In Proc. 2004 International Conference on Software Engineering, Edinburgh, Scotland, May 2004.

AcmeStudio: Supporting Style-Centered Architecture Development.

Bradley Schmerl and David Garlan
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. 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. In this research demonstration, we describe AcmeStudio, a style-neutral architecture development environment that can be easily specialized for architectural design in different domains.

1. Introduction

Developing a software architectural model is 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, simulation, and protocol analysis.

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 applicable across many domains, 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 (ADLs) and tools we saw the introduction of dozens of notations and analytical methods, each specialized for some particular family of systems. Constructing a new tool from scratch for each domain incurs a prohibitive cost, but on the other hand it is not desirable to require architects to use tools that are not suitable to their domains. 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.

2. A Tour of AcmeStudio

AcmeStudio is an architecture development environment, written as a plugin to IBM's Eclipse IDE Framework [3], that supports the Acme ADL [1]. Acme is a generally purpose ADL supporting component and connector architectures. It provides a system for defining new component and connector types, rules about their composition, and facilities for packaging these up in *architectural styles*.

2.1. Developing Architectural Models

Figure 1 shows AcmeStudio displaying three architectural models and one architectural style being developed. This figure illustrates AcmeStudio support for 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 model. The style used for this model is different from the other two depicted in the top and bottom right windows. These models are in a style developed for Ford Motor Company 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.

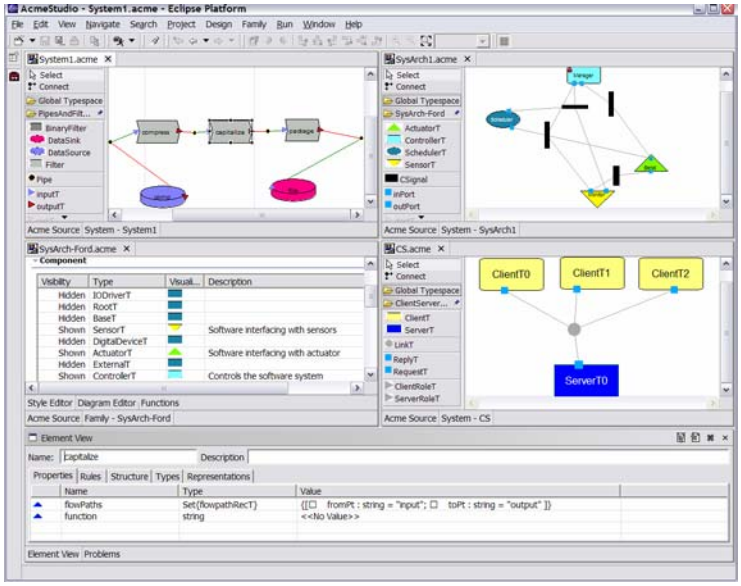

Figure 1. Using AcmeStudio for architectural design.

2.2. Developing Architectural Styles

A key aspect of AcmeStudio is the ability to easily and quickly create new architectural styles. Having styles as first-class entities in AcmeStudio allows architects to iteratively design, tailor, and combine styles.

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 of these types. Furthermore, rules can be assigned to this style to define the correct composition of an architecture in this style. Rules are defined in a first-order predicate logic language extended with architectural primitives [2]. For example, the following is the definition of a complex rule for an architectural style for autonomous robots:

```
(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)…)
```

This rule partially specifies that 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 by AcmeStudio as an architectural model of this style is created, providing feedback to architects about the correctness of their architectures.

In addition to Acme types, AcmeStudio provides editors for defining visualizations of these types, allowing an architect to decide the shape, color, size, icons, labels, layout policies, etc., that should be used in the architectural models.

2.2.2 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 appropriate architectural styles. Integrating these analysis tools allows AcmeStudio to provide architectural analysis beyond the built-in rule evaluation. A key requirement for this ability is to provide an interface that allows analysis tools access to architectural models, and provide notifications of changes to the model to update analysis. AcmeStudio also allows plugins to add analysis-specific views, such as tables, and actions to the user interface. This interface is built on top of the Eclipse plugin infrastructure. Examples of analyses that we have integrated are performance analysis based on queuing theory and schedulability analysis based on rate monotonic theory.

2.3. Applications

We have recently been working with NASA Jet Propulsion Laboratory (JPL) and Ford Motor Company to provide tailored architectural environments based on AcmeStudio for their domains. AcmeStudio, in addition to a number of plugins, is available from http://www.cs.cmu.edu/~acme/AcmeStudio/AcmeStudio.html.

3. The Demonstration

Our demonstration will illustrate how AcmeStudio supports style-centric architecture design and 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. We will demonstrate these analyses and views.

In addition, we will demonstrate extensions to AcmeStudio that we developed to support automotive architectural design for Ford Motor Company and the design of autonomous robots for JPL.

References

[1] 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.

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

[3] Object Technology Internation Inc. "Eclipse Platform Technical Overview." http://www.eclipse.org/whitepapers/eclipse-overview.pdf, 2003.