Acme – An overview

Katrin Hunt

00-707-000
katrin@hunt.ch

Institut für Informatik
Universität Zürich

Abstract: Acme is an ADL (Architecture Description Language) that was developed to create a common interchange format in the software engineering community. Acme is used as ADL itself, as a basis to develop new ADLs and environments or as a basis for translation between two different ADLs. Acme comes along with several tools. AcmeStudio can be downloaded as an Eclipse plug-in and supports visual design of architectures. AcmeLib is a library that supports new development of architecture design tools or environments.

1 Introduction

The fact that a solid software architecture is crucial to the success of large-system development has been acknowledged by the software engineering community a long time ago. For a long time though, there were virtually no tools to assist the software engineer in developing architecture. This, and carelessness when documenting the architecture, led to sloppy and informal software architecture documentations. Software architectures that have been jotted down in an informal way are often ambiguous, allowing different interpretations by the different developers and stakeholders of the project. Furthermore they are also not able to withstand any analysis for consistency or completeness. This stresses the demand for formal, non-ambiguous software architecture documentation.

Numerous research communities have taken up the matter and come up with a wide array of solutions and suggestions to support the architecture-based development. Clements et al [3] stated the following seven rules for sound software architecture documentation:

1. Write Documentation from the Readers point of view
2. Avoid unnecessary repetition
3. Avoid ambiguity
4. Use a standard organization
5. Record rationale
6. Keep documentation current, but not too current
7. Review documentation for fitness and purpose

In order to keep the above specified standards several architecture description languages (ADL) and their accompanying toolsets were / are being developed by academic circles, as well as in the industry. Famous examples are Aesop, C2, Darwin, Rapide, UniCon and Wright.

Most of these languages focus on a certain aspect of software architecture development. Unfortunately they have all been developed quite independently and are therefore not compatible in any way. To profit from different ADLs and their toolsets software engineers have to model their entire architecture in each desired ADL.

To conquer this problem Acme [1] was developed in 97 as a common interchange format for architecture design tools.

2 ADLs in general

An ADL is a language that provides features for modeling a software system's conceptual architecture, distinguished from the system's implementation. [2] ADLs usually provide a concrete syntax, tools for parsing, un-parsing, displaying, compiling, analyzing or simulating architectural descriptions written in the coherent notation. When speaking of ADLs, one often hears of an actual proliferation of ADLs.

On the one hand, it is nice to have so many different ADLs with different strengths. On the other hand it also means, that to make use of them, the software engineer has to master them.

3 Importance of Tools

The main drive behind developing an ADL, is that their formality allows reasoning and manipulation by software tools. These software tools can provide a great benefit for the software engineer. For example a tool that can visualize different views of a software architecture is valuable to simplify communication among the stakeholders of the software project.
The following rule can be stated:
The better the tools that accompany an ADL, the more useful the ADL itself.
Table 1 shows that Darwin is the only ADL that covers all the aspects in question.

	Active Specifi-cation	Multiple Views	Analysis	Refine-ment	Implemen-tation Generation	Dynamism
ACME		X				
Aesop	X	X	X		X	
C2	X	X	X	X	X	X
Darwin	X	X	X	X	X	X
MetaH	X	X	X	X	X	
Rapide		X	X	X	X	X
SADL		X	X	X		
UniCon	X	X	X	X	X	
Weaves		X	X		X	X
Wright		X	X			

Table 1 [2]

4 ACME

The ACME project was started in early 1995. The main initiators are David Garlan, Bob Monroe, and Drew Kompanek at Carnegie Mellon University, and Dave Wile at USC's Information Sciences Institute. The primary goal was to provide a common language, that could be used to support the interchange of architectural descriptions between a variety of architectural design tools.[5]

Acme is a simple, generic language for describing software architectures. It provides a foundation that is general and domain neutral and can be used to develop new tools and notations.

ACME comes with its own notation and toolkit. Currently these are the three most important tools:

1. **ACMEStudio**
   AcmeStudio is a design environment, that is available as an eclipse plug-in. It simplifies the designing process immensely, as one can design on a drag and drop basis without having to actually know the ACME syntax.

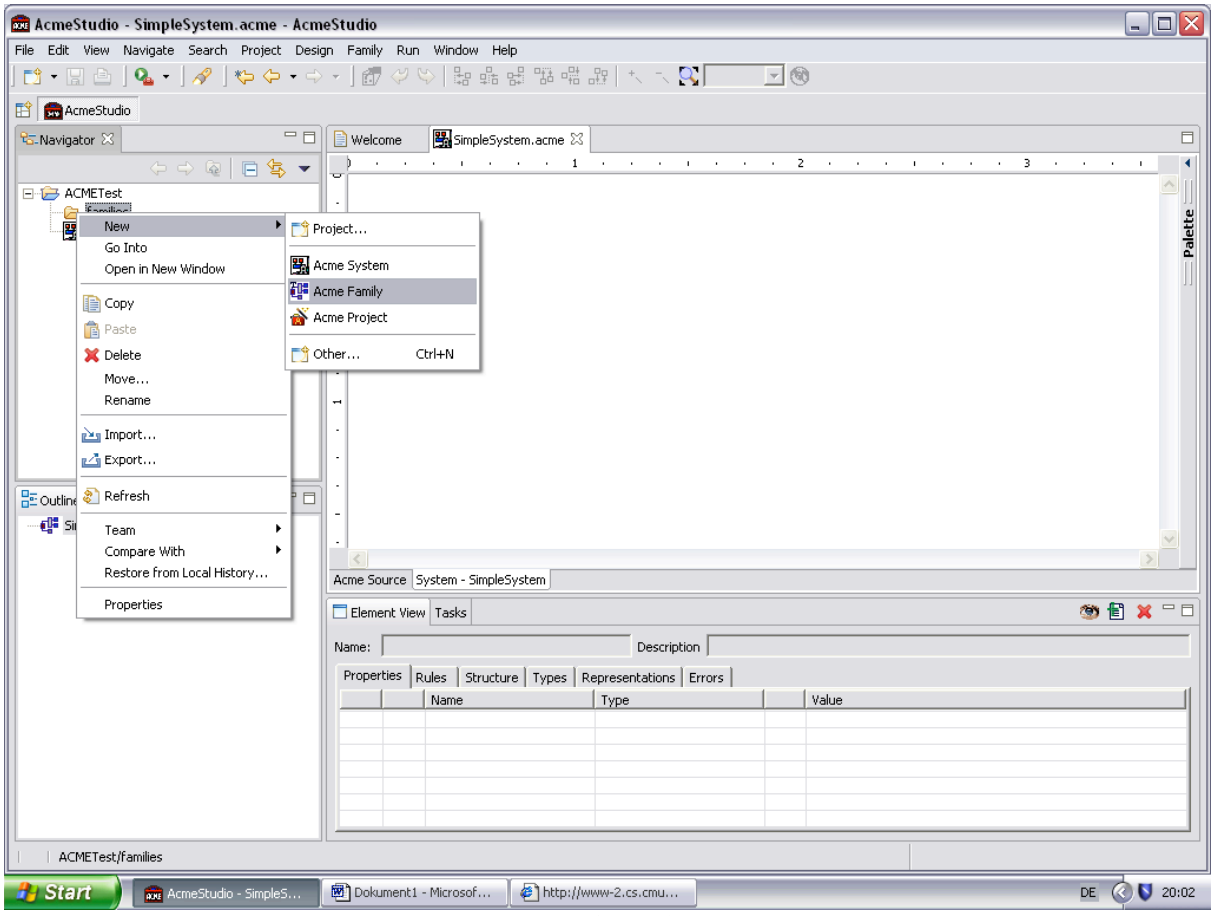

figure 1

2. **AcmeLib**
The Acme Tool Developers Library is available in Java and C++. The library provides the infrastructure for describing, representing, generating, and analyzing software architecture descriptions. It makes it a lot easier to build new software architecture analyzing tools.

3. **Acme-web html documentation generator**
The weblet generator reads in an Acme description and generates a linked collection of HTML documents that visually depict the ACME description. The weblet generator is a read-only visualization solution. Acme designs cannot be modified or created with the webgen tool.[5]
Unfortunately it no longer seems to be free for download, as all links are broken on the ABLE Group page.

According to Garlan, Monroe and Wile [4], there are a dozen more tools as well as three other environments based on Acme in development.

5 Motivation for ACME

As stated in the introduction, with time numerous different ADLs have been developed. Most of these ADLs have been designed to serve an explicit purpose, to enhance a special feautre, model a certain approach.
Software Engineers could profit in different ways from different ADLs. Unfortunately the ADLs are not compatible und to profit from several different languages, the software engineer would have to re-implement the architecture in each language from scratch. To do this the software engineer would have to be knowledgeable up to a certain degree in all languages he wants to make use of.

Acmes aim is to close this gap. It is being developed as a joint effort of the software community to provide a common intermediate representation for a wide variety of architecture tools. [1]
As seen in *table 1*, most ADLs do not offer a wide range of tool support. With an interchange format software engineers would be able to make use of a variety of tools belonging to several ADLs with only one formal description.

6 Goals and Aims

The main goal of ACME is to provide an interchange format for architectural development tools and environments. [2] Next to this primary aim of interchange, a number of secondary aims can be found [1]:

1. *To provide a representational scheme that will permit the development of new tools for analyzing and visualizing architectural structures.*

2. *To provide a foundation for developing new, possibly domain-specific, ADLs.*

3. *To serve as a vehicle for creating conventions and standards for architectural information.*

4. *To provide expressive descriptions that are easy for humans to read and write.*

7 Key Features

The Acme language provides the following key features [5]:

1. an ***architectural ontology*** consisting of seven basic design elements

2. a flexible ***annotation mechanism*** supporting association of non-structural information using externally defined sublanguages

3. a ***type mechanism*** for abstracting common, reusable architectural idioms and styles
   The type mechanism doesn't seem to have been put into use much, as one only comes across it in scientific papers.

4. **an *open semantic framework*** for reasoning about architectural descriptions

7.1 Acme ontology

Acme contains seven types of entities for architectural representation: components, connectors, systems, ports, roles, representations and rep-maps. [5] These seven classes of design elements are fit for defining a structure of an architecture. The ontology as described in the following paragraph could also be called the least common denominator of basically all ADLs [2].

Components: Components represent what one would intuitively draw as boxes in a box-and-line description; the centres of computation. Typical examples are clients, servers, filter, objects etc.

Connectors: Connectors represent interactions between components. Typical examples are pipes, procedure calls etc.

Systems: Systems represent configurations of components and connectors.

Ports: Ports are the interfaces of components, therefore define the point of interaction among component and environment.

Roles: Roles define the interfaces of the connectors.

Representations: Each component or connector can be represented by a lower-level description. These descriptions are called representations. This mechanism permits the use of hierarchical descriptions. Also see *figure 3*

Rep-map: Rep-map stands for representation maps. Rep-maps define the way an internal system representation corresponds to the external interface of the represented component or connector. Also see *figure 3*

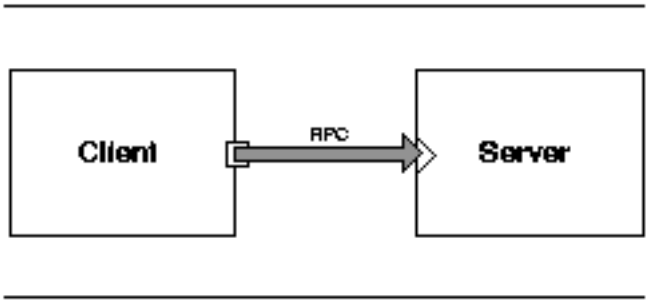

figure 2

Below you see an Acme code example of a simple client server system that describes *figure 2*. The attachments are used to define the topology of the system.

```
System simple_client_server = {

          Component client = {

             Port send-request;

          };

          Component server = {

             Port receive-request;

          };

          Connector rpc = {

             Role caller;
```

```
            Role callee;

          };

        Attachment {

            client.send-request to rpc.caller;

            server.receiver-request to rpc.callee;

}

}
```

7.2 Annotation mechanism

As discussed earlier in this paper, ADLs typically focus on a certain aspect or problem in software architecture design. Therefore additionally to the above explained core elements, which are practically identical with every ADL, there comes auxiliary information corresponding to the ADLs intent and focus. Acme allows to include this auxiliary information in its description by using properties. Each property has a name, an optional type, and a value.[4] Also see *figure 2*

The component of the example above enhanced with properties to correspond UniCon and Aesop could look like this:

```
Component client = {

        Port send-request;

        Property Aesop-style : style-id = client-
        server;

        Property UniCon-style : style-id = cs;

};
```

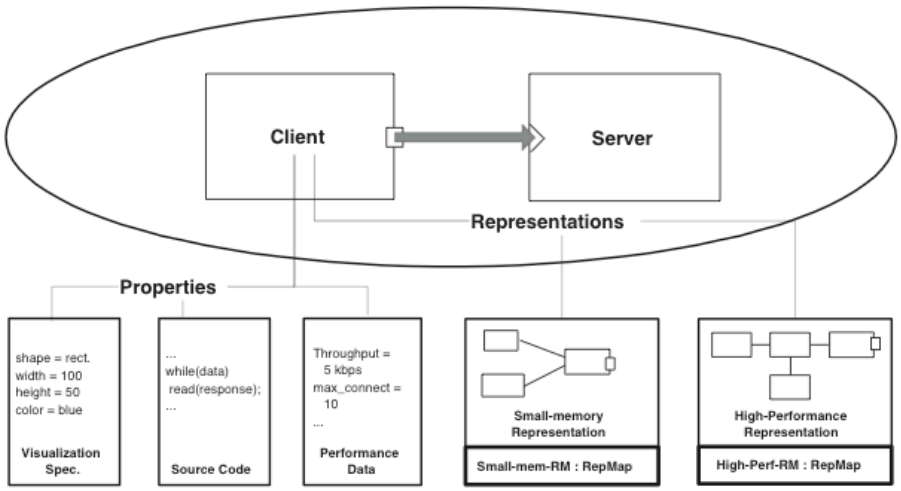

figure 3 [1]

8 An Acme Transformation

As mentioned before, one of Acme's primary goals is to be an ADL interchange format. At the moment there are facilities to translate Acme descriptions into Aesop, Rapide, Wright and back.

Figure 4 shows the translation between Wright and Rapide. The following steps have to be taken in order to take full advantage of all depicted tools:

1. Translate the specification from ADL 1 into an Acme specification with the corresponding annotations / properties from ADL 1.
2. Translate the annotated Acme specification into an Acme specification with annotations from ADL 2.
3. From there the Acme code can directly be translated into the ADL 2 specific description.

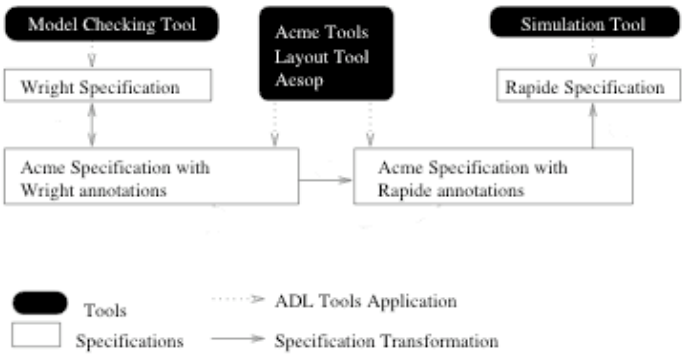

figure 4 [7]

During the process of developing a translator the following crucial decisions have to be considered to simplify or even permit success [7]. One important factor is to limit the class of systems of the ADLs for translation purposes. Another key factor may be not to attempt bi-directionality, as this can complicate the process vastly. Unfortunately this may render the translation tool less useful.

A further problem is that Acme comes with certain constraints that cannot satisfy approaches of some ADLs [2]. For example in Acme a connector cannot be directly linked to another connector. In C2 this is possible.

9 Conclusion

Currently Acme is being used in 3 different ways:

- For starters it is used as a normal ADL. With the AcmeStudio as Eclipse plug-in it provides an easy to grasp environment. The Acme language is also quite intuitively understood and not too difficult to learn.
  Still, to use Acme as ADL only doesn't seem desirable for several reasons. Acme represents the least common denominator of existing ADLs rather than a definition of an ADL. As seen in *table 1* there are not sufficient tools to support analysis form different points of interest.

- In a second role Acme is used as the basis for new architecture design and analysis tools. The AcmeLib may certainly simplify the process of developing a new environment. Nevertheless one is inclined to grade Garlan, Monroe and Wiles [7] observation of a wide, community-spread acceptance as vastly optimistic.

- Acmes third role is that of an integrator between different tools. With translation infrastructures between Aesop, Rapide and Wright a good step

towards that goal has been made. The translations though are not easy and often important and painful trade-offs have to be made, to permit the success of such translations.

Looking back at the 4 goals stated in chapter 6, one can conclude that they have been met. This is nevertheless useless, if the community doesn't make use of Acme. So far Acme didn't experience the predicted [4] break through.

Currently efforts are being made to map Acme into UML [6]. Possibly this evolution could lead to much broader acceptance among the engineering community, as most engineers are familiar with UML. The above discussed translation problems would not vanish though, and therefore it is questionable whether UML can solve the acceptance problem single-handedly.

References

[1] David Garlan, Robert Monroe and David Wile.
ACME: An Architecture Description Interchange Language
In Proceeding of CASCON`97 , November 1997

[2] Nenad Medvidovic and Richard N. Taylor
A Classification and Comparison Framework for Software Architecture Description Languages
University of California, Irvine 1997

[3] Clements P, Bachmann F, Bass L, Garlan D, Ivers J, Little R, Nord R, Stafford J
Documenting Software Architectures: Views and Beyond
Pearson Education, Inc. 2003

[4] David Garlan, Robert Monroe, David Wile
ACME: Architectural Description of Component-Based Systems
2000

[5] Offizielle ACME Homepage
http://www-2.cs.cmu.edu/~Acme/

[6] Miguel Goulão, Fernando Britto e Abreu
Bridging the gap between Acme and UML 2.0 for CBD
Faculdade de Ciencias e Tecnologia- Portugal 2003

[7] David Garlan, Zhenyu Wang
A Case Study in Software Architecture Interchange
Carnegie Mellon University 1999