Bridging the gap between Acme and UML 2.0 for CBD

Miguel Goulão
Departamento de Informática
Faculdade de Ciências e Tecnologia - UNL
2825 Monte de Caparica, Portugal

miguel.goulao@di.fct.unl.pt

Fernando Brito e Abreu
Departamento de Informática
Faculdade de Ciências e Tecnologia - UNL
2825 Monte de Caparica, Portugal

fba@di.fct.unl.pt

ABSTRACT

Architecture Description Languages (ADLs) such as Acme (a mainstream second generation ADL which contains the most common ADL constructs) provide formality in the description of software architectures, but are not easily reconciled with day-to-day development concerns, thus hampering their adoption by a larger community. UML, on the other hand, has become the de facto standard notation for design modeling, both in industry and in academia. In this paper we map Acme modeling abstractions into UML 2.0, using its new component modeling constructs, its lightweight extension mechanisms and OCL well-formedness rules. The feasibility of this mapping is demonstrated through several examples. This mapping bridges the gap between architectural specification with Acme and UML, namely allowing the transition from architecture to implementation, using UML design models as a middle tier abstraction.

Keywords

Component-based architectures, component specification, ADLs, Acme, UML.

1. INTRODUCTION

Software architectural descriptions provide an abstract representation of the components of software systems and their interactions. There are three main streams of architectural description techniques: ad-hoc, OO techniques and ADLs.

Ad-hoc notations lack formality, preventing architectural descriptions from being analyzed for consistency or completeness and for being traced back and forward to actual implementations [2].

To overcome those drawbacks, one can use ADLs, such as Aesop [3], Adage [4], C2 [5], Darwin [6], Rapide [7], SADL [8], UniCon [9], MetaH [10], or Wright [11]. Although with a considerable overlap on the core, each ADL focuses on different aspects of architectural specification, such as modeling the dynamic behavior of the architecture, or modeling different architectural styles. This diversity provides different approaches to solve specific families of problems. However, the interchange of information between different ADLs becomes a major drawback. Developing a single ADL providing all the features of the various ADLs would be a very complex endeavor. Instead, an ADL called Acme [12] emerged as a generic language which can be used as a common representation of architectural concepts in the interchange of information between specifications with different ADLs [13].

Although ADLs allow for architecture in-depth analysis, their formality is not easily reconciled with day-to-day development concerns. OO approaches to modeling, on the other hand, are more widely accepted in industry. In particular, the UML [14] has become both a *de jure* and *de facto* standard. Using it to describe software architectures could bring economy of scale benefits, better tool support and interoperability, as well as lower training costs.

OO methods have some advantages in the representation of component-based systems, when compared to ADLs. There is a widespread notation, an easier mapping to implementation, better tools support and well-defined development methods. But they also have some shortcomings. For instance, they are less expressive than ADLs when representing connections between components.

Several attempts to map ADLs to UML have been made in the past, as we will see in section 2. One motivation for such attempts is to bring architectural modeling to a larger community, through the use of mainstream modeling notations. Another is to provide automatic refinement mechanisms for architectures. UML can be used as a bridge from architectural to design elements [15]. In this paper we will present a more straightforward mapping from Acme to UML, when compared to previous attempts, due to the usage of the new UML 2.0 metamodel.

We will represent the concepts covered by Acme using the candidate UML 2.0 metamodel, which has been partially approved by the OMG recently. It includes UML's infrastructure [16], superstructure [17] and OCL 2.0 [18]. This increases our modeling power due to the new features of the upcoming standard version, mainly in what concerns the representation of components, ports, interfaces (provided or required), and the hierarchical decomposition of components.

This paper is organized as follows. Related work is discussed in section 2. Section 3 contains a formal specification of the mapping between Acme and UML. Section 4 includes a discussion of the virtues and limitations of that mapping. Section 5 summarizes the conclusions and identifies further work.

2. RELATED WORK

A number of mappings among the concepts expressed in ADLs and their representation with UML have been attempted.

A possible strategy is to use UML "as is", in the mapping. In [19], UML is used to express C2 models. In [2], Garlan presents several UML metamodel elements as valid options to express each of the structural constructs defined in Acme. Each mapping becomes the best candidate depending on the goals of the translation from Acme to UML. The semantic mismatch between the ADL and UML concepts is the main drawback of this strategy.

An alternative is to modify the UML metamodel, to increase the semantic accuracy of the mapping [20]. Unfortunately, this drives us away from the standard, and consequently sacrifices existing tool support.

An interesting compromise is to use UML's extension mecha-

nisms to mitigate conceptual mismatches, while maintaining compatibility with the standard metamodel. Examples of this strategy can be found in [15] (C2SADEL to UML), [1] (Acme to UML-RT), and [21] (C2 and Wright to UML). The latter uses OCL constraints on the metamodel elements which is close to the one proposed in this paper, but requires a mapping for each ADL and uses an older and notably less expressive version of UML).

The approach discussed in this paper bridges the gap between software architecture and design using an OO modeling notation. All of the above mentioned mappings were performed with UML 1.x, whereas in our paper we use the new UML 2.0 metamodel elements, which enhance the language's suitability for component-based design.

3. MAPPING ACME INTO UML

From now on we will assume the reader is familiar with Acme, UML and OCL. Due to space constraints, we omit the OCL definition of predicates such as `IsAcmeComponent()`, `IsAcmeConnector()`, `IsAcmePort()`, `IsAcmeRole()`, `IsAcmeProperty()` and others with self explanatory names that will be used in our mapping presentation. `HasNoOtherInterfaces()` is a predicate that denotes that no other interfaces except for the ones defined in ports will be available for a particular component.

3.1 Components

An Acme **component** has **ports**, which act as the component interfaces, **properties**, a **representation** with several bindings (defined as **rep-maps**) and a set of **design rules**. The closest concept in UML is the one of component. To avoid mixing Acme's components with other concepts that we will also represent with UML components, we created a stereotype for Acme components named <<AcmeComponent>>, using `Component` as the base class. Invariant 1 assures these components only have interfaces through Acme ports or properties.

```
context Component inv: -- Invariant 1
   self.IsAcmeComponent() implies
      self.ownedPort->forAll(ap|
         ap.IsAcmePort() or
         ap.IsAcmeProperty()) and
      self.HasNoOtherInterfaces()
```

3.2 Ports

Acme's ports identify points of interaction between a component and its environment. They can be as simple as operation signatures, or as complex as collections of procedure calls with constraints on the order in which they should be called. UML ports are features of classifiers that specify distinct points of interaction between the classifier (in this case, the component) and its environment (in this case, the rest of the system). UML ports have required and provided interfaces, which can be associated to pre and post conditions. We use a combination of UML port and corresponding required and provided interfaces to express Acme's port concept. Acme ports can only be used with Acme components and they have one provided and one required interface.

```
context Port inv: -- Invariant 2
   self.IsAcmePort() implies
      self.owner.IsAcmeComponent() and
      (self.required->size()=1) and
      (self.provided->size()=1)
```
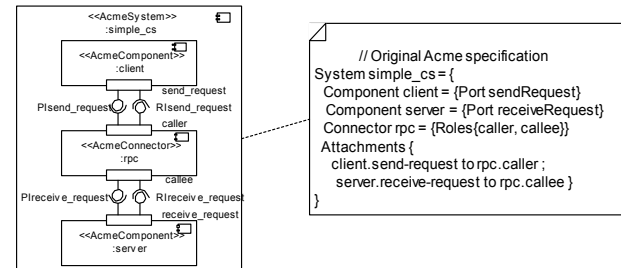
3.3 Connectors

Acme **connectors** represent interactions among components. They are viewed as first class elements in the architecture community. Representing them using UML's assembly connector would be visually appealing, but we would loose expressiveness because Acme connectors may be much more complex than a simple interfaces' match. They can be, for example, a protocol, or a SQL link between two components (a client and a database). Moreover, when reusing components built by different teams it is normal that their interfaces do not match exactly. The connector may provide the required glue between the components and this must be made explicit in the design. In order to represent the concept of connector, which has no semantic equivalent in UML, we use a stereotyped component named <<AcmeConnector>> and ensure that it has no other interfaces than the ones defined through its roles and properties.

```
context Component inv: -- Invariant 3
   self.IsAcmeConnector() implies
      self.ownedPort->forAll(ap|
         ap.IsAcmeRole() or
         ap.IsAcmeProperty()) and
      self.HasNoOtherInterfaces()
```

Although representing a connector with a stereotyped component clutters the outcoming design, it offers the ability to represent the connector as a first class design element, with flexibility in the definition of any protocols it may implement. Consider the example in Figure 1, where the components `client` and `server` have interfaces that do not match, but the `rpc` connector implements a protocol to make both components interact. We have included provided and required interfaces in both ends of the connector, to illustrate that it provides bi-directional communication abilities.

Figure 1 – Using the <<AcmeConnector>>

3.4 Roles

In Acme, **roles** are related to connectors the same way as ports are related to components. Thus, it makes sense to represent Acme roles as constrained UML ports, through the use of the <<AcmeRole>> stereotype.

```
context Port inv: -- Invariant 4
   self.IsAcmeRole() implies
      self.owner.IsAcmeConnector() and
      (self.required->size()=1) and
      (self.provided->size()=1)
```

3.5 Systems

An Acme system represents a graph of interacting components. The UML's concept of package (with the standard <<subsystem>> stereotype) represents a set of elements, rather than the structure containing them and is not suitable for defining system-

level properties. To avoid such problems we use the constrained component stereotype <<AcmeSystem>>, with the following constraints:

```
context Component inv: -- Invariant 5
   self.IsAcmeSystem() implies
      self.contents()->select(el|
         el.IsKindOf(Component))->asSet()
            ->forAll(comp|
               comp.IsAcmeComponent() or
               comp.IsAcmeConnector())

context Component inv: -- Invariant 6
   self.IsAcmeSystem() implies
      self.contents()->select(el|
         el.IsKindOf(Port))->asSet()
            ->forAll(prt|
               prt.IsAcmePort() or
               prt.IsAcmeRole() or
               prt.IsAcmeProperty())
context Component inv: -- Invariant 7
   self.IsAcmeSystem() implies
      self.ownedPort->forAll(ap|
         ap.IsAcmePort() or
         ap.IsAcmeRole() or
         ap.IsAcmeProperty()) and
      self.HasNoOtherInterfaces()
```
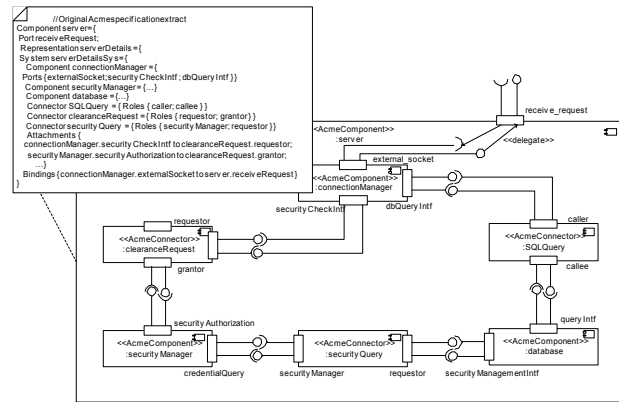
3.6 Representations

Acme's **representations** provide the mechanism to add detail to components and connectors. Acme **rep-maps** are used to show how higher and lower-level representations relate to each other. We will use the features for packaging components of UML 2.0 to express representations. UML provides two wiring elements (in the UML specification, they are referred to as "specialized connectors"): `assembly` and `delegation`. The former provides a containment link from the higher level component to its constituent parts, while the latter provides the wiring from higher level provided interfaces to lower level ones, and from lower level required interfaces to higher level ones. A `delegation` corresponds to Acme's rep-map concept. To ensure components are connected to other components through connectors, we need to constrain all assembly connectors to link ports to roles.

```
context connector inv: -- Invariant 8
  self.kind = #assembly implies
    self.end->(exists(cp|cp.role.IsAcmePort())
          and exists(cr|cr.role.IsAcmeRole()) )
```

Figure 2 depicts the specification of `server`. The wiring between the internal structure of server – a system which contains a topology with three components and the connectors among them – and the server's own ports is achieved with the usage of the <<delegate>> connectors. Although Acme explicitly uses the concepts of representation and system for defining subsystems, we make them implicit in our mapping. Making them explicit would not improve the expressiveness of the resulting design and would clutter the diagram by creating an extra level of indirection.

Figure 2 – Detailing a component specification

3.7 Properties

Properties represent semantic information about a system and its architectural elements. To allow automatic reasoning on them, using OCL, we can make these properties available outside the component's internal scope. Ports can be typed with a provided interface that allows the component user to access its properties. The downsides of representing Acme properties as UML ports are that by doing so we are cluttering the design and extending the interfaces provided by the design element. An <<AcmeProperty>> port owns a single provided interface that must provide get and set operations for the property's value and type.

```
context Port inv: -- invariant 9
   self.IsAcmeProvided() implies
      (self.required->IsEmpty()) and
      (self.provided->size()=1)
```

3.8 Constraints (invariants and heuristics)

Constraints allow the specification of claims on how the architecture and its components are supposed to behave. While invariants are conditions that must hold at all times, heuristics are constraints that should hold, although breaking them is possible. In UML, we can express design constraints through OCL. These constraints can be pre-conditions, post-conditions or invariants. Acme's notion of invariant can be directly mapped to its OCL counterpart. However, there is no direct UML semantic equivalent for the notion of heuristic. This could be circumvented by creating the <<AcmeConstraint>> stereotype as a specialization of the UML Constraint metaclass. The former would have an enumerated attribute with two allowed values: `invariant` and `heuristic`.

3.9 Styles and Types

An architectural style defines a vocabulary of design elements and the rules for composing them. It is used in the description of families of architectures. Since we have created stereotypes for the several UML constructs used in this Acme to UML mapping, we can now specify architectural styles using these stereotyped elements.

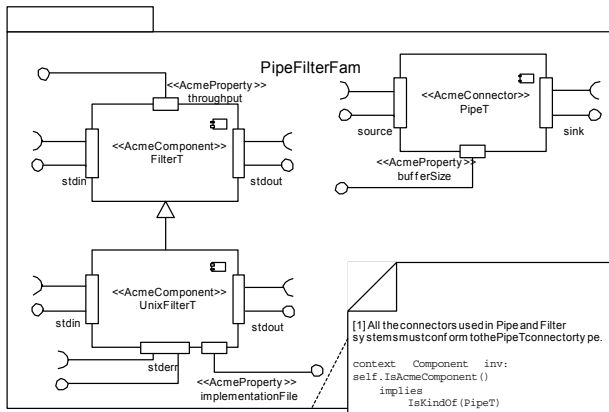**Figure 3- The pipe and filter family**

Figure 3 represents the pipe and filter family, an architectural style that defines two types of components, `FilterT` and `Unix-FilterT`, a specialization of `FilterT`. The architectural style is defined by means of a UML package, as the family definition does not prescribe a particular topology. It does, however, establish an invariant that states that all the connectors used in a pipe and filter system must conform to `PipeT`.

4. DISCUSSION

The presented mapping from Acme to UML is more straightforward than previous approaches. This mainly results from the increased expressiveness provided by the new UML 2.0 design elements. From a structural viewpoint, representing a topology is fairly simple when using UML. This is mainly due to the relative closeness of the sort of structural information that we want to express both at the architectural and design levels. In both cases we have to identify components and the connections among them, possibly at different levels of abstraction.

However, while a connector is regarded as a first class design element by the architecture community, it has no direct mapping in UML 2.0. Our proposal is to promote connectors to first class design elements, by representing them as stereotyped components. This seems to be a good option, considering that the evolution of CBD should provide us with an increasing number of off-the-shelf components and that, the complexity of building component-based software is shifting to the production of glue code. Representing connectors as stereotyped components gives us the extra flexibility to meet this challenge.

The representation of properties is not an easy nut to crack. Perhaps they could be more suitably defined at the meta-level, rather than using the `<<AcmeProperty>>` ports for this purpose, but this still requires further research.

Heuristics are also complex to map directly to UML, as UML provides no direct representation for this concept, although we can use OCL to deal with this problem.

Since Acme does not provide a direct support for component dynamics specification, in this paper we do not address it. Nevertheless, we could use properties to annotate the architectural entities with information on their expected behavior. For instance, a connector may have a property specifying its protocol with some formalism (e.g. Wright). We could use UML's behavioral modeling features similarly, thus complementing the structural

information in the mapped specification with a behavioral specification of the design elements used.

5. CONCLUSIONS

We have shown the feasibility of expressing architectural information expressed in Acme using the UML 2.0. It is possible to obtain a mapping from a given ADL to UML, through a two-step approach. We could first map the architecture from the original ADL to Acme and then use the mapping proposed in this paper to obtain the corresponding specification in UML. Details lost in the ADL to Acme conversion can always be added later to the resulting UML specification.

The proposed mapping builds upon the added expressiveness of UML 2.0 for architectural concepts, when compared to UML's previous versions. The availability of components with ports typed by provided and required interfaces has proved to be a step forward in the exercise of bridging the gap between architectural and design information. This improves traceability between architectural description and its implementation, using the design as a middle layer between them. This traceability is relevant for keeping the consistency between the architecture, design and implementation of a software system.

The proposed mapping focuses mainly on structural aspects and design constraints. Although it also points out to ways of dealing with the definition of system properties, including semantics and behavioral specification, further research is required to provide more specific guidance on these aspects.

REFERENCES

[1] S.-W. Cheng and D. Garlan, "Mapping Architectural Concepts to UML-RT", PDPTA'2001, Monte Carlo Resort, Las Vegas, Nevada, USA, 2001.

[2] D. Garlan and A. J. Kompanek, "Reconciling the Needs of Architectural Description with Object-Modeling Notations", <<UML>> 2000, York, UK, 2000.

[3] D. Garlan, R. Allen, and J. Ockerbloom, "Exploiting style in architectural desing environments", SIGSOFT'94: The Second ACM Symposium on the Foundations of Software Engineering, 1994.

[4] L. Coglianese and R. Szymanski, "DSSA-ADAGE: An Environment for Architecture-based Avionics Development", AGARD'93, 1993.

[5] N. Medvidovic, P. Oreizy, J. E. Robbins, and R. N. Taylor, "Using object-oriented typing to support architectural design in the C2 style", SIGSOFT'96: Fourth ACM Symposium on the Foundations of Software Engineering, 1996.

[6] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, "Specifying distributed software architectures", Fifth European Software Engineering Conference, ESEC'95, 1995.

[7] D. C. Luckham, L. M. Augustin, J. J. Kenney, J. Veera, D. Brian, and W. Mann, "Specification and analysis of system architecture using Rapide", *IEEE Transactions on Software Engineering*, vol. 21, No.4, pp. 336-355, 1995.

[8] M. Moriconi, X. Qian, and R. Riemenschneider, "Correct architecture refinement", *IEEE Transactions on Software Engineering*, vol. 21, No. 4, pp. 356-373, 1995.

[9] 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 Transactions on Software Engineering*, vol. 21, No. 4, pp. 314-335, 1995.

[10] P. Binns and S. Vestal, "Formal real-time architecture specification and analysis", Tenth IEEE Workshop on Real-Time Operating Systems and Software, New York, USA, 1993.

[11] R. Allen and D. Garlan, "A Formal Basis for Architectural Connection", *ACM Transactions on Software Engineering and Methodology*, vol. 6, pp. 213-249, 1997.

[12] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural Description of Component-Based Systems", in *Foundations of Component Based Systems*, G. T. Leavens and M. Sitaraman, Eds.: Cambridge University Press, 2000, pp. 47-68.

[13] M. R. Barbacci and C. B. Weinstock, "Mapping MetaH into ACME", Carneggie Mellon University / Software Engineering Institute, Technical Report CMU/SEI-98-SR-006, July 1998.

[14] OMG, "OMG Unified Modeling Language Specification. Version 1.5", Object Management Group March 2003.

[15] A. Egyed and N. Medvidovic, "Consistent Architectural Refinement and Evolution using the Unified Modeling Language", 1st Workshop on Describing Software Architecture with UML, co-located with ICSE 2001, Toronto, Canada, 2001.

[16] U2-Partners, "3rd revised submission to OMG RFP ad/00-09-01: Unified Modeling Language: Infrastructure - version 2.0", U2-Partners January 2003.

[17] U2-Partners, "2nd revised submission to OMG RFP ad/00-09-02: Unified Modeling Language: Superstructure - version 2.0", U2-Partners January 2003.

[18] Boldsoft, Rational, IONA, and Adaptive, "Response to the UML 2.0 OCL RfP (ad/2000-09-03) - Revised Submission, Version 1.6 - OMG Document ad/2003-01-07", OMG 2003.

[19] N. Medvidovic and D. S. Rosenblum, "Assessing the Suitability of a Standard Design Method for Modeling Software Architectures", First Working IFIP Conference on Software Architecture, 1999.

[20] B. Selic, "On Modeling Architectural Structures with UML", ICSE 2002 Workshop Methods and Techniques for Software Architecture Review and Assessment, Orlando, Florida, USA, 2002.

[21] J. E. Robbins, N. Medvidovic, D. F. Redmiles, and D. S. Rosenblum, "Integrating Architecture Description Languages with a Standard Design Method", International Conference on Software Engineering (ICSE98), Kyoto, Japan, 1998.