

Architectural Aspects in UML

Jon Oldevik and Øystein Haugen

Institute of Informatics, University of Oslo, Norway

jonold at ifi.uio.no, oysteinh at ifi.uio.no

Abstract

We propose a method for describing architectural aspects in UML and show how binding specifications are used to compose aspects with base models. UML classes, parts, ports, and connectors are used for aspect structural description and sequence diagrams and state machines for behaviour. We discuss the relationship with standard UML constructs and the requirements for the binding language of architectural aspects.

1. Introduction

Architecture is the heart of any larger piece of software; if the architecture is in good shape, the chances are good that the complete system is working well; conversely, if the architecture is weak, the chances are equally good that the system is not performing well. The importance of architecture in software development has long been recognised by the software engineering community, as seen in a wide range of books (e.g. [1]), articles ([2, 3]), and standards (IEEE 1471[4]).

To support the need for architecture descriptions, many different architecture description languages (ADLs) have evolved over the years, such as Koala[5] and ACME [6]. Some of them targets specific domains or aspects of architecture, others are more special-purpose languages. Architectures are naturally coloured by the domain or system family within which they are used, which often gives rise to specific requirements to architecture descriptions.

Architecture and Aspects. The popularity of aspect-orientation has also brought aspects into the software architecture community. The match is not unnatural, as the notion of aspects can be useful to represent independent or semi-independent architectural pieces; architectural styles might be represented as aspects that that can be used to evolve software architectures more flexibly than is possible today. In [7], Pinto et al present the language *DAOP-ADL* and the *Component Aspect Model*, which combines a architecture description and aspects. They provide a formalism to specify component architectures, aspect and composition rules for these descriptions. In [3] Baniassad el al give their perspectives on how aspects can be manifested as concerns that cross cut architectural views, and how to identify and capture architectural aspects.

UML and Architecture Description. Being a general purpose modelling language, and the standard for graphical software modelling, UML is also being used for architecture description. In [8], Perez et al evaluate the usage of UML1.4 and UML2 for representing architecture connectors in the context of some specific architectural styles. One of their observations is that the connector concept in UML2 is not powerful enough to represent architectural connectors. The workshop 'Software Architecture Description and UML' in 2004 [9] discussed and presented various facets of UML used for architecture description. Roh et al [10] describe a UML2 profile for architecture modelling where a number of extensions are made to

support architecture modelling, for example providing a new type of connector based on collaborations.

We will investigate how UML constructs can be used to model architectural aspects and how these can be composed with base architectures using a binding specification. Structured classed with ports, connectors, and behaviour are used to specify both aspects and base architecture. We further discuss the relation between an aspect approach and standard UML constructs and elaborate on the requirements for the binding language.

2. UML Architectural Aspects

We will use an example called *ICU(I see You)* to illustrate an approach to architecture modelling and aspects in UML.

2.1 The ICU System

The ICU System is a buddy positioning application based on mobile messaging (SMS), where users can register, manage their buddies, and perform positioning services on those buddies. Users use mobile terminals to interact with the system, which runs at some server location.

The system is described completely in terms of UML models. UML classes with composite structures describe the system architecture while UML state machines describe the behaviour. Code generation is applied to generate the system from the UML architecture.

The ICU System is a state-machine-based, reactive system with asynchronous messaging between system objects. The system architecture is defined by classes and decomposition of classes into parts. The main system architecture (figure 1) consists of three parts: The *SMSManager*, which receives and sends SMS messages from and to the user terminals. The *ICUController* does the further handling of the messages, and the *DBManager* handles persistent storage of data. We will use this top-level architecture to illustrate how architectural aspects can be modelled and integrated in UML.

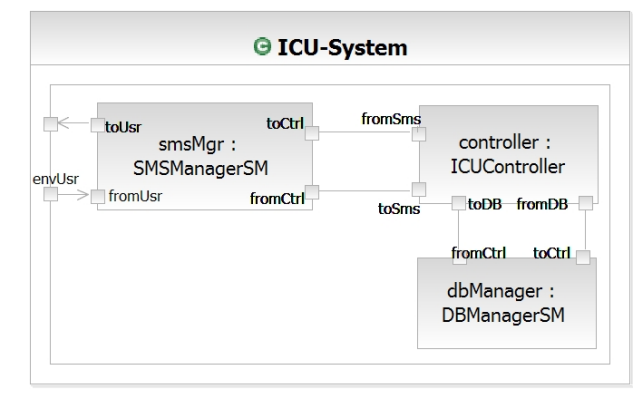


Figure 1. The ICU System Architecture

2.2 Architectural Aspects

We will look at a couple of different examples of aspects: Access control and distributed proxy aspects.

Access Control Aspect. First, we would like to introduce access control as a new aspect in our original architecture. An access control component needs to intercept messages before they arrive at the SMSManager, so we would like to insert this component between the environment input and the SMSManager. Figure 2 shows the Access Control Aspect, where the components of the aspect are parts within an aspect class.

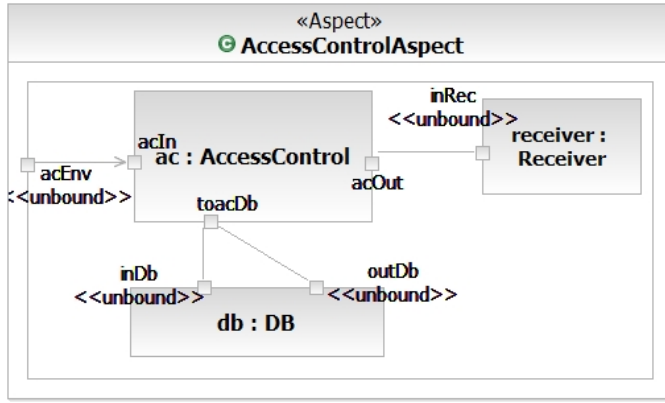


Figure 2. Access Control Aspect

In this aspect, only the *ac* part of type *AccessControl* represents a new architectural element to the system. The other parts, the *receiver* and *db* are existing parts in the *base model architecture*, which are replaced when the aspect is bound to a base model (They are placeholders for parts in the base model). The access control implementation is encapsulated within the *AccessControl* class and is implemented by a state machine. The ports that are stereotyped as *<<unbound>>* should also be bound to the base model. These ports belong to the parts that should be replaced.

Binding the aspect. To bind the aspect, a mapping of ports and parts are provided in a textual description (embedded in a model constraint). The mapping binds the unbound ports and the placeholder parts to the base model. Here, binding the ports will indirectly also bind the parts.

```
AspectBinding {
  aspect: AccessControlAspect;
  base: ICU-System;
  PortMap (acEnv, envUser)
  PortMap (inRec, fromUser)
  PortMap ((inDB, outDb), new Port (dbManager))
}
```

The AccessControl behaviour. The behaviour of the access control aspect is described by a sequence diagram which outlines the interactions (figure 3) and a state machine within the *AccessControl* class (figure 4).

The challenge with this aspect is that it imposes new structure and new behaviour on an already existing component - the database. Thus, the database also needs extension with behaviour to handle the new access control signals, which are *CheckAccess* and *NoAccess*. Other signals (messages) described in the sequence diagram/state machine refers to existing ones in the base model.

The partial behaviour for the database part can be specified within the placeholder *DB* class, which describes the desired behaviour. The only "new" properties in this partial behaviour is a single transition with an effect (which contains the actions to take when checking access).

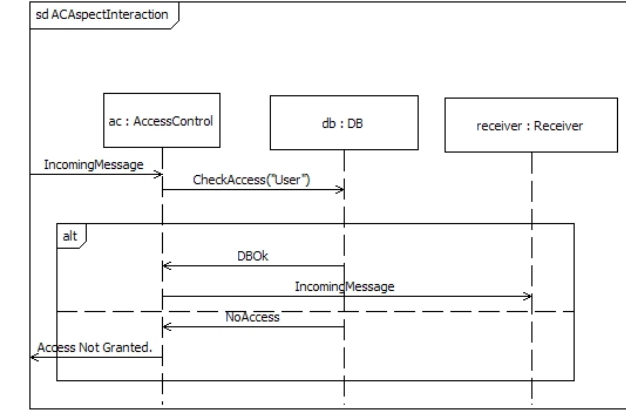


Figure 3. Access Control Sequence Diagram

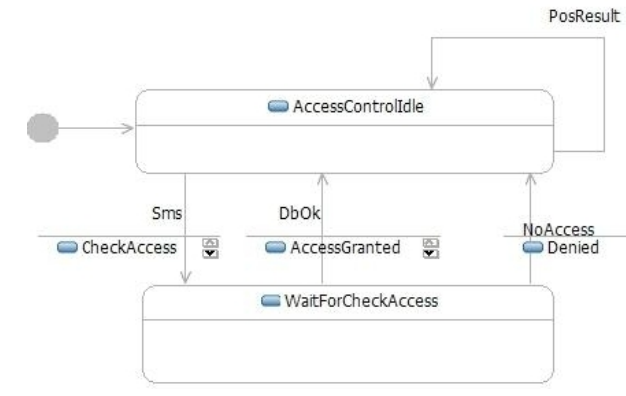


Figure 4. Access Control Behaviour

In order for the partial behaviour of the database to be composed with the base model, the states and transition within the partial behaviour must be matched with the behaviour of the base model class. This might be specified in the binding specification or be subject to some automatic matching routine. In order to support more advanced weaving of general state machine patterns, the work by Cazzola et al[11] and Klein et al[12] should be investigated.

Distributed Proxy. The second aspect we want to impose on our architecture is a distributed proxy aspect to provide a distribution level between the SMSManager and the ICUController.

A distributed proxy component consists of a local proxy part and a remote skeleton part (figure 5). The *DistributedProxyAspect* contains three parts: a local part, the distributed proxy, and a remote part. The local and remote parts have one *<<unbound>>* port each.

Since the communication between the base model components (SMSManager and ICUController) is asynchronous and two ways, the aspect must be bound twice, where both components can be both local and remote with respect to each other.

The Bindings. The bindings of the distribution aspect is shown below. Two bindings of the aspect is done: One where the SMS-Manager is local and one where the controller is local.

```
AspectBinding {
  aspect: DistributedProxyAspect;
  base: ICU-System;
  PortMap ((lp, rp), (toCtrl, fromSms))
  PortMap ((lp, rp), (toSms, fromCtrl))
}
```

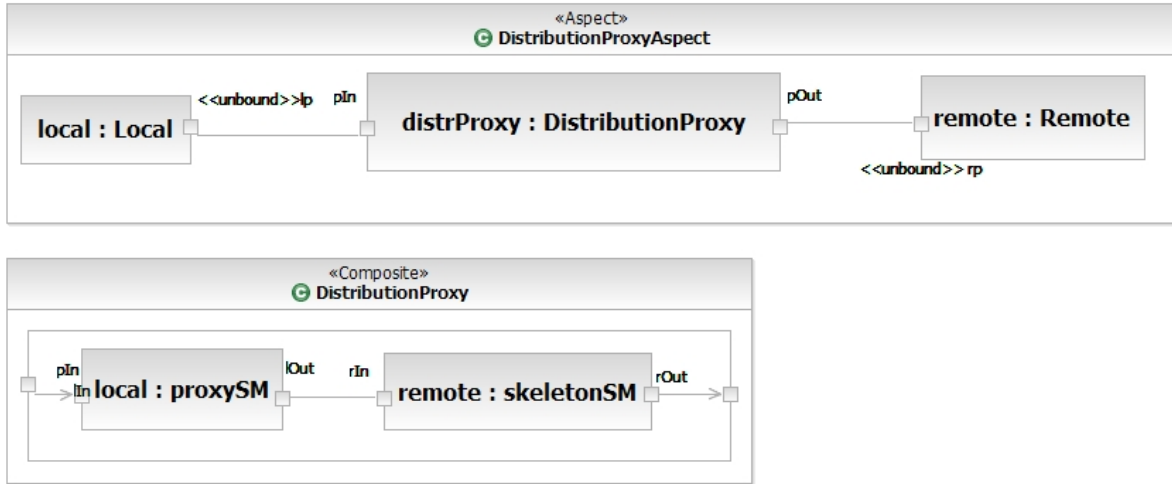


Figure 5. Distribution Aspect

In the distributed proxy example, a class and a part is made to handle the connection between the local and remote participants. If we had the ability to associate structure and behaviour directly to the connectors, we could instead have used a specialised connector concept to describe this. This simple example seemingly illustrate the points made in [8, 9, 10] that the connector concept in UML might not be powerful enough. In UML2.1[13], however, the Connector type for components has been extended with a property (the 'contract' property) to associate behaviour with the connector. By defining special types of connector (e.g. in a profile) and adding behaviour, more of the connector requirements can be met. It will still be difficult to add structure to the connector.

The Composed System. The composed system resulting from binding the two aspects to the original ICU System is shown in figure 6. The AccessControl part receives input from the environment, checks the access rights and forwards input to the smsMgr. The two distribution proxies are placed between the smsMgr and the controller, handling the messaging both ways.

2.3 Further On Aspect Binding

Crosscutting. Often, the reason for defining architectural aspects is to capture concerns that crosscut several parts of a base model architecture. It may for example be a special kind of connector or port that should be used many places in the architecture. More flexible forms of port or connector bindings will then be appropriate, e.g. to quantify over a set of target elements, similar to the abilities in most pointcut languages. Example of expressions that should be possible to express are: *All parts of a specific type, all parts with ports of a specific type, all ports associated with all connectors between ports of specific types, etc.*

Binding ports, connectors, and parts. When the various features of an aspect is bound, different semantics of a binding is possible. Ports may be replaced entirely, created, or their type(s) (provided/required interfaces) may be changed or merged with aspect ports. Similarly for connectors and parts, which may be replaced, stereotyped or similar.

A binding language for architectural aspects must cater for all these and more properties. It should define semantics to query an architecture in a meaningful manner and provide meaningful bindings for those queries.

2.4 Comparing With UML Mechanisms.

Aspects vs. Specialisation. By using specialisation and redefinition of virtual elements, a lot of different system configuration possibilities are open. However, the architectural extensions made in the example are not legal in a specialisation relationship in UML, because it requires changes to existing elements (connectors) that are not legal redefinitions in UML specialisations. A redefinition of a port, part, or connector can only be done with another, type compatible port, part, or connector, respectively.

In the example, the new, composed system architecture all changes have characteristics not compatible with specialisation. Another option with specialisation is to redefine the existing parts parts to contain the added complexity, such that for example, a redefined SMSManager contains the access control aspect. Redefined parts, ports and connectors must be type/interface-compatible with the redefined ones. The redefinition approach requires a full specification of the "new" system and its wiring. Concerns can still be described as separate components (aspects), but the benefits of quantification and cross cutting is lost, and the architecture is different from our composed result.

Another possibility for using standard specialisation is to take an approach closer to product line design, where possible future "aspect" extensions already are identified in the architecture as "virtual parts" that can be overridden in specialisations.

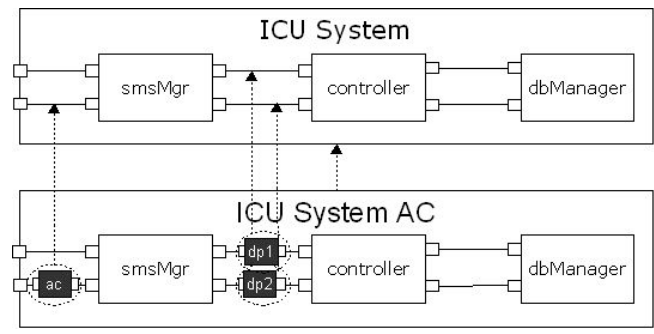


Figure 7. System Refinement

In conclusion, we see the obvious that the intent provided by an aspect-oriented approach is a different one than that captured by inheritance and virtuality. In this case, the transformation resulting

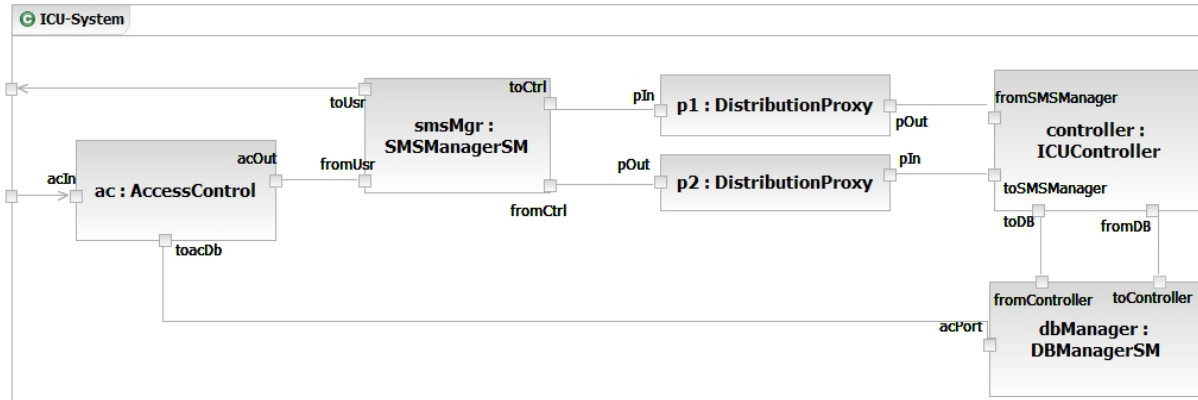


Figure 6. The Composed System

in the composed model from the aspects and the base model can be viewed as a refinement of the original system. More interesting is that in this case, the refinement can be viewed as three separate refinements of three connectors (figure 7).

Extended Virtual Connectors. Connectors in UML are virtual in that specialisations of a class can *redefine* its connectors (which is also the case with ports and parts). A connector can only be redefined by another connector, and the redefining connector and its ends must be associated with compatible types. It might be useful to extend the connector in UML, so as to support the kind of redefinition done by the distributed proxy example, where in principle connectors are replaced by a more complex structure. It could suffice to extend the semantics of connector redefinition to support transformations from simple connectors to more advanced structures.

Aspect Binding vs. CollaborationUse. UML provides a mechanisms for binding roles in collaborations with parts in a class. This could allow for an alternative way of describing the binding of architectural elements. There are however difficulties in representing the binding of parts in the aspect to something that do not already exist in the base model, for example the access control, which leads to the creation of a new part in the base model. Furthermore, it is difficult to express the binding of ports or creation of new ports. An extension to the UML's CollaborationUse would be needed.

3. Summary

We have described how aspects can be introduced in UML architecture descriptions and compared the approach with standard UML modelling mechanisms such as specialisation, collaboration bindings, and connector extensions. Furthermore, we have discussed how aspect bindings for architecture descriptions could be specified.

References

- [1] Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1998)
- [2] Garlan, D., Shaw, M.: An introduction to software architecture. In Ambriola, V., Tortora, G., eds.: Advances in Software Engineering and Knowledge Engineering, Singapore, World Scientific Publishing Company (1993) 1–39
- [3] Baniassad, E., Clements, P., Araujo, J., Moreira, A., Rashid, A., Tekinerdogan, B.: Discovering early aspects. Software, IEEE **23**(1) (2006) 61–70

- [4] IEEE: Ieee std 1471:2000recommended practice for architectural description of software-intensive systems. Technical report, IEEE (2000)
- [5] van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The koala component model for consumer electronics software. Computer **33**(3) (2000) 78–85
- [6] Garlan, D., Monroe, R., Wile, D.: Acme: an architecture description interchange language. In: CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research, IBM Press (1997) 7
- [7] Pinto, M., Fuentes, L., Troya, J.M.: A dynamic component and aspect-oriented platform. The Computer Journal **48**(4) (2005) 401–420
- [8] Perez-Martinez, J., Sierra-Alonso, A.: Uml 1.4 versus uml 2.0 as languages to describe software architectures. In: Software Architecture, Springer (2004) 88–102
- [9] Avgeriou, P., Guelfi, N., Medvidovic, N.: Software architecture description and uml. In: UML Modeling Languages and Applications. Volume 3297/2005 of LNCS., Lisboa, Portugal, Springer (2004) 23–32
- [10] Roh, S., Kim, K., Jeon, T.: Architecture modeling language based on uml2.0. In: APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), Washington, DC, USA, IEEE Computer Society (2004) 663–669
- [11] Cazzola, W., Pini, S.: Join point patterns: a highlevel join point selection mechanism. In: Aspect Oriented Modeling Workshop. (2006)
- [12] Klein, J., Jzquel, J.M., Plouzeau, N.: Weaving behavioural models. In In First Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD at ECOOP 2005, Glasgow (2005)
- [13] OMG: Unified modeling language: Superstructure, version 2.1. Standard ptc/06-01-02, OMG (2006)