acme architectural styles, including types, depictions of these types, and rules governing their correct compositions; and

[1] Details about Eclipse can be found at http://www.eclipse.org.

2. Acting as an integration framework for plugging in domain specific architectural analyses, providing access to the underlying object model representing the architecture, and hooks for extending the user interface with new views, dialogs, actions, etc.

Having styles as first-class entities in AcmeStudio allows architects to iteratively design, tailor, and combine styles, and have their effects immediately propagated in AcmeStudio. This is in contrast to our previous approach with Aesop [3], and with other architecture tools such as ArchStudio [2], where the iterative nature of style development is hindered by the necessity to recompile the tools.

2.2.1 Defining the Acme style

The bottom left window of Figure 1 shows an architectural style under development. The style editor allows component, connector, and interface types to be defined, as well as required properties and substructure or these types. Furthermore, rules can be assigned to this style that define the correct composition of an architecture in this style. Rules are defined in a first-order predicate logic language extended with architectural primitives [6]. Figure 2 shows the definition of a complex rule for the an architectural style for NASA JPL. This rule partially specifies that there if there is a connection of type C1 between a component of type A and a component of type B, then there must also be a connection of type C2. These rules are checked iteratively by AcmeStudio as an architectural model of this style is created, providing immediate feedback to architects about the correctness of their architectures.

In addition to Acme types, AcmeStudio also provides editors for defining visualizations of these types. This allows an architect to decide the shape, color, size, icons, labels, etc., that should be used in the architectural models (c.f., the different depictions in Figure 1).

Any changes to the architectural style are immediately reflected in all models of that style. In this way, AcmeStudio provides lightweight tailoring and iterative development of styles and models.

2.2.2 Plugging in domain-specific analyses

Typically, different types of analysis will be amenable to different styles. AcmeStudio allows different analysis tools to be integrated, and applied to certain architectural styles. Examples of these analyses that we have integrated are performance analysis based on queuing theory, which requires components and connectors to have properties

```
(Forall b :! B in sys.components |
  (Forall cnp :! P1T in b.ports |
    (Forall a :! A in sys.components |
      (Forall cnr :! P2T in a.ports |
        (connected(cnp, cnr) -> (Exists cqr :! P3T in b.ports |
          Exists cqp :! P4T in a.ports |
            connected(cqr, cqp) ) )) ) ) ) ;
```

Figure 2. Defining an architectural rule.

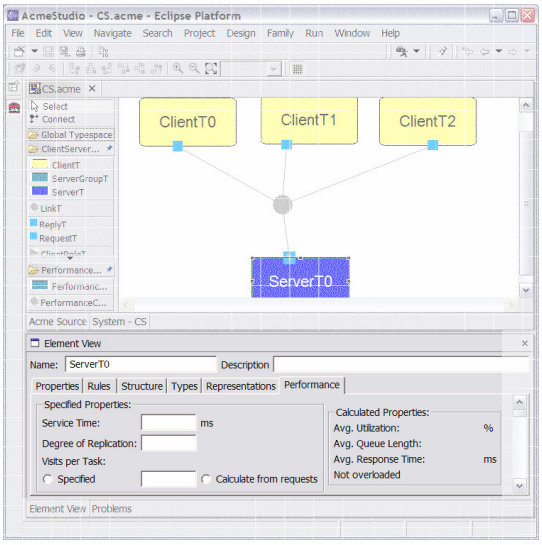

Figure 3. Extending the user interface for specific domains.

defining their service time, delay times, replications, visits, response times, etc; schedulability analysis, which requires components to specify the priorities, deadlines, periodicity, etc;

A key requirement for this ability is to expose certain aspects of AcmeStudio. For example, analysis tools have access to architectural models, and may be notified of changes to the model to update their analysis. Furthermore, AcmeStudio allows analyses to add analysis-specific views and actions to the AcmeStudio user interface, through a combination of standard Eclipse and custom AcmeStudio extension points. Figure 3 gives an example of a new view added for performance analysis. This view extends the Element View window to allow display of the properties pertaining to performance analysis, and allow changes only to those properties that are not automatically calculated by the analysis.

3. Fitting into Company Processes

As an architecture development environment, AcmeStudio provides a comprehensive set of tools for an architect to design software systems. However, no tool is an island; an ADE works best if it can be integrated into a company's software process. AcmeStudio allows this again through plugins that can relate the architectural model to elements not contained in the architecture, and also allow generation of alternative models from an architectural model. We briefly describe our experience with two companies: Ford Motor Company and NASA JPL.

3.1. Ford Motor Company

Ford Motor Company uses Simulink models of its software system components to simulate the behavior of software that will be placed in automobiles. However, creating unified Simulink model out of all the small components in a reliable way is tedious and error prone. We helped Ford split this process into two levels of architectural design, and provided plugins to AcmeStudio to generate unified Simulink models. Briefly, a Ford Engineer constructs a high level architecture of the software, without reference to particular implementations, and then connects them together with abstract connectors. The engineer then associates specific Simulink models with each component in the system. The engineer finally uses AcmeStudio to generate a detailed model, with detailed component interfaces (based on the Simulink definition of components) and detailed connections linking up specific ports in components. This detailed architecture can be checked by style rules for well-formedness and completeness in AcmeStudio. Finally, the plugin generates the composite Simulink model of the whole system. In this way, AcmeStudio dovetails nicely with the existing Ford process and existing tools (i.e., Simulink). This work is more fully described in [9].

3.2. NASA Jet Propulsion Lab

At NASA JPL, we have tailored AcmeStudio to provide an architectural style for their Mission Data System (MDS) architectural framework for autonomous space vehicles. AcmeStudio checks architectural models as changes are made to check some 39 different architectural rules.

Our work now is concentrating on how to integrate AcmeStudio with other tools in JPL's process. Together with JPL, we are developing plugins that integrate AcmeStudio with a database of states that represent the systems engineering part of JPL development. During this stage, states, and their effects on other states, are captured. This information is reflected in the architectural model, and also affects the analyses that can be conducted.

JPL has a significant body of reusable framework code that is used to implement their space systems. Previously, they developed a rudimentary architectural language that could be used to describe the architecture of a system and from which code could be generated. Unfortunately, it was difficult to reason about architectures in this language. We are integrating AcmeStudio with their code generation tools so that code can be generated directly from Acme descriptions that have been analyzed and shown to conform to their stylistic rules.

4. Conclusion

We have illustrated how AcmeStudio supports style-centric architecture design, can be tailored to support multiple architectural styles. Through its use of Eclipse, AcmeStudio serves as an integration framework, allowing domain-specific analyses and views to be incorporated into the architectural design process, and also allows AcmeStudio to be extended to integrate with other tools as part of a company's development process. We have illustrated this with two case studies.

References

[1] Allen, R., Garlan, D., and Ivers, J. "Formal Modeling an Analysis of the HLA Component Integration Standard." Proc. 6th International Symposium on the Foundations of Software Engineering (FSE-6), 1998.

[2] Dashofy, E.M., van der Hoek, A., and Taylor, R.N. "An Infrastructure for the Rapid Development of XML-based Architecture Description Languages." Proc. 24th International Conference on Software Engineering (ICSE2002), 2002.

[3] Garlan, D., Allen, R., and Ockerbloom, J. "Architectural Mismatch or, Why it's hard to build systems out of existing parts." *IEEE Software* 12(6), 1995.

[4] Garlan, D., Monroe, R.T, and Wile, D. "Acme: Architectural Description of Component-Based Systems." *Foundations of Component-Based Systems*, Leavens, G.T, and Sitaraman, M. (eds), Cambridge University Press, 2000.

[5] Luckham, D.C., Kenney, J.J., Augustin, L.M., Vera, J., Bryan, D., and Mann, W. "Specification an Analysis of System Architecture Using Rapide." *IEEE Transactions on Software Engineering,* 21(4):336-355, 1995.

[6] Monroe, R. "Capturing Architecture Design Expertise with Armani." Carnegie Mellon University School of Computer Science Technical Report CMU-CS-98-163, 1998.

[7] Roshandel, R., Schmerl, B.R, Medvidovic, N., Garlan, D., and Zhang, D. "Using Multiple Views to Model and Analyze Software Architecture: An Experience Report". Submitted for publication, 2003.

[8] Spitznagel, B., and Garlan, D. "Architecture-Based Performance Analysis." Proc. the 1998 Conference on Software Engineering and Knowledge Engineering (SEKE'98),1998.

[9] Steppe, K., Bylenok, G., Garlan, D., Schmerl, B., Abirov, K., and Shevchenko, N. "Two-tiered Architectural Design for Automotive Control Systems: An Experience Report." Submitted for publication, 2003.

[10] Taylor, R.N., Medvidovic, N., Anderson, K.M., Whitehead, E.J., Robbins, J.E., Nies, K.A., Oriezy, P., and Dubrow, D.I.. "A Component- and Message-Based Architectural Style for GUI Software." *IEEE Transactions on Software Engineering* 22(6):390-406, 1996.

[11] Vestel, S. "MetaH Programmer's Manual, Version 1.09." Technical Report, Honeywell Technology Center, 1996.