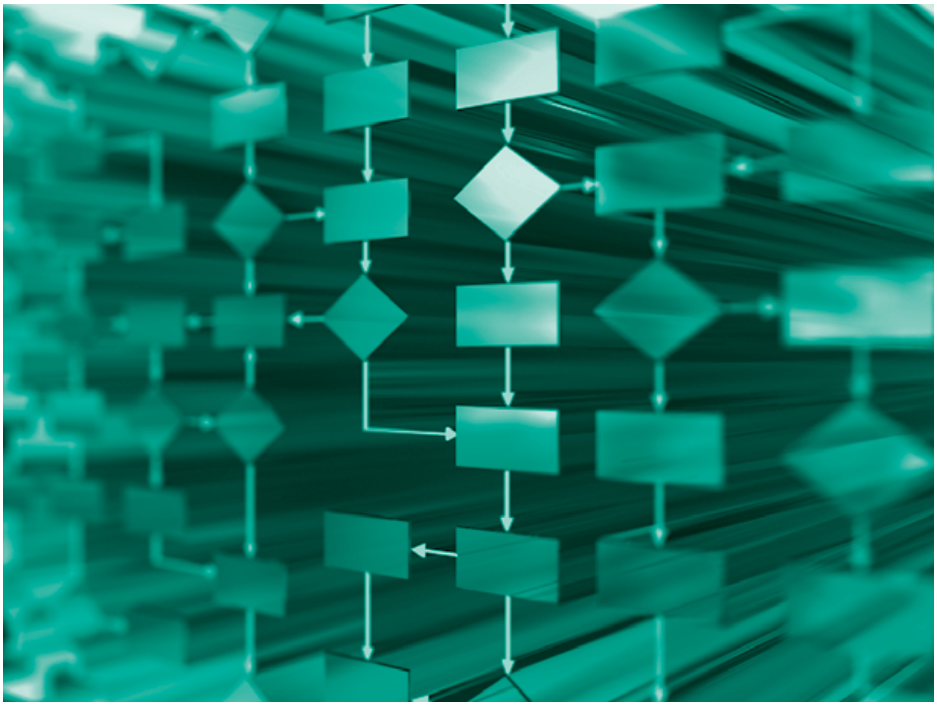
PhD Theses in Experimental Software Engineering

Vol. 33

Thorsten Keuler

# An Aspect-Oriented Approach for Improving Architecture Design Efficiency





FRAUNHOFER VERLAG

# PhD Theses in Experimental Software Engineering
## Volume 33

Zugl.: Kaiserslautern, Univ., Diss., 2011

Printing:
Mediendienstleistungen des
Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart

Printed on acid-free and chlorine-free bleached paper.


ISBN 978-3-8396-0225-6
Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 80 04 69, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon +49 711 970 - 25 00
Telefax +49 711 970 - 25 08
E-Mail verlag@fraunhofer.de
URL http://verlag.fraunhofer.de

# An Aspect-Oriented Approach for Improving Architecture Design Efficiency

Beim Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

**Doktor der Ingenieurwissenschaften (Dr.-Ing.)**

genehmigte Dissertation
von

**Dipl.-Inf. Thorsten Keuler**

Fraunhofer-Institut für Experimentelles Software Engineering
(Fraunhofer IESE)
Kaiserslautern

| | |
|---|---|
| Berichterstatter: | Prof. Dr. Dr. h.c. Dieter Rombach<br>Prof. Dr. Colin Atkinson |
| Dekan: | Prof. Dr. Karsten Berns |
| Tag der Wissenschaftlichen Aussprache: | 26.11.2010 |

**D 386**

*"Design is not just what it looks like and feels like.*
*Design is how it works."*

*Steve Jobs*

# Acknowledgment

Looking back over the past years, it became clear to me that the process of getting a Ph.D. at Fraunhofer is characterized by fluctuation rather than convergence. Although job variation seems to be intriguing in terms of day-to-day work, it does not align well with the clear goal of getting a Ph.D. Considering this fact a personal challenge, it turns out that self-motivation combined with the will to embrace change needs to be rekindled every once in a while. I believe this is impossible to do without the continuous support of other people.

First of all, I want to thank the members of my Ph.D. committee: Colin Atkinson, Karsten Berns, and Dieter Rombach. In particular, I want to thank Dieter Rombach for giving me the opportunity to contribute to the field of software engineering at a renowned research institution such as Fraunhofer IESE.

Second, I want to thank my colleagues at Fraunhofer IESE who supported my work in a multitude of ways, such as scientific discussions or giving feedback on Ph.D. rehearsals. In particular, I want to thank Michalis Anastasopoulos, Martin Becker, Jörg Dörr, Isabel John, Jens Knodel, Dirk Muthig, Matthias Naab, Marcus Trapp, Mario Trapp, and Christian Webel (in alphabetical order). I am also thankful for the professional support with respect to the conduction of empirical studies provided by Marcus Ciolkowski and Michael Kläs. Special thanks go to Stephan Thiel and Sonnhild Namingha. Stephan supported me in printing and binding preliminary versions of this thesis; and thank you, Sonnhild, for doing a great job in improving the English grammar of this thesis on short notice.

Third, I want to thank the student workers who eagerly developed the software realizing many ideas described in this thesis. In particular, I want to thank Yury Kornev, Alexander Kabanov, and Mateus Volkmer Nunes Gomes. In addition, I want to thank all students who participated in the controlled experiments conducted in the context of the course "Software Architectures for Distributed Systems" in the years 2008 and 2009, respectively.

Finally, I want to thank my family for supporting me all these years. I am deeply grateful for having been given the chance to take the path I took. Last but not least, I want to thank my soon-to-be wife Andrea for her continuous support, understanding, patience, and love, throughout the final and most challenging phase of the Ph.D. process. Thank you.

# Abstract

Our experience shows that the discipline of software architectures must face the challenge of scaling with the ever increasing size and complexity of requirements imposed on system development in order to be able to deliver high-quality products on time. That is, software architecture methods need to provide practical support for efficiently and effectively predicting, controlling, and evolving required product properties throughout the entire lifecycle of software-intensive products.

Traditionally, software architecture utilizes the principle of separation of concerns to cope with the challenge of scalability of solutions to complex problems. For instance, by encapsulating functionality into self-contained units and applying information hiding, large and complex systems are decomposed into modules that can be developed and integrated independently. In the context of non-functional concerns, however, separation of concerns does not work as well. Since the focus during decomposition is on the identification of components that aggregate semantically cohesive chunks of functionality, the inter-component dependencies remain side-affected by solutions addressing non-functional concerns. This is particularly true for component interdependencies that specify inter-component communication. The interactions among components become more critical determinants of system properties as components become more complex and heterogeneous.

The main solution idea of this thesis is to leverage aspect-oriented concepts at the architectural level in order to provide a means for effectively and efficiently separating communication concerns in the context of architectural design. The primary contribution of this thesis comprises a formalization of architectural models and a method that supports the design process with sophisticated extensions of a commercial architecture design tool. The aspect-oriented architecture model, the design method, and the tool support are fundamental ingredients for efficiently implementing the principle of separation of concerns at the level of architectural interconnection.

The proposed solutions are validated by means of a controlled experiment showing that the aspect-oriented separation of communication concerns indeed improves the design efficiency of architectures.

# Table of Contents

# List of Figures

# List of Tables

# 1 Introduction

*"Architecting is about Balancing", Randy Stafford*

Today, software development organizations are faced with the continuously increasing size and complexity of the systems to be developed. Consequently, software systems engineering methods must be scalable regarding size and complexity in order to be able to deliver products of required quality on time. That is, software systems engineering methods need to provide practical support for efficiently and effectively predicting, controlling, and evolving quality throughout the entire lifecycle of such software-intensive products. Throughout this thesis, the term "system" will be used to refer to software-intensive products.

The reasons underlying the demand for the scaling of engineering methods are three-fold:

- First, the number and complexity of requirements used as input for the development of a system are increasing. As a result, modern systems contain several millions of lines of code, thousands of modules, tens of thousands of interrelationships between these entities, and involve hundreds of developers, potentially distributed all over the globe.

- Second, the relevance of non-functional requirements is heavily increasing. For instance, the systems themselves need to be extensible, secure, interoperable, and so forth. At the same time, they need to be controlled, maintained, and evolved in continuously changing environments during runtime.

- Third, the design for reuse in the context of product lines calls for the effective, efficient, and sustained handling of variants throughout system development, which imposes additional complexity on the development process [Bos00].

It is the goal of software engineering [Som01] to cope with these scalability drivers by imposing structures on systems in order to enable efficient development while, at the same time, providing a high degree of confidence that the required quality properties are achievable. For these

reasons, the discipline of software and system architectures has emerged since the 1990s [Sha96], [PW92], [GS94].

**Definition – Software Architecture**

A software system's architecture is the set of principal design decisions made about the system [TMD09].

According to this definition, one of the main goals in architecting is to provide a set of design decisions that, in combination, address a given set of architectural concerns [TA05].

**Definition - Concern**

A concern about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture [MEH01].

Design decisions as such are decisions about the shape and texture of solutions to particular architectural concerns. The consequences of wrong decisions made at the architectural level will be uncovered during implementation the earliest, which increases the likelihood for costly rework across the entire development lifecycle. Since concerns potentially contradict or are in conflict with one another, for the benefit of a particular concern, a design decision is sometimes accepted despite the fact that other concerns are impacted negatively. Such decisions are referred to as trade-off decisions. The major challenge in making design decisions in general, and trade-off decisions in particular, is to keep track of direct and indirect impacts revealed among solutions that address different concerns.

## 1.1 Problem Statement

Making architectural design decisions needs to be supported by a process that scales with the number of concerns at hand. In other words, the efficiency of the design process needs to scale with the number of team members involved in the design.

**Practical Challenge: Scalability of Team Sizes**

In practice, architectural design efficiency does not scale with the size of the team developing the architecture.

In other words, the time for creating an architecture design does not necessarily decrease because of new resources assigned to the architecture team. The main reason for the scalability problem is due to a poten-

tially high number of inter-dependencies among architectural quality concerns. Considering parallel teamwork in architecture design, all solutions to quality requirements are locally designed, evaluated, and optimized. At one point in time, these partial solutions need to be consolidated into one single architecture though. Without communication among the separate teams, there is only a slight chance that the solutions will not impact each other. Hence, the effort decrease achieved by parallel teams working on different facets of the architecture, in turn, causes significantly higher effort during consolidation of partial solutions. In fact, it is the side effects of the partial solutions on each other that need to be checked every time before they can be integrated into an architecture design.

In the following, a set of additional influencing factors is presented that make the endeavor of architecture development challenging in terms of scalability.

The first important influencing factor is that architectures are designed iteratively and incrementally [Boo07]. That is, at the beginning of any iteration the architects need to select a subset of requirements, which are then addressed by taking appropriate design decisions. For making design decisions, the architect also needs to consider the increments designed during previous iterations.

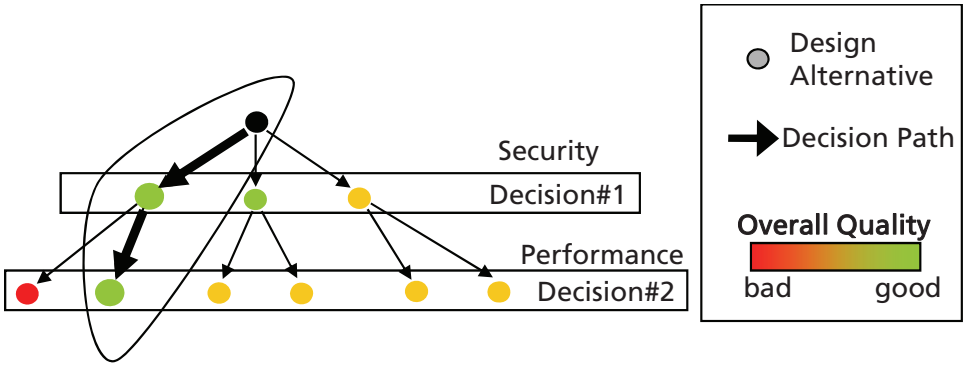



Figure 1 Design decisions are based on previous increments

In this context, one fundamental assumption is that architecture is designed relative to a set of requirements. Hence, an architecture increment is the totality of design decisions that address the requirements at a particular point in time. The example shown in Figure 1 illustrates that the decisions made for security and performance yield the best combination of design alternatives up to that point (the green fill color indicates a good design at that particular increment).

The iterative nature of the process, however, imposes an additional source of complexity on the design of architectures, namely the limited scope that the design can be evaluated in. Since iterations imply that the architecture develops over time, the validity of the analysis of the design is volatile.

**Practical Challenge: Limited Design Scope**

In the context of a new increment, there is a high probability that previous design decisions need to be revised.

As shown in Figure 2, based on the current increment, there is no way to reach the global optimum merely by selecting a design alternative for availability (Decision#3). Previous decisions (in this case Decision#1) would have to be revised in order to get the best combination of design alternatives.

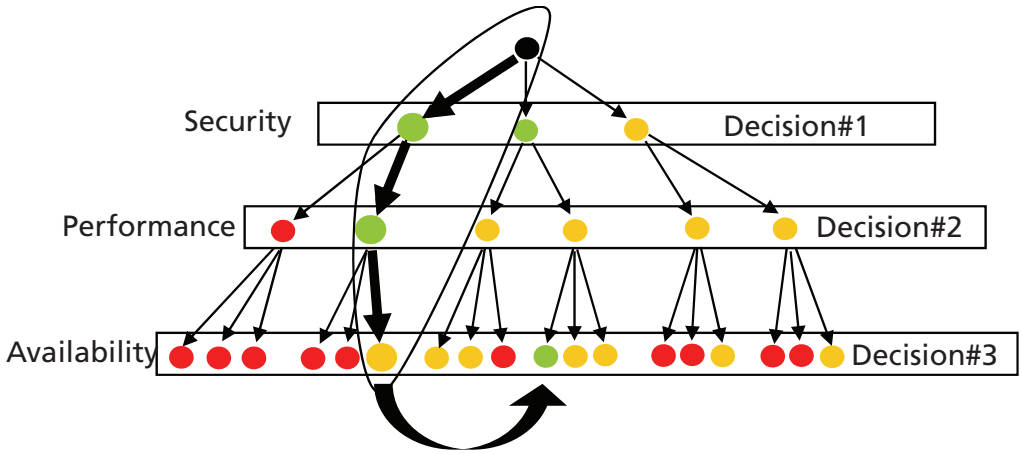



Figure 2 Confined decision making in incremental design

In order to know what combination would be the best, theoretically all possible combinations would have to be compared after each iteration. I stress the fact that it is not realistic to strive for the global optimum; In practice it is necessary to know what would be the minimum set of changes that need to be performed in order to yield an architecture of required quality. The resulting design space complexity can be expressed by the following formula, with *n* being the number of separate design decisions and $o(d_i)$ the number of options for the design decision $d_i$:

$$\textbf{Possible combinations of design decisions =: } \prod_{i=1}^{n} o(d_i)$$

In other words, the formula describes the number of distinct paths through the design space. Applying this formula to the example as shown in Figure 2, we end up with 3*2*3 = 18 distinct combinations of design decisions. Since we assume the exploration of at least two options per design decision, we expect each factor of the formula to be equal to or greater than two.

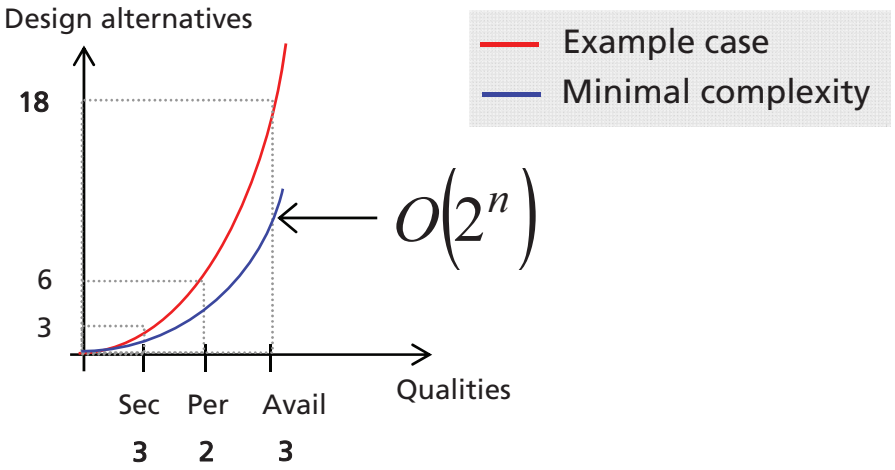



Figure 3 Complexity of design space exploration

Due to the incremental nature of architecture design, an issue arises that is of practical relevance as well: The tacit knowledge of the decisions that lead to a particular design. That is, the steps describing the decisions taken are not explicit. Thus, the architecture design only reflects the impact of the aggregation of all design decisions made.

**Practical Challenge : Traceability of Design Decisions**

An architectural design is always the result of a combination of design decisions that have been made (deliberately or not) up to a particular point in time. However, tracing how the final design was created is not necessarily recoverable from the final result.

Besides issues regarding the incremental nature of design, another major influencing factor is given by the fact that architecture needs to be developed under conditions of constant change: "The set of decisions comprising the architecture forms in concert with the requirements and continues to expand through evolution. The architecture is thus under constant change" [TMD09].

**Practical Challenge : Continuously Changing Requirements**

In practice, architectural concerns are never stable, and thus require continuous adaptation of architectural decisions throughout the lifecycle of the system.

That is, in realistic settings architectural concerns change constantly, either in terms of changed priorities, or in terms of new or modified concern specifications. Knowing that requirements change over time leads to the conclusion that architectural design decisions need to be revised

as well, which imposes additional complexity on design space exploration.

As illustrated in Figure 4, a change of requirements potentially leads to a different evaluation of intermediate design decisions, which in turn requires a change of decisions already made. Projecting this into the real world, this could mean that an architecture that was designed for security and performance suddenly needs to address maintainability as well. As a consequence, all existing design decisions would have to be revisited and checked for the stated security concern, and if feasible, changed accordingly.

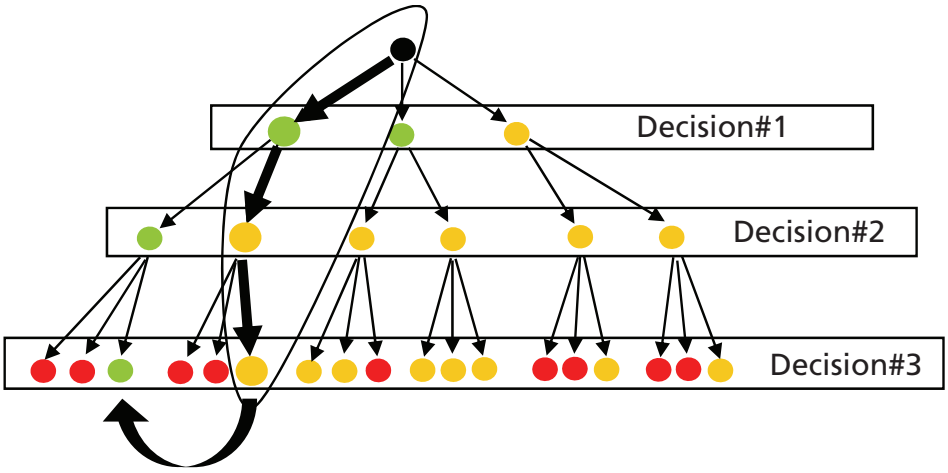



Figure 4 Changes in priorities imply revisions of design decisions

If requirements become obsolete (see Figure 5), existing designs need to be revisited as well. By removing concerns, and thus their impact, from the queue of design decisions, changes to other design decisions might be a likely result in order to obtain a better set of design decisions.

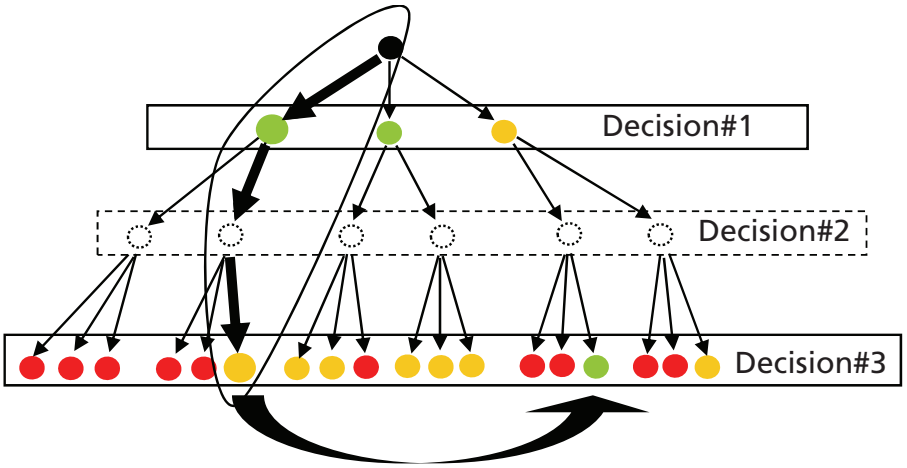



Figure 5 Obsolete requirements imply a change of architectural decisions as well

In the context of product lines ([WL99], [CN02]) the complexity of the design space is even higher. That is, since the product line architecture needs to be designed in such a way that a set of products with shared as well as individual requirements can be built, variants of architectural designs need to be created, evaluated, and evolved over time.

**Practical Challenge: Designing for Variation**

Product line architectures impose an additional dimension of complexity on the design space exploration, since the variant parts need to be explicitly designed, evaluated, and evolved in parallel to the common parts.

The main reason that makes all these issues hard to address in practice is the lack of effective and efficient separation of architectural concerns.

**Definition – Separation of Concerns**

In computer science, separation of concerns is the process of separating a computer program into distinct features that overlap in functionality as little as possible. [Dij82]

Separation of Concerns has traditionally been achieved through modularity and encapsulation with the help of information hiding [Par72]. The main drawback of such decomposition approaches is that they only allow modularizing the system in one dimension at a time. As a result, a particular concern becomes dominant and most of the other concerns such as quality concerns that do not align with the main decomposition are scattered across many system modules. This phenomenon is called "tyranny of the dominant decomposition" [THO0]. One implication of this statement is that what works well for functional concerns does not suffice for the quality requirements. The reasons for this are two-fold:

First, solutions to quality requirements potentially interfere with one another. For example, in order to achieve maintainability, modular structures could be introduced to minimize change impacts. However, the very same structures introduced for maintainability might decrease performance at the same time.

Second, solutions to quality requirements do have significant implications on functional structures. For example, in case security is an architectural concern, there are potentially many functions in the system that are required to be compliant with the security policy prescribing rules for accessing and changing information. The relationship between solutions to quality requirements and functional structures is the reason why such kinds of concerns are also knows as "cross-cutting concerns".

**Definition – Cross-Cutting Architectural Concern**

At the architecture design level, a crosscutting concern could be any concern that cannot be effectively modularized using the given abstractions of an Architecture Description Language (ADL) [GS94].

We stress the fact that concerns themselves do not cut across the architectural structure per se; it is rather the solutions to the concerns that make them eventually cross-cutting. Typical examples of cross-cutting solutions to quality requirements are authentication, encryption, compression, access control, transactions, or persistence. Cross-cutting concerns have several implications on architecture development.

**Technical Need: Effective and Efficient Separation of Cross-Cutting Architectural Concerns**

There is no effective and efficient way for both separating **and** composing cross-cutting solutions to architectural concerns.

Solutions to cross-cutting concerns usually comprise multiple parts that affect architectural entities in structural as well as behavioral regards. In terms of architectural concerns, runtime quality attributes require solutions that change the behavior of architectural entities such as the control flow or coordination behavior. This implies that inter-component communications are potentially affected by such cross-cutting solutions [FEC04], [CRF+06]. There is a common understanding that "arriving at design decisions describing proper component interactions in a system can be even more challenging than those restricted to the development of functionality" [TMD09], [CRF+06].

**Solutions to Runtime Quality Attributes are Impacting Inter-Component Communications**

Runtime concerns are heavily dependent on the coordination and communication capabilities of architectural entities.

In principle, all externally visible properties that are exposed during a system's runtime must potentially be considered, since they are influenced by component interactions [FEC04]. In case of security concerns, for instance, the component communications specify the way of how and when data is encrypted and decrypted in any context of data access or transfer. Such a specification needs to be followed by many, if not all, of the components in the system. Since not all quality requirements relate to communication properties per se (e.g., maintainability), for the remainder of this thesis, I will focus on those quality requirements that have an impact on inter-component communication.

## 1.2 Solution Idea and Research Questions

The phenomenon of cross-cutting concerns exists on other levels of abstraction as well, for instance at the requirements or at the code level. Most existing solutions target cross-cutting concerns on the code level. The most prominent example is Aspect-oriented Programming (AOP) [HHK00]. AOP aims at solving cross-cutting problems on the source code level by separating the cross-cutting portions from those entities affected by them and providing a composition mechanism that automatically composes the separate parts into a final code base.

The main idea of aspects is to exploit the fact that the cross-cutting solutions affect many places of the overall system in the same or in a similar way. Hence, the cross-cutting solution can be implemented once and is then propagated by replicating the very same solution in different places of the design.

**Aspect Orientation in General allows for Separating Concerns that Cut across a System**

Aspect orientation addresses cross-cutting concerns that affect a number of places of the system in similar ways.

This is especially true for architectural solutions that cut across inter-component communication specifications, for example authorization or persistence. The aspect-oriented programming paradigm brings an efficient composition mechanism that enables developers to quickly enhance or modify source code with so-called aspects. The composition itself is also known as “aspect weaving”, denoting that aspect code is introduced into the system code. As a consequence, in case a cross-cutting concern is to be revised or extended, there only one place needs to be changed: the aspect.

The main solution idea of this thesis is to leverage aspect-oriented concepts at the architectural level in order to provide a means for effectively and efficiently creating architectural design models that reflect inter-component communications. The architectural models shall be composed from a set of designs that address non-functional requirements separately from each other.

To that end, I provide composition mechanisms akin to AspectJ [HHK00], but at the modeling level. Based on these mechanisms, model compositions can be specified based on a number of sequence diagrams. As a consequence, I am able to generate any given combination of design decisions in terms of inter-component communication at any point in time. As shown in Figure 6, we can navigate through the design space by

simply selecting any combination of design decisions. The final design model is then generated automatically. The generation of design models enables easy changes of design decisions, without having to backtrack through the decision tree.

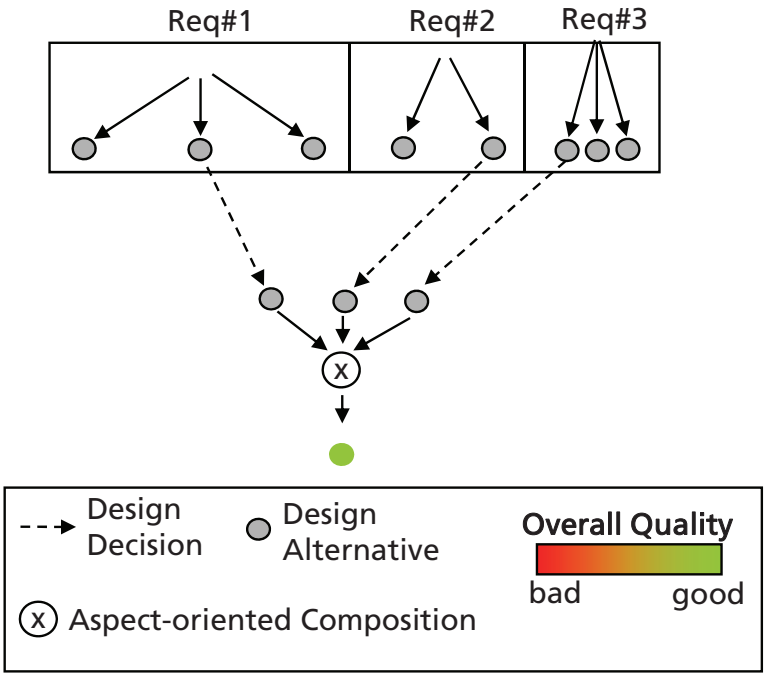



Figure 6 Solution Idea

The solution idea addresses all problems introduced above by providing the ability to flexibly and efficiently combine design decisions in the realm of architectural inter-component communication. Referring to the tree structure of the design space depicted in Figure 2, I am now able to break the dependencies between the layers between the iterations.

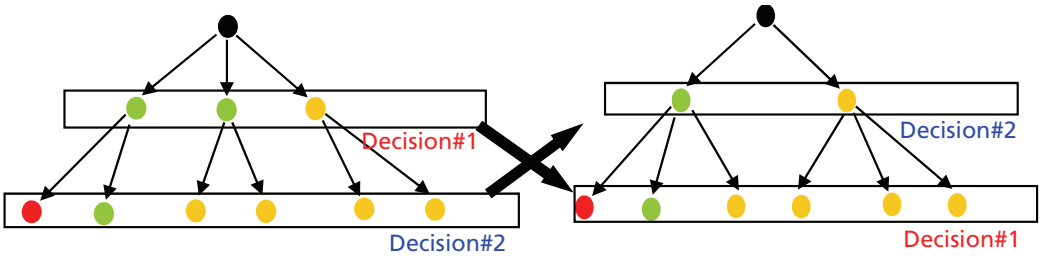



Figure 7 Dissolving the dependencies between "layers" in the solution space

The underlying solution hypothesis can then be defined as:

**$H_S$**: Aspect-oriented concepts can be used for efficiently manifesting and changing design decisions in the realm of inter-component communications.

The expected benefits lie in the ability to efficiently create and change architectural design models specifying inter-component communication. Therefore, the research presented in this thesis is geared towards two main research questions:

1. Can aspect orientation be utilized at the architectural level for separating cross-cutting solutions for inter-component communications?

2. Can aspect-oriented modeling improve the efficiency of the creation and alteration of models specifying inter-component communications?

In order to answer these questions, the following research questions arise:

| **Research Questions** |
|---|
| • How do the concepts of aspect-oriented programming map to architectural concepts? |
| • What concerns can be treated in an aspect-oriented way? |
| • What should a composition mechanism provide? |
| • How should architectural composition be defined? |
| • How is a correct composition defined? |
| • How can compositions be checked for correctness? |
| • How efficient is the composition? |
| • How flexible is the composition with respect to change? |

The results presented in this thesis answer these research questions. However, during the evolution of the solutions as presented in this thesis, a number of research challenges revealed themselves and had to be tackled appropriately. In the following, I will describe these research challenges that I needed to cope with, discuss related work and state-of-the-art solutions, and briefly explain the contributions of the work presented in this thesis.

## 1.3 Research Challenges and Contributions

The research challenges as addressed by the work presented in this thesis are related to one of the following four fields:

1. **Theoretical foundations** and models that I built the solution on

2. **Methodical approaches** that support the process of utilizing the formalized models in order to increase scalability and efficiency

3. **Engineering approaches** realizing the formalized models in such a way that the methodical support can be leveraged in practical settings

4. **Empirical evaluation** of the proposed solutions using experimental approaches as described in [Bas93].

For each of these fields, I provide a number of scientific challenges as well as engineering challenges that I came across along the way, explain the respective shortcomings of existing state-of-the-art concepts, and show the solutions I came up with in order to address the scientific challenges.

### 1 – Theoretical Foundations and Model Building

| **Scientific Challenge: Formalization of Architecture** |
|---|
| **State of the Art**<br>In the literature, there exist solutions that formalize partial aspects of architecture design; however, a conceptualized model interrelating entities that are of importance in the context of the problems motivating this thesis is missing. |
| **Contribution of this thesis**<br>Based on a consolidation of existing formalizations as described by the current state-of-the-art, I defined a meta-model that relates all facets of architecture that are of relevance for separating cross-cutting solutions from core architectural entities. The conceptualized model provides the basis for defining specific extensions addressing the architectural issues regarding improvement of separation of concerns at the architectural level. |

In order to find solutions to the practical problems described in the previous sections, the practical issues are mapped to a conceptualized model that could be taken as a starting point for building a solution.

That is, in a first step I needed to build a model that formalizes and interrelates concepts within the realm of architecture design. The second step of formalization was concerned with model building at the solution level. Since I aim at leveraging aspect-oriented concepts at the architectural level, I needed to overcome a number of current shortcomings existing in the realm of aspect-oriented modeling in general and aspect-oriented architectures in particular.

Looking at the problem statement regarding cross-cutting concerns at the level of inter-component communication, I found that an appropriate solution needs to provide a means for expressing cross-cutting concerns at the architectural level in the form of behavioral models. In that context, I found that existing aspect-oriented design solutions at the architectural level were realized as extensions to architectural description languages (ADLs) [GCB06]; however, "model compilers" that produce an integrated solution out of the aspect-oriented architecture descriptions are missing. In addition, I found that existing approaches are only appropriate for working on the type level of model elements rather than on the instance level. This shortcoming is relevant for two reasons: First, architectural element types that are defined after concrete models have been created cannot be easily propagated to existing instances. This is particularly important in the context of improving the scalability of teams by increasing parallelism. Second, the spectrum for defining specific aspects is limited. Either the types are extended or modified, or a new type needs to be defined that is appropriate for the particular context. In the following table, I summarize the state-of-the-art shortcomings and list the contributions provided by this thesis to overcome or mitigate the problems identified.

**Scientific Challenge: Utilize Aspect Orientation at the Architectural Level**

**State of the Art**

(a) Regarding aspect-oriented modeling, a number of contributions already exist that leverage aspect-oriented concepts at the modeling level. However, most approaches address structural models only.

(b) There are solutions that provide aspect-oriented extensions to architecture description languages; however, there is no support for compiling the language into an integrated solution.

(c) Existing solutions work at the type level of model elements only. That is, aspect-oriented modeling can be integrated at the level of type definitions; however, existing model elements cannot be treated as existing instances of types introduced at a later point in time.

**Contributions of this thesis**

(a) I base this work on the modification of UML sequence diagrams.

(b) In this thesis, I present a solution that is able to compile a number of solutions to crosscutting concerns into a consistent architecture design.

(c) The solution presented in this thesis comes with means for specifying aspects to operate on the type as well as on instance level of the base model elements. I can propagate new or changed element types into the model and achieve a finer-grained spectrum between the worlds of types and instances. In addition, I leverage meta-model relationships as defined in the architectural meta-model in order to be able to provide a powerful means for efficiently selecting huge portions of an architectural design for specifying model compositions.

Compositional interactions denote the situation when aspects interfere with one another within the base model. Since aspects potentially operate on the same base model elements, there is a high potential that two or more aspects will change the model in such a way that the resulting aspectual composition does not reflect the intended model properties. Regarding the state-of-the-art, I found that there is only limited support for detecting, resolving, or preventing aspect interactions at the modeling level.

**Scientific Challenge: Treatment of Compositional Interactions**

**State of the Art**

There is only one aspect-oriented modeling approach that is capable of detecting compositional interactions at the modeling level. However, that approach only works for structural models and does not provide any means for resolving the detected interactions.

**Contributions of this thesis**

- Utilization of fluent linear temporal logic for checking compositional interactions at the modeling level.
- Provision of a set of strategies for resolving conflicts during model composition.

## 2 – Methodological Support

In general, a formalization of real-world phenomena in terms of a model also requires a process that shows how to use the model for solving real problems at hand. However, existing aspect-oriented modeling approaches found in the literature usually do not provide methodological guidance [SSK06].

**Scientific Challenge: Lack of Methodological Guidance**

**State of the Art**

Existing work in the area of aspect-oriented modeling in general and aspect-oriented architectures in particular does not provide methodological guidance.

**Contribution of this thesis**

In this thesis, I provide a set of extensions to an existing architectural design method. That is, I define a process that explains how to use aspect-oriented modeling within the process of architecture design.

In terms of model evolution, aspect-oriented approaches usually suffer from the drawback that they need to keep track of changes to the base model in order to keep the composition specification consistent. This problem is also referred to as *pointcut fragility* [CGB09]. In fact, there is only limited support regarding pointcut fragility at the modeling level.

**Scientific Challenge: Evolution of Aspect-oriented Models**

**State of the Art**

There is only limited support for addressing the problem of pointcut fragility at the modeling level.

**Contributions of this thesis**

- The thesis comes with a set of guidelines addressing pointcut evolution scenarios.
- Provision of a reflective model that allows for checking base model changes against pointcut specifications.

### 3 – Engineering solutions

I used the engineering method [WRH00] to determine the capabilities, drawbacks, and limitations of current solutions in the realm of architecture design. Since existing solutions are realized as extensions to ADLs, they have not been adopted in practice yet [WH05].

**Engineering Challenge: Industry Acceptance of Architecture Description Languages**

**State of the Art**

Existing approaches utilizing aspects at the architectural level extend formal architecture description languages (ADLs).

**Contribution of this thesis**

The solution provided by this thesis uses the UML, which can be considered a de facto industry standard.

An important aspect of engineering solutions regarding aspect-oriented modeling is the specification of the composition. Usually, a so-called pointcut definition language is used for specifying the model composition. Since these languages are based on a formal syntax, pointcut expressions are hard to create and to evolve in the face of change.

**Engineering Challenge: Setting up Weaving Specifications**

**State of the Art**

State-of-the-art approaches rely on the manual construction of formal pointcut expressions.

**Contribution of this thesis**

In this thesis, I provide a concept for interactively selecting the model elements that are considered the places for modifying the respective behavior of architectural elements. Based on that selection, a formal pointcut expression used by the model composition algorithms can be generated automatically.

Another challenge regarding the engineering of aspect-oriented model creation is related to the actual model weaving. I found that almost no approach as of today supports effective weaving of models. Most approaches defer the actual weaving to the programming level [SSK06].

**Engineering Challenge: Weaving of Aspect-oriented Models**

**State of the Art**

Existing aspect-oriented modeling approaches do not support the actual weaving of behavioral models.

**Contribution of this thesis**

In this thesis, I provide an implementation of algorithms that actually realize the weaving of behavioral models.

As described before, the basis for checking compositional properties is fluent linear temporal logic. That is, I utilize the concepts existing in the realm of labeled transition systems [GM03] to check the resulting models for time-varying properties. In practice, however, the specification of a formal model describing a labeled transition system including formal assertions is a manual task and prone to errors.

**Engineering Challenge: Manual Creation of Fluents and Assertions for Model Checking**

**State of the Art**

The creation of formal models that can be used by model checking approaches requires manual effort.

**Contribution of this thesis**

I defined a set of heuristics that allow for automatically deriving formal specifications of model properties in terms of fluents and assertions. Besides the fact that I can generate a labeled transition system out of the interaction specifications defined by sequence diagrams, I am also able to hide the formal complexity from the user, which is positive in three ways: 1. Automatic generation tremendously improves the efficiency of model checking. 2. The designer does not need to be familiar with formal model checking techniques. 3. The automatic generation is less, if at all, error prone.

### 4 – Empirical Validation

In the literature, there is only little evidence regarding the effectiveness and efficiency of aspect-oriented concepts in the realm of model-based development in general and model-based architectures in particular.

**Empirical Challenge: Lack of Evidence regarding the Effectiveness and Efficiency of Aspect-oriented Approaches in Model-based Development**

**State of the Art**

There is only limited evidence regarding the effects of aspect-oriented modeling.

**Contribution of this thesis**

A controlled experiment was conducted that revealed gains in efficiency due to the use of aspect-oriented separation of concerns in the context of modeling-related activities.

In summary, the contribution of this thesis comprises:

- A formalization of architectural design elements for defining aspect-oriented extensions such that cross-cutting concerns can be treated systematically.

- Aspect-oriented extensions to the architecture meta-model in order to be able to separate cross-cutting concerns effectively and efficiently from base models. The aspect-oriented meta-model enables efficient composition of sub-solutions yielding a consolidated design overall. The aspect-oriented meta-model presented in this thesis

  i. is based on UML sequence diagrams.
  ii. exploits meta-model relationships of the architecture in order to provide a powerful means for supporting pointcut specifications.
  iii. comes with means for specifying aspects to operate on the type- as well as on the instance-level of the base model elements.
  iv. comes with a set of weaving algorithms.
  v. utilizes fluent linear temporal logic for checking compositional interactions at the model level.

- A method that guides an architect through the process of designing and evaluating trade-off decisions. That is, I provide guidelines for

  i. how to use aspect-oriented modeling within the process of architecture design.
  ii. pointcut evolution.
  iii. resolving semantic conflicts during model composition.

- A tool that enables efficient model-based design and formal analyses by supporting semi-automated model compositions and evaluations. The solution provided by the tool comprises:

  i. Aspect weaving based on UML sequence diagrams.
  ii. Interactive selection of model elements for weaving, including automatic generation of a formal pointcut expression that can be used by the model weaving algorithms.
  iii. Implementations of the algorithms that actually realize the model weaving.
  iv. Automatic derivation of formal specifications of model properties in terms of fluents and assertions, which hides the formal complexity from the user while improving the efficiency of

model checking and significantly reduces error-proneness due to the automatic generation of formal expressions.

v. Integrated meta-model relationships in order to provide powerful means for efficiently selecting huge portions of a model for weaving.

vi. A reflective approach for automatically checking base model changes against pointcut specifications.

- The validation of the impact of aspect orientation on the efficiency of architectural model handling. I compared the aspect-oriented approach and an integrated modeling approach.

## 1.4 Practical Implications

Given the proposed solution approach and the problem statements as discussed in sections 1.1 and 1.2, improvement hypotheses for the practical issues can be defined as follows:

**$H_{P1}$**: By separating cross-cutting architectural solutions using aspect-oriented concepts, the ability of scaling the team size with the concerns is significantly improved.

Since the composition approach supports the integration of partial solutions, the approach significantly improves the scalability of parallel teamwork. According to the problem statement described in section 1.1, the main factor that hampers scalability is the number of distinct solutions describing inter-component communications that need to be integrated into a single one. By applying the aspect-oriented solution as presented in this thesis, we assume that the increased scalability of team size depends on the increase of the number of runtime quality attributes to address and the respective number of design alternatives. In other words, if we have only a single quality attribute to design for and we only have a single solution to investigate, the aspect-oriented approach does not unfold its power. However, in practice, architectures need to serve a multitude of quality requirements and thus motivate the exploitation of aspect-oriented concepts.

**$H_{P2}$**: By separating cross-cutting architectural solutions using aspect-oriented concepts, the impact of the limited design scope is significantly reduced.

Because all design alternatives can be created by simply generating the resulting architectural designs from a composition specification, there are no increments that other (partial) solutions need to build on. This avoids ripple effects from changing previous designs. However, even

though the impact of the limited design scope is significantly reduced, the challenge is shifted towards the composition of the partial solutions. That is, the effort for navigating through the theoretical design space as depicted in section 1.1 depends on the effort needed for composing a respective selection of partial solutions. The amount of effort saved for navigating through the design space scales with the number of interdependencies among the partial designs created for each quality attribute.

**$H_{P3}$**: By separating cross-cutting architectural solutions using aspect-oriented concepts, the design decisions that lead to a particular design are always explicit.

If the solutions to particular concerns can be separated in terms of aspects, it is possible to operate on the selection of decisions only. As a result, architects do not depend on a particular resulting design to make the next decision, and it is clear which decisions finally make up the architecture design.

**$H_{P4}$**: By separating cross-cutting architectural solutions using aspect-oriented concepts, the responsiveness to changing requirements is significantly improved.

Through its ability to generate designs from basic solutions, the approach increases its resistance to change. That is, if priorities of quality requirements change, the change is local to the selection of a particular solution. In case requirements become obsolete, the corresponding solutions are dismissed from the composition specification. I hypothesize that the significance in efficiency improvements is dependent on the kind of change. That is, the more scattered the solution is that is subject to change, the more significant are the efficiency gains if aspect-oriented concepts are applied.

**$H_{P5}$**: By separating cross-cutting architectural solutions using aspect-oriented concepts, the efficiency of creating and maintaining architectural designs in the context of product lines is significantly improved.

The controlled and efficient separation of modular solutions to quality requirements supports the handling of variant models in the context of product lines as well. In the case of single system development, the selection of design alternatives leads to a set of design options that are discarded. However, the very same mechanism can be utilized in the context of product lines, since selecting different design options for different quality concerns can be interpreted as instantiating generic architectures to a particular product instance. For example, if a product line offers products ranging from low-end to high-end solutions, the most expensive variants are supposed to exhibit a high level of performance as compared to cheaper variants of the product line.

## 1.5 Outline

Based on the research questions and challenges described above, the thesis is structured as follows:

In Chapter 2, the foundations for the work presented in this thesis are explained. In particular, I point out the drawbacks of current state of the art concepts for realizing the principle separation of concerns at the architectural level.

Chapter 3 defines an architectural meta-model and relates all terms and notions for architectural design and modeling.

In Chapter 4, the aspect-oriented model for architecture design is introduced. The core idea is to modularize architectural solutions that would cut across the architecture otherwise. To that end, an extension of the meta-model is presented showing how aspect-oriented concepts map to architectural elements as defined in the meta-model.

In Chapter 5 the architecture design method is presented, showing how to utilize the meta-models defined in the previous chapters. The method addresses compositional issues of aspect-oriented architectures and gives guidance on how to design architectures in the context of change.

Chapter 6 presents the tooling that supports the method by automating important steps as described in the design method. The tool support is realized as extensions to the IBM Rational Software Architect®.

In Chapter 7, the validation of the approach is presented. To validate the hypotheses regarding efficiency and effectiveness of the proposed approach, an experiment was conducted. This chapter presents the experiment design, execution, analyses, and interpretations of the results.

Chapter 8 concludes this thesis by summarizing the findings and contributions and providing an outlook on future topics for further research.

# 2 Foundations

In order to relate the phenomenon of cross-cutting concerns to software architecture, I introduce the foundations of software architecture as described in the software engineering literature. In addition, I scope out what the reasons for cross-cutting concerns are and how they impact architectural design in general. In order to clearly distinguish the contributions of this thesis from existing work, an overview of state-of-the-art concepts is given that support separation of concerns at the architectural level and show the reasons why existing concepts are insufficient for dealing with cross-cutting concerns at the architectural level.

## 2.1 Software Architecture Definitions

In research as well as in practice, there exist many definitions for architectures. However, all definitions, in one way or another, refer to a common set of things: elements or components, relationships between them, the properties or form of all elements, as well as the rationale describing the existence and appearance of the resulting structures given a set of requirements to be fulfilled.

In order to come up with a meta-model that interrelates architectural concepts, the definition is based on two definitions of software architecture most frequently referred to in the software engineering literature. The first one stems from the Software Engineering Institute [CK03].

**Definition – Software Architecture**

The architecture of a software-intensive system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

The second definition is defined by the IEEE 1471-2000 standard for architecture description [MEH01].

**Definition – Software Architecture**

The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

The most interesting aspect that is added by the definition of IEEE 1471-2000 is that architecture comprises principles that guide a system in its design and evolution. Hence, besides describing what architecture is, the importance of the rationale is confirmed by that definition as one of the fundamental concerns in architecture design.

As mentioned in the previous chapter, architecture design is about making design decisions, balancing design decisions, and trading off architectural concerns [CKK01]. In order to do all these things, architecture design decisions need to be manifested in one form or another. Without manifestation, architectures would be implicit and very limited regarding their leverage. Therefore, architectural design decisions are recorded using so-called architectural models [TMD09]. However, the facts that "every architectural model is an approximation of reality" and therefore "is only partially accurate and partially complete", need to be taken into account whenever architectures are to be exploited [RW05].

**Definition - Architectural Model**

"An architectural model is an artifact that captures some or all of the design decisions that comprise a system's architecture. Architectural Modeling is the reification and documentation of those design decisions. Depending on the level of rigor and formality, architectural models are more or less expressive for visualizing, evaluating, and evolving an architecture." [TMD09]

Architectural models are created using modeling notations or modeling languages. There exist quite a number of modeling languages, ranging from highly ambiguous languages to strictly defined formal languages. There is a natural trade-off between expressiveness and formality of modeling notations. If a modeling notation is very expressive, practitioners are willing to adopt such a language without any problems. Natural languages mark the one side of the spectrum that spans expressiveness and formality. Since natural languages are very expressive but not formal, they cannot be processed by machines to validate architectural design decisions. Therefore, validation of design decisions needs to be done manually through inspections. Such kinds of inspections are open to misinterpretation and it is questionable if complexity can be coped with if models are described by natural languages.

On the other end of the spectrum, there are modeling notations that are very formal, supporting machine executable and therefore quantifiable validations. Most of today's so-called architecture description languages (ADLs) belong to this category of modeling notations. Most ADLs were developed during the 1990s when there was a rising understanding that architectures needed appropriate descriptions in order to be effective. However, almost all ADLs originate from research efforts and are rarely applied in industrial practice today [WH05]. Not surprisingly, there is almost no or only limited tool support for ADLs.

The Unified Modeling Language (UML) [UML08] is a graphical modeling notation that can be placed in the middle of that spectrum. UML is syntactically rich and it is more precise than natural or purely graphical notations (e.g., MS Powerpoint™, MS Visio™, etc.). However, since the UML is still ambiguous, UML offers so-called stereotypes and tagged values that add information to the models in order to increase the semantics and reduce the degree of misinterpretation. In practice, there exist numerous tools supporting the UML and its extensions since the UML is considered a de-facto standard in industry.

Given these facts, the work presented in this thesis aims at utilizing the UML in order to benefit from both ends of the spectrum. On the one hand, UML is considered a de-facto standard in industry; on the other hand, UML can be geared towards more rigor and formality.

## 2.2 Design Dimensions

Architecture in general is used for different purposes at different places throughout the lifecycle of a system [TMD09]. Depending on the stage of development, architecture needs to support different activities and therefore answer different kinds of questions. The activities that I focus on are activities of architecture design. Architecture design deals with the manifestation of design decisions that address non-functional requirements stated by a set of stakeholders. That is, the stakeholder concerns determine what to model and to which level of detail.

In order to systematically connect stakeholder concerns to architecture design activities, stakeholder concerns are classified according to the goals they aim at. For example, according to [ISO01], stakeholder concerns can be mapped to external and internal product properties. External properties are properties that are exhibited by the product during runtime. Internal properties are “typically static measures of intermediate products” [ISO01]. Hence, stakeholder concerns need to be addressed by appropriate architectural models that allow reasoning about internal and external properties alike.

For these reasons, architectural models are used for answering either one of the following types of questions:

1. How is the software structured as a set of elements that have runtime behavior and interactions?

2. How is the software structured as a set of implementation units?

3. How does the software relate to non-software elements in its environment?

Consequently, there are three main dimensions of design, each dimension giving answers to different questions. For the remainder of this thesis, I will refer to the dimensions as component-connector, module, and allocation dimension (Figure 8). The **component-connector** dimension answers questions related to runtime structures of the system, the **module** dimensions answers questions related to the structure of implementation units, and the **allocation** dimension relates all software concerns to non-software elements. Typical examples of allocations are team assignments relating implementation units to teams, or deployments relating software units to execution environments.

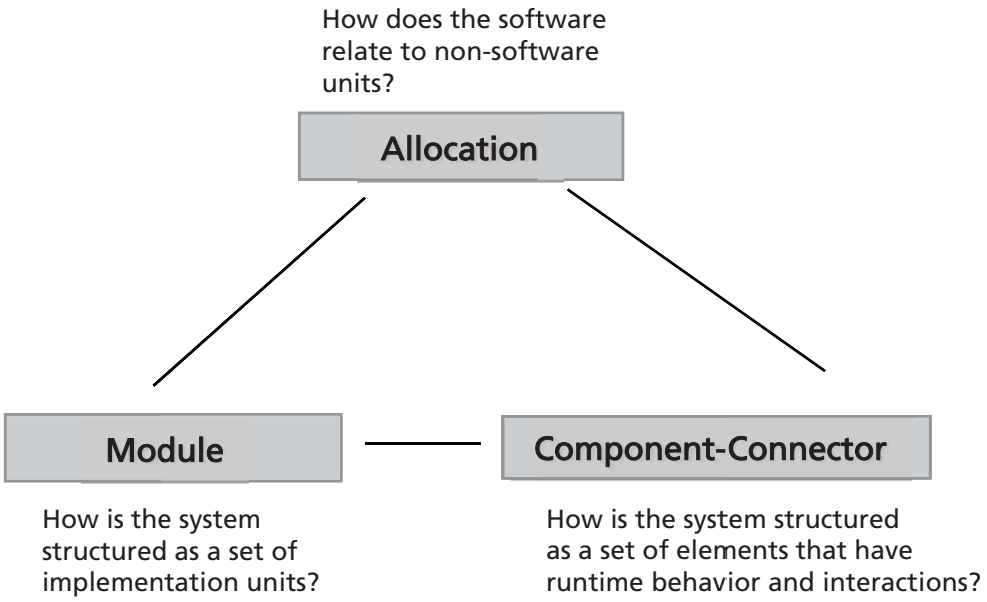



Figure 8 Main dimensions of architecture design

In the following, these design dimensions will be described in more detail.

### Component-Connector Dimension

In the component-connector dimension (C&C), the system is described in terms of its runtime behavior. That is, based on components that have runtime presence, it can be specified how the collaboration among these components has to take place. Typical questions that can be answered using the C&C dimension are: "What is the mean response time of a specific function?" or "How do elements collaborate to achieve a certain goal?".

According to [TMD09], components and connectors (C&C) address the following concerns:

- **Processing**, which can be referred to as functionality or behavior (functional responsibility assignments)
- **State**, which may be referred to as information or data
- **Interaction**, which may be referred to as communication, coordination, or mediation

In order to specify the collaboration behavior of the component types, the dynamic structure of components has to be modeled. During runtime, these components are interconnected by so-called connectors. Connectors specify the coordination behavior via so-called connector protocols. Basically, connector protocols show how components are supposed to work together in terms of coordination. In terms of modeling, the coordination behavior can be modeled by using message sequence charts, UML sequence diagrams, or UML state charts.

### Module Dimension

The module dimension denotes a collection of principles that can be applied when a static model of the architecture is supposed to be created. Looking at architectural principles, modules are structured using decomposition and by establishing implementation relationships between them. The module dimension describes how different implementation units depend on one another. As a consequence, modules are considered to be close to a system's implementation. By capturing these relationships, dependencies can be analyzed that might exist among modules in the presence of a change scenario, for instance. Here, I assume that concerns like functional decomposition and dependencies between modules are defined in such kinds of views.

The main concerns addressed by the module architectural dimension are:

- Functional decomposition of a system into subsystems
- Module interface definitions
- Information flow and data structures

The module structures are thus the starting point for estimating the impact of a particular change.

### Allocation Dimension

The allocation dimension is mainly concerned with assigning software artifacts to other entities. In the case of the definition given by [CGB+02], the allocation dimension comprises three distinct concerns: deployment, implementation, and work assignment.

• Deployment: The deployment shows how software units are mapped to the HW environment comprised of processing nodes, network interconnections, or disk storage facilities.

• Work assignments: Modules are mapped to teams.

• Implementation: Modules are mapped to infrastructure entities (e.g., configuration management) to support multi-team development.

As can be seen from this explanation, allocation is relevant for answering questions about both runtime and development-time quality attributes. If runtime properties are addressed, components and connectors need to be considered in combination with deployment models. Likewise, in the case of development-time properties, module structures in combination with work assignments and implementation allocations need to be considered.

## 2.3 Architectural Tactics, Patterns and Strategies

In architecture design, the architect needs to come up with solutions that adequately address the stakeholder concerns at hand. For addressing non-functional concerns, the main underlying principle of solution design is called *operationalization*. That is, non-functional requirements are refined and eventually mapped to new functional requirements. In general, operationalization is driven by an *architectural strategy*.

**Definition – Architectural Strategy**

An architectural strategy is a high-level decomposition of a quality goal into sub-goals.

Each of the sub-goals is then addressed by an appropriate set of solutions. Architecturally speaking, there are two notions of relevance regarding architectural solutions: architectural tactics and patterns.

**Definition – Architectural Tactic**

An architectural tactic is a design decision that influences the control of a quality attribute response [CK03].

In other words, tactics are design options for the architect that support the achievement of non-functional concerns. Looking at the design dimensions of architectures in general, architectural tactics potentially have a multi-dimensional nature. That is, an architectural tactic most likely not just influences one dimension (e.g., component and connector) but might also influence the allocation dimension in terms of deployment. Looking at the "Heartbeat" example illustrated in Figure 9, it can be seen that the tactic comprises structural (a) as well as behavioral (b) aspects. The purpose of the "Heartbeat" tactic is to provide a solution to the problem of detecting the failure of particular components in a system. The failure of a component is deduced from the missing heartbeat the component is supposed to send within a specified time span. The structural models of the tactic illustrate what elements need to collaborate, whereas the behavioral models show how the elements are supposed to work together.

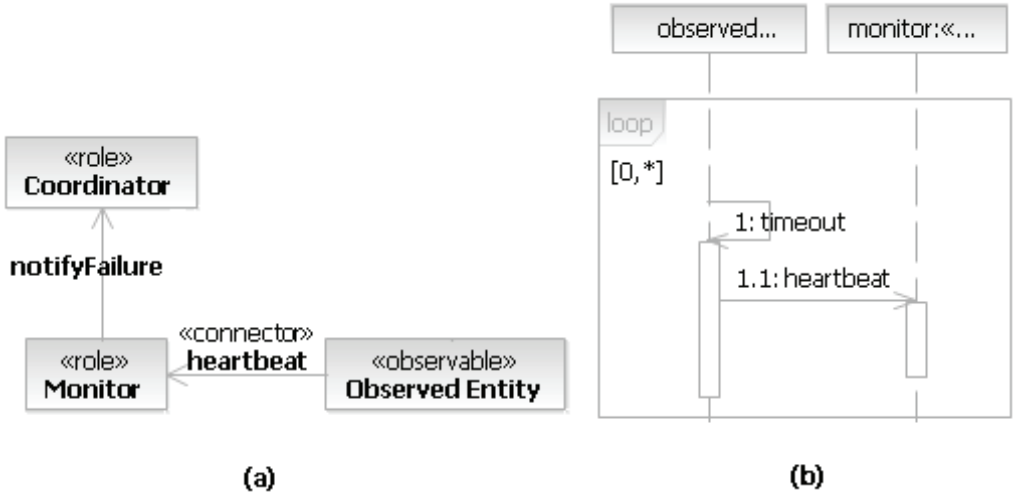



Figure 9 Sample tactic – "Heartbeat"

Although the notion of patterns has been coined by the design pattern community [GHJ+95], the concept of a pattern also exists at the level of architecture design. In that sense, architectural patterns are reusable solutions or concepts that can be transferred to different contexts.

**Definition – Architectural Pattern**

An architectural pattern packages a set of architectural tactics [CK03].

According to this definition, the sample pattern as depicted in Figure 10 captures design knowledge by packaging two tactics into one, namely the "Heartbeat" tactic and the "Active Redundancy" tactic. The heartbeat tactic in the pattern corresponds to its description as given above. The "Active Redundancy" tactic aims at switching resources in terms of

functional responsibilities. In the example case, the "Active Redundancy" tactic switches the connector behavior of the *Communication* connector if the monitor detects a failure of the observed entity. All entities communicating with the *Observed Entity* that failed are re-routed to the *Backup Entity*.

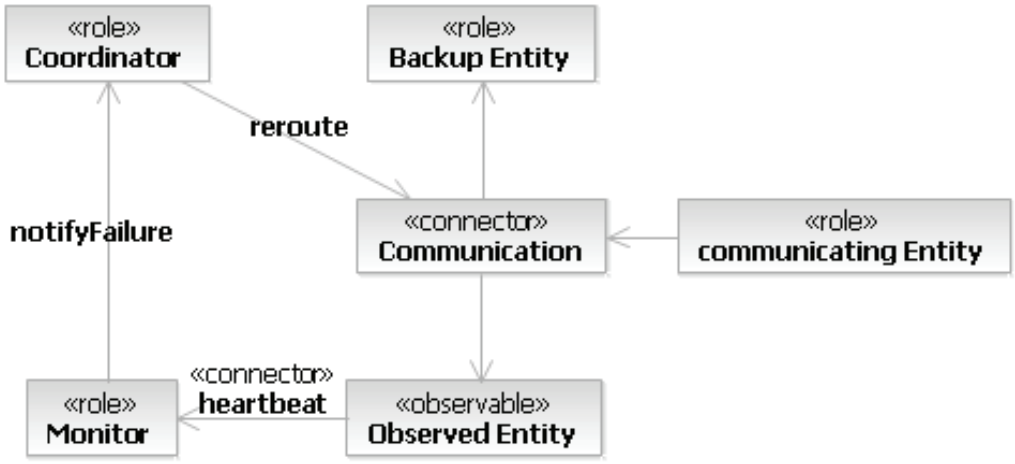



Figure 10 Architectural Availability: combining "Heartbeat" and "Active Redundancy"

As can be seen in Figure 11, the architectural pattern defines additional assumptions in terms of its multi-dimensionality. Here, the pattern states that the *Observed Entity* needs to be deployed on a different physical node than the *Backup Entity*.

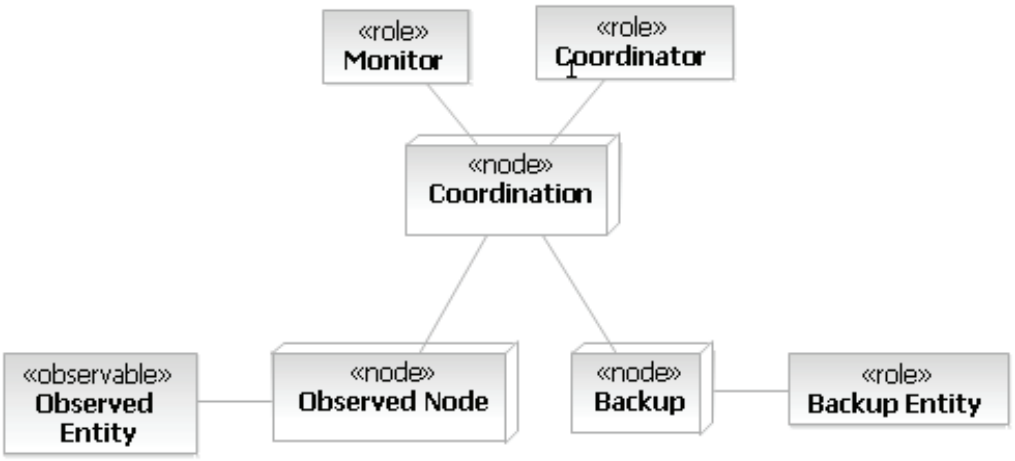



Figure 11 Architectural Availability Pattern: deployment implications

# 2.4 Architectural Separation of Concerns

## 2.4.1 Views and Perspectives

Traditionally, separation of concerns at the architectural level has been addressed by applying so-called architectural views [MEH01], [Kru95], [RW05].

**Definition – Architectural View**

An architectural view is a representation of one or more aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders [RW05].

In other words, an architectural view "is a set of design decisions related by a common concern (or set of concerns)" [TMD09]. There exist several works in the literature that emphasize the need for different views during architecture design and evaluation. The most prominent ones are Kruchten's 4+1 view model [Kru95], SEI's Documentation Scheme [CGB+02], Siemens-4-Views [HNS00], as well as the IEEE 1471-2000 Standard [IEEE1471], [MEH01]. In this context, the IEEE 1471-2000 standard introduced the term viewpoint for the first time.

**Definition – Architectural Viewpoint**

A specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis [MEH01].

Viewpoints help in deriving views; however, there is no general set of views that adequately and universally describe the architectures of software-intensive systems. One of the challenges in architecting is to select those views that convey information which is useful for analyses or communication. The selection itself depends on questions like:

- Which kind of system does the architecture describe?
- What concerns do the stakeholders have?
- Which analyses should be possible?

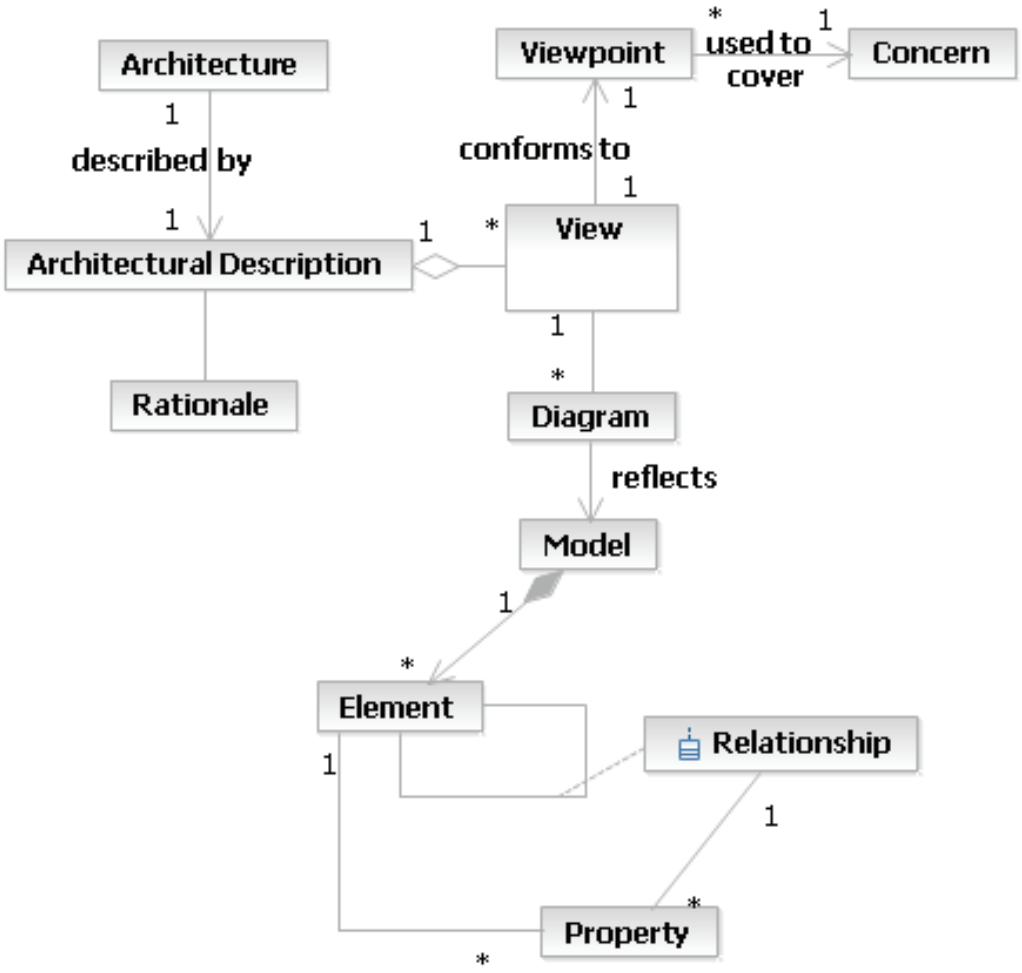



Figure 12 Architectural views in context

Even though there is no general answer to which views to select, our experience has shown that the following set of views represents a good starting point in any architecture endeavor:

1. **Context Views:** Describe the context of the system to be developed. That is, they show all surrounding systems that interact with the system to be developed. In short, it is the purpose of the context view to show what the scope of the system is by distinguishing it from the systems that it is connected to.

2. **Component-Connector Views:** The component-connector views describe the decomposition of the system into functional components and connectors allowing the components to communicate.

3. **Module Views:** The module views describe the decomposition of the system into implementation units, also referred to as modules. From this view, the assignment of tasks to developers can be derived.

4. **Deployment Views:** The deployment view associates runtime artifacts like components and connectors with concrete computational units and communication paths. That is, components are assigned to nodes and (implicitly) connectors to communication paths.

5. **Team Allocation Views:** Team allocation views allocate resources to development activities and related artifacts.

This set of views mainly supports the decomposition of a system into smaller units, which somehow makes these views form the "quasi-standard" set of views. Up to this point, all views described have not explicitly addressed non-functional concerns. Since we need to be able to answer questions about the fulfillment of non-functional concerns, we need to collect information from different views and extend or analyze that information in a very special way. This is what perspectives are used for.

**Definition – Architectural Perspective**

An architectural perspective is defined as "a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views" [RW05].

For instance, in order to answer questions about the maintainability of a system, we need to combine information about the team structure and its elements, including their properties (e.g., productivity, availability, etc.), with information taken from the module view describing the relationships of implementation units of the system under investigation. In order to answer questions about performance, we need to use information from the deployment view, telling us about hardware properties (e.g., bandwidth) and physical separation of software units in combination with information about the logical interactions of components (taken from the component-connector view).

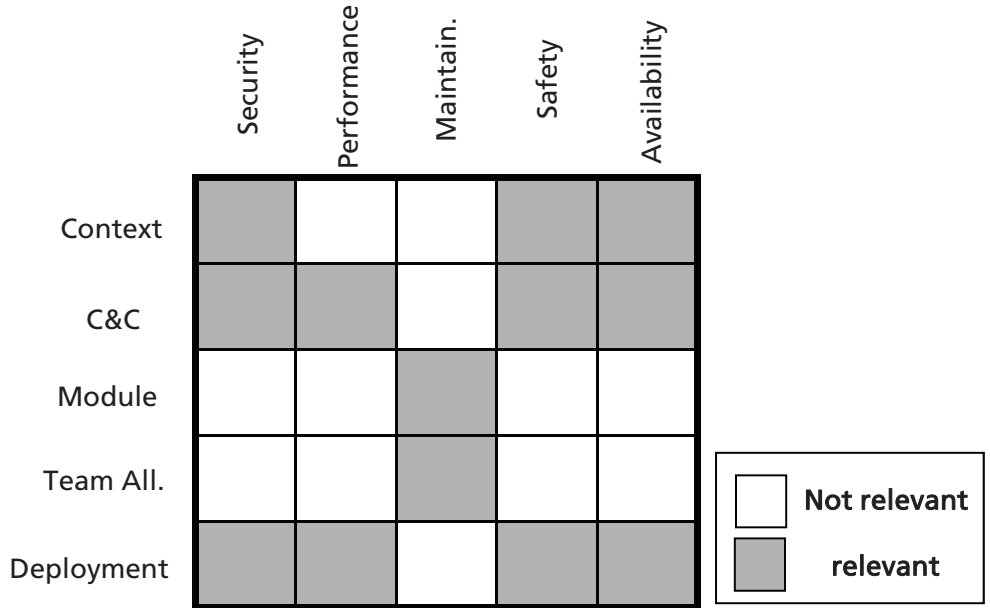



Figure 13 Quality specifics cut across views

The resulting challenge regarding views and perspectives for separating concerns is due to the fact that all views and perspectives need to operate on the same architectural model. The creation and evolution of architectural views based on a consistent architectural model turns out to be the major challenge in that regard.

In the context of architectural views, there exist so-called architecture frameworks support view-based architecture development. Architecture frameworks are a means for packaging existing knowledge and best practices such that architects have appropriate support for architecture design, in particular architecture documentation. However, the frameworks usually only provide a starting point in terms of a set of questions, processes, guidelines, or a set of viewpoints that an architect might consider during architecture design.

In the following, I will briefly describe the most prominent examples of architecture frameworks, and characterize them regarding their support with respect to cross-cutting concerns.

### DoDAF

The Department of Defense Architecture Framework (DoDAF) [DoD04] defines a standard for documenting system architectures. DoDAF leaves many choices to the user of the framework such as notation selection, consistency checks, and so forth. DoDAF comes with a set of viewpoints that are recommended to be considered during architecture documentation, namely:

- Operational view: Describes the context of the mission of the system
- Systems view: Describes the system parts in detail
- Technical standards view: Shows what technical standards are applied
- All views: Captures cross-cutting concerns, but at a very high level of abstraction

DoDAF, however, does not mandate (and thus also does not provide) any method to be applied, and especially does not provide any support regarding the handling of cross-cutting concerns.

### TOGAF

The Open Group Architecture Framework (TOGAF) [ToG03] is a framework for developing enterprise architectures. That is, the framework supports enterprise architects in their need to design an IT–based infrastructure that supports the implementation of specified business goals. TOGAF does consider factors beyond software and hardware, such as human factors as well. In short, TOGAF consists of three major items:

- ADM: Architecture development method
- Virtual repository: Comprises models, patterns, architecture descriptions, etc.
- Resource base: Comprises guidelines, templates, etc.

Similar to DoDAF, TOGAF does specify what to do but leaves the decision about how to do it to the implementer of the framework. Regarding the handling of cross-cutting concerns, TOGAF does not provide any means per se.

### RM-ODP

The Reference Model for Open Distributed Processing (RM-ODP) [ISO98] is an ISO standard for describing open, distributed processing information systems developed to run in an environment of independent, heterogeneous processing nodes connected by a network [TMD09]. The RM-ODP defined five viewpoints to be considered during architecture design, namely:

- Enterprise (context): Describes the enterprise's context
- Information: Shows the distribution and processing of information
- Computational: Shows computational entities and their relationships
- Engineering: Shows processes that interrelate the computational entities
- Technology: Describes the technologies employed

The framework contributes to ensuring that the architect thinks about and documents relevant aspects of the architecture that impact various qualities. In general, the RM-ODP is similar to DoDAF: There is no methodical support in terms of how the viewpoints are supposed to be used, and there is no support for dealing with cross-cutting concerns.

### Zachman Framework

The Zachman Framework [Zac87] is a widely used approach for developing and/or documenting enterprise-wide Information Systems Architectures. The framework comes with multiple perspectives of the overall architecture for supporting the organization, access, integration, interpretation, development, management, and changing of a set of architectural representations of the organization's information systems.

Akin to other architecture frameworks, the Zachman framework does not provide any concrete methodical guidance of how to instantiate the framework. Any appropriate approach, standard, role, method, technique, or tool may be applied. In fact, the Framework can be viewed as a tool for organizing any form of metadata for the enterprise.

In conclusion, the frameworks as such make a contribution at the conceptual level of architecting; however, they do not provide any means for facilitating design activities, especially regarding the systematic treatment of cross-cutting concerns.

### 2.4.2 Issues with Cross-cutting Concerns

In general, cross-cutting concerns come with three major challenges.

**Issue #1: Identification of Cross-Cutting Concerns**

In order to cope with the issue of cross-cutting concerns, they need to be identified in a first step.

That is, depending on the stage of development or on the kind of development artifacts at hand, it is not obvious whether or not a particular concern is actually cross-cutting. Cross-cutting concerns can exist at all levels of abstractions [TMA+04]; however, the existence of a cross-cutting concern at a particular level of abstraction does not imply the existence of the same cross-cutting concern at other levels.

**Issue #2: Separation of Cross-Cutting Concerns**

In order to tackle cross-cutting concerns effectively, they need to be separated from other concerns.

This is where architectural views as described in the previous sections come into play. As already indicated, architectural views in general are a means for separating concerns. However, since the views are defined on the same architectural model, they are likely to expose elements that are affected by other views as well. It has been recognized in research that "the drawbacks of crosscutting with respect to architectural views is similar to the drawbacks with respect to code, i.e. hampering reuse, maintenance and evolution of the architecture" [BH06].

**Issue #3: Integration of Cross-Cutting Concerns**

The third challenge is the integration of the separated concerns into a coherent system.

With the term "system" as used in the above problem statement, I refer to either one of the following: system model, system implementation, or executing system. Without integrating the concerns into a single system, there would be no realization of cross-cutting concerns in the end. However, the issue of concern integration is the main reason why separation of concerns using state-of-the-art view-based architecture approaches does not suffice. Currently, all view-based approaches support the separation issue; however, the integration issue remains unsolved.

View-based approaches only separate the concerns; they do not provide any means for integrating the concerns.

Conceptually, view-based approaches are similar to database queries. The only difference is that the role of the database is taken by the architectural model. Architectural views define queries on an architectural model and the query result comprises a partial representation of information stored in the architectural model. An important assumption of a query-based approach to separation of concerns is that the model comprises much more information that is indirectly related to the information displayed within a view. Consequently, when the architectural model is changed based on the partial information rendered by that view, the underlying reasoning for the change is only of limited value because of the limited scope of the information. An informed change of an architectural design based on the views needs to take all relevant architectural information into account that is impacted by that change. However, this stipulation is not fulfilled by state-of-the-art view-based architecture approaches.

# 3 Architecture Meta-Model

*"As far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality." Albert Einstein*

In the literature, there exists no common meta-model that relates all important facets of architecture in a conceptualized way. However, in order to address the research issue of cross-cutting concerns in the realm of architectural interconnection, a meta-model is needed that provides a basis for understanding the relationships between the different architecture design dimensions. In this chapter, I describe such a conceptualized meta-model for architecture that integrates existing concepts regarding architecture modeling and aligns them according to the design dimensions presented before.

The first part of this chapter reflects the main architectural design dimensions (C&C, Module, and Allocation) as introduced in Chapter 2 in the context of architecture modeling. The second part of this chapter establishes the interrelationship between the different meta-models created in their own right. The third part shows how architectural tactics relate to the meta-model and what entities are potentially affected by cross-cutting concerns.

## 3.1 Element and Relation Types

The architectural dimensions as introduced in Chapter 2 define runtime, development, and allocation elements and structures to be considered during architecture design. As shown in Figure 14, there are always three types of elements to be defined in all of the high-level areas: software entities, processes, and contextual entities. For each of the areas, the entities have a different manifestation. For instance, in case of development, contextual entities need to be defined that support the development of a system; in case of runtime, contextual entities need to be defined that are of relevance during runtime, such as the runtime environment. Although the main focus of this thesis is on the software elements within the runtime space, I present a complete architecture meta-

model in order to provide a context that is broad enough to fit the particular extensions.

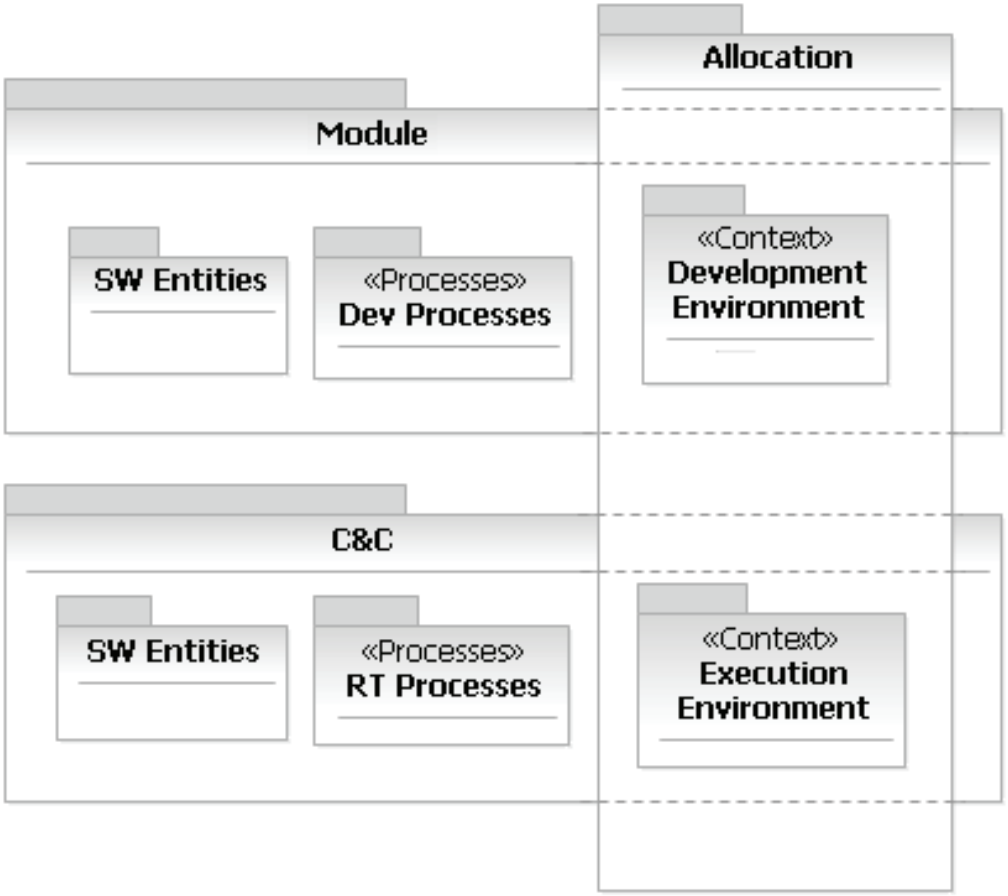



Figure 14 Design dimensions to be reflected by the architectural meta-model

### 3.1.1 Components and Connectors

As described before, components and connectors (C&C) describe runtime entities of a system and show how these elements are supposed to collaborate. Given that definition, the purpose of an architectural model comprised of components and connectors is to represent dynamic aspects in terms of behavior of entities and interactions among those entities. In the following, these elements and relations are defined in more details.

**Components**

Since there is no commonly agreed on definition of a component, we base our architectural meta-model on aspects postulated by one of the most prominent definitions of components.

**Definition - Software Component**

A *software component* is a unit of composition with **contractually specified interfaces** and context dependencies only. A software component can be **deployed independently** and is subject to composition by third parties [Szy98].

Concerning relation types, there is only one relation type that is of relevance in the context of components and connectors: the connector.

**Connectors**

The elements that specify how components interact at the architectural level are referred to as "connectors". Connectors allow architects and engineers to compose heterogeneous functionality, developed at different times, in different locations, by different organizations [MMP00]. As such, connectors can be thought of quite appropriately as "the guards at the gate of separation of concerns" [TMD09].

**Definition - Software Connector**

A software connector is an architectural element tasked with **effecting and regulating interactions among components**. In other words, a connector performs transfer of control and data among components. [TMD09]

Examples of simple connectors are "procedure call" or "shared data access" [CGB+02]. I stress the fact that connectors are first-class entities with potentially high complexity since they might also offer non-simplistic services like transactions, messaging, and persistence for instance [Sha94]. In terms of architectural decisions, connector design is a non-trivial endeavour, since the architect gets lots of options to choose from. For instance, a connector could be broadcasting messages or events to any component that is interested or only to a subset of components. Further, the connector could stipulate that the sending component suspends its operation until the transmission process is complete. In case of volatile networks, a connector could also be in charge of collecting, reordering, assembling, compressing, encrypting, or filtering the messages it is supposed to transmit. When these responsibilities are taken into account, it can be seen that connectors can be sophisticated and highly complex design entities that need to be constructed and verified with care.

The meta-model for components and connectors as depicted in Figure 15 defines that a component provides a number of ports and can be decomposed into a number of (sub-)components. Ports are the communication end points between connected components. A port provides and/or requires a number of interfaces. "A connector type is defined by

a set of *roles* and a *glue* specification. The roles describe the expected local behavior of each of the interacting parties. That is, they act as a *specification* that determines the obligations of each component participating in the interaction [AG97]". In the meta-model as depicted below, the role is used for connector specification without a concrete set of components that should be connected using that connector. When the connection is established, the roles are replaced by the respective ports that need to comply with the role specification. The connector protocol (sometimes also referred to as "glue") coordinates the interaction of the connected components.

Meta-model:

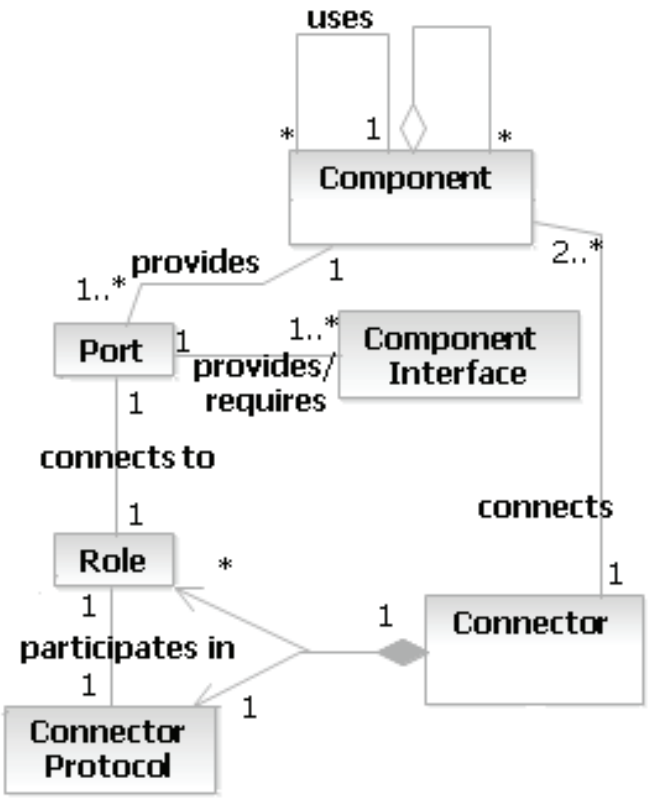



Figure 15 Component-connector meta-model

Figure 16 shows the notational elements for modeling components and connectors. For representing components and interfaces, the notational elements are used as defined by the UML.

Notation:

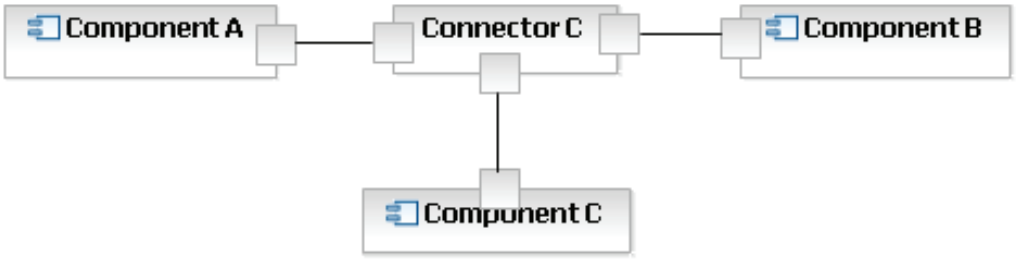



Figure 16 Notation for C&C structures

Figure 17 shows a sample model that specifies a client-server component structure. The upper part of the figure shows the structure of the components. The connection between the components is established by the *serverCall* connector. The *serverCall* connector itself is specified explicitly as illustrated in the lower part of Figure 17. I stress the fact that connector semantics can be modeled by means of UML sequence diagrams.

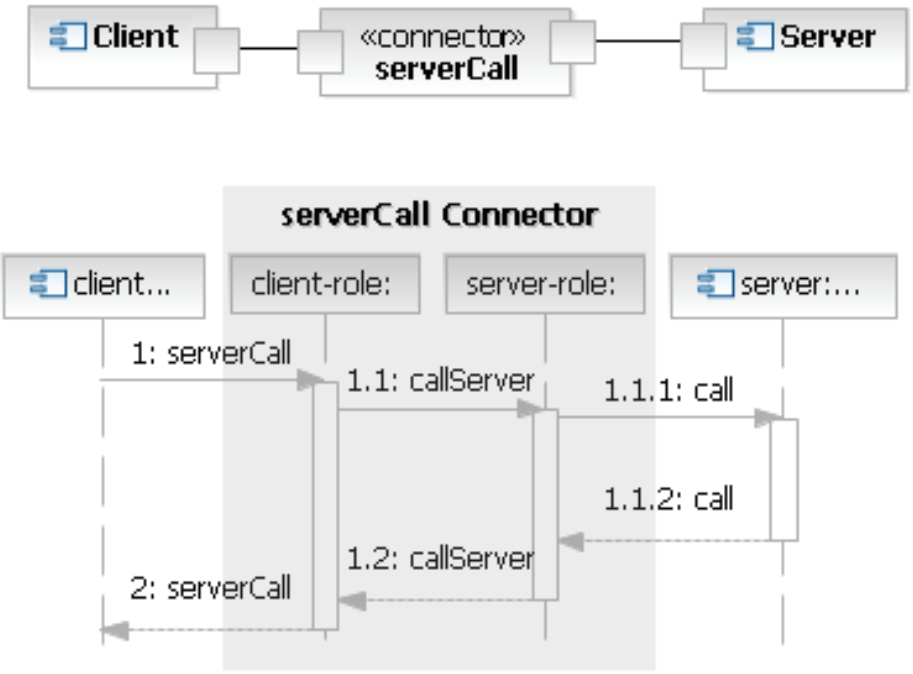



Figure 17 Example C&C model

### 3.1.2 Modules

In contrast to components, modules refer to design-time entities. A module suggests encapsulation properties, whereas components might not be well encapsulated [GAO95]. Additionally, it might not be possible to deploy a module independently from other entities. This is due to the fact that modules usually depend on lots of other modules in order to be compiled and installed.

**Definition - Module**

A module is a code unit that implements a coherent set of responsibilities. A module can be a class, a collection of classes, a layer, or any decomposition of the code unit [CGB+02].

From the perspective of a developer, it is the modules that are considered the blueprint for the implementation rather than the components as defined by component-connector models.

**Dependencies**

Abstractly speaking, in the realm of modules there exists only one type of relation: module dependencies [CGB+02]. A dependency relation might be instantiated to more concrete forms of specific relations.

Meta-model:

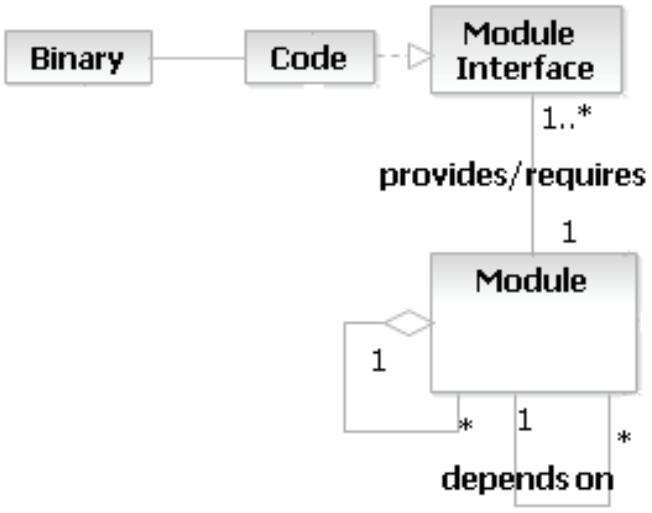



Figure 18 Module meta-model

A module dependency relation between modules denotes a (set of) concrete relationship(s) at the implementation level that establish a dependency among these modules. The most relevant concrete dependencies

on the module level are <<use>>-relations, <<is-allowed-to-use>>-relations, and <<is-a>>-relations. The module meta-model as depicted in Figure 18 defines that modules potentially aggregate other modules. This is important, since modules are decomposable into sub-modules. In addition, modules might depend on other modules. A module provides or requires a set of interfaces that specify the usage possibilities of the module. The provided module interfaces are eventually realized by program code that is translated into an executable binary format.

A sample model showing the use of the UML for describing module structures is shown in Figure 19. The *Clients* module aggregates two modules (*Remote-UI* and *PC*) and specifies a usage relation between client modules and service modules. Note that the use-relation might be subject to decomposition as well, since there might be more concrete use-relations between modules on lower levels.

Notation:

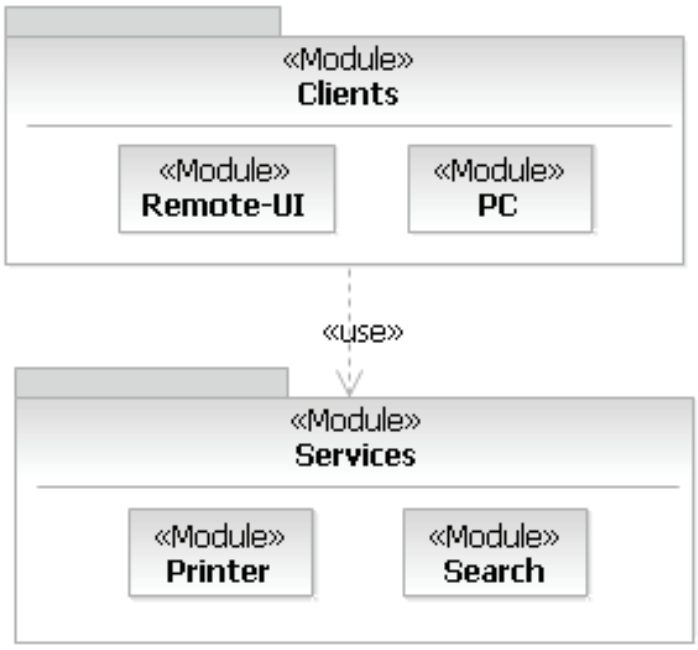



Figure 19 Example model using the module notation

### 3.1.3 Allocation Types

There exist a number of element and relation types belonging to the category of allocations. According to the definition given by [CGB+02], allocations are mainly concerned with assigning software artifacts to non-software entities. Non-software entities either refer to hardware, external systems, infrastructure, or people in the lifecycle of the system. In the following, these non-software entities will be described in more detail.

#### 3.1.3.1 Hardware

"Hardware aspects of the system under consideration are seldom far from the software architect's mind" [CK03]. This statement underpins that it is very important to know about the relevant properties of hardware elements that are involved in rendering the final system functionality. In case certain hardware properties were to be neglected, the outcome of the development effort would run the risk that the final system might not achieve its goals. Depending on the concrete requirements, different hardware properties might be of interest to the architect. For example, in the case of performance, the processing power of physical nodes as well as the bandwidth of the physical communication media are of relevance. If availability is an issue, the failure probabilities of the hardware units are of interest. If energy efficiency is a requirement, the energy consumption of all hardware units needs to be known. In general, all hardware elements that influence or are influenced by the software are potentially relevant for modeling.

**Element Types**

In terms of elements, there is a clear connotation to components and connectors, since components need to execute and interconnect using appropriate hardware units. Therefore, hardware components and physical connectors are the main elements in the hardware dimension.

**Relations**

In terms of relations within the hardware model, the link to the software world needs to be established. Using the terms as defined by the UML, the deployment of the software entities to the hardware units needs to be considered.

Meta-model:

The meta-model as depicted in Figure 20 shows the elements and relations that exist in the hardware allocation dimension. A hardware component (*HW component*) is a processing unit capable of hosting and executing software units packaged as so-called artifacts. Hardware components host one or more runtime environments that actually execute the software pieces of the system. The hardware components are interconnected by means of *Physical Connectors*. Physical connectors represent physical communication channels such as wireless LAN or bus systems as they are found in embedded systems.

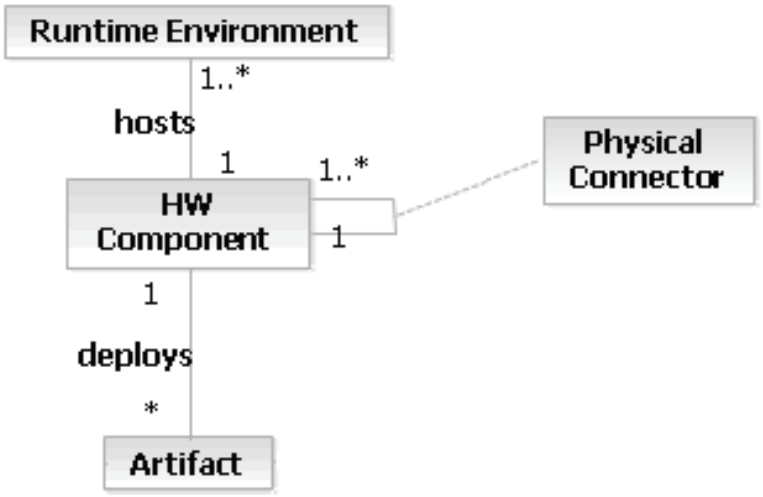



Figure 20 Hardware meta-model

The notational elements of the hardware allocation type are illustrated by the sample model depicted in Figure 21.

Notation:

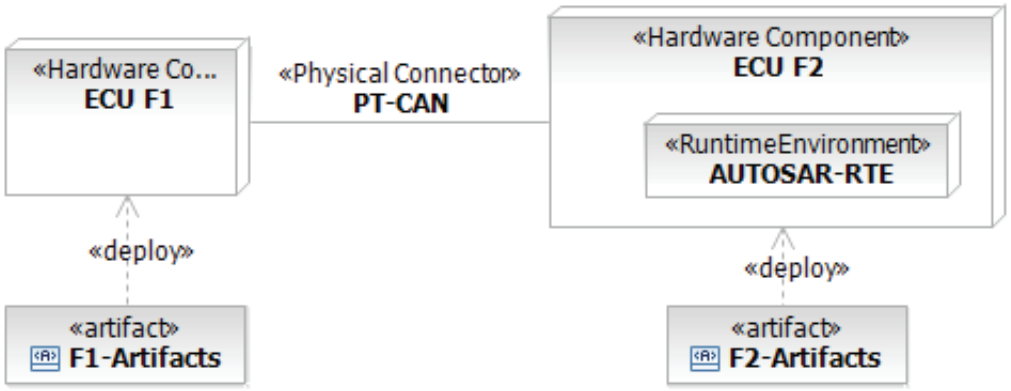



Figure 21 Example model using hardware notational elements

The sample model as shown in Figure 21 shows two hardware components that deploy one artifact each. The hardware component "ECU F2" also hosts the runtime environment "AUTOSAR-RTE".

### 3.1.3.2 External Systems

Usually, today's systems are not operated in isolation anymore. New systems are supposed to collaborate with existing systems like third-party or legacy systems. Despite the fact that the existing systems are usually not under control by the architect, there is an information need about existing systems that the new system needs to interface with. Here, the situation is similar to the hardware world: Depending on the requirements, different properties of existing systems need to be accessible to the architect.

Meta-model:

The meta-model of external systems as depicted in Figure 22 shows that an external system potentially provides one or more interfaces. Furthermore, an external system uses one or more communication paths.

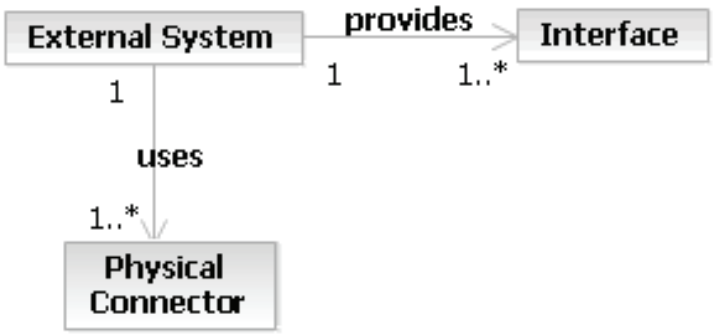



Figure 22 External systems meta-model

When an architecture is about to be designed, all relevant external systems need to be identified and relevant properties need to be specified as concretely as possible.

### 3.1.3.3 People

One of the most easily overlooked factors during software development is the human factor. Even though many steps during software development can be automated, lots of activities and decisions remain that have to be made by people. The set of involved roles and persons assigned to the roles have properties that need to be taken into consideration as well. For instance, when modules are assigned to teams for implementation, the team members themselves heavily influence the speed and quality of the development. The business goal of achieving a particular time to market is heavily dependent on the capabilities of the development and quality assurance teams. The pure software structures and dependencies are important; however, it is the combination of software structures with people that determines the final result in terms of quality and time.

#### Element types

The element types relevant in the people meta-model are roles, resources, tasks, and software artifacts. In general, there might be different roles involved in software development; however, the important aspect is that the architect is aware of what roles to consider and what properties of the persons that are assigned to the roles need to be taken into account. A typical list of roles involved in a software development project are requirements engineers or business analysts, architects, designers, developers, testers, and maintainers.

### Relations

Each of the roles has to perform a (set of) particular tasks defined on the software artifacts, such as elicits, develops, tests, maintains, evolves, and so forth.

Meta-model:

The meta-model as depicted in Figure 23 shows that resources are assigned to one or more roles. Each role has a number of responsibilities expressed by a need to perform certain tasks on a defined set of software artifacts.

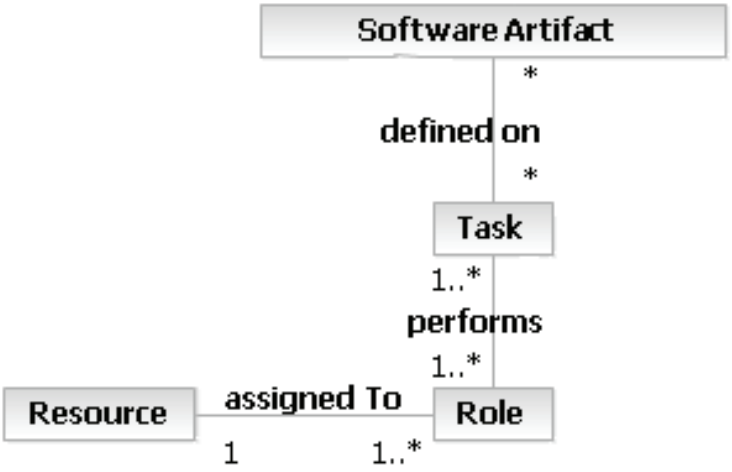



Figure 23 People meta-model

Notation:

As depicted in Figure 24, an instance of the people meta-model comprises resources, roles, and tasks. Here, a single resource (John) is assigned to two roles, namely *Developer* and *Tester*. The resource properties that are of relevance to the architect are attributes like "productivity" or "steepness of learning curve". These attributes can be used when role assignments need to be made in the face of pressing deadlines.

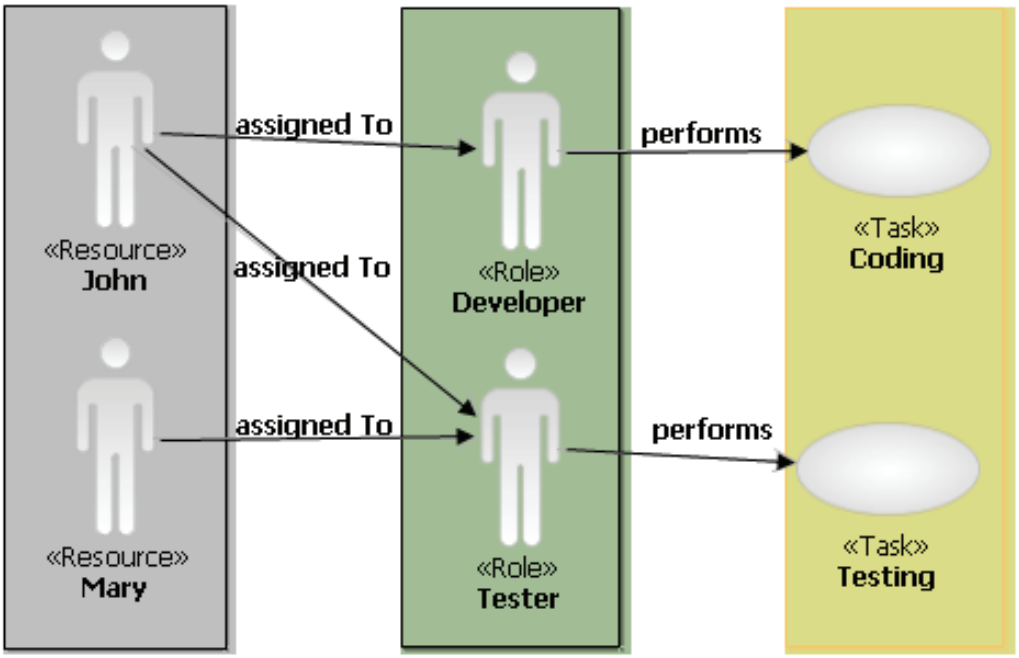



Figure 24 Example model showing people allocations

### 3.1.3.4 Infrastructure

At development and maintenance time, there are systems external to the software system that need to be taken into consideration as well. For instance, when maintainability is an issue, the infrastructure of the development teams is supposed to support the maintenance activities. Typically, in large projects a configuration management system is used for enabling concurrent development activities. At one point, technical decisions about the development environment need to be taken, and most importantly, these decisions should not conflict with other quality attributes the software is required to exhibit.

#### Element types

According to that description, infrastructure elements like frameworks and tools can be identified. Prominent instances of these categories in practice are *Configuration Management Systems, Code Development Frameworks, and Test Frameworks.* In general, these elements support lifecycle activities in the context of the product.

#### Relation types

The relation types that exists in the infrastructure are the <<supports>> and the <<performs>> relationships.

Meta-model:

As depicted in Figure 25, the infrastructure meta-model defines that frameworks and tools support tasks that are defined on the respective software artifacts.

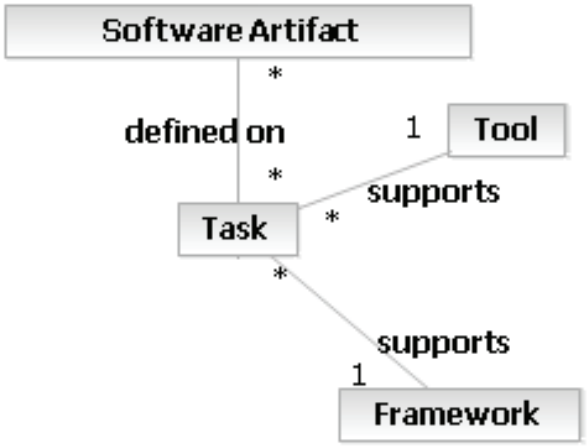



Figure 25 Infrastructure meta-model

Notation:

The example model shown in Figure 26 depicts an instance of the infrastructure meta-model as defined above. There exists a number of tasks supported by a configuration management tool (SVN: subversion [Sub10]) and a testing framework (JUnit [JUN10]) that support a set of the lifecycle activities (here: *Coding* and *Testing*).

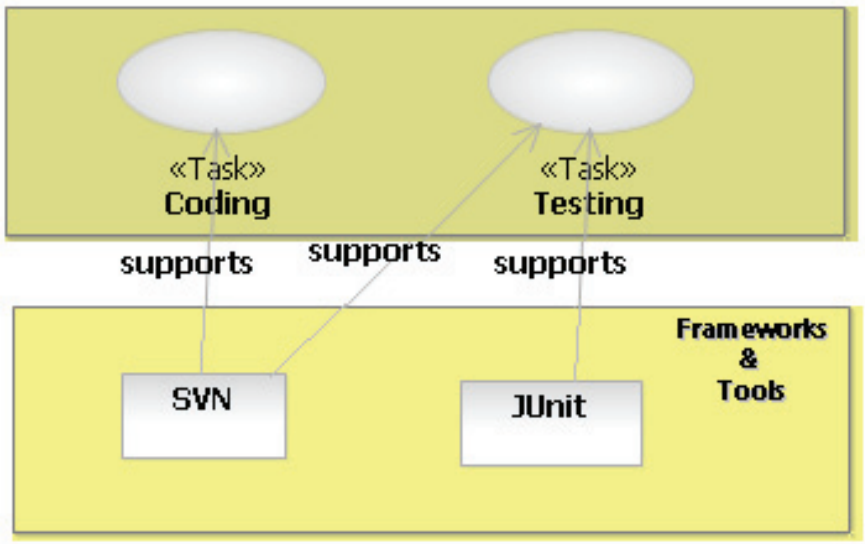



Figure 26 Example model for infrastructure

## 3.2 Inter-Element Type Relations

In this section, I investigate how the respective meta-models relate to each other. By systematically investigating relationships between the meta-model elements as presented in the previous sections, I aim at a coherent picture of architecture that appropriately relates all facets as described earlier. I start by mutually mapping each of the design dimensions to each other before all identified relationships are combined into a coherent overall picture.

### 3.2.1 C&C – Modules

The C&C architectural dimension and the module architectural dimension are related to each other since components and connectors are eventually realized by modules (see Figure 27).

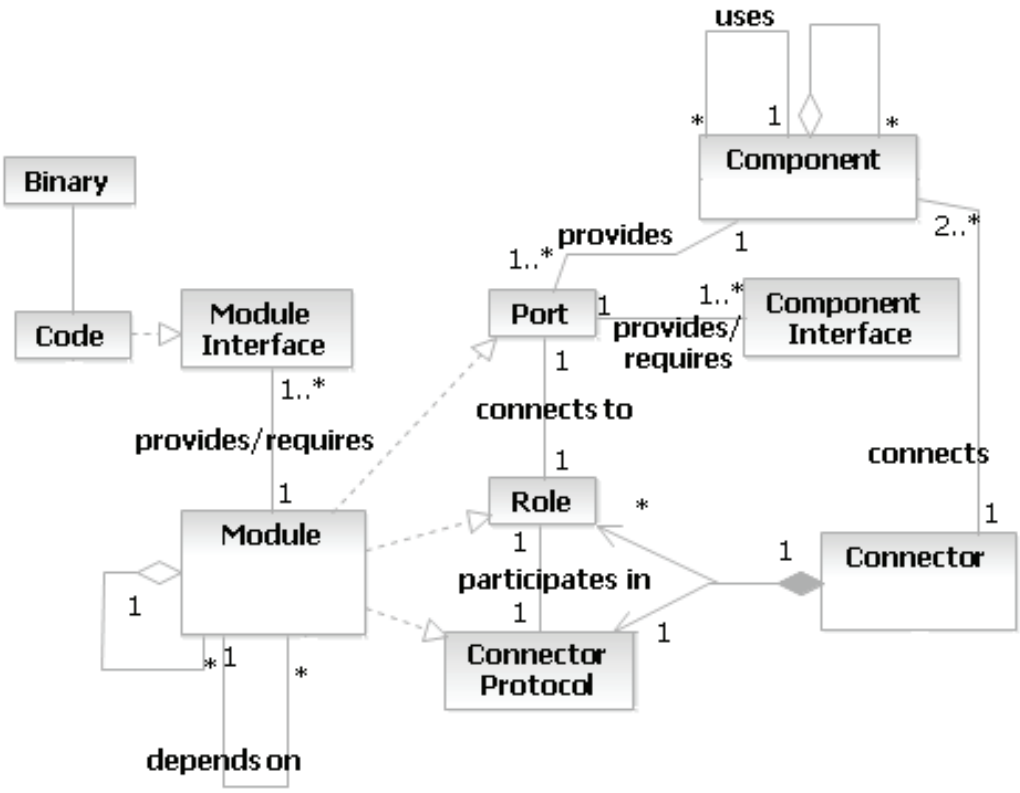



Figure 27 C&C and Modules: Meta-model relationships

That is, while a component represents a runtime entity, the realization of such an entity is a development-time module. In this case, a module or a set of modules eventually realizes the interfaces provided by a particular component port. The same holds for connectors; connectors define the interaction mechanisms of the components that exist at runtime, how-

ever, the interaction mechanisms need to be mapped to units of implementation or infrastructure services that realize the required communication facilities. In this case, a set of modules eventually realizes the role- and connector protocols.

### 3.2.2 C&C – Allocations

According to the definition given in Section 3.1.3, the allocation dimension denotes how software entities relate to non-software entities. In case of C&C, runtime elements are mapped to non-software elements. The most obvious relationship between the C&C and the allocation dimension is the deployment of components. The deployment defines the mapping of runtime entities to hardware units. Here, the element *Artifact* bridges the two meta-models, since an *Artifact* manifest components. Since there are allocations in terms of people and infrastructure as well, the element *SoftwareArtifact* links the meta-models of people and infrastructure.

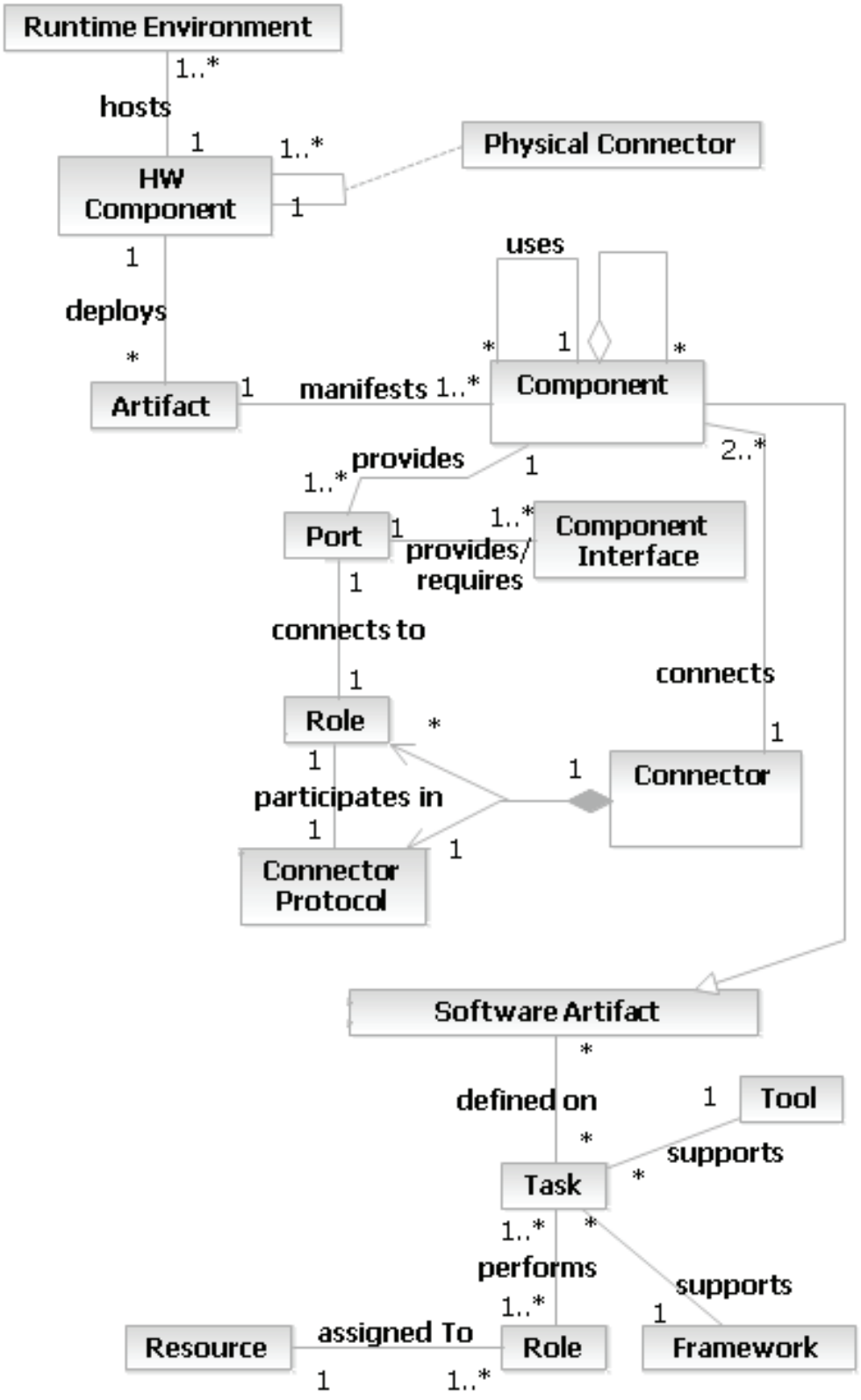



Figure 28 C&C and Allocations: Meta-model relationships

### 3.2.3 Modules – Allocations

Modules are eventually transformed into code units, which are in turn used for producing executable binaries. These binaries are then to be manifested in terms of artifacts that can be deployed (see Figure 29). As compared to the component-connector deployment relationships, a refined deployment of the architecture is created allowing for impact analyses regarding the module structures on runtime properties.

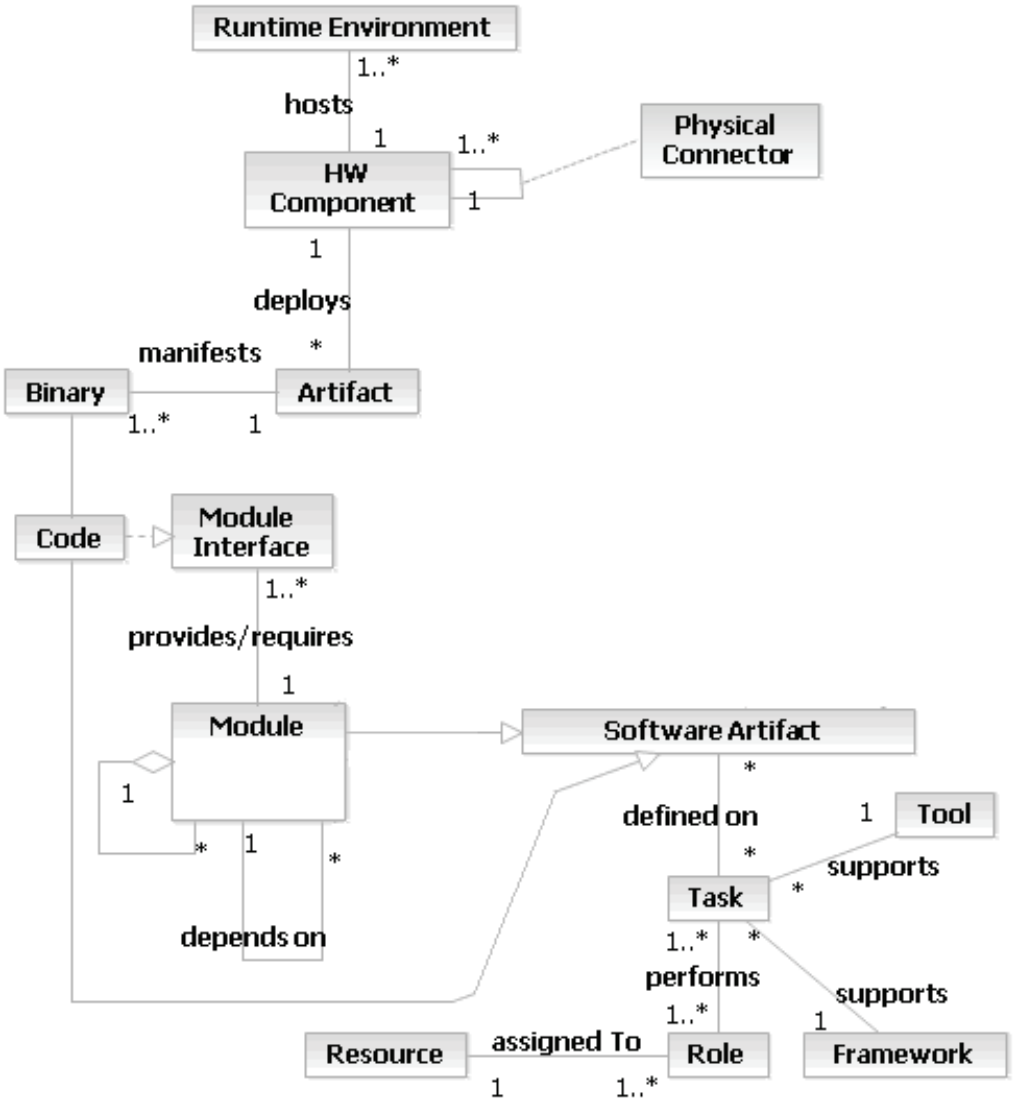



Figure 29 Modules and Allocations: Meta-model relationships

In addition to being allocated to physical entities, modules are to be allocated to other non-software entities as well, namely to teams or team members in terms of role assignments. Using this kind of mapping allows for analyses of development-time issues like buildability or maintenance. Another important allocation instance is the mapping of modules to infrastructure entities like configuration management systems, coding frameworks, or test frameworks. This mapping is realized by the element *SoftwareArtifact,* which forms a bridge from the module meta-model to

the allocation meta-models in terms of people and infrastructure. In Figure 30, an overview of the resulting meta-model for architecture development is depicted.

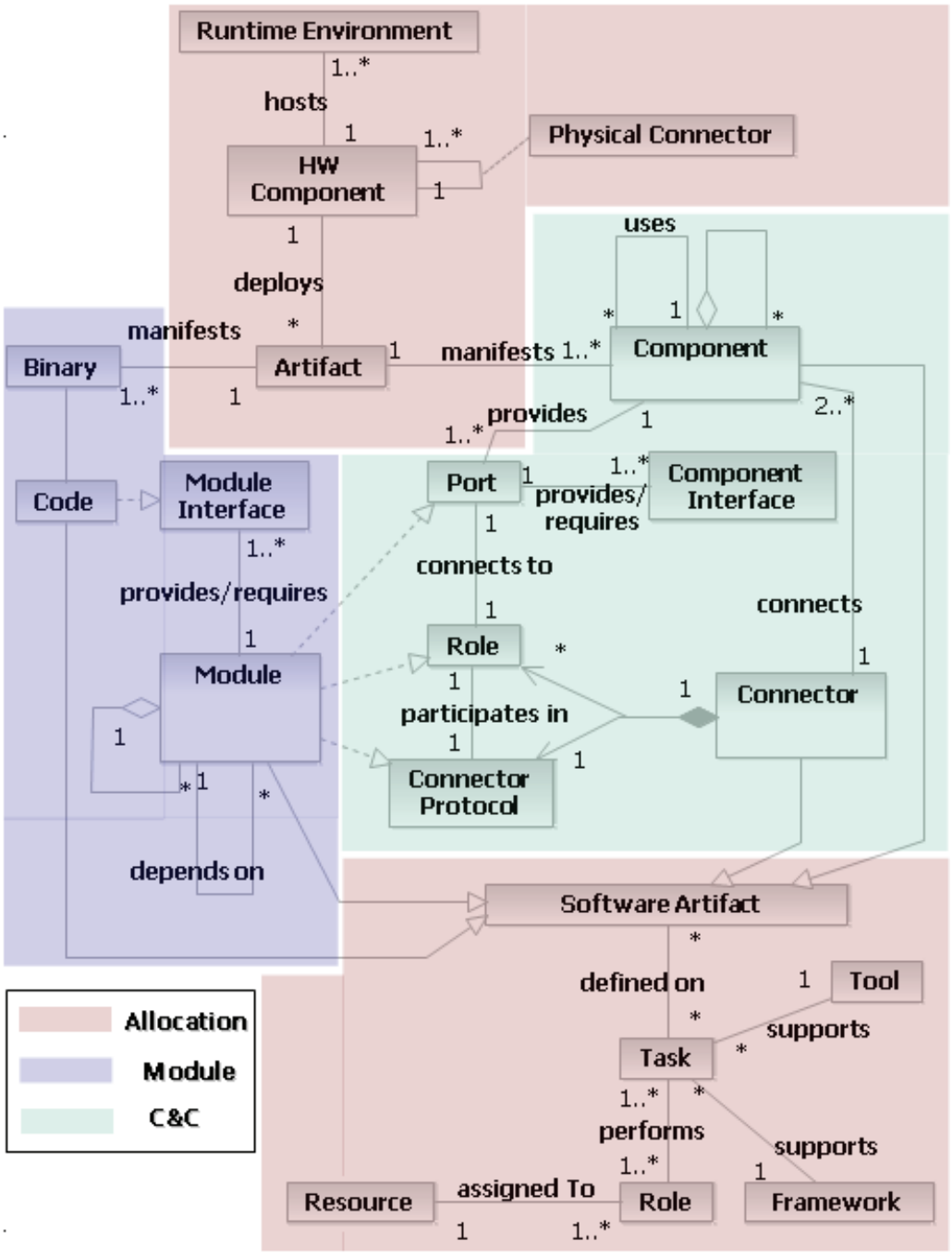



Figure 30 Architecture meta-model – Overview

## 3.3 Tactic Meta-Model

Having defined the architectural meta-model, the tactic meta-models need to be defined as well. As defined in Section 2.3, a tactic is a design option for the architect. However, design options need to be applied to the architectural design in order to be effective. This also implies that the application of tactics affects a number of architectural elements already existing in the design. In order to examine the potential impact that the different kinds of tactics have on architectural design, tactics in the context of the architectural meta-model are investigated.

### 3.3.1 Implementation Tactics

In terms of design dimensions, implementation tactics aim at improving the efficiency and effectiveness of activities that teams need to perform on the implementation units.

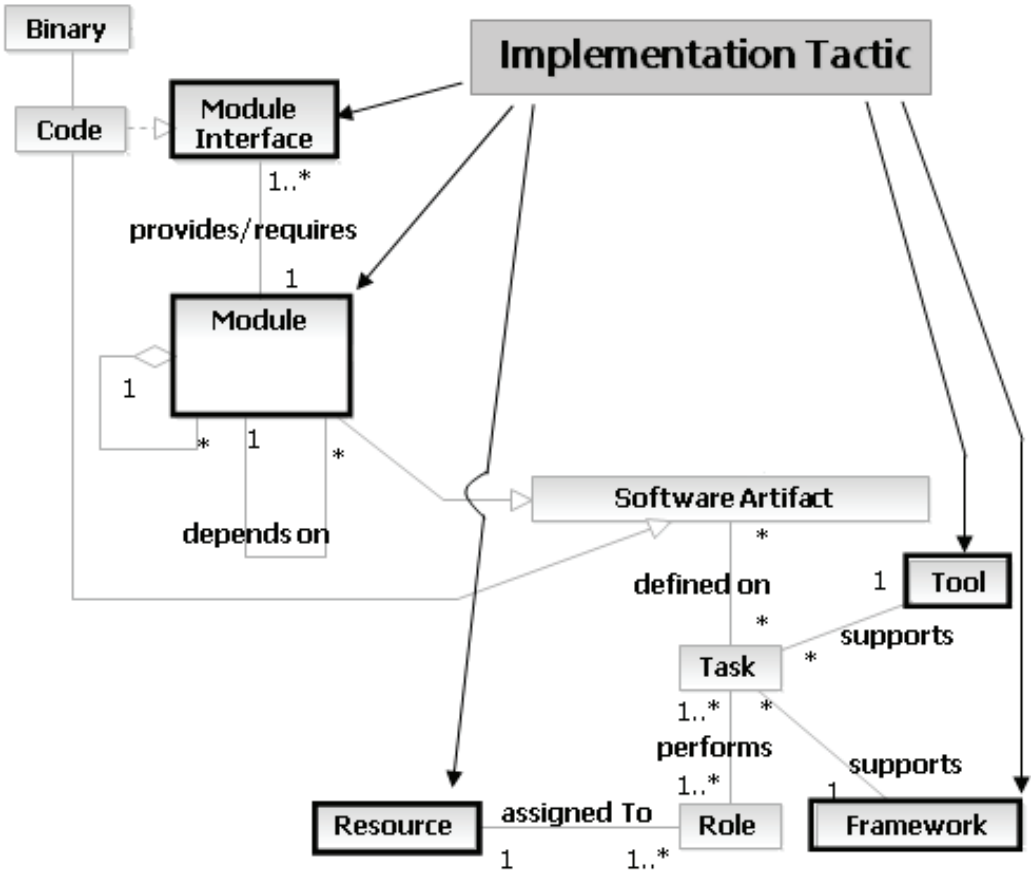



Figure 31 Implementation tactics affecting elements of the architecture meta-model

In terms of the architecture meta-model as defined in the previous section, implementation tactics either apply to modules themselves, mean-

ing that tactics might introduce new modules or change the structure of existing ones, or they may as well cause changes to interfaces provided by modules. Regarding the team allocations and infrastructure elements, implementation tactics potentially target resources by adding resources to a team or changing resources depending on the required skill set. In addition, implementation tactics may also refer to tools or frameworks supporting specific tasks throughout system development.

In general, implementation tactics are categorized into "Localization", "Prevention", and "Binding time" tactics [CK03]:

### Localization of change

Localization of change describes tactics that aim at minimizing the number of elements to be modified in case a change request needs to be implemented. According to [CK03], localization of change assumes that "restricting modifications to a small set of modules will generally reduce the cost". Modularization of concerns is considered a general approach that serves the goal of localizing change.

### Prevention of ripple effects

Tactics belonging to this class strive for the elimination of ripple effects in the context of a change to a system. According to [CK03], a ripple effect of a modification is defined as "the necessity of making changes to modules not directly affected by it [the initial change]". Implementation tactics might introduce new interfaces in order to absorb changes such that other modules are not affected in case of a change.

### Defer binding time

Deferring the binding time of decisions categorizes implementation tactics that aim at "allowing non-developers to make changes" [CK03]. That is, a product that can be extended in the field without the need to be redesigned, implemented, and tested, is very changeable. The drawback of deferring the binding time is that the system itself as well as the environment of the system needs to be prepared for those kinds of changes. According to [CK03], "Deferring binding time supports both of those scenarios at the cost of requiring additional infrastructure to support the late binding."

### 3.3.2 Execution Tactics

In general, execution tactics address runtime properties such as performance or availability. Thus, execution tactics potentially affect two design dimensions: the component and connectors and the deployment allocations (see Figure 32).

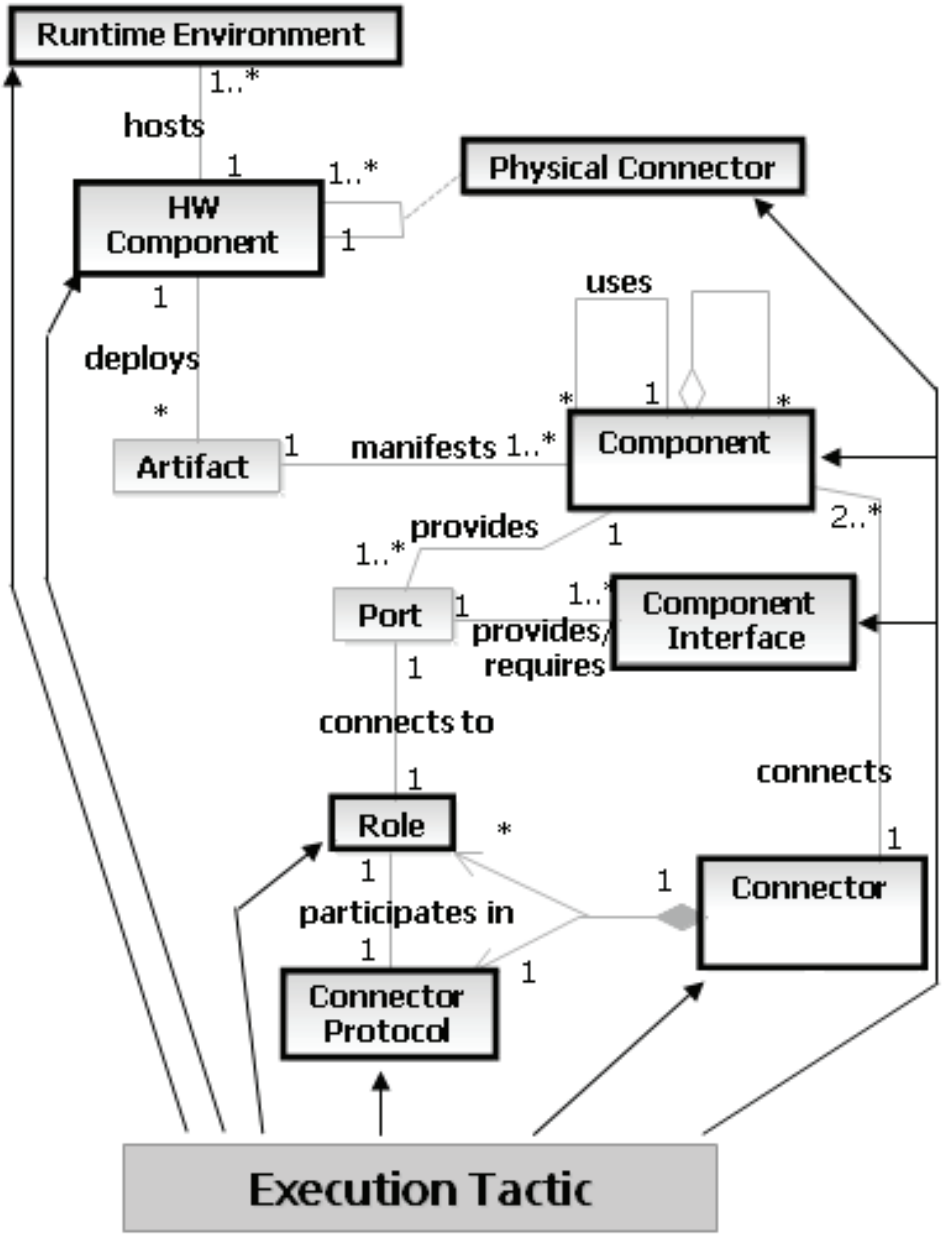



Figure 32 Execution tactics affecting the architecture meta-model

The component structure is usually impacted by introducing new components or by adding or changing responsibilities of existing components in terms of interface changes. The same holds for deployment entities like nodes or runtime environments. Most importantly, execution tactics most likely influence connectors. That is, an execution tactic potentially

introduces new roles and changes the connector protocol. In other words, the connectors are impacted by changes of coordinating information exchanges of the component structures.

Similar to implementation tactics, execution tactics are classified into different categories: detection, recovery, prevention.

### Detection tactics

Detection tactics deal with solutions that allow for detecting incidents during runtime that the system needs to identify in order to react to these incidents. An example tactic for incident detection is the "Heartbeat" tactic depicted in Figure 9 on page 29. Detection tactics usually impact components, component interfaces, connectors, hardware components, and physical connectors.

### Recovery tactics

Recovery tactics are strongly coupled with detection tactics, since they express the way the system is supposed to react to an identified incident. Usually recovery tactics aim at minimizing the negative consequences of an incident that has already occurred. For instance, if a component fails, a recovery tactic would be to switch to a replacement component that takes over the responsibility of the component that failed. Recovery tactics usually impact components, component interfaces, connectors, hardware components, and physical connectors.

### Prevention tactics

In contrast to detection and recovery tactics, prevention tactics aim at minimizing the likelihood that a particular incident will occur. Usually, the system can be analyzed with respect to certain probabilities that describe the likelihood of a particular incident. Depending on the goal of the prevention tactic, the probabilities can be reduced by applying systematic approaches like failure mode effects analyses or fault tree analyses. As a result, prevention tactic usually impact structures and properties of architectural elements such as redundant communication channels.

# 4 Aspect-oriented Architecture Model

*"I believe consistency and orthogonality are tools of design, not the primary goal in design."*
*Yukihiro Matsumoto*

In this chapter, I present specific extensions to the architectural meta-model in terms of how to separate cross-cutting concerns from one another. To that end, I apply ideas of aspect-oriented programming [KLM+97]. First, I give an overview of aspect orientation in general, then I look into aspect orientation at the modeling level. After that, the aspect-oriented extensions to the architectural meta-model are defined. That is, I map all relevant concepts of aspect orientation to elements at the architectural level as defined in Chapter 2.

## 4.1 Foundations

### 4.1.1 Aspect Orientation

The term aspect orientation was mentioned at the European Conference on Object-Oriented Programming (ECOOP) in 1997 for the first time [KLM+97]. At that time, the authors of [KLM+97] tried to solve programming issues like reuse and evolution of code by improving modularization and separation of concerns. For these reasons, the term aspect orientation was used interchangeably with aspect-oriented programming. The authors of [KLM+97] had realized that software programs were hard to modularize and hard to change because of the existence of so-called cross-cutting concerns. Cross-cutting concerns often cannot be cleanly separated from the rest of the system in both design and implementation, and can result in either scattering (code duplication), tangling (significant dependencies between systems), or both [BC05].

The solution idea of aspect orientation is to factor out the cross-cutting concerns into a single, well-defined module (the so-called aspect) that can be automatically composed with the rest of the program (the so-called "core" or "base"). Every aspect-oriented technique is defined by a so-called join point model (JPM) [Caz06]. According to [MKD02], a JPM

is characterized by three elements: join points, join point selection mechanism, and a means for specifying semantics at join points.

1. **Join points**: A join point is a point in a program where behavior can be altered, augmented, or removed. In other words, the join points are considered the interface type between aspects and base elements. Examples of join points at the programming level are method calls, field accesses, or constructor calls.

2. **Join Point Selection Mechanism**: On the programming level, join point selection is done by using so-called pointcuts. A pointcut can be defined as a query on the program itself. The most simple and trivial way to capture join points is to enumerate them using either their names or ids or whatever property can be utilized for their exact identification. Another technique that can be seen as an extension to the enumeration of join points is to capture desired join points by providing a pattern (mostly a name or signature pattern). The most common means used at the programming level is the wildcard symbol. A concrete example in AspectJ would be:

```
pointcut threadCreation(Runnable runnable)
        : call(Thread.new(Runnable)) && args(runnable);

// This pointcut captures all calls to the
// constructor of class Thread using an argument of type Runnable.
```

Listing 1 Sample pointcut in AspectJ

3. **Adaptation**: The third important element in the JPM is the adaptation, a means of specifying semantics at join points. Once join points have been selected, the programmer usually alters the code at those points either by adding or by changing existing functionality. In AO terminology, the adaptation is called "advice". In general, there are three ways of how the advice can be introduced: **before**, **after**, and **around**.

```
Thread around(Runnable runnable) : threadCrea-
tion(runnable){
    Thread myThread = pool.get();
    if (myThread == null) {
       myThread = new ThreadPool.DelegatorThread();
        }
        myThread.setDelegatee(runnable);
        return myThread;
    }}
// In this advice, the original calls to the Thread-constructor as collected
// by the pointcut threadCreation will be replaced by the code fragment
// as listed within the advice declaration.
```

Listing 2 Sample advice declaration in AspectJ

"Before" denotes that the adaptation is supposed to be introduced right before the control flow reaches the respective join point, for instance, before a method is about to be called. The designator "after" is defined accordingly, intercepting the control flow right after the join point has been reached. "Around" implies a special behavior, since the behavior exposed at the selected join point can be completely exchanged or even removed.

4. **Weaving**: The transformation process that puts base elements and aspects together is called "weaving". If the aspect is woven as defined into a sample class *MyThreadClass* as defined in Listing 3, we get a modified behavior according to Listing 4.

```
class MyThreadClass extends Thread {

    public MyThreadClass(Runnable runnable){
             super(runnable);
    }

    Thread getNewThread(Runnable runnable){

         return new MyThreadClass(runnable);
    }
}
```

Listing 3 Sample class describing the weaving target

```
class MyThreadClass extends Thread {

    public MyThreadClass(Runnable runnable){
             super(runnable);
    }

    Thread getNewThread(Runnable runnable){

      Thread myThread = pool.get();
      if (myThread == null) {
             myThread = new ThreadPool.DelegatorThread();
          }
          myThread.setDelegatee(runnable);
          return myThread;
    }
}
```

Listing 4 Modified behavior of class "MyThreadClass" after weaving

As a result, all constructor calls will return an assigned thread from the thread pool. The type *Thread* of the around advice denotes the return type of the advice execution.

### 4.1.2 Aspect-oriented Modeling

The ideas of aspect-oriented programming found adoption in model-based software development as well. The discipline dealing with concepts of aspect orientation on the modeling level is called Aspect-oriented Modeling (AOM). According to [SSK06] and [CRS+05], most existing aspect-oriented modeling approaches base their aspect oriented modeling proposal on the UML [SHU02a], [SHU02b], [SHU02c], [SHU06], [AEB03], [BGL04], [FS06], [RTT04], [ZHZ02]. Furthermore, the majority of the evaluated approaches in [SSK06] base their work on UML versions prior to the UML 2.0 specification. For customizing the UML towards aspect-oriented concepts, UML's inherent profiling mechanism is used.

In addition, most approaches as compared in [SSK06] address structural diagrams only. There are only few approaches that make use of behavioral diagrams in order to demonstrate behavioral features of aspects or to specify when cross-cutting behavior should occur relative to the base behavior. Since our goal is to support the efficient design of architectural connectors, we will eventually need to operate on behavioral diagrams.

In the following, I define all relevant terms in the realm of AOM, which are then mapped to the architectural meta-model as described in Chapter 3.

**Concern**: According to [MEH01], a concern is "a requirement, an intention, an objective, or an aspiration a stakeholder has regarding a particular product". Concerns are either cross-cutting or non-cross-cutting. That is, a concern is cross-cutting if the concern is scattered across a modular structure. The principle separation of concerns aims at modularization of concerns such that each concern can be treated separately in order to reduce complexity. In that context, concern composition also deals with re-composing a system by assembling the separated concerns effectively and efficiently.

**Base**: A base is a unit of modularization formalizing a non-cross-cutting concern [SSK06]. Usually, the base is the result of having selected a particular dominant decomposition dimension (e.g., functional decomposition). In such cases, the functions are usually well modularized and not cross-cutting. Ultimately, they are most likely to be influenced by cross-cutting concerns though.

**Aspect:** An aspect is a unit of modularization formalizing a cross-cutting concern [BC05]. In other words, the aspect describes a concern, which would be scattered across the system otherwise, in a modular way.

**Weaving**: In aspect-oriented software development, the composition of aspects with other concerns, which in turn are either bases or aspects, is called weaving [SSK06]. The weaving process is the concern composition

process, combining the cross-cutting concerns with the non-cross-cutting concerns in order to form an integrated model.

**Adaptation:** An adaptation specifies in what way an aspect adapts the concern's structure or behavior. Adaptation denotes enhancement, replacement, or deletion. In other words, an aspect can extend the behavior of a base concern; it might replace the base concern with the aspect behavior; or it might just delete behavior as defined by the base.

**Adaptation Subject:** In general, an adaptation subject describes where to introduce an aspect's adaptations [SSK06]. In terms of aspect-oriented programming, the adaptation subject is the join point. At the modeling level, two kinds of join points are defined: structural join points and behavioral join points. Structural join points are defined on model elements that represent system structure (e.g., module structures), whereas behavioral join points are defined on model element representing behavioral elements of a modeling language (e.g., activities).

**Adaptation Rules**: Adaptation rules are part of a particular kind of weaving and introduce an aspect's adaptations at certain points of other concerns. In a sense, an adaptation rule combines a particular (set of) adaptation(s) with a set of adaptation subjects. In addition, adaptation rules specify how the adaptation affects the base, also referred to as **adaptation effect**. Since aspect-oriented software development has clearly been driven by the emergence of aspect-oriented programming languages such as AspectJ, most AOM approaches rely on the asymmetric paradigm [SSK06]. That is, the adaptation effects transferred from the programming level are: before, after, and around. According to [SSK06], only few approaches do consider the effect of aspect adaptations at all.

In the following conceptual model, I relate all relevant terms as defined above. It is important to note that the adaptation subject is defined by a pointcut expression that specifies a join point selection. In the subsequent sections, I map the notion of a join point to architectural elements.

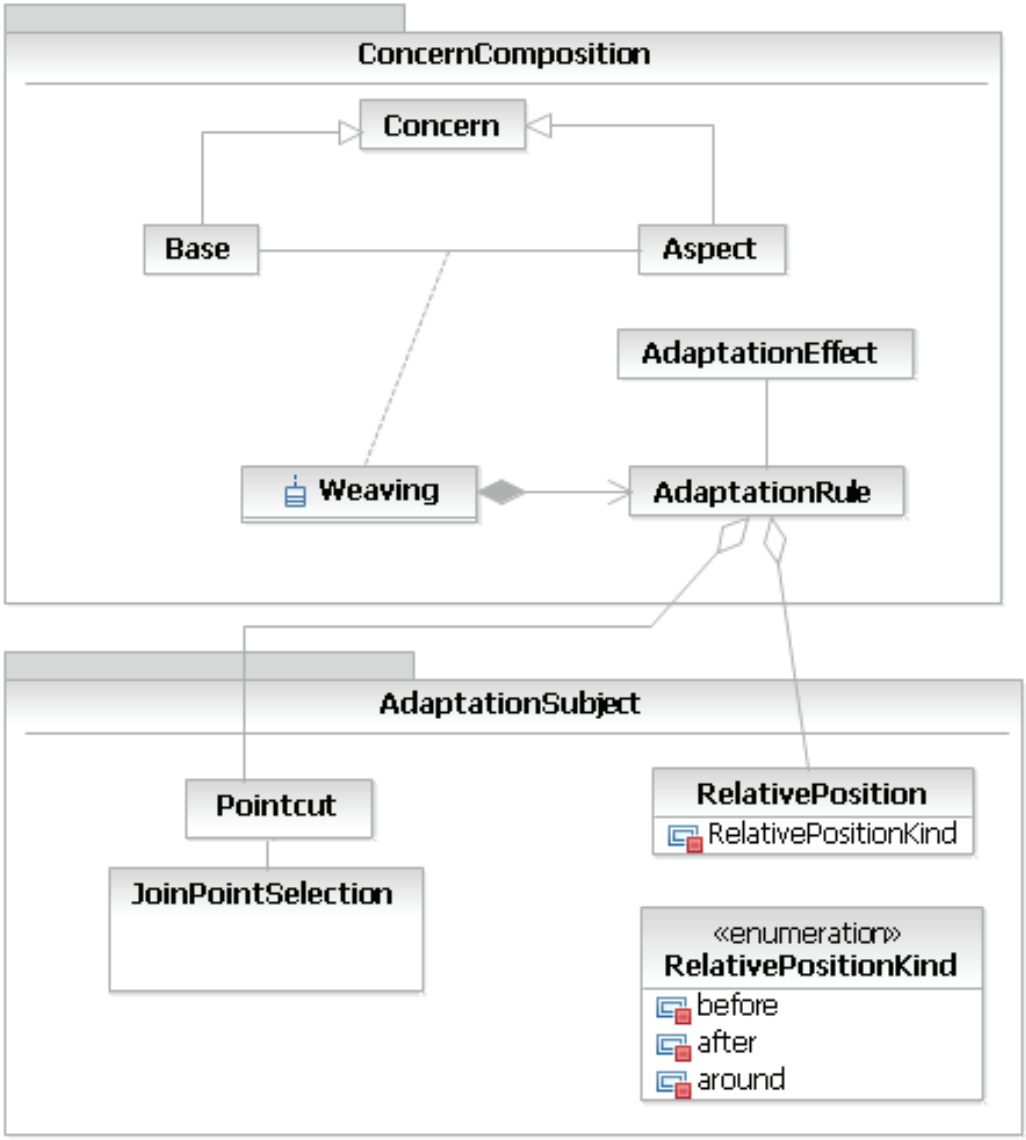



Figure 33 Conceptual model for aspect-oriented modeling (adapted from [SSK06])

## 4.2 Related Work

### 4.2.1 Architectural Aspects

In the realm of architectures, there exist several works that aim at leveraging aspect notions in order to systematically capture cross-cutting architectural concerns [KTG+06], [NPM+02], [CPF05], [Per06], [NPM09]. Most of the existing approaches aim at enhancing the description languages (ADL) [Cle96] in order to express cross-cutting concerns. That is, the approaches focus on the extension of connector specifications, like *AspectualACME* [GCB06], *DAOP-ADL* [PFT03], *AO-ADL* [PF07], or *PRIS-*

*MA* [PAC+06] so the connectors can be defined within the context of so-called aspectual components. Aspectual components are in charge of capturing otherwise cross-cutting concerns.

However, in terms of connecting the aspects to the base models, most existing approaches do not mention in their proposal the need for supporting important AO properties such as quantification (e.g., pointcuts), interferences between aspects, and heterogeneous aspects [GCB06]. In case of AspectualACME, the authors provide a quantification mechanism; however, it is solely based on syntactical expressions and comes with all the drawbacks of pointcut fragility. Another issue is that there is no way of specifying precedence of aspects in case of collisions. In general, I agree with the findings of the authors of [GCB06] that there is little consensus on how Aspect-oriented Sofware Development (AOSD) and ADLs should be integrated, especially with respect to the interplay of aspects and architectural connection abstractions [BCG+06].

The major difference between existing work in the area of aspectual architecture description languages and the work presented in this thesis can be found in the goal that we strive for. We use aspect-oriented concepts to make it possible to efficiently modify and extend connector protocols throughout the genesis of an architecture. The focus of the work presented in this thesis is clearly on the integration of cross-cutting concerns with a base architectural model. As stated in Section 2.4.2, cross-cutting concerns exhibit three challenges: identification, separation, and integration.

In case of aspect-oriented ADLs, none of the solutions provides any support for integrating the concerns, that is, for generating a compiled representation of the aspects and base architectural concerns.

### 4.2.2 Connector Composition

The work presented by [SK04] is similar in terms of the general idea: enhancing basic connectors with selected adaptations to produce a more complex connector. The authors apply a set of transformation operators to enhance the basic connectors. However, the goal as well as the solution of the approach differ in several ways:

Whereas the main goal of the work of [SK04] is to generate implementations of connector compositions, the work presented in this thesis aims at generating models of connectors. Another significant difference is that the transformation approach presented in [SK04] works on the connector-type level only. That is, connector instances are not subject to change by other kinds of transformations. The work presented in this thesis, however, leverages a type-based specification mechanism in order to navigate to the adaptation subjects. Another fundamental issue of the approach presented in [SK04] is the incremental construction of con-

nectors. There is no solution to the incremental problems as stated in Chapter 1, and most importantly, there is no explicit handling of cross-cutting concerns. Finally, the approach of [SK04] does not provide any means for checking compositional properties.

The work presented in [LWF03] is based on category theory [LWF01]. Their approach for composing connectors works on the type level only, similar to the approach proposed by [SK04], and comes with the drawbacks of evolutionary introduced cross-cutting concerns. In addition, the formalisms that the composition is based on are less intuitive to architects in practice.

The approach presented in [RGG01] aims at designing communication protocols in distributed systems using the Specification and Description Language (SDL). The underlying idea is to modularize the design of complex protocols by using collaboration roles that can be composed into more complex protocols. With the exception of the shortcoming of working on the type level only, and thus lacking solutions to incremental issues, the approach of decomposing protocols into smaller units looks promising in terms of being complimentary to the approach presented in this thesis.

## 4.3 Architectural Join Point Model

As stated by the research questions in section 1.3, I aim at leveraging aspect-oriented concepts at the architectural level. Therefore, appropriate architectural elements are identified that aspect-oriented concepts can be mapped to.

### 4.3.1 Architectural Base

As explained in section 1.1, one reason for the inefficiency of model creation is that architectural solutions for quality concerns potentially cut across a variety of architectural design artifacts. This is particularly true for quality concerns that affect component interactions [FBL02]. Referring to the architectural meta-model as defined in Section 3.1.1, architectural connectors specify interaction protocols between components. Connectors are crucial in terms of cross-cutting concerns, since connectors are responsible for all kinds of communication and coordination among components. Connectors can be described using formal specification techniques [All97] as illustrated in Figure 34-b, or graphically using a modeling language like the UML, as shown in Figure 34-a. I stress the fact that the sequence-based specification can be enhanced by using techniques for transforming a set of sequences into a single-state machine as presented in [WS00].

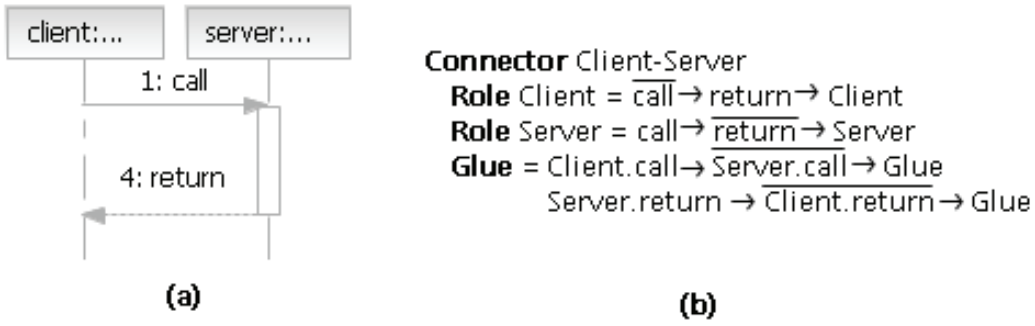



Figure 34 Sample connector definition: Client-Server

For the remainder of this thesis, the architectural connectors are referred to as the base models that modularize a (communication) concern.

### 4.3.2 Architectural Aspects

I map the concept of an aspect to architectural tactics as defined by the architectural meta-model shown in Section 3.3.2. Since execution tactics enhance an existing connector description in order to support a certain quality attribute, execution tactics are treated as architectural aspects. More specifically, it is the interaction specifications of the execution tactics influencing the connectors that are considered architectural aspects. They are eventually composed with the base communication model expressed by the connector design as exemplified above. More formally, an execution tactic $A$ is used to augment an existing connector description by a number of tactics $T_A \subseteq T$ and pointcuts $P_A \subseteq P$.

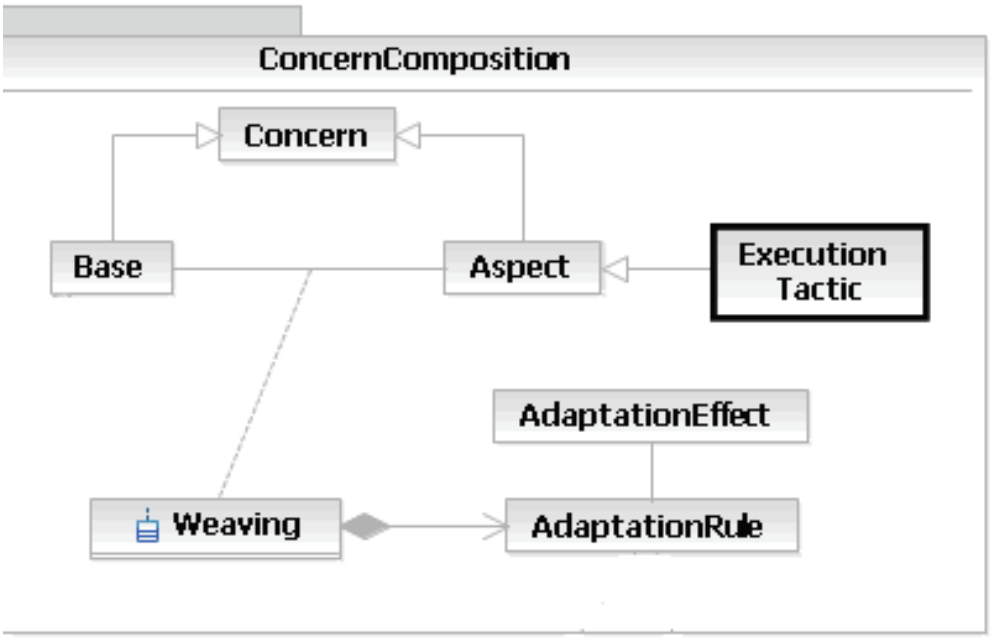



Figure 35 Mapping of aspects to execution tactics

The composition of execution tactics with the base models is referred to as "weaving" (see Figure 35). The weaving specifies the way a set of execution tactics should be composed with a base model.

### 4.3.3 Architectural Adaptation Subjects

In light of such connector models, architectural design decisions concerning component interaction and coordination potentially require many model elements to be identified, created, altered, and validated. In the context of connector design, an adaptation subject (or join point) would be a place in the base communication model that can be altered, augmented, or removed by an aspect (see Figure 36). Hence, the application of an architectural pattern (as defined in section 2.3) can be described by a number of tactics $T_A \subseteq T$ and respective pointcuts $P_A \subseteq P$.

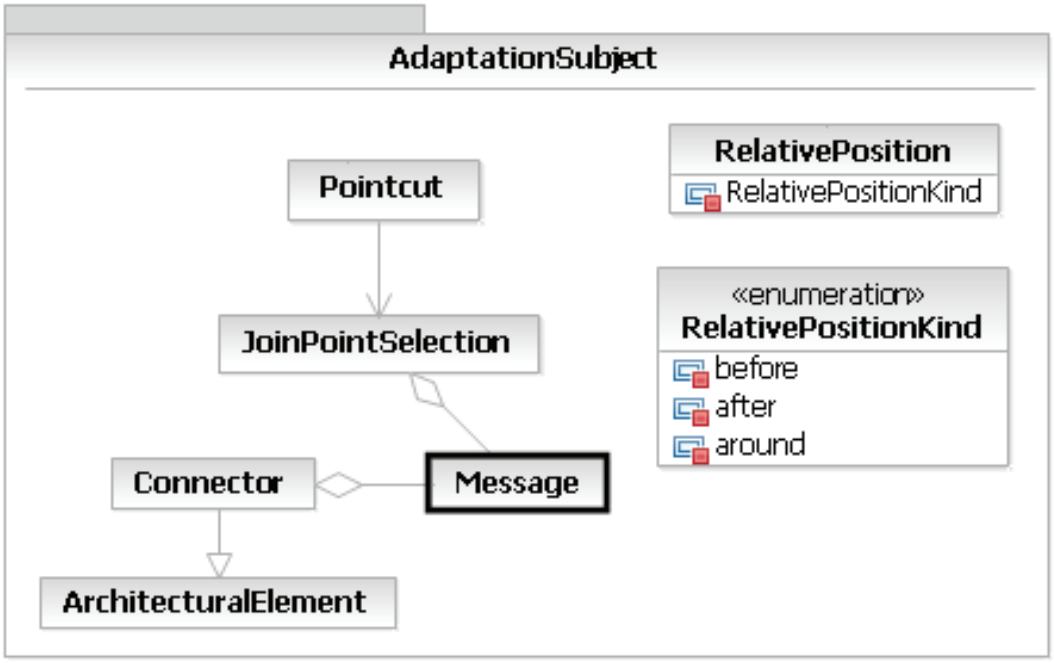



Figure 36 Messages as adaptation subjects

Based on the fact that connectors describe component interactions by showing message passing [TMD09], I mapped the notion of connector protocols to a message-centric paradigm [KMU08]. As a result, a component-connector design comprises a special data structure of messages that encode all information required in a connector protocol. In that sense, messages are abstract constructs representing communication events, enriched with information about involved roles and their collaboration protocol.

A connector design can be defined as:

1. A finite set C of components

2. A sequence S that denotes the allowed order of message exchanges among components

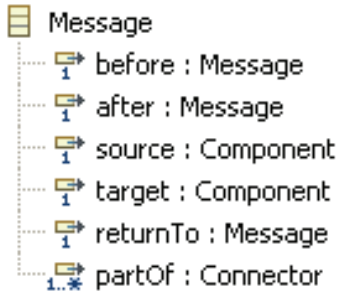



Figure 37 Data structure for messages in connector protocols

Each message encodes its own source and target components. Besides, the position of the message in the protocol sequence is encoded using before and after attributes.

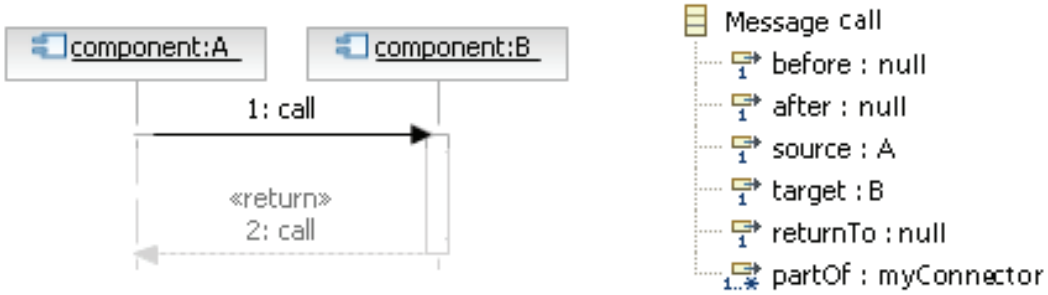



Figure 38 Sample mapping to message data structure

Let message *n* be message m's follow-up message. Then, the before-attribute of message *m* in the connector protocol sequence refers to message *n*. The after-attribute of message *n* references message *m* in turn. In the example, we see that the message *call* encodes the sequence of the diagram as shown in Figure 38. Since there is no preceding and succeeding message in the sequence, the references are null. However, the source and target attributes are set to *component A* and *component B,* respectively. By looking at the complete set of messages, we know the overall sequence of the collaboration protocol by just looking at the respective message attributes. The *partOf* property is used for tracking messages to their respective connector. In case a message is the return message for a particular message, that message can be referenced using the *returnTo* attribute. For modeling connector protocols, I use sequence diagrams as defined by the UML 2.1. For complex protocols, it is thus possible to model the sequences hierarchically.

Looking at Figure 36, a pointcut selects messages and thus determines where execution tactics (as defined in Section 3.3.2) apply. Hence, the pointcut retrieves a set of messages that are considered during weaving.

Thus, an execution tactic $A$ is a set of pairs over $2^{(T,P)}$ where

- $T \subseteq Msg_T$ is a finite set of messages including a special *joinPoint* message
- $P \subseteq Msg_B$ is a selection of messages from the base model

During weaving, the *joinPoint* message is used to define where the target model and the tactics meet. Note that $M^A \subseteq Msg_B \cup Msg_T$, i.e., the target model, is enriched by new messages and collaborations. A weaving is specified using adaptation rules as described in Section 4.1.2.

### 4.3.4 Pointcut Definition Language

As defined in Section 4.1.2, an adaptation rule specifies where and how to introduce an aspect's adaptations at certain points of other concerns. Where to introduce the adaptation is determined by a selection of adaptation subjects, or join points, that can be specified by the respective pointcuts. In fact, what makes aspect orientation powerful in terms of efficiency is its ability to specify a multitude of different places in a system with a relatively simple expression [FF00]. However, in order to benefit from such an approach, a number of challenges need to be addressed. First, it should be easy to create a valid pointcut expression using formalized language. Second, the pointcut expressions need to support easy maintenance over time. In the following, I define a pointcut definition language that strives to mitigate the challenge of both creating and maintaining of pointcut expressions.

#### 4.3.4.1 Type-based Pointcuts

Today, composition modeling techniques rely mostly on syntax-based pointcuts, meaning that they rely on specific naming conventions or structural properties of elements. These kinds of pointcuts have become widespread in aspect-oriented programming. The main identified drawback of syntax-based techniques is the so-called pointcut fragility [Caz06]. Pointcut fragility refers to the fact that in case of changes to a base model, the respective pointcut would have to be changed as well.

In order to address the issue of pointcut fragility, the pointcuts need to operate on a higher level of abstraction. To that end, I define the pointcuts by using the connector protocols at the type level of messages. By doing so, I decouple the pointcut definitions from concrete messages as specified in the connector models and mitigate the problem of

pointcut fragility, which causes inconsistency between the aspects and the core models.

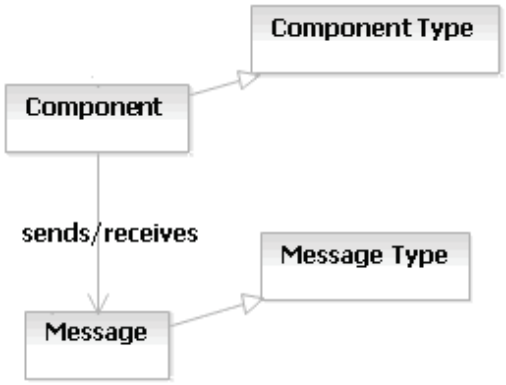



Figure 39 Creating pointcut expressions based on component and message types

By using such a type system, we are able to select a number of concrete messages just by referring to their types. Looking at Figure 40, we can select the messages *getOtherData* and *update* by simply denoting that we would like to capture all messages of the type <<call>>.

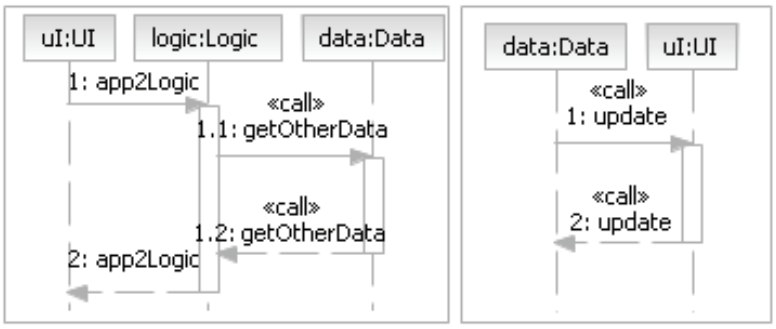



Figure 40 Sample sequences for illustrating the effects of type-based selections

The pointcut definition language (PDL) as defined in this section implements the type-based pointcut design as well as the navigation capabilities provided by the integrated architecture meta-model as presented in Chapter 3 [KRK09]. The syntax of the PDL is defined analogously to the Atlas Transformation Language (ATL) [ATL10]. Moreover, it supports reflective access to join points, meaning that messages are referenced by name (instance-level) or by stereotypes (type-level pointcuts). For referring to elements within the architectural model, I will use the following notations:

```
metaelement!instance(name)

and

metaelement!type(name)
```

In these statements, the meta-element refers to the elements as defined by the UML. The elements of relevance in this context are *UML::Message*, *UML::Node*, *UML::Component*, *UML::Artifact*, and *UML::CommunicationPath*. The respective instance or type following the "!"-symbol refers to a particular element stereotype or element instance. For specifying pointcuts, however, we use abbreviations that map to the UML element definition as follows:

| Abbreviation | UML Element |
|---|---|
| **jp** | UML::Message |
| **node** | UML::Node |
| **comp** | UML::Component |
| **art** | UML::Artifact |
| **cp** | UML::CommunicationPath |

Table 1 Mapping of PDL to UML

Pointcut expressions can be composed using these operators: "**&**","**|**", "**!**", and "**( )**". For instance, by denoting pc_A & pc_B, those join points are retrieved that are captured by both pointcuts. In other words, the intersection of both sets is retrieved. By denoting pc_A | pc_B, the set union is returned, that is, all join points, even those that are captured by either one of the pointcuts. By denoting !pc_A, all join points captured by that particular pointcut are excluded. By using the brackets (pc_A), sets of join points can be grouped logically.

#### 4.3.4.2 Deployment-based Pointcuts

Even though type-based pointcut expressions are a powerful means for selecting huge numbers of message instances, experience has shown that the message types alone might not be sufficient for being efficient in practice. For this reason, I aim at utilizing architectural information during the process of pointcut specification in addition to the message types. In order to utilize architectural facets of messages for defining pointcuts, I exploit the fact that connectors belong to the component-

connector dimensions and therefore have a relationship to the deployment view of architectures. In the following, I show how to take advantage of these relationships to specify pointcuts. Looking at the definition of component type and connector type, I can exploit the fact that connectors, and thus messages, are operating between components. That is, by looking at a particular component we are able to trace the messages that the component sends and receives. In order to exploit the deployment relationships, we take advantage of the element types as defined by the meta-model in section 3.1.3.1. Since components are manifested by artifacts that are eventually deployed to particular hardware components, the deployment offers two additional elements that can be used for tracing the messages exchanged between components: artifacts and nodes.

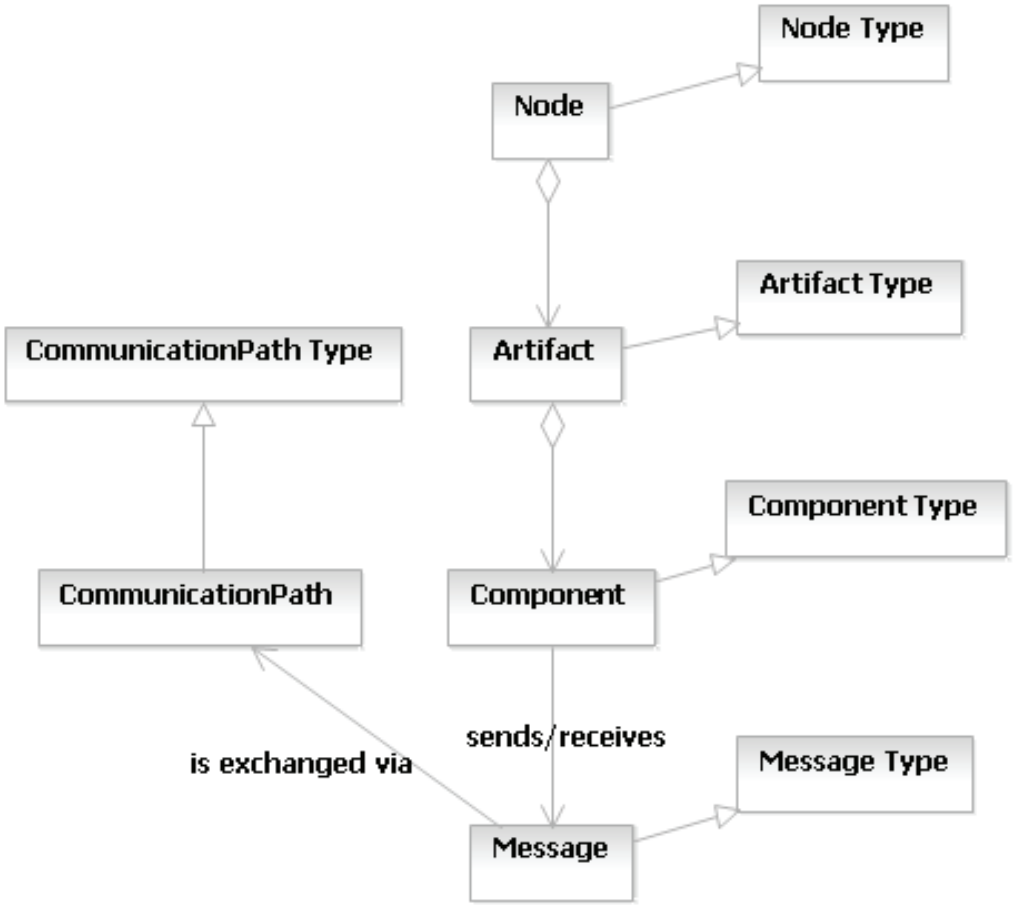



Figure 41 Navigating to messages using the deployment

The basis for navigation are the elements hardware component, artifact, and component. We can take advantage of these facts since components are defined as being interrelated by connectors, the connectors themselves are defined using message sequences. By referring to components, we have another kind of abstraction mechanism for mechanisms, namely, all messages that are sent or received by a particular component. That principle applies to deployment entities as well. Semantically, we can specify a collection of messages that are exchanged

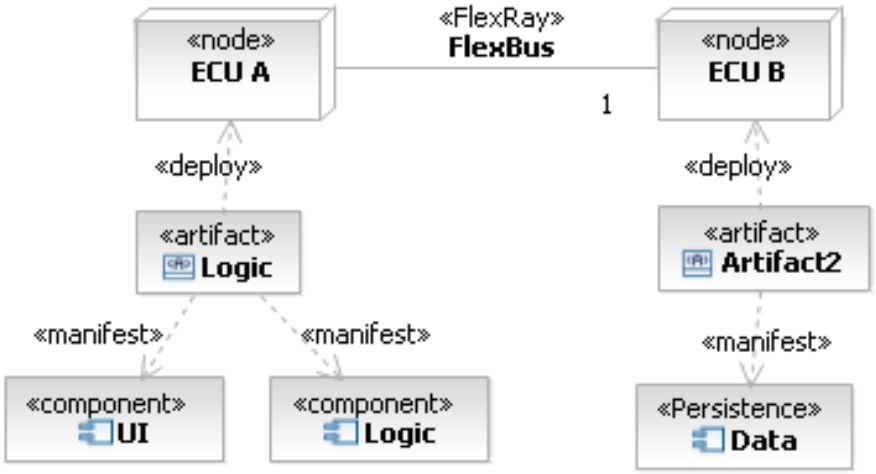



Figure 42 Sample deployment illustrating the effects of the join point selection operators

between two hardware components, or messages that are sent or received by a particular hardware component. That is, we use the designators *source*, *target*, and *all* for specifying a message selection given one of these elements.

The semantics of the designators are defined in the following:

• **source** (element): returns all messages originating from the specified element (instance or type). Valid assignments of elements need to be either node, artifact, or component.

```
source(node!instance(ECU B))
```

Given the example as depicted in Figure 14 and Figure 42, this pointcut specification returns the message *update.*

• **target** (element): returns all messages received by the specified element (instance or type). Valid assignments of elements need to be either node, artifact, or component.

```
target(component!type(ArchProfile::Persistence))
```

Given the example as depicted in Figure 14 and Figure 42, this pointcut specification returns the messages *getOtherData.*

• **all** (element): returns all messages that are passed through the specified element (instance or type). The only valid assignment of the element is communication path.

```
all(node!type(ArchitectureProfile::FlexRay))
```

Given the example as depicted in Figure 14 and Figure 42, this pointcut specification returns the messages *update* and *getOtherData.*In addition, attributes of elements can be accessed through the "dot-notation" as follows:

```
metaelement!type(string).attribute
```

For example, the expression

```
cp!type(ArchitectureProfile::FlexRay).bandwidth
```

accesses the attribute named *bandwidth* of the *FlexRay* stereotype as defined by the UML profile *ArchitectureProfile* specializing the UML metaelement *CommuncationPath*. With access to element attributes like this, conditional statements can be constructed that utilize the element properties for assembling a particular pointcut. Here I apply the comparison operators <, >, =, >=, <= in order to evaluate numeric expressions.

For instance, given the following pointcut:

```
jp!type(ArchitectureProfile::secureMessage).size>10
```

This pointcut expression selects all the messages of the type <<secureMessage>> that have an attribute *size* whose value is greater than 10.

The pointcut expression

```
source(comp!type(ArchitectureProfile::Peer).id=E12)
```

selects all messages that are sent by all components of the type <<Peer>> having the attribute value of *id* set to E12.

As described in [KRK09], the PDL offers the possibility to provide upper- and lower-range values of the same attribute or to utilize different attributes of the same stereotype by combining simple pointcuts. For example, given the pointcut

```
jp!type(ArchitectureProfile::IService).size >10 &
 jp!type(ArchitectureProfile::IService).size <100
```

selects all messages having an attribute *size* set between 10 and 100.

Besides the idea of type-based pointcut specifications, the selection of join points may be performed across architectural views. Using elements from different views in the pointcut language provides additional possibilities for capturing join points on different semantic levels. First of all, combining elements from component-connector and deployment mod-

els makes it possible to capture the cross-cutting structure of some concerns in a more natural way. For instance, in a Client-Server system, pointcuts can be described by *select all messages where a Server is communicating with a Client* or *select all messages that trigger a remote operation*. Using the architectural meta-model, it is possible to create formal expressions that exactly realize the natural query for messages, since the information about the distribution of the system is manifested in the deployment views of the architecture. (Note that a message is sent through a communication path only if sender and receiver are deployed on different nodes that are connected.) The central element in the approach for defining pointcuts across architectural views is the component element. Components enable interconnections between component-and-connector and deployment models because of the relationship <<manifests>> in the architecture meta-model. Using the component element as a navigation link, it is possible to navigate to all messages starting from the deployment model. As a result, the pointcut language spans the following design space in terms of types and architectural views:

| | **Message-based** | **Deployment-based** | **Cross-View** |
|---|---|---|---|
| **Type-based** | jp!type(AP::call) | all(cp!type(AP::CAN)) | all(cp!type(AP::CAN)) \| jp!type(AP::call) |
| **Instance-based** | jp!instance(secure*) | source(node!instance(ECU A)) | source(node!instance(ECU A)) & jp!instance(secure*) |
| **Mixed** | jp!type(AP::call) \| jp!instance(serviceA) | all(cp!type(AP::CAN)) & source(node!instance(ECU A)) | all(cp!type(AP::CAN)) & jp!instance(secure*) |

Table 2 Poincut design space

### 4.3.5 Weaving of Architectural Models

For the actual weaving, the adaptation effects as described in Section 4.1.2 are used, namely *before*, *after*, and *around*. The before-designator represents the point in the sequence right before the message occurrence. The after-designator captures the point right after the message execution, i.e., after the reply has been sent (synchronous messages) or the end of the execution flow is detected (asynchronous messages).

Weaving always implies that connector protocols are altered by means of adding or replacing message sequences in a connector protocol instance. To exemplify the mechanisms, it is shown how the interaction sequences are modified by each of the adaptation effects. In fact, the weaving algorithm works on the message data structure as defined in Section 4.3.3. A sample base with the respective join point (message m = "call") is depicted in Figure 43.

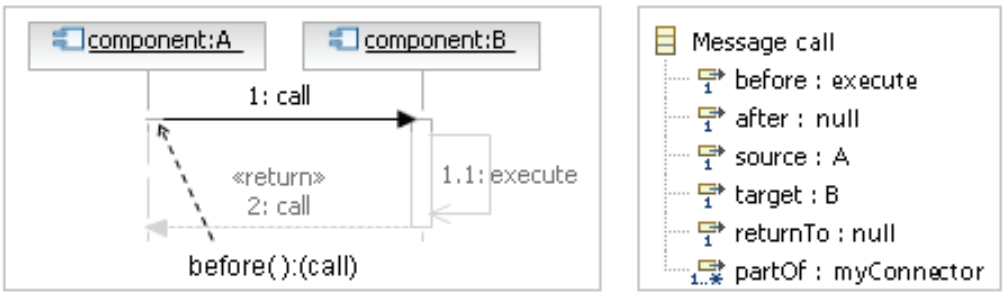



Figure 43 Join point semantics of before() directive

In the following, the weaving algorithm is described for each of the adaptation effects in detail.

Adaptation Effect: **Before**

For describing the weaving algorithms, I use an object-oriented notation. In case of a before-directive (see Listing 5), the before-reference of the last message in the tactic T is changed to the message m (lines 1-2). Note that, since sequence is represented as an array, the size()-operation returns the number of elements in the respective array. Then, the before-reference of message m's after-reference (m.after.before) is changed to the first message in the sequence of the tactic (line 3). The after-reference of the message m, the source- and the target references are changed, respectively (lines 4-6).

```
before():(Message m){

1 T.sequence [sequence.size()-1].before = m;

2 T.sequence [0].after = m.after;

3 m.after.before = T.sequence [0];

4 m.after = T.sequence [sequence.size()-1];

5 T.sequence [0].source = m.source;

6 T.sequence[sequence.size()-1].target = m.source;}
```

Listing 5 Weaving algorithm for before-semantics

Figure 44 shows a sample tactic for weaving. Note that the tactic has a representation in terms of the message data structure as well.

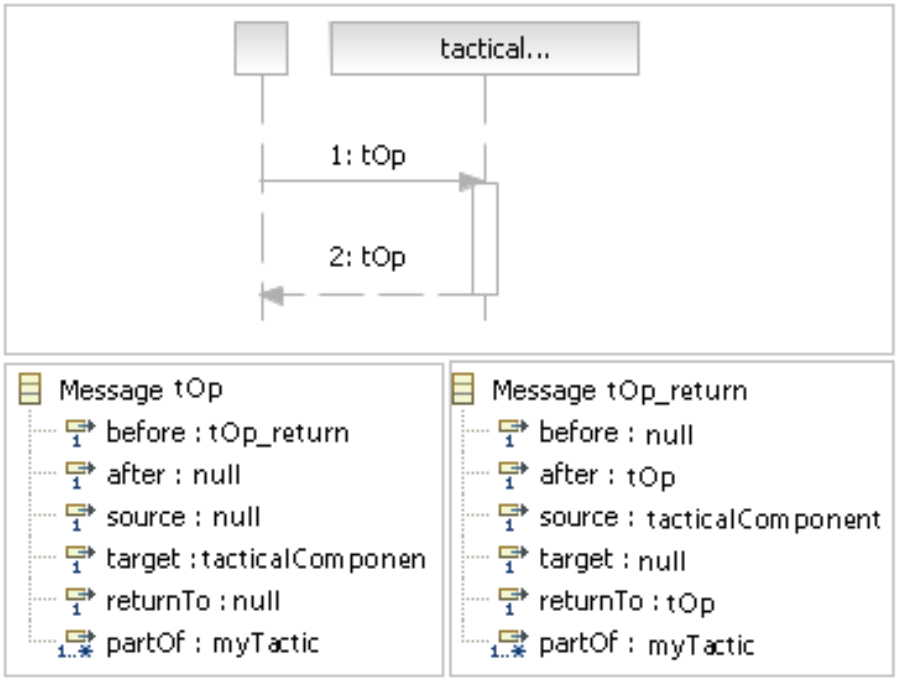



Figure 44 Sample tactic - graphical and message notation

For illustration purposes, a sample weaving can be defined as follows:

```
before(): jp!instance(myConnector.call) -> myTactic
```

The semantics of this weaving specification is interpreted as follows: "Enhance the interaction specification of the connector *myConnector* with the interactions as specified by *myTactic* right **before** all places where a message with the name *call* occurs."

The graphical result of the weaving is shown in Figure 45. Figure 46 shows the resulting messages in terms of field changes indicated by blue highlight. As can be seen, the message structure of the join point itself ("call"), as well as the message structure of the tactics ("tOp") have been changed according to the weaving algorithm as shown in Listing 5.

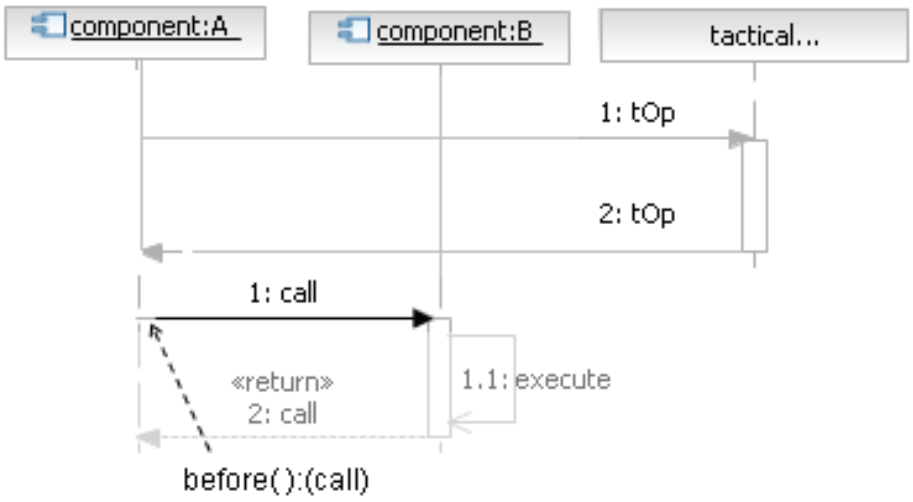



Figure 45 Graphical weaving result of the before()-directive

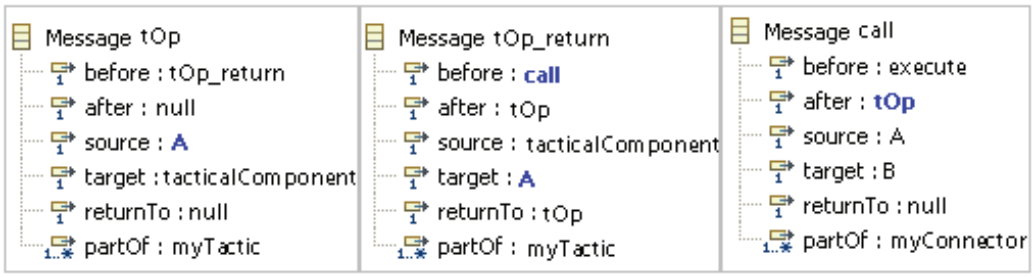



Figure 46 Message structure of the weaving result - before

## Adaptation Effect: **After**

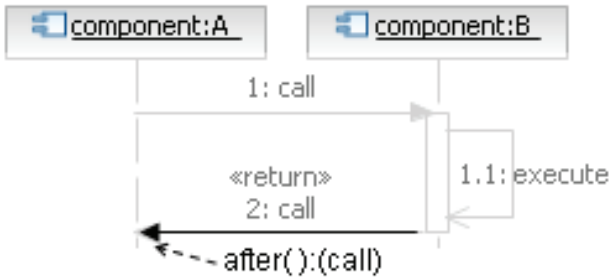



Figure 47 Join point semantics of after() directive

In case of an after-directive (see Listing 6), the before-reference of the last message in the tactic is set to the before-reference of the return message of m (return is indicated by m*) (line 1). Then, the after-reference of the first message in the sequence of the tactic is set to the return message (line 2). The before-reference of the message that was originally after-referenced by the return message (m*.after) in the protocol sequence is changed to the last message in the tactic (line 3). The before-reference of the return message m* as well as the source and target

components are changed accordingly (lines 4-6). The semantics of the after-directive are depicted in Figure 47.

```
after():(Message m){

1 T.sequence [sequence.size()-1].before=m*.before;

2 T.sequence [0].after = m*;

3 m*.before.after = T.sequence [sequence.size()-1];

4 m*.before = T.sequence [0];

5 T.sequence [0].source = m*.target;

6 T.sequence[message.size()-1].target = m*.target;

}
```

Listing 6 Weaving algorithm for after-semantics

For illustration purposes, a sample weaving is defined as follows (the tactic myTactic as depicted in Figure 44 is used):

```
after(): jp!instance(myConnector.call) -> myTactic
```

The semantics of this weaving specification is interpreted as follows: "Enhance the interaction specification of the connector *myConnector* with the interactions as specified by *myTactic* right **after** all places where a message with the name *call* occurs."

The graphical result of the weaving is shown in Figure 48. Figure 49 shows the resulting messages in terms of field changes indicated by blue highlighting.

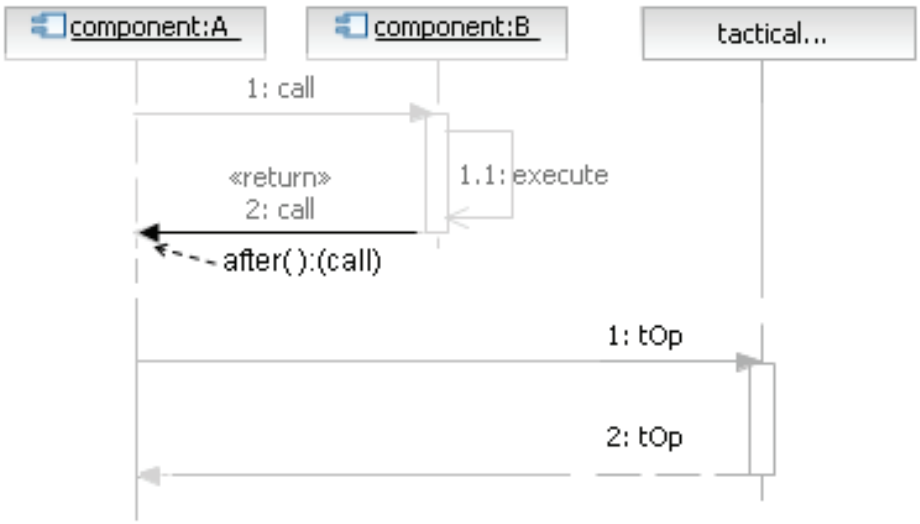



Figure 48 Graphical weaving result of the after()-directive

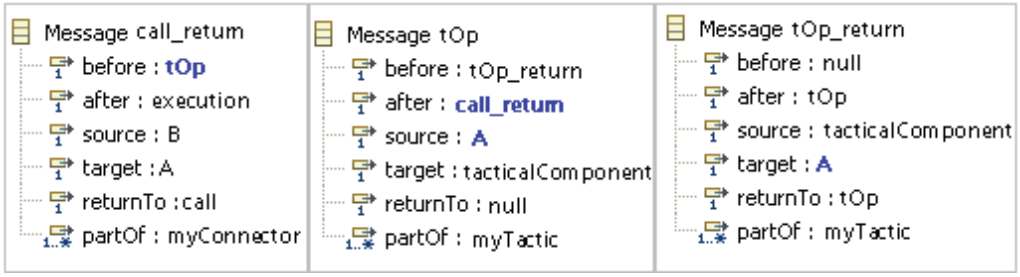



Figure 49 Message structure of the weaving result - after

## Adaptation Effect: **Around**

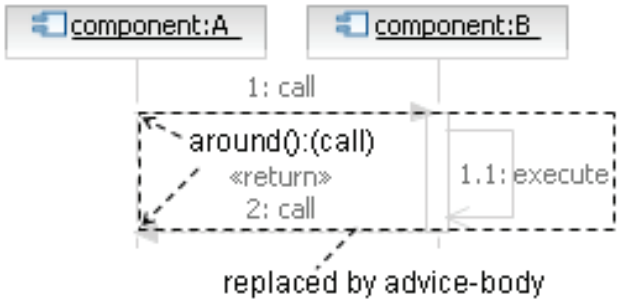



Figure 50 Join point semantics of around() directive

The around-advice sets the before-reference of the last message in the advice to the before-reference of the original return message (see Listing 7, line 1). Then, the after-reference of the first message in the advice is set to the after-reference of the original sequence (line 2). The before-reference of the originally preceding message (m.after.before) is set to the first message in the sequence of the tactic (line 3). The after-reference of the follow-up message of m (m*.before.after) is changed to the last message in the tactic (line 4). The source and target references

of the tactic are changed accordingly (lines 5-6). The semantics of the around-directive are depicted in Figure 50.

```
around():(Message m){

1 T.sequence[sequence.size()-1].before=m*.before;

2 T.sequence [0].after = m.after;

3 m.after.before = T.sequence [0];

4 m*.before.after = T.sequence [sequence.size()-1];

5 T.sequence[sequence.size()-1].target = m*.target;

6 T.sequence [0].source = m.source; }
```

Listing 7 Weaving algorithm for around-semantics

The tactic shown in Figure 51 defines a special keyword "thisJoinPoint". This keyword is used for accessing the join point context during weaving. Information about the join point message itself (denoted by the keyword "thisJoinPoint"), as well as the join point's source and target components (accessible via "thisJoinPoint.source/thisJoinPointTarget", respectively) are particularly important.

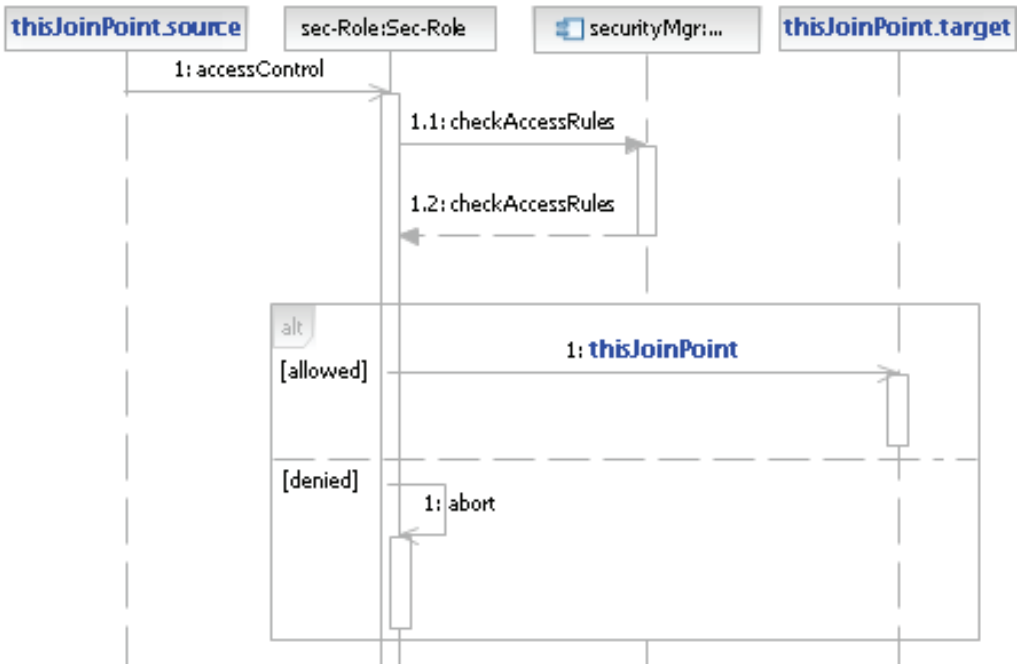



Figure 51 "authorizationTactic" with special keywords for around() directives

For illustration purposes, a sample weaving is defined as follows (the tactic "authorizationTactic" is used as depicted in Figure 51):

```
around():jp!instance(myConnector.call)->authorizationTactic
```

The semantics of this weaving specification is interpreted as follows: "Enhance the interaction specification of the connector *myConnector* with the interactions as specified by *authorizationTactic* in such a way that all places where a message with the name *call* occurs are wrapped according to the tactic's context".

The graphical result of the weaving is shown in Figure 52. Figure 53 shows the resulting messages in terms of field changes indicated by blue highlights. It is remarkable that the original sequence is maintained. The around-directive caused a wrapping effect since the tactic was designed to do so. However, the power of the around-advice allows to completely remove the join point itself as well.

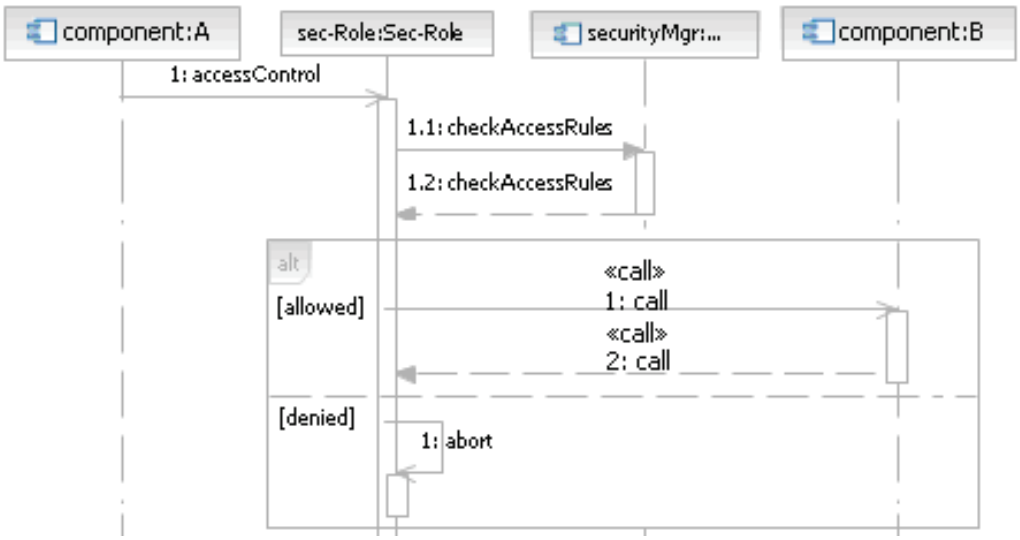



Figure 52 Graphical Weaving Result of the around()-Directive

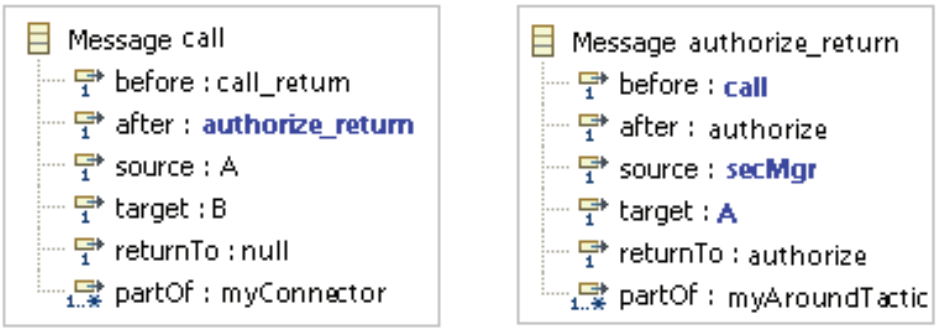



Figure 53 Message structure of the weaving result – around

# 5 Aspect-oriented Design Method

*"Most assuredly, there is no magic potion, and an adequate approach to design problems demands care, thought, creativity, discipline, experience, and method." [TMD09]*

In this chapter, we present the solutions to the challenges we found when the aspect-oriented architectural model was applied to architecture design. In order to relate the proposed design approach to existing work, I briefly give an overview of existing architecture design methods at the beginning of this chapter. In particular, I show what the commonalities and challenges of existing methods are. Then, I investigate two major challenges that arise when aspect-oriented concepts are to be applied in general and at the modeling level in particular:

1. Compositional challenges in terms of aspect interference, since aspects are working on the same base model.

2. Evolutionary challenges, since the base model potentially evolves independent of the aspect definitions.

## 5.1 Foundations

Architecture design approaches that have found adoption in practice are Attribute-driven Design (ADD) [BB01], Siemens' 4 Views method [HNS00], Rational Unified Process (RUP) [Kru03], Business Architecture Process and Organization (BAPO) [ARO03], Architectural Separation of Concerns (ASC) [Ran00], Functionality-based Architecture Design method [Bos00], and the Architecture Design Process as defined by Fraunhofer IESE, [EKK+10].

In the literature, there exist architecture design processes that prescribe when which activity leads to what artifacts. The authors of [HKN+07] provide a comparison of five industrial approaches for architecture design. They outline their commonalities as well as the differences. As a result, they derive a general model for architecture design.

In general, there are many commonalities among these architecture design methods. Most of the existing design approaches follow an iterative process model. We always find a preparation or analysis phase, an actual design or synthesis phase, and a validation phase. Synthesis is a well-known concept in traditional engineering disciplines and involves the construction of sub-solutions for distinct, loosely coupled sub-problems and the integration of these sub-solutions into a complete solution. It is the synthesis phase, as we will see, that covers the challenge of efficiently and effectively composing quality requirements solutions.

A mapping of the general phases to the Fraunhofer Architecture Process is shown in Figure 54. The analysis phase as defined by the general model maps to the elicitation of architectural concerns in terms of business goals, quality requirements, key functional requirements, and constraints. A stakeholder analysis determines the questions that stakeholders need to have answered by the architecture and thus determine what models need to be created throughout the architecture design. The synthesis phase maps to the realization phase. During realization, the solutions to particular concerns are designed and consolidated (synthesized). The documentation and assessment steps represent the architectural analysis phase as defined by the general model for architecture design.

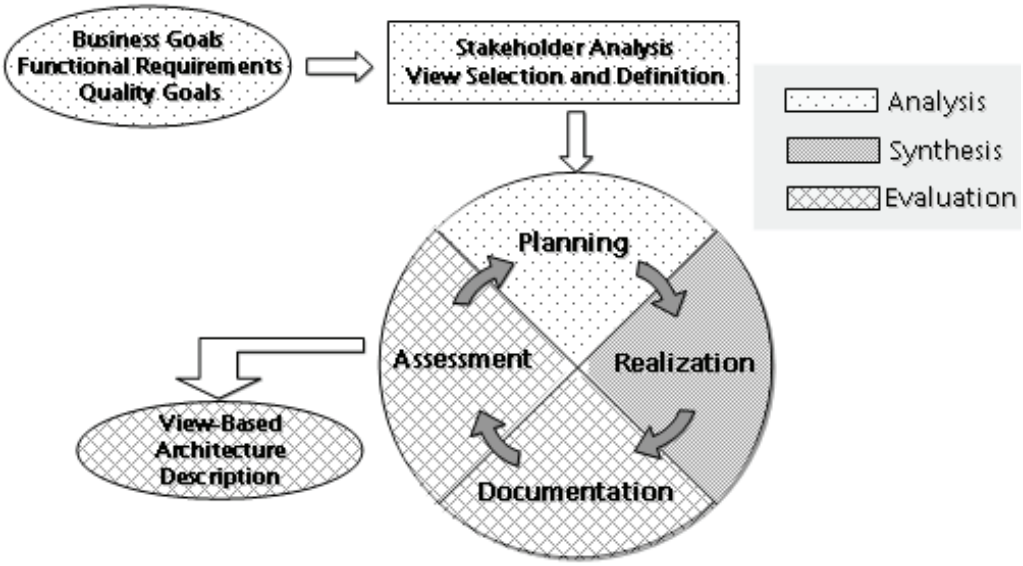



Figure 54 High-level depiction of the Fraunhofer Architecture Design process

Reflecting the need for separating and composing cross-cutting solutions into non-functional requirements, none of the methods analyzed in [HKN+07] addresses this issue. Besides differences in levels of abstraction, existing aspect-oriented approaches that aim at tackling cross-cutting concerns usually do not provide any methodical guidance. According to [SSK06], there are only two approaches that come with a process description [CB05] or guidelines ([FRG+04] and [EAB05]). However, [FRG+04] do not consider behavioral models (e.g., sequence dia-

grams) at all, and the authors of [CB05] and [EAB05] do not provide any tool support and model composition at all.

## 5.2 Design Method Overview

Our approach for connector design seamlessly integrates with the Fraunhofer Architecture Design process. Since connector design is an issue of the realization phase, refined view of the realization phase in terms of concrete realization activities in shown in **Error! Reference source not found.**.

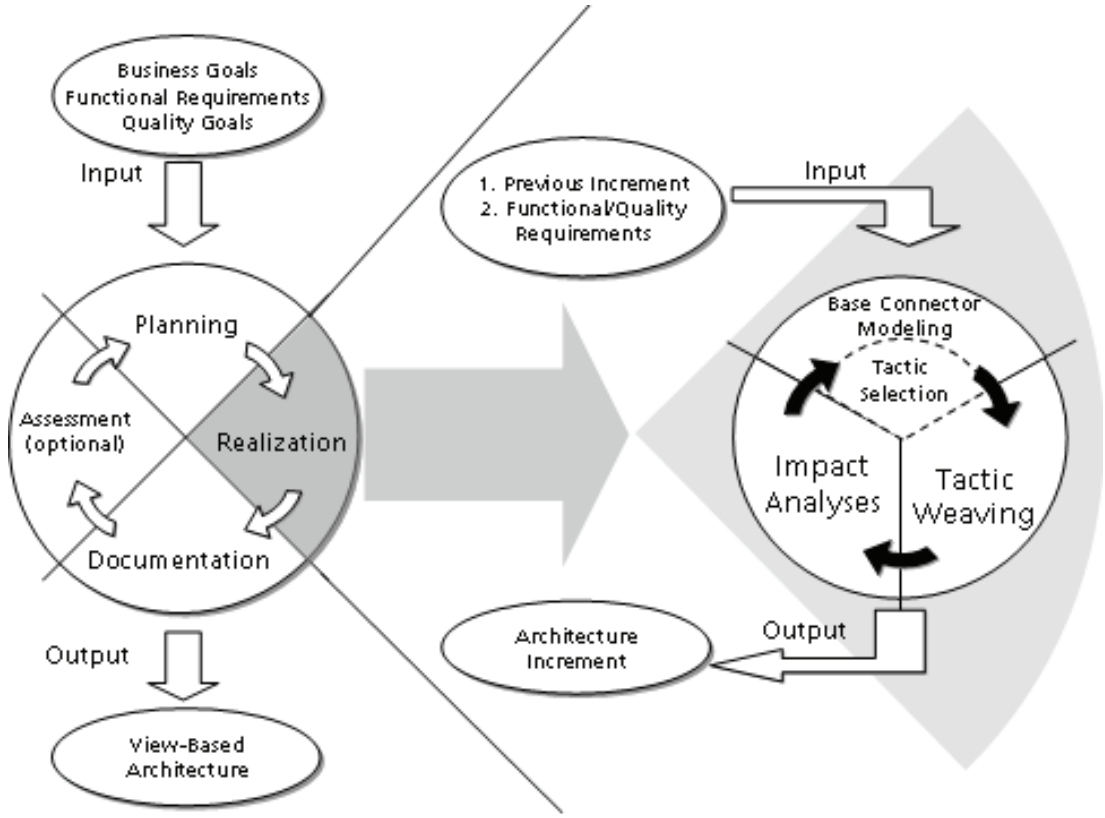



Figure 55 Connector design in the Fraunhofer Architecture Design process

The connector design takes architectural increments and a set of functional and/or non-functional requirements as input. The process itself decomposes into four phases, where the first two phases are executable independently from each other:

**Phase 1 – Base connector modeling**

In the first phase, a base connector model is designed that reflects core functional interaction behavior. For instance, consider a simple Publish-

Subscribe connector that describes the core functional interactions as shown in the following diagrams:

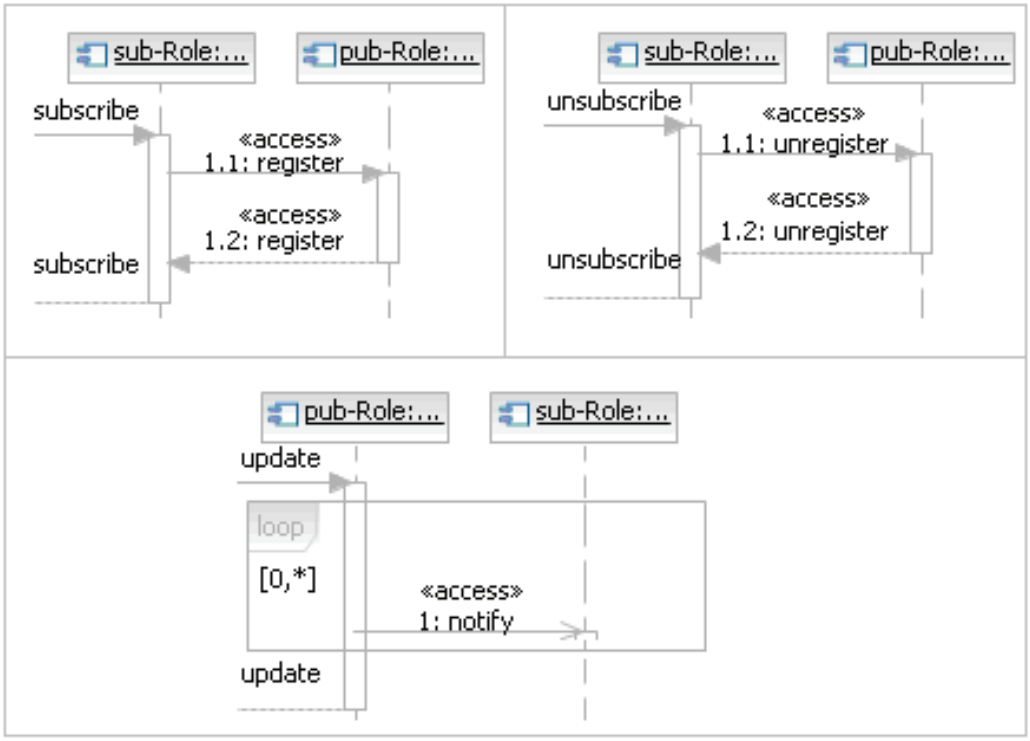



Figure 56 Base connector specifying Publish-Subscribe semantics

### Phase 2 – Tactic selection

The tactics used in the connector design process are execution tactics as defined in Section 3.3.2. The tactics themselves might be defined in other architectural projects as well, since they are not related to a particular architectural model at all. Similar to architectural patterns, tactics can be just selected in case they already exist; otherwise, they can be adapted or created from scratch.

### Phase 3 – Tactic weaving

During tactic weaving, the tactics are finally composed with the base connector models. The weaving itself requires a set of pre-processing steps that need to be executed manually. The pre-processing steps yield a so-called weaving configuration that specifies which tactics are supposed to go where in the connector models. The specification of where the tactics need to be woven is a manual step; however, I extended the modeling tool in such a way that the architect is able to interactively work with the model elements of relevance. The interactive selection of join points is efficient, since I utilize the architectural meta-model. That is, sets of messages can be assembled by selecting elements that aggregate, or indirectly refer to, messages that we want to intercept in order

to apply selected execution tactics. For example, by selecting a hardware component in a deployment diagram, the architect is able to select all messages that are sent or received by components deployed on that particular node without having to select all messages individually. This is possible because the relationship between components, artifacts, and nodes is clearly defined within the architectural meta-model.

**Phase 4 – Impact Analyses**

Impact analyses aim at estimating what the impact of the newly introduced tactic is. In other words, we need to check the suitability of the solutions for the given requirements and their compatibility with design decisions made in earlier iterations. For this reason, the resulting model needs to be checked for certain properties. The model properties to be checked depend on the quality that we need to check the impact on. One approach that seems promising for impact analyses is simulation. By simulating behaviors based on synthesized architectural models, we have the chance to perform cross-impact analyses that are objective and repeatable.

## 5.3 Designing Connectors under Constant Change

In this section, I show that the incremental problems as stated in Chapter 1 did not dissolve completely. The approach following the aspect-oriented separation of concerns still has to face these issues in one form or the other. In the following, I show the remaining problems and discuss corresponding solutions in detail.

### 5.3.1 Compositional Issues

One of the initial motivators for the work presented in this thesis is given by the fact that architecture development follows an iterative and incremental process. One of the main reasons that make increments problematic is given by the cross-cutting nature of architectural solutions to non-functional requirements. Depending on increments makes a revision of previous design decisions difficult.

Although we are now able to separate and compose architectural design decisions in the realm of connectors, the problem of increments has not been resolved; rather, it shifted in terms of the point in time we need to deal with it. That is, although tactics separate and modularize cross-cutting concerns, eventually they have to be composed with a base model. In this case, we are not depending on previous increments in the definition of execution tactics and the application to connector models; however, the composition itself represents the concentrated version of

incremental issues. The reason for this becomes clear when we look at the composition steps in more detail.

If we have two tactics: A security tactic as defined in Figure 51 on page 84, and a simple tactic for billing as defined in Figure 57.

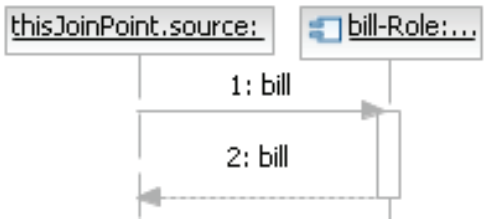



Figure 57 Billing tactic

According to the weaving specification given by:

```
around (jp!type(access)): Authorization-Tactic

after (jp!type(access)): Billing-Tactic
```

Listing 8 Weaving specification example

both tactics are woven into the base connector model (shown in Figure 56 on page 90).

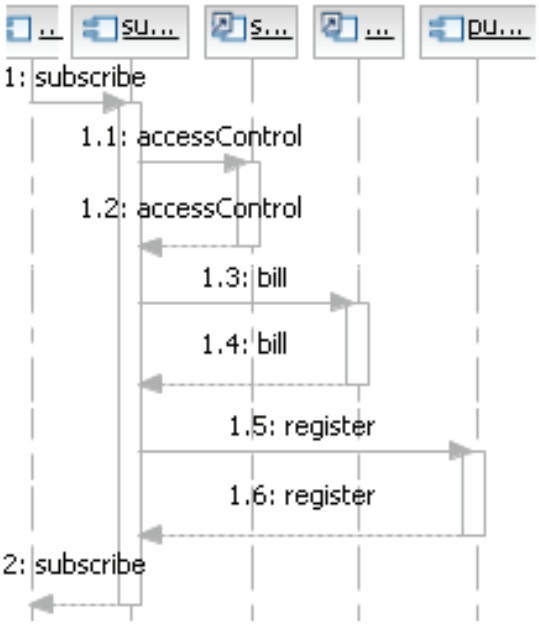



Figure 58 Authorization tactic interrupted by billing tactic

The composition itself, however, is going to be sequential because the connector model is augmented by one tactic at a time. As a consequence, we might end up in a situation that the tactics interfere at the

modeling level. In the aspect-oriented world, such interference effects are also called aspect interactions [STJ06].

Interactions refer to any situation in which the behavior of any tactic as specified in isolation is altered by some other tactic. In this example, we see that the billing tactic interferes with the security tactic since it interrupts the authorization check necessary for the billing (see Figure 58). (Here, we show the subscribe sequence only.)

### State of the Art Approaches towards Aspect Interactions

To this day, only few proposals have rigorously addressed the problem of aspect interaction detection. Most of the work concentrates on the syntactical level, i.e., on potential conflicts between aspects caused by the syntax. In [WJ08], the authors present MATA, a UML aspect-oriented modeling tool that uses graph transformations to specify and compose aspects. They use critical-pair analysis to detect dependencies between aspects on the syntactical level by looking at graph rule dependencies. The approach in [BE07] introduces Aspect Interaction Charts on top of Live Sequence Charts to specify aspects interactions in a modular manner. A formal framework for interaction detection is presented in [KFG04], including a method for the static analysis of aspect interaction and conflict resolution of parallel composition. This work is extended in [DFS02] by adding variables to aspects in order to define aspects more precisely and thus to eliminate spurious conflicts, and by introducing a generic composition operator that supports interaction. A trace-based approach is presented in [SKB03], where runs on the base and the woven system are compared with identical input to test if an aspect works as intended.

These approaches mainly concentrate on static program analysis and therefore often fail to decide about semantic conflicts between behavioral aspects. The authors of [DFS04] propose formal annotations, such as pre- or post-conditions, for analyzing aspect interactions in models written in Aspect-UML from a semantic point of view. However, besides the fact that all models have to be supplied with annotations, this approach does not support the checking of safety or liveness properties, which are essential for model checking semantic aspect interactions. In [MV07], the authors also use model-checking to verify aspect advices, but only for the state machines generated by the aspect code itself. Neither the entire system nor overridden aspects are verified. Katz et al. [GK06], [GK07] use linear temporal logic to verify that a single aspect will provide the desired properties when it is woven into an appropriate base model. However, their approach only addresses one single aspect and pointcut descriptor, and hence does not cover aspect interactions and their detection.

### 5.3.2 Change Issues

Looking at the aspect-oriented architectural model as defined in Section 4.3, the underlying concept is a clear separation of architectural tactics from architectural core models. Consequently, both of the separated entities might evolve, or change, in isolation from one another. This imposes additional complexity on the mechanism that relates both worlds: the pointcut.

When the architectural base models change, there are two symptoms that might impact the validity of specified compositions.

1. The existing pointcut definition **does not capture** (newly introduced) architectural join points that **should be captured.**

2. The existing pointcut **does capture** (newly introduced) architectural join points that **should not be captured.**

The reason for these issues is due to the fact that in general, join points of a system are strongly coupled with the respective pointcuts, which would cause problems during iterative architecting as well.

In the world of aspect-oriented software development, this phenomenon is referred to as the *fragile pointcut problem*, which is due to the high coupling between advice and the base system [Caz06]. This means that modifications of the base model may change the semantics of pointcuts even though the pointcut definition itself remains unaltered [KMB+06].

The example shown in Figure 59 shows a change in the message type (*access* is replaced by *call*) that would lead to a reduced join point set in case the pointcut definition (see Listing 8) would not be adapted.

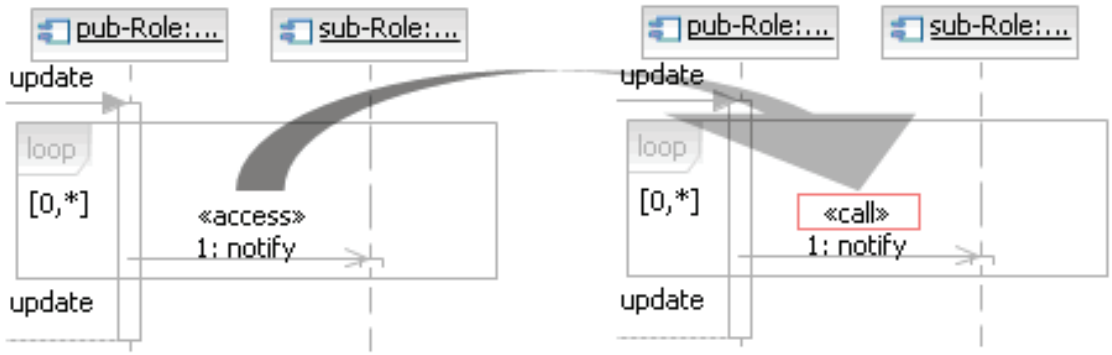



Figure 59 Changed base model causes a join point "miss"

In the subsequent sections, proposed solutions to the issues mentioned above, namely aspect interactions and pointcut fragility, will be explained in detail.

## 5.4 Managing Compositional Interactions

As exemplified in the previous sections, if tactics are to be composed that are not independent of each other, it is most likely that so-called tactic interactions will occur. Tactic interactions potentially result in unintended connector behaviors.

Since aspect orientation at the modeling level is quite a young discipline, so is interaction detection. If we look at existing approaches for aspect-oriented separation of concerns at the modeling level, there are only few that consider interactions at all. In the realm of architecture description languages, there are no such approaches that address interaction detection or resolution at the modeling level. According to [SSK06], there is only one aspect-oriented modeling approach that allows detecting conflicts; other approaches simply provide some guidelines for avoiding interactions. According to [SSK06], the conflict resolution mechanisms of existing methods are quite simplistic as well. Since interactions can either be on the syntactical or on the semantic level, existing model-based approaches address the syntactical interactions only. The work presented in this thesis goes one step further: I provide an approach that is capable of detecting semantic interactions as well.

### 5.4.1 Classification of Interactions

In order to find appropriate solutions for the interaction problem, I first classify interactions according to indications and impact. Interactions can occur when several tactics are woven into the same base communication model. I distinguish between three types of interactions: *dependencies*, *collisions*, and *conflicts*. Since the definitions given by [STJ06] are informal, I formalize the definitions of *dependency* and *conflict* in order to have a basis for defining appropriate resolutions strategies. *Dependencies* arise if one tactic is applicable to a model element introduced by another tactic, i.e. the number of join points for a given tactic increases after the application of another tactic. This can be formalized as follows:

**Definition – Weak Direct Dependency**

Let $A_1$, $A_2$ be two tactics, and $P_1 = \cup_{i \leq n} p^1_i$ and $P_2 = \cup_{i \leq m} p^2_i$ the corresponding sets of pointcuts. $A_1$ is said to be *weakly directly dependent* on $A_2$ if the following predicate holds:

$$DDep_{weak}(A_1,A_2) =_{DF} P_1(A_2) \neq \emptyset$$

where $P_i(M) : M \rightarrow 2^{Msg}$ denotes the application of a pointcut $P_i$ to a base communication model $M$.

In other words, $A_1$ is *weakly directly dependent* on $A_2$ if $A_2$ entails elements that are captured by $A_1$. Given the weaving specification

**after (jp!type(access)): Billing-Tactic**

the example tactic shown in Figure 51 on page 84 would introduce join points that would be captured by the pointcut used for weaving the Billing-Tactic. Here, we encounter a weak dependency between the Billing-Tactic and the Authorization-Tactic.

**Definition – Strong Direct Dependency**

Let $M = (S, Msg, \delta, s_0)$ be a base communication model. A tactic $A_1$ *strongly directly depends* on a tactic $A_2$ with respect to $M$ if the following condition is true:

$$DDep_{strong}(A_1,A_2,M) =_{DF} DDep_{weak}(A_1,A_2) \wedge P_1(M) = \varnothing$$

Strongly dependent means that a tactic requires a model element that is only introduced by another tactic.

In the event that two tactics are mutually directly dependent on each other, we encounter a so-called cyclic dependency. Weak and strong cyclic dependencies can be formalized as follows:

**Definition – Cyclic Dependency**

$$CDDep_{weak}(A_1,A_2) =_{DF} DDep_{weak}(A_1,A_2) \wedge DDep_{weak}(A_2,A_1)$$

$$CDDep_{strong}(A_1,A_2,M) =_{DF} DDep_{strong}(A_1,A_2,M) \wedge DDep_{strong}(A_2,A_1,M)$$

Two or more tactics may also interact *indirectly* via a base communication model. Indirect dependencies are determined by the respective pointcut specification. That is, we always need to have a reference to a base model in order to check if an indirect dependency between tactics exisits. I formalize this as follows:

**Definition - Indirect Dependency**

Let $A_1$, $A_2$ be two tactics, $P_1 = \cup_{i \leq n} p^1_i$ and $P_2 = \cup_{i \leq m} p^2_i$ the corresponding sets of pointcuts, and $M = (S, Msg, \delta, s_0)$ a base communication model. $A_1$ is said to be *indirectly dependent* on $A_2$ for a given $M$ if the following predicate holds:

$$IDep(A_1,A_2,M) =_{DF} P_1(M) \cap P_2(M) \neq \varnothing$$

In other words, $A_1$ is indirectly dependent on $A_2$ if $A_2$ alters a subset of the same join points altered by $A_1$.

Looking at the example given by:

**around (jp!type(access)): Authorization-Tactic**

**after (jp!type(access)): Billing-Tactic**

we can see that both tactics are operating on the same join points (specified by "access"). According to $_{DF}$ $P_1(M) \cap P_2(M) \neq \varnothing$, we encounter an indirect dependency between these two tactics.

The different kinds of dependencies that exist are indicators of potential problems. However, the main problems arise when two or more tactics are composed into one base model. This is where symptoms like collisions and conflicts occur.

*Collisions* are interactions between tactics on the syntactical level. A collision is declared if one tactic modifies a base communication model in such a way that a second tactic cannot be applied any longer or only in a limited fashion.

In the following, I refer to the composition of a base communication model $M$ with a tactic $A_x$ as $M \otimes A_x$, or $M^{Ax}$ for short.

**Definition 3 - Collision**

Let $A_1$, $A_2$ be two tactics, $P_1$ and $P_2$ the corresponding sets of pointcuts, $M = (S, Msg, \delta, s_0)$ a base communication model $M$. $A_1$ collides with $A_2$ at $M$ if the following predicates hold:

$$Col_{weak}(A_1,A_2,M) =_{DF} |P_1(M^{A2})| < |P_1(M)|$$
$$Col_{strong}(A_1,A_2,M) =_{DF} Col_{weak}(A_1,A_2,M) \wedge P_1(M^{A2}) = \varnothing$$

A collision between two tactics $A_1$ and $A_2$ is weak if the number of join points for $A_1$ in the base communication model is decreased after the weaving of $A_2$, compared to only applying $A_1$. A strong collision states that a tactic can no be longer be applied after enhancing the base communication model with another tactic. In such a case, a tactic would remove all join points of another tactic by using the around directive as explained in Section 4.3.5 without applying the *thisJoinPoint* keyword. For example, applying the weaving specification to the Publish-Subscribe connector, all join points (register, unregister, notify) would be replaced with the "replacementMessage".

**around (jp!type(access)): Authorization-Tactic2**

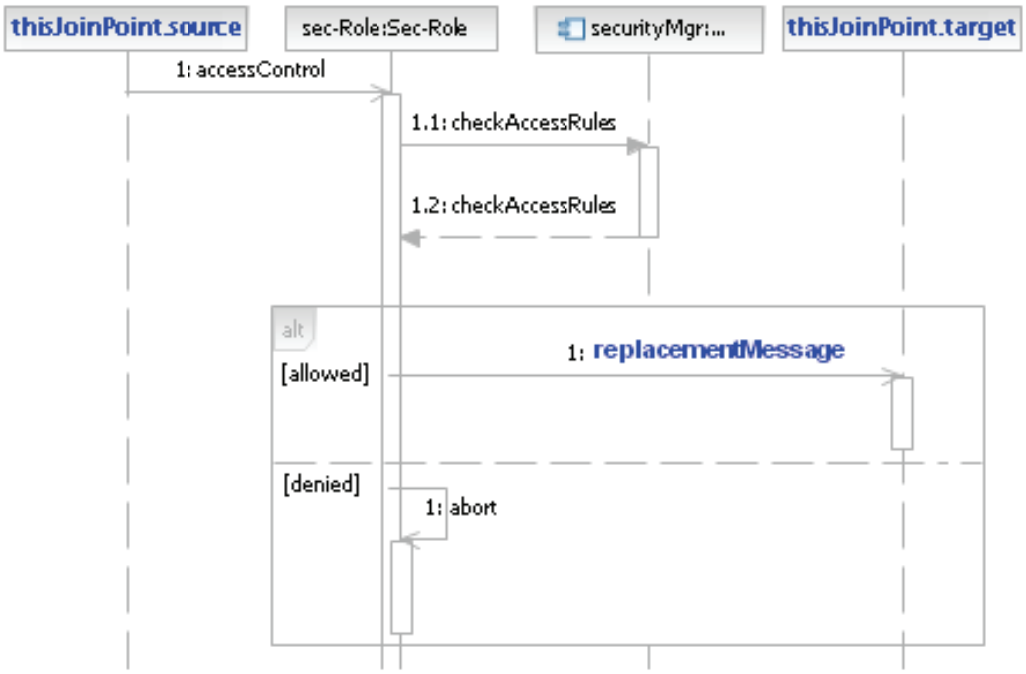



Figure 60 Authorization-Tactic2

The modified Authorization-Tactic as shown in Figure 60 is woven using the around-directive, while at the same time omitting the *thisJoinPoint* keyword referencing the respective join point.

Interactions on the semantic level are called *conflicts*. In the following, I use the notation of $M \models \varphi$ to denote that $\varphi$ is satisfied by a base communication model $M$. A formula $\varphi$ could express that a particular tactic is not interrupted by any other tactic, for example.

**Definition 4 - Conflict**

Let $A_1$, $A_2$ be two tactics and $\varphi_1$ and $\varphi_2$ the corresponding assertions that have to hold after the weaving of $A_1$, resp. $A_2$. Then, $A_1$ *conflicts* with $A_2$ at a model $M$ if the following predicate holds:

$$Con(A_1, \varphi_2, M) =_{DF} M \models \varphi_2 \wedge \neg(M \otimes A_1 \models \varphi_2)$$

In other words, a property of a model (expressed by $\varphi_2$) holds in the first place ($M \models \varphi_2$). After composing this model with another model (e.g., a tactic), that property does not hold anymore $\neg(M \otimes A_1 \models \varphi_2)$.

Relevant properties can be described by stipulating that specific messages occur at a particular point in time relative to the tactic specification. For instance, if the authorization is woven into the model, no other interaction must be woven in between the authorization message itself and the join point message.

Cyclic conflicts arise if two tactics alter the model during the weaving such that they mutually invalidate the stipulated property of the resulting model. In such cases, conflict resolution strategies need to be applied.

A *cyclic* conflict can be formalized as:

$$CCon(A_1,A_2,M) =_{DF} Con(A_1,A_2,M) \wedge Con(A_2,A_1,M)$$

The weaving of two tactics into a base communication model $M$ with $CCon(A_1,A_2,M)$ always implies undesired behavior in the resulting augmented models $M \otimes A_1 \otimes A_2$ or $M^* \otimes A_2 \otimes A_1$.

Note that dependencies can be examined and detected before the actual weaving, collisions and conflicts only afterwards. Consequently, the predicate $DDep_{weak}$ is universally valid since dependencies between tactics are independent of any base communication model. All other interactions are always defined with respect to a specific base.

Dependencies, collisions, or conflicts between two or more tactics usually imply that these tactics should be composed with the base communication model in a particular order, since the resulting models are different. Analyzing interactions helps to determine a *feasible* order of composing tactics with a base communication model. Since interactions on the syntactical level do not necessarily result in undesired system behavior, I define feasible weaving on the semantic level as follows:

**Definition 5 - Feasible Weaving**

Let $A_1,\ldots A_n$ be tactics that have to be woven into a base communication model $M$, $\varphi_i$ the property that has to hold after the weaving of $A_i$, and $M^i = M \otimes A_1 \otimes A_2 \otimes \ldots \otimes A_i = M \otimes_{1 \le k \le i} A_k$ the base communication model resulting from the sequential composition of $M$ with tactics $A_1,\ldots, A_i$. A sequence of tactics $w = (A_1,\ldots A_n)$ is called a *semantically feasible weaving* if the following condition holds:

$$\forall\, i \le n \,.\, M^n \models \varphi_i$$

This means that all formulas established up to step *i-1* must still hold when the model is augmented at step i. Hence, all elements of $w$ are pair-wise conflict-free, i.e.,

$$\forall i \le n\ \forall j < i \,.\, \neg Con(A_i, A_j, M^{i-1}) \wedge M^n \models \varphi_i$$

Feasible weaving is important, since a weaving configuration needs to be checked for feasibility in order to prevent conflicts in the resulting model.

### 5.4.2 Interaction Detection

In this section, I describe the approach for detecting collisions and conflicts [KW09]. Collision detection is straightforward, as we can check the properties defined above by examining the respective sets of join points. For detecting conflicts, I use fluent linear temporal logic as a formal foundation. In the following, I will describe the approaches in more detail.

**Detecting Dependencies and Collisions**

Each of the classes of interaction described in the previous section can be detected by different kinds of approaches. Dependency detection between two tactics $A_1$ and $A_2$, for instance, can be addressed by looking at the join point collections of $A_1$ and $A_2$, respectively. To check for weak dependencies between $A_1$ and $A_2$, the join points are analyzed that are collected by each of the pointcuts $P_1(A_2)$ and $P_2(A_1)$. If the collection of join points is not empty in either one of the cases, a weak dependency is detected.

In case of strong dependencies, we check the join points captured by the pointcuts mutually applied to the tactics. If any result set is not empty, we have found an indicator of a strong dependency. Then we check if the pointcut as applied to the base model returns an empty set of join points. To that end, we can check for the conditions stipulated by definition $DDep_{strong}$. More concretely, if the predicate $DDep_{weak}$ holds and the collection of join points gathered by $P_1(M)$ is empty, we have detected a strong dependency. Indirect dependencies can be detected by comparing the pointcuts applied by different tactics to the same base model. If we find pointcut expressions that return identical subsets of join points, we have detected an indirect dependency. Collisions can be detected in a similar way, by just looking at the sets of join points that are collected by pointcuts of different tactics applied to the same base model.

**Detecting Conflicts**

The main idea for checking semantic interactions of aspects, or tactics in our case, is based on formal model checking techniques on the basis of Labeled Transition Systems and Fluent Linear Temporal Logic (FLTL) [GM03]. As described in Section 5.3.1, there is one approach that uses FLTL to check particular model properties; however, the approach does not consider the detection of interactions. Our approach pushes the envelope of semantic interaction detection in two regards: First, I enable the specification of compositional properties of aspects that can be model-checked, and thus, extend the traditional property checks of *safety* and *liveness*. Second, I provide a set of heuristics that can be used for deriving the formal specifications of properties automatically. A *labeled transition system* (LTS) can be used to model the communicating behav-

ior of different entities in (distributed and concurrent) systems. More formally, a labeled transition system *M* is described by a quadruple ($S$, $A$, $\delta$, $s_0$) where:

- $S$ is the finite set of possible states
- $A \subseteq$ *Action* is the finite communicating alphabet of *M*,
- $\delta \subseteq S \times A \cup \{\tau\} \times S$ is a transition relation from state $s_1$ to state $s_2$ labeled with an action $a \in A$
- $s_0 \in S$ is the initial state

where *Action* is a global set of visible actions or events and $\tau$ a local action that is unobservable. A sequence of messages over $2^A$ (including the $\tau$ message) that *M* can perform in its initial state is called an *execution* of *M*. The semantics of a LTS are described by the set of all executions. I use the formalism of LTS in the context of connector specifications, since the formal connector foundations stem from the theory of communicating sequential processes (CSP) [AG97]. Formally speaking, a connector is described by a set of interacting roles and a glue specification. Roles represent entities involved in an interaction defined by a connector. Each role has to follow a protocol specification that expresses the local behavior of the respective role. The glue specification, in turn, determines the interplay of different collaborating roles. In terms of models, so-called connector automata (CA) can be used for formally specifying connectors. A CA coordinates the functional behavior of the roles by denoting how the entities work together. A CA describes a glue specification as a set of tasks represented as states and possible transitions between states where each transition is labeled by a message. We can map connector models as presented in Section 4.3.5 to LTS in a straightforward fashion. Hence, let $Msg_B$ be the set of possible messages $m = (p, s, src, trgt)$ that can occur between two communicating entities *src* and *trgt*, enclosed by a preceding message *p* and a succeeding message *s*, and let $\tau$ be a local message that is not observable. A *Connector Automaton* C is a discrete *Labeled Transition System* described by a quadruple ($S$, $M$, $\delta$, $s_0$) where:

- $S$ and $s_0$ are a set of states and the initial state, respectively,
- $M \subseteq Msg_B$ is the finite communicating alphabet of *C*,
- $\delta \subseteq S \times M \cup \{\tau\} \times S$ is a transition relation from state $s_1$ to state $s_2$ labeled with a message *m*

An example LTS specifying the Publish-Subscribe connector protocol as described by the diagrams shown in Figure 56 on page 90 is depicted in Figure 61.

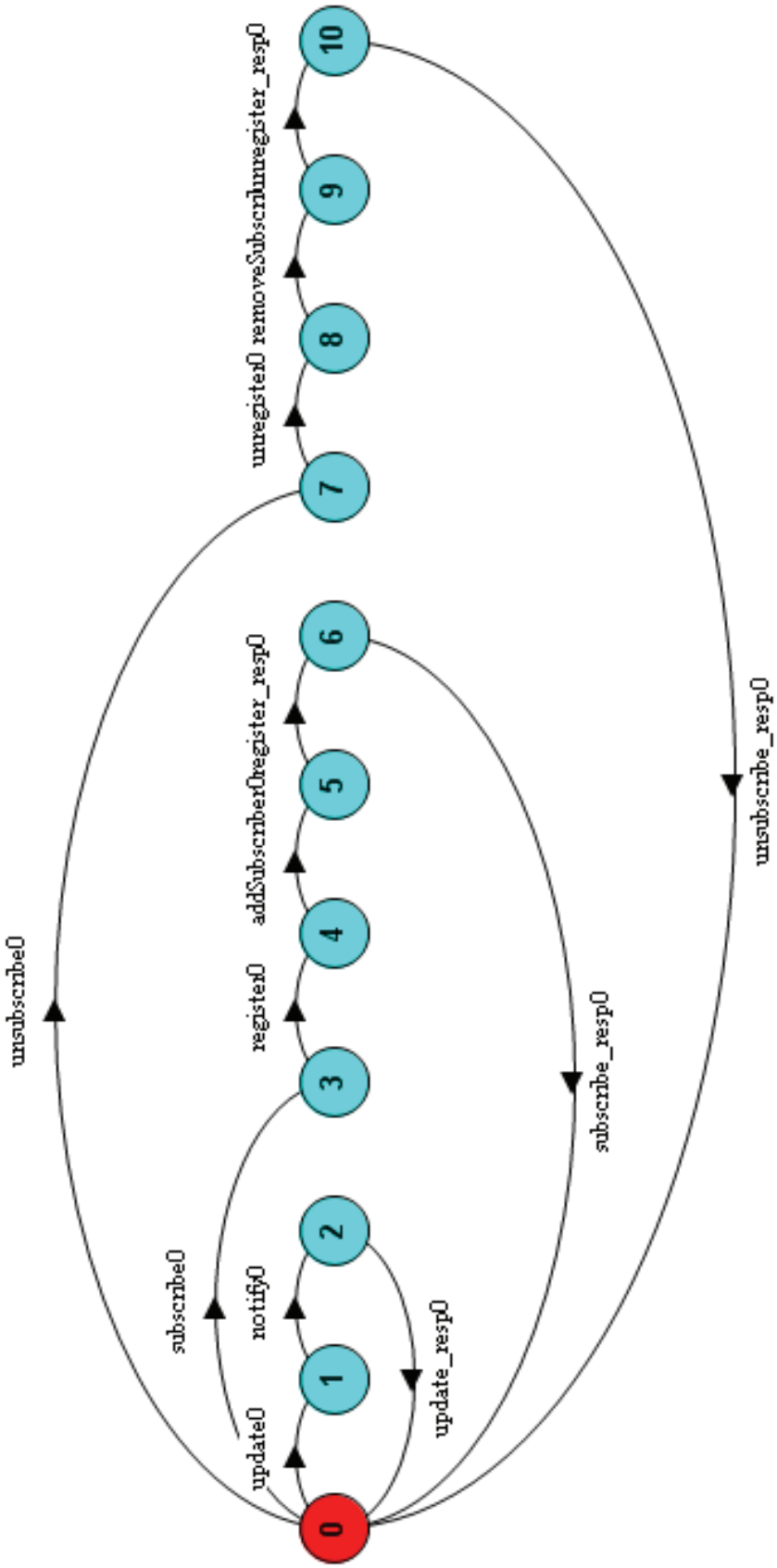



Figure 61 LTS representation of a Publish-Subscribe connector

The LTS resulting from the weaving of security into the Publish-Subscribe connector is depicted in

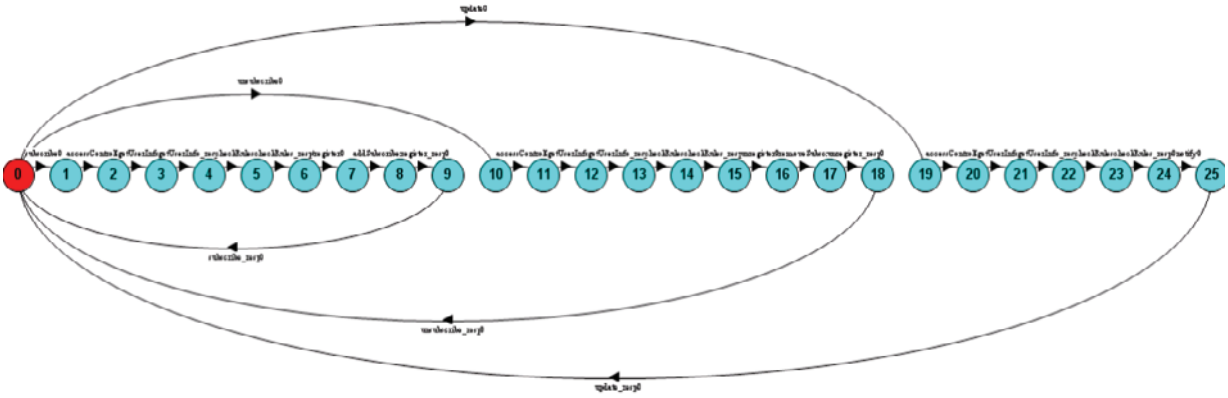



Figure 62 Woven Publish-Subscribe connector with authorization tactic

I use Fluent Linear Temporal Logic (FLTL) to specify state-based temporal logic properties over an event- or message-based computational model. A *fluent* [MS99] is a time-varying property of the world that is true at some time instant $t$ if it has been initiated by an action or message at a time instant $t_p$ prior to $t$ and has not been terminated by another action or message in the meantime. More formally, a fluent $F$ is a proposition defined by a set of initiating actions *Init* and terminating actions *Term,* and an optional attribute *Initially* that states whether the fluent $F$ is true or false at time zero:

$F =_{DF} \langle Init, Term \rangle$ with $Init, Term \subset Action$

The set of initiating and terminating actions has to be disjoint. The default value of a fluent is false.

The set of fluents $\Phi_{Fl}$ is the set of the atomic propositions which built a FLTL formula. Analogous to LTL, an FLTL model $m_{FL}$ is an infinite sequence over $2^{\Phi_{Fl}}$. By means of $\Phi_{Fl}$ it is possible to define a mapping from action-based to state-based traces. Therefore, let $tr_a$ be an action trace, i.e., a sequence of actions that can occur in a corresponding state machine and let $tr_a(i)$ denote the $i$-th element. This can be formalized as:

$\forall\, i \in \mathbb{N}_0, \forall\, F \in \Phi_{Fl}\, .\, F \in m_{FL}(i)$ iff

(a) $\quad Initially \wedge (\forall\, k\, .\, 0 \leq k \leq i, tr_a(k) \notin Term) \vee$

(b) $\quad \exists\, j \mid (j \leq i) \wedge (tr_a(j) \notin Init) \wedge (\forall\, k\, .\, j < k \leq i\, .\, tr_a(k) \notin Term)$

Condition (a) states that the fluent *F* is true at position *i* if it is initially true and no terminating action has occurred before position *i*, condition (b) stipulates that *F* is true at *i* if some initiating action occurred before position *i* and no terminating action has happened since then. In the following, I use the notation of $tr_a \models Assert$ to denote that an FLTL assertion *Assert* is satisfied by an action trace $tr_a$.

Every action $a \in Action$ implicitly defines a fluent $F(a)$ as follows:

$$F(a) =_{DF} \langle \{a\}, Action \setminus \{a\} \rangle$$

The fluent $F(a)$ becomes true at the time instant *a* occurs and becomes false with the first occurrence of any other action or event.

Bounded FLTL operators [LKM+05] extend FTLT by timed properties. For example, the property $\Diamond_{\leq 5}\, \varphi$ means that $\varphi$ holds at some future point in time not more than 5 ticks or time units away from the current position in the state-based trace (now). To determine this time elapsed between two positions of the state-based trace, a temporal distance function over a timed event-based model is used.

In order to check the semantic properties of composed models, we define assertions about events, or sequences of events, that should never arise or always be true. Assertions refer to state-based temporal logic properties specified over an event-based operational model that have to be satisfied by a state machine after tactics have been woven. Properties refer to properties of the tactics as well as those of the base communication model itself. For instance, we can define assertions that stipulate having *event* $e_1$ always followed by e*vent* $e_2$.

Considering the Publish-Subscribe connector, we could define a fluent that specifies that there should be an event *granted* right before the message *register*. This stipulation is then defined as:

FLUENT AUTHREG = ⟨granted, InvalidateSet ∪ {register_rsp} ⟩

WITH

InvalidateSet = $Msg^*$ ∪ {*accessControl*, *abort, denied*}

and the assertion

□ (*register* → AUTHREG)

The following sequence shows the values of the fluent according to message occurrences along the sequence. The fluent changes to true as soon as the message *granted* occurs. At the time the message register

occurs, the fluent is still valid, so the assertion □ (*register* → AUTHREG) holds.

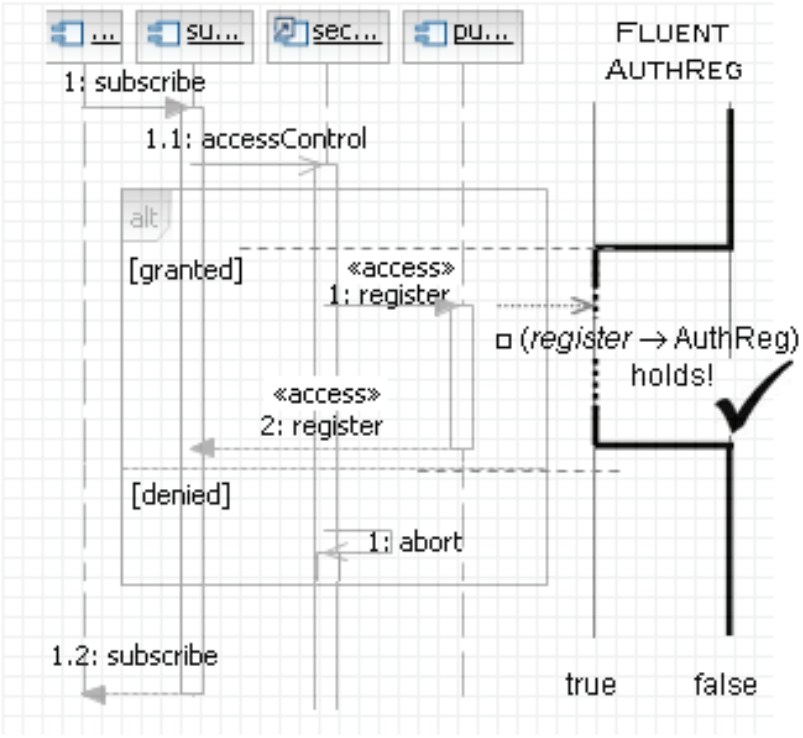



Figure 63 The assertion holds since the fluent is evaluated to true

### 5.4.3 Generating Fluents from Weaving Specifications

Since the manual creation of such fluents and assertions is a tedious and error-prone task, we aimed at an automation of that task. To that end, we specify a set of heuristics that allow for generating such fluents and assertions automatically from a given weaving specification. Since the weaving specification indicates some properties already (e.g., by the adaptation effect), we use this information for deriving some fluents that express standard properties to be true after weaving.

For deriving assertions, we need to know in what way the tactics are supposed to be woven into the base communication model, as the way tactics are woven implies contextual properties of that tactic with respect to the base communication model. The derived assertions capture these contextual properties and make them checkable during further modifications by other tactics.

As already indicated, the contextual properties are heavily dependent on the adaptation effect selected for particular tactics. In the following, we will elaborate on each of the adaptation effects and define a set of rules that are useful for the automatic generation of fluents and assertions.

**Before-Advice:**

Let *A* be a tactic and *P* be the corresponding pointcut. When defining an assertion that describes the desired behavior of a connector augmented by tactic *A*, we have to distinguish between two cases: synchronous interaction and asynchronous interaction at the join point described by *P*. In the following, let *join point* denote the message where the base communication model and the tactic meet. For synchronous communication, let *join point_rsp* denote the return message caused by a synchronous message *join point*. A feasible assertion for the before-advice is given by

□ (*joinPoint* → BEFOREADVICE), with
FLUENT BEFOREADVICE = ⟨ *validatorMsg*, *joinPoint_rsp* ⟩

where *validatorMsg* ∈ $Msg_A$ is a message that validates the fluent. In other words, the validator message is the message that occurs right before the join point. The assertion defined in the before-case stipulates that the fluent BEFOREADVICE is always true at the time when the respective join point message occurs. The fluent BEFOREADVICE is defined to be true when a particular validator message (*validatorMsg*) occurs, and is falsified when the respective response message of the join point occurs (*joinPoint_rsp*). (Note that this rule implies a synchronous message exchange.)

In general, when interacting asynchronously, there is no response message that can be used to invalidate the fluent. For that reason, all messages *m* = (*p*, *s*, *src*, *trgt*) that may occur directly after the *join point* are used. Let *Join pointNext* = {*m* ∈ $Msg_B$ | *p* = *join point*}. A feasible assertion for asynchronous communication can then be defined as follows:

□ (*joinPoint* → BEFOREADVICE), with
FLUENT BEFOREADVICE = ⟨ *validatorMsg*, *JoinPointNext* ⟩

The assertion just defined is, in a sense, not very strong. There is no strict coupling between the validation of the fluent and the *join point,* i.e., messages or events that do not influence the fluent may occur. In the majority of the cases, this is not a problem; On the contrary, it is the only possibility for tactics to be applied successively. But sometimes, strict coupling is mandatory, for instance if for security reasons, an authorization is required immediately before a login, without any further events or messages in between. In that case, the set *JoinpointNext* has to be extended by all messages ∈ $Msg_T$ that belong to the tactic and may occur between *validatorMsg* and *join point.*

Therefore, let *o* be a non-empty sequence of messages of length *n* denoting the allowed order of message exchanges over *A* and let $m_i$ de-

note the *i*-th message of that sequence. Then, $\forall\ i < n$ is $m_i = p_{i+1}$. Let $Ind_o : Msg \rightarrow N$ be an index function over $o$ where $Ind_o(m)$ denotes the position of $m$ in the sequence $o$ and $j = Ind_o(validatorMsg)$ and $k = Ind_o(join\ point)$, respectively. For strong assertions, the set of messages *Inv* that invalidate the fluent is defined as $Inv = Join\ pointNext \cup Msg^*_A \not\in Msg^*$, where $Msg^*_A = \{m \in Msg_A \mid Ind_o(m) < Ind_o(validatorMsg)\}$ and $Msg^* = \{m \in Msg_{A1} \cup \ldots \cup Msg_{An} \mid m \neq join\ point \wedge m \neq validatorMsg \wedge m \notin Msg^*_A\}$. The latter describes the set of messages defined by other tactics to be injected, and thus might be woven in between the validating message of *A* and the join point. So the resulting fluent for a before-advice is given by

$$\text{FLUENT BEFOREADVICE} = \langle\ validatorMsg, Inv\ \rangle$$

In case of asynchronous communication, a system model with two succeeding messages of the same type would satisfy the above assertions, although the assertion is obviously violated. This is due to the fact that when the second message occurs, the fluent is still valid, since no response message *"_rsp"* will invalidate it. However, this can only happen if tactics depend on each other (cf. Definitions 1 and 2) or if they are woven in incorrectly. The former can be detected statically without weaving; the latter may occur if the weaving algorithm is incorrect. A more precise assertion avoiding the aforementioned drawbacks can be defined by using the next operator or a global clock such that

$$\Box\ (joinPoint \rightarrow \text{BEFOREADVICE} \wedge \Box\neg joinPoint), \text{ or}$$
$$\Box\ (joinPoint \rightarrow \text{BEFOREADVICE} \wedge \Box_{\leq 1}\ \neg joinPoint)$$

Obviously, it is impossible to successfully enhance one join point in the base communication model by two different before-advices whose assertions are strong, since the resulting model cannot satisfy both properties.

**After-Advice:**

Two possibilities for specifying assertions for an after-advice for synchronous interactions are given by

$$\Box\ (joinPoint\_rsp \rightarrow \bigcirc\ \text{AFTERADVICE}) \text{ or}$$
$$\Box\ (joinPoint\_rsp \rightarrow \Box_{\sim X}\ \text{AFTERADVICE}), \text{ with}$$
$$\text{AFTERADVICE} = \langle\ validatorMsg, \text{in}validatorMsg\ \rangle$$

where $\sim \in \{<, >, \leq, \geq\}$.

However, the first assertion requires the advice to be executed immediately after the join point and is, in addition, not closed under stuttering. The second assertion requires a global clock to synchronize tick events. A

feasible assertion without the next operator or a global clock can be defined as follows:

$$\Box\,(\mathit{joinPoint\_rsp} \rightarrow (\neg(m_1 \vee \ldots \vee m_n)\ \mathbf{W}\ \textsc{AfterAdvice})\,), \text{ with}$$
$$\textsc{AfterAdvice} = \langle\, \{\mathit{valid}_1,\ldots,\mathit{valid}_n\},\{\mathit{invalid}_1,\ldots,\mathit{invalid}_m\}\,\rangle$$

The above assertions state that after every occurrence of the *join point* event, the advice has to be executed before any of the messages $m_1$ to $m_n$ occur. The messages $m_i$ are to be chosen according to the context. For instance, a strong assertion can be derived from the above by choosing all $m_i \in \text{Msg} \setminus \{\mathit{jointPoint\_rsp}\}$. The messages $\mathit{valid}_i$ and $\mathit{invalid}_j$ are appropriate messages taken from the tactic, i.e., $\forall\ i,j\ .\ \mathit{valid}_i, \mathit{invalid}_j \in \mathit{Msg}_T$.

In the asynchronous case, *join point_rsp* has to be replaced by *join point*, since no explicit return message exists. As in the before-advice, a system model with two succeeding messages corresponding to join points, but only one after-advice, satisfies the above assertion, although the model does not show the desired behavior. A more sophisticated assertion can be defined in the following two ways:

$$\Box(\mathit{joinPoint} \rightarrow (\neg(m_1 \vee\ldots\vee m_n)\ \mathbf{W}\ \textsc{AfterAdvice})) \wedge \Box\neg\mathit{joinPoint}, \text{ or}$$
$$\Box\,(\mathit{joinPoint} \rightarrow (\neg(m_1 \vee \ldots \vee m_n)\ \mathbf{W}\ \textsc{AfterAdvice})\,) \wedge \Box_{\leq 1}\neg\mathit{joinPoint}$$

**Around-Advice:**

An around-advice replaces any join point within the base model with the interactions described by the advice, either by keeping the join point and adding some "interactions" before and after, or by completely substituting the join point by one interaction. For the first case, the around-advice can be seen as a sequential composition of appropriate before- and after-advices. A feasible assertion is given by:

$$\Box\ (\textsc{BeforeAssert} \wedge \textsc{AfterAssert})$$

where BeforeAssert and AfterAssert are corresponding assertions for a before- and an after-advice as described in the previous subsections. Further, the join point may also be renamed, that is, the join point does not necessarily have to be in $\mathit{Msg}_B$, but can also be in $\mathit{Msg}_T$. In the second case, the presented heuristic cannot be used to (automatically) derive fluents and assertions based on the interaction advice. This is due to the fact that in the resulting model (after the advice injection), there is no particular trigger or message coupled with the advice and therefore the assertions cannot be specified without knowing the embedding context. In that case, the assertion has to be defined manually. Eventually, for every join point that is captured by pointcuts of a tactic, we generate

one individual assertion that reflects the properties of the tactic in that particular join point context.

An example of deriving fluents and assertions for around-adaptations is given by the following weaving specification:

**around (access): Authorization-Tactic**

Here, we refer to the base model as defined in Figure 56 on page 90 and the Authorization tactic as defined in Figure 51 depicted on page 84. In the example, the Authorization tactic introduces no model elements after the thisJoinPoint message. For that reason, an assertion for the Authorization advice may be defined according to (A8) by □(access → StrictAuth). Since Authorization is a security-critical task, we model the generic assertions as strict (A3) to prevent any message interactions between authorization and registration, leading to:

FLUENT StrictAuth_Sync = ⟨*granted*, *InvalidateSet* ∪ {*joinPoint_rsp*} ⟩
FLUENT StrictAuth_ASync = ⟨*granted*, *InvalidateSet* ∪ *JoinPointNext* ⟩,
with
*InvalidateSet* = $Msg^*$ ∪ {*accessControl*, *checkAccessControl*,
*checkAccessControl_rsp, denied*}

When applying the pointcut to the base model, the message joinPoint_rsp and the set JoinPointNext have to be instantiated by corresponding messages of the base model. In the example case, this leads to the following assertions and fluents for all messages of type <<access>>, namely (a) register, (b) unregister, and (c) notify:

(a) □ (*register* → StrictAuth_Sync _Reg), with
FLUENT StrictAuth_Sync _Reg =

⟨ *granted*, *InvalidateSet* ∪ { *register_rsp*} ⟩

(b) □ (*unregister* → StrictAuth_Sync_ UReg), with
FLUENT StrictAuth_Sync_ UReg =

⟨ *granted*, *InvalidateSet* ∪ { *unregister_rsp*}⟩, and

(c) □ (*notify* → StrictAuth_ASync No) with
FLUENT StrictAuth_ASyncNo =

⟨ *granted, InvalidateSet* ∪ { *updating_rsp*} ⟩

With these rules, we are now able to automatically generate a set of fluents and assertions such that conflicts can be checked before the "actual" weaving.

### 5.4.4 Composition Process

Our interaction detection process, as depicted in Figure 64, entails four distinct steps (right-hand side of the picture). The identification of syntactical interactions (dependencies and collisions) is done independent of the semantic analyses. In the following, we will describe each of these steps in more detail.

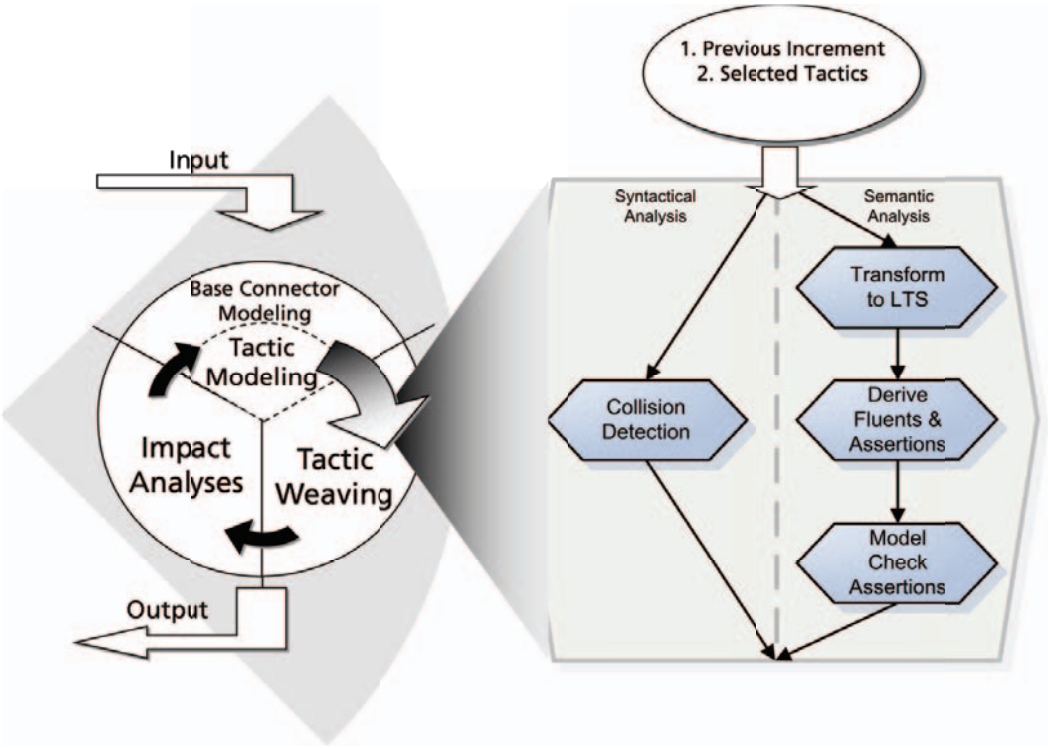



Figure 64 Interaction detection in the process of connector design

**Collision Detection:** For syntactical analyses, we check the properties that are stated by definitions given in Section 0. That is, we analyze sets of join points gathered by pointcut expressions.

**Transform to LTS**: To detect interactions on the semantic level, the sequences describing the connector semantics have to be mapped to FSP (Finite State Processes) in order to derive a computational model expressed as LTS. That is, we transform the sequence-based connector specification into an FSP representation. The FSP specification serves as one input (besides the fluents and assertions) for the LTS Analyzer for model-checking the augmented connector model.

**Derive Fluents & Assertions:** In this step, we derive fluents and assertions based on the tactics selected and related weaving specifications.

**Model Check Assertions:** In the last step, we run the LTS Analyzer in order to check if the defined assertions hold for the augmented base model. For this purpose, the FSP specification, the assertions, and the

fluents serve as input to the tool. The LTSA uses the FSP description to generate an LTS behavior model that is equivalent to our connector automaton. Finally, we run the LTS analyzer and check if the defined assertions hold for the augmented base communication model.

### Interaction Resolution Strategies

If we encounter interactions, a resolution for the respective conflicts needs to be found. In the following we present effective strategies that can be pursued when interactions need to be resolved.

1. Changing the weaving sequence of tactics

Depending on the adaptation effect (before, after, around), there are different impacts of the tactics with respect to the selected sequence in which the tactics are woven into the model. For instance, in case an around-advice is used, the tactic is likely to drastically change the connector model at the join point, whereas in case of before- and after-advice the connector model is just extended. One resolution strategy is to find a sequence that alters the base models as non-intrusively as possible from one step to the next. In general, around-advices are woven as late as possible, since they might be intersected by other tactics otherwise.

2. Changing the strictness of assertions

One strategy for resolving conflicts is to trade-off strictness requirements among tactics. It might not be okay if a security tactic is interrupted by a logging tactic; however, it might be reasonable for a billing tactic to be interrupted by an encryption tactic.

3. Renaming the join points (types could be renamed)

A resolution strategy for collisions is to create a distinction between the join points matched by a couple of tactics. In case the join points are semantically disjoint, a renaming of the respective join points, or a change of types, might be a good way to go.

4. Combining critical tactics into one module before weaving

If cyclic conflicts occur, a resolution for such a situation might be to integrate the interacting tactics at the source. That is, by combining both tactics into one model, they are woven into the base model in the way they are supposed to work together.

## 5.5 Pointcut Evolution

The second major issue besides compositional challenges as mentioned at the beginning of this chapter was the connector design under constant change [PAC+05]. The problem of pointcut fragility is the main reason for aspect-oriented design to be specified incorrectly.

Since fragility is defined relative to the model modifications performed, we scope the change scenarios that may occur, and analyze the impact of these scenarios on pointcut definitions in general. In addition, we derive strategies for evolving pointcuts in the face of change. The concept of type-based pointcuts as introduced in Section 4.3.4.1 already contributes to the stability of pointcuts, as we will see.

### 5.5.1 Evolution Scenarios

Regarding the scope of change scenarios, we look at changes that might happen to the base models themselves, namely the sequence diagrams describing the interactions between components as well as model elements (e.g., the deployment) that indirectly influence the connector model.

Regarding the connector models themselves, we find three change scenarios that potentially impact pointcut specifications:

1. New elements/messages are **added**

2. Existing elements/messages are **changed**

3. Existing elements/messages are **removed**

Regarding model entities of related design dimensions that have an indirect impact on the pointcut stability are:

4. Existing elements are **redeployed**

Besides the base model elements that might change over time, there is a chance that the strategy of how a particular tactic should be applied to the system might change as well.

5. **Strategy** of tactic application is **changed**.

For reacting to these evolution scenarios, we define a set of guidelines that support the designer in adjusting the respective pointcut definition.

### 5.5.2 Pointcut Evolution Guidelines

The goal of pointcut redefinition is to have an optimal level of complexity in terms of the elements and operators used in the pointcut specification itself. We can formulate heuristics for revising pointcuts in order to keep the definitions' complexity at an optimal minimum level during model evolution or construction. The general heuristic that we can use for revising the pointcut can be formulated as follows: Always capture as many elements as possible by applying higher-level constructs that aggregate instances of that particular type. If messages are captured unintentionally by a higher-level construct, exclude them explicitly at the message level as long as the instance-based exclusions do not exceed the messages captured by the type expression.

Here, we give a list of concrete strategies for optimizing pointcuts regarding the number of elements and operators.

- If several messages are of the same type, then we aggregate these messages by using the type-designator, e.g., jp!type(message_type).

- If several messages have similarities in their names, we use the wildcard designator "*", e.g. jp!instance(M_name*).

- If several messages originate from or target the same component, we aggregate these messages using the source designator, e.g., source(comp!{instance, type}(M_source)). This applies to higher-level elements as well (artifacts and nodes), e.g., target(node!{instance, type}(Node)).

- If several messages are sent through the same communication path, we use the cp-designator, e.g., all(cp!{instance, type}(CP)).

- If several messages can be aggregated on the attribute level, we use respective attribute designators, e.g., source(component!type(P::ST).attrib > 0).

In the context of the change scenarios described above, we need to consider such kinds of optimization goals concurrently with the pointcut adaptations. In the following, we will refer to the strategies shown in [KRK09] that address the revision of pointcuts in the context of change scenarios 1-6.

**Scenario 1.1*:* A new message M is added to the base model**

Pointcut Error: Unintended join point capture

In case the message needs to be removed from the join point set, we apply the general strategy of excluding types that ultimately refer to the respective instances to be included. That is, in case the message is sent from a component deployed on a node that is not involved in sending any other messages captured by the pointcut, we can simply refer to the node element for exclusion of the message. We apply this scheme along the aggregation hierarchy of the components in the deployment specification. We do also consider message types as well. In case the message type is not used in the current pointcut specification, we can simply exclude the message type from the pointcut expression using the expression "&! jp!type(M_type)". In case the type is already used by the pointcut, we exclude the respective message on an instance basis using the expression "&! jp!instance(M)". If we assume or know at this point that all existing and possible future messages with this type will be rather excluded from the pointcut, the obvious choice is to rule the type out. If the M is rather an exception, then excluding the instance directly is the right path.

Pointcut Error: Accidental join point miss

In case of accidental join point miss, we also have the possibility to add the instance directly ("| jp!instance(M)") or to add the type of the message ("|jp!type(M)"). Using the type-based specification, we need to check for possible resulting unintended join point captures. In case we find additional messages revealing similarities, we aim at aggregating them using logical operators as defined in Section 4.3.4.1.

**Scenario 1.2: A new component C is added**

In a sense, this scenario is similar to scenario 1.1, except that we possibly add several messages with one component. Then, all the messages are aggregated by C and we can use them in the pointcut directly.

Pointcut Error: Unintended join point capture

In case the current pointcut unintentionally captures messages introduced by the new component, we can exclude the messages directly by means of the designator jp or use the exclusion operator on the component, e.g. &!source(comp!{type, instance}(C).

Pointcut Error: Accidental join point miss

In case we accidentally miss the messages introduced by the component, we can include the messages by including the component type if it is not already used as an inclusion criterion within the pointcut.

**Scenario 2.1: The name of a message is changed**

In both cases, the unintended join point capture and the accidental join point miss, the pointcut definitions have to be adjusted. If the pointcut just refers to a deployment element only (artifacts, components, nodes), the pointcut remains the same. If the pointcut already uses message names, that pointcut definition would have to be revised.

**Scenario 2.2: The type of message is changed**

In general, a change of message types only affects a pointcut if the pointcut already uses the respective message type. Pointcuts remain unaffected if they refer to deployment elements only.

Pointcut Error: Unintended join point capture

If the new message is captured by the current pointcut definition, we can exclude the message according to the strategy described in scenario 1.1.

Pointcut Error: Accidental join point miss

If we do not capture the message anymore with the unchanged pointcut, we can handle the problem by treating the missed message as a new message according to the strategy described in scenario 1.1.

**Scenario 2.3: The name of component C is changed**

Here, the pointcut is affected if instance-based declarations of the respective component are used.

Pointcut Error: Unintended join point capture

In the event the component name is changed so that it is now matched by a name pattern specified by the pointcut, the name pattern needs to be adjusted. An alternative is to explicitly exclude the component instance using "&!{source, target}!instance(C)".

Pointcut Error: Accidental join point miss

If the new name is not captured by the respective declaration in the pointcut, the name pattern needs to be adapted. Alternatively, the component can be included by referring directly to the instance.

**Scenario 2.4: The type of component C is changed**

In this case, if the poincut uses instances, name-based or deployment related elements only, the pointcut is not affected and might remain unaltered.

Pointcut Error: Unintended join point capture

In this case, the poincut portion referring to the component type that C is changed to needs to be adapted. Alternatively, the component is directly excluded by referring to the instance, or we find an appropriate way of uniquely referring to all messages related to that component.

Pointcut Error: Accidental join point miss

In case of accidental miss, the component can be included back by extending the pointcut with the type of C or the component instance, or add its message or message types.

**Scenario 3.1: An existing message M is removed**

Removal of a particular message potentially reduces the complexity of the pointcut, even if the message was not explicitly excluded or included. In case the pointcut refers to the message by instance, the message has to be removed from the pointcut.

**Scenario 3.2: Component C is removed**

Removal of a component implies that all portions of the pointcut that are referring to the component instance need to be adjusted appropriately. All other portions of the pointcut are potentially unaffected.

**Scenario 4: Component C is redeployed**

If we do not refer to any deployment element (node, artifact, component, communication path), the pointcut remains the same.

Pointcut Error: Unintended join point capture

In case of unintended capture of join points, we can exclude the component as in scenario 1.2. We also refine the existing pointcut by explicitly stating what components to exclude, e.g., source (comp!instance(C_1)) | source(comp!instance(C_2))).

Pointcut Error: Accidental join point miss

Here we apply the strategy described in scenario 1.2.

**Scenario 5: Strategy of tactic application is changed**

If the strategy of the tactic application changes, all pointcut definitions related to that tactic are potentially subject to change. There is no general guideline that supports a revision of tactic application strategies.

We implemented the strategies for these scenarios (except Scenario 5) via the pointcut generation algorithm. That is, based on a selection of messages, the pointcut is generated automatically and reflects the solutions as presented in this section.

### 5.5.3 Reflective Pointcut Expressions

The idea of reflective pointcut expressions leverages the fact that we have a formalized model for selecting join points. That is, a join point set is always comprised of a number of messages exchanged by particular components, coordinated by respective connectors. The pointcut specification approach as described in Section 4.3.4 is interactive and works on the architectural meta-model, exploiting the fact that deployment relationships can be used for navigating to concrete messages.

Given these facts, the reflective pointcut expression works the following way: If a pointcut expression has been defined for a particular base model and the base model is about to change, then the current set of messages captured by the pointcut expression is cached. This join point set is the reference set for the changed base model. If the base model is modified, then the pointcut fragility phenomenon as described in Section 4.3.4 is about to occur.

Here, we have to examine two cases: Unintended join point capture and accidental join point miss. In case of accidental join point miss, we can automatically refactor the pointcut expression by examining the architectural model regarding navigation possibilities to the respective set of messages. In case of unintended join point capture, we can display the newly captured messages to the architect, and based on the decision to include or exclude, the pointcut expression is automatically adapted.

# 6 Tool Support

*"Any sufficiently advanced technology is indistinguishable from magic." Arthur C. Clarke*

In this chapter, we show the tool support we provide in the context of this thesis. The tooling issue regarding model-based architecture design is important, since the concepts explained in the previous chapters would not be realizable in practice without appropriate tool support.

First, we give an overview of the steps of the design process that are automated or tool-supported. Then we illustrate for all of these steps what the tool interface looks like, how the tool can be used, and how the results can be exploited by an architect.

## 6.1 Automated Steps

Here, we show which steps in the design process are manual steps, and which steps are tool-supported and automated.

The modeling of the base connectors and the tactics is tool-supported; however, these steps are manual steps. Automation kicks in when the composition is to be defined. That is, once the tactics have been selected, the composition can be defined. The composition definition step is semi-automatic in two ways:

1. Selection of join points

2. Generation of a valid pointcut expression.

Once the composition is defined, the tool automatically checks for interactions as described in Section 5.3.1. That is, the tool automatically generates an FSP representation of the connector models accompanied by appropriate fluents and assertions that can be model-checked.

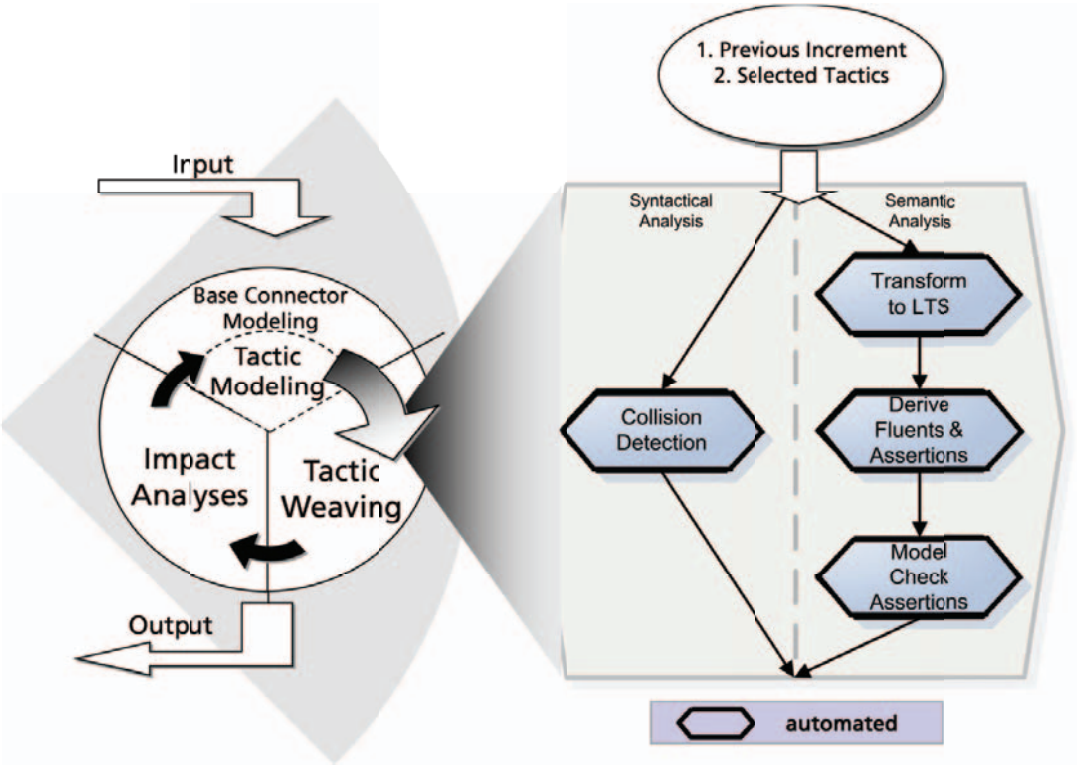



Figure 65 Automated steps in the design process

If a composition specification is to be realized, the weaving algorithms as defined in Section 4.3.5 generate the resulting models automatically. Here, we stress the fact that existing approaches lack appropriate tool support for weaving at the modeling level [SSK06]. According to the survey conducted by the authors of [SSK06], there is only one approach that supports weaving at the modeling level [FRG+04]. However, the authors of [FRG+04] only consider structural diagrams; they do not provide any support for weaving behavioral diagrams such as sequence diagrams or state charts. All other approaches defer the actual weaving to the programming level.

Based on the generated models, the impact analysis is conducted. Here, we use model-based simulations as a way of analyzing the change of design decisions in terms of connectors. The simulation engine itself takes a particular simulation format as input. This input is generated automatically from the architectural model and is comprised of connector definitions, component structures, and deployment descriptions.

The automation of several steps as described above has been realized as plugin extensions to the commercial modeling tool by IBM, Rational Software Architect (RSA) [IBM10]. The design of sequence diagrams is a standard functionality; however, for synthesizing tactics and for interaction detection, we extended the RSA with extra plug-ins designed for exactly these purposes. The synthesis is accomplished by a sequence diagram weaver. Based on a so-called "weaving configuration", we can set up the order of tactics as well as the pointcuts denoting where the tac-

tics are supposed to go. It turned out that it is useful to be able to save weaving specifications for later reuse or modifications. The configuration can be stored and replayed, so we can quickly iterate over a set of different design decisions.

In Table 3, we show some facts regarding the tool support itself in order to provide an idea of the magnitude of extensions that were necessary to realize the design process as specified.

| Tool Characteristics | |
|---|---|
| **# (distinct) Plugins** | 3 Plugins (Interaction-Detection, Weaving, Simulation) |
| **Numer of statements (all plugins)** | 16.000 |
| **Number of classes** | 292 |
| **Total development effort** | 2.8 person years |

Table 3 Tool characteristics

In the subsequent sections, we will show some details regarding the tool extensions for the supported phases. First, we show the steps supported by the tool in terms of model composition, entailing interactive join point selection, pointcut generation, model checking, and model weaving. Then, we show an integration of a simulation engine supporting the impact analysis phase.

## 6.2 Composition Process

As already mentioned, the model composition entails a set of activities that are supported by the tool extensions. The first step is to look into the process of how the pointcuts can be created by selecting appropriate join points from the base models.

### 6.2.1 Interactive Join Point Selection

The interactive selection of join points is motivated by the fact that the pointcut expressions based on the pointcut definition language as shown

in 4.3.4 tend to get quite complex, even for simple pointcut expressions [KRK10]. The main idea behind the interactive definition was to provide a means that intuitively guides the designer in selecting the join points directly on the model, and defer the pointcut construction itself to an automatic generator that is capable of transforming a selected set of model elements into a valid pointcut expression according to the PDL.

The join point selection itself is session-based. That is, we collect join points only for a single tactic at a time. The session for collecting join points is triggered by selecting the respective tactic and starting the visual collection session.

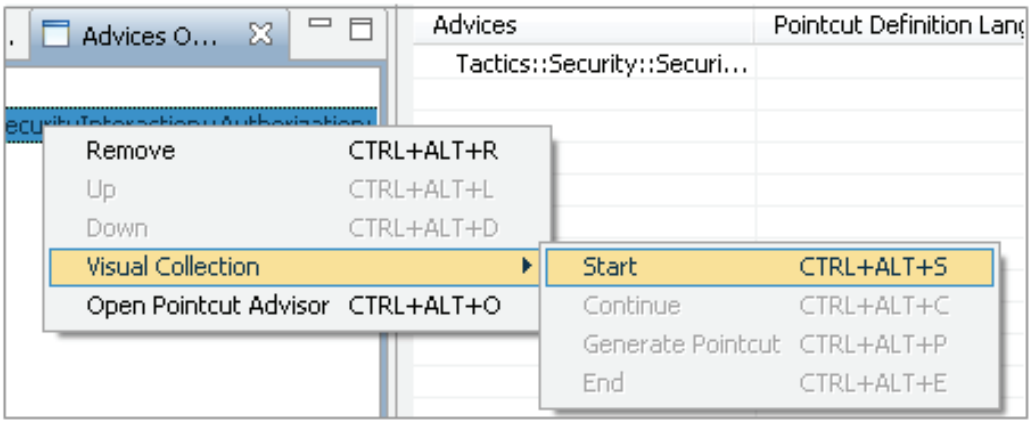



Figure 66 Starting an interactive join point collection session

The join point selection implements the semantics of the type-based and deployment-based join points as defined in Sections 4.3.4.1 and 4.3.4.2. That is, based on the elements selected in the diagrams, we are eventually able to navigate to messages. As shown in Figure 67, by simply selecting the node, we can capture all messages related to the component *Client*, since this component is deployed on the selected node *MobilePhone*. The effect of selecting a particular element is shown in Figure 67. Here we selected component C1 in the diagram. On the left side of Figure 67, we see a number of elements in a tree structure. That tree structure reflects the containment hierarchy as described in Section 4.3.4.2. That is, we analyze the type of the component (upper part of the tree) and show all instances of the component type, including related messages that exist in the architectural model. In the lower part of the tree, we show the particular component instance and the messages related to that component. When elements are selected within the tree structure, the affected elements are marked in all the diagrams. The affected components are marked in the respective deployment diagram (as depicted in Figure 67).

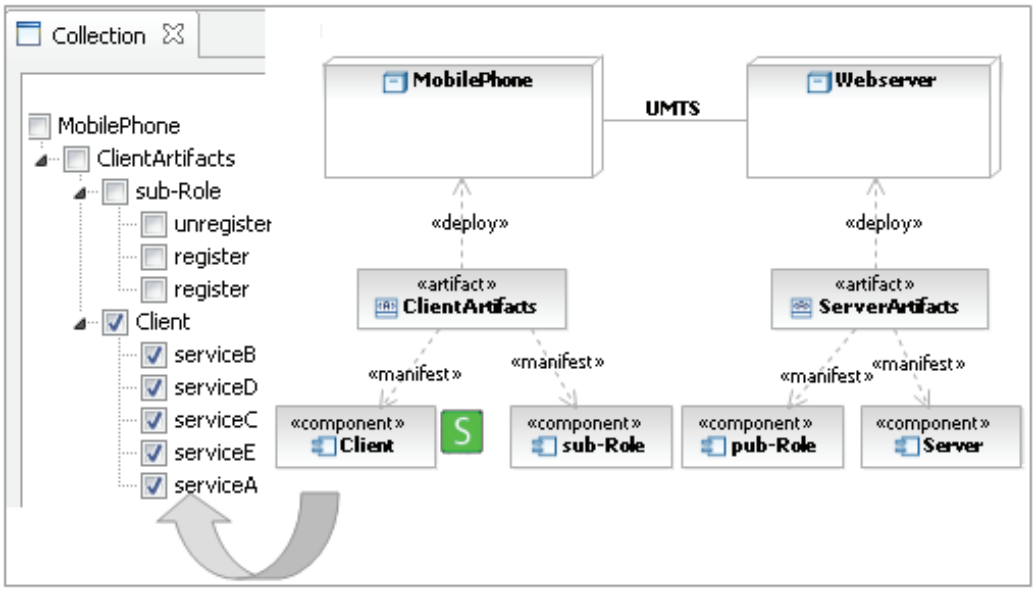



Figure 67 Displaying messages related to a selected architectural element

A selection of the component Client results in a change of the appearance of a sequence diagram containing the respective message as well. An example is shown in the following illustration:

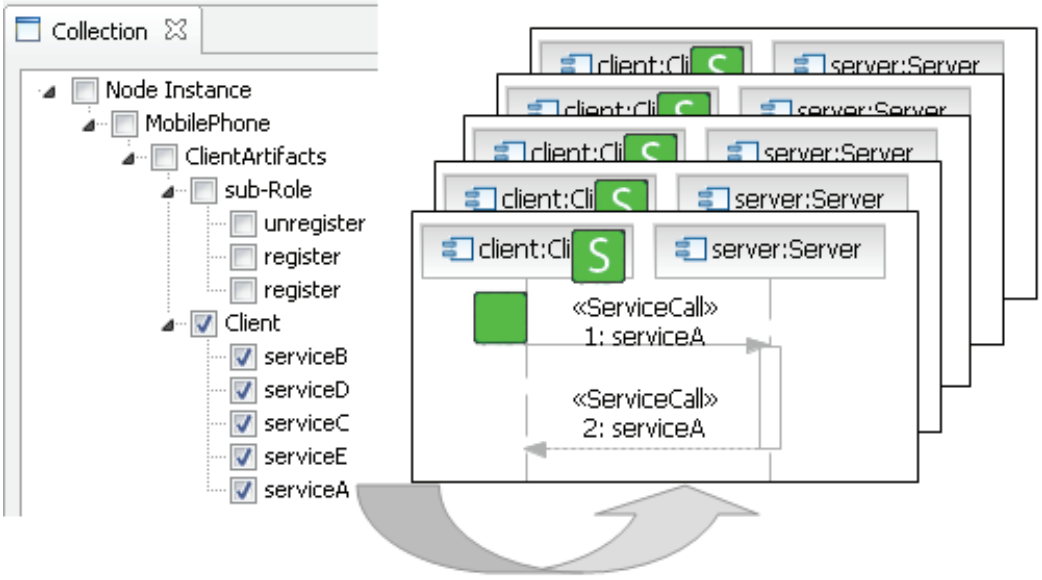



Figure 68 Example selection of messages affecting the diagrams containing the messages

### 6.2.2 Pointcut Generation

Pointcut generation is straightforward since we map the selected join points to the PDL as defined in Section 4.3.4. We can activate the generation of a pointcut for a join point set by selecting the respective option during a collection session.

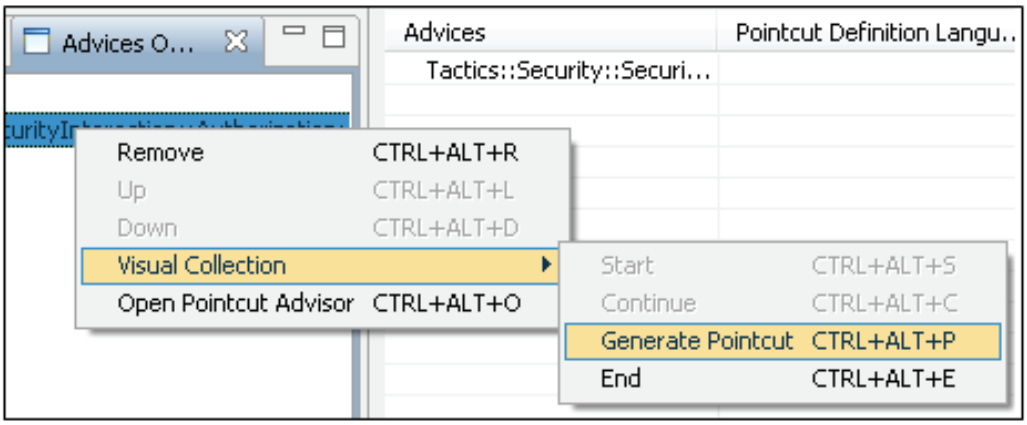



Figure 69 Generating pointcut expressions during the join point collection session

### 6.2.3 Model Checking

Model checking as described in Section 5.4.3 leverages the heuristics for deriving fluents and assertions from a given weaving specification. Consequently, model checking comprises a two-step process:

1. Derivation of fluents and assertions

2. Generation of an FSP based on the sequences defined in the model

We use the Eclipse-based version of the LTSA for actually checking the properties; however, the semi-automatic specification of the FSP and the respective fluents and assertions is one of the contributions of this thesis.

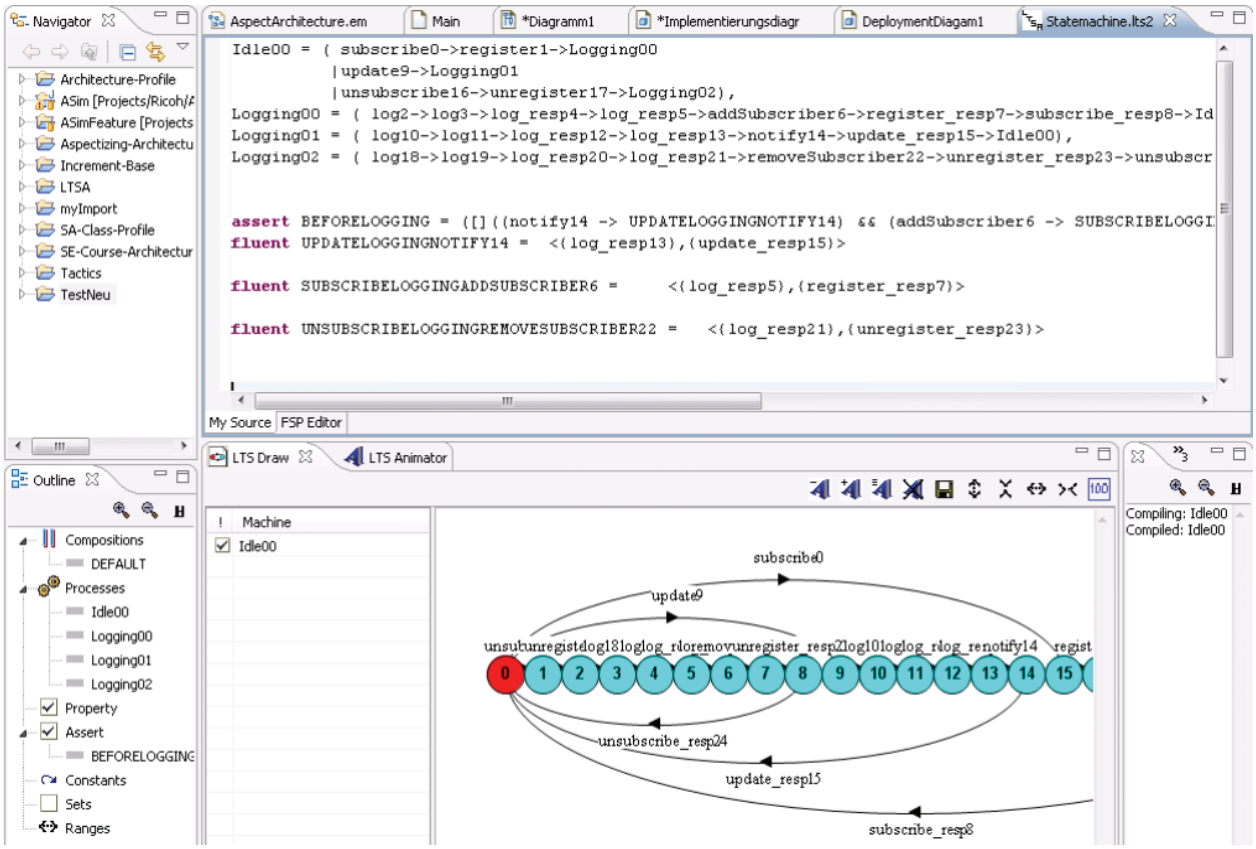



Figure 70 Integrated model checking capabilities

## 6.2.4 Model Weaving

The weaving is implemented according to the algorithm specified in Section 4.3.5. That is, given a weaving specification comprised of a set of tactics, pointcuts, and a base model, the algorithm produces a woven model that manifests all tactics according to the pointcut definition.

We are able to generate the resulting model into a new model (the so-called target model), so the original base model is not polluted with the tactics. If we want to compare design alternatives, we simply change the weaving specification and generate a new model. Then we are able to compare the properties of a set of given design alternatives.

The dialog for specifying the weaving configuration is depicted in the following figure.

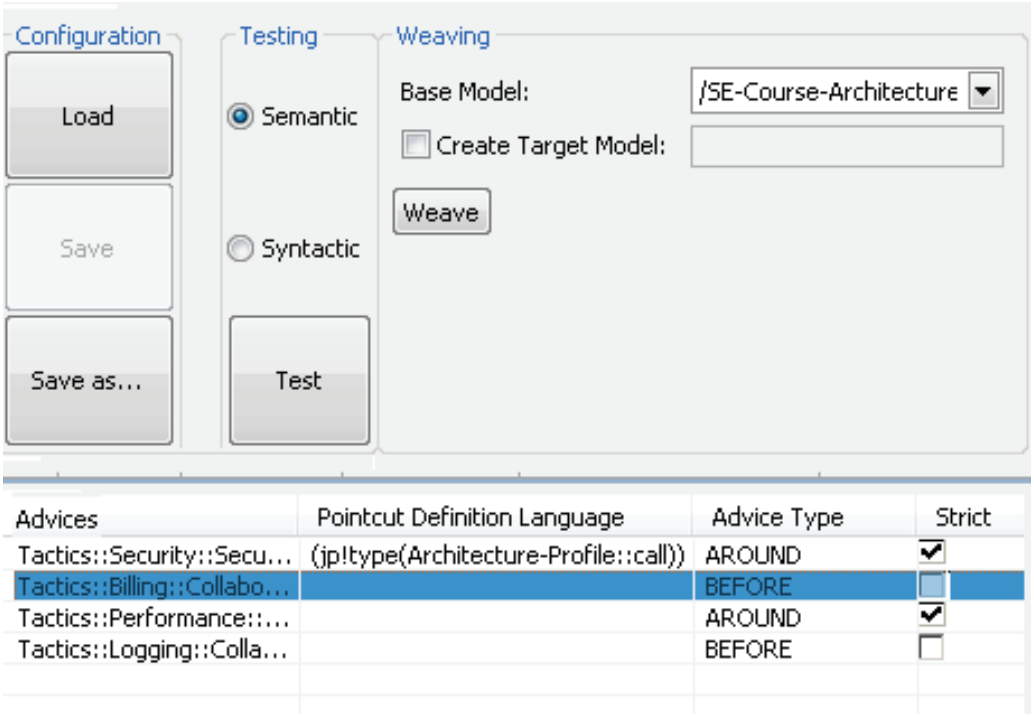



Figure 71 Weaving specification dialog

The dialog shows the tactics that are to be woven into the base model. The base model needs to be selected in the first place, otherwise there would be no possibility of selecting respective join points for the weaving. Using the "Save as …" option, we can save the current weaving configuration for future reuse or change. Clicking on the "Save" button gives us a convenient mechanism for quickly overwriting the selected weaving specification file with the current one. We also integrated model checking into the weaving specification. We can simply click on semantic or syntactical testing, and the respective models are generated for LTSA-based model checking. When the "weave" button is pushed, the specification is realized by the model weaver.

## 6.3 Simulation

The simulation is one technique that can be used for impact analyses. In this case, since we are creating models that describe runtime interactions of components, a simulation approach seemed to be promising, since we would be able to evaluate a resulting design model objectively given a set of metrics that can be used for comparing different design alternatives.

For simulation itself, we use the Java-based simulation framework "DesmoJ" [Des10]. Similar to the model checking approach, we utilize the simulation plugin as is; however, we provide a generative approach

for compiling respective models that can be simulated based on the architectural descriptions.

The simulation setup as shown in Figure 72 comprises information about the sequences to be considered for simulation. (By default, these are all sequences involved in the particular deployment.)

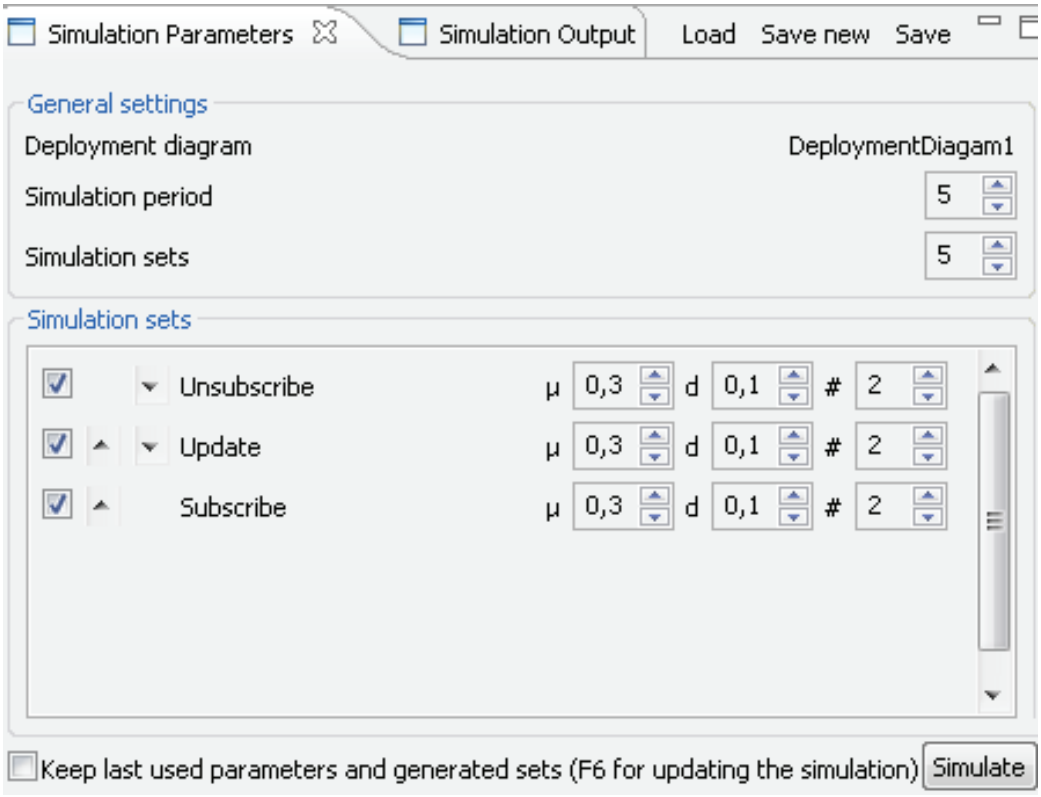



Figure 72 Simulation setup

Then we can specify the number of distinct simulation runs, given a statistical distribution of message occurrences denoted by μ and a standard deviation.

As the result of a simulation, we obtain a graphical depiction of each simulation run as shown in Figure 73. The thick black line is the average calculated over all simulation runs. We can also trace back from the respective message to the model (see tool tip) by simply double-clicking the graph. If we want to know why there is a peak at a particular point in time (see Figure 73), this feature helps us to understand the consequences of design decisions in terms of inter-component communication models.

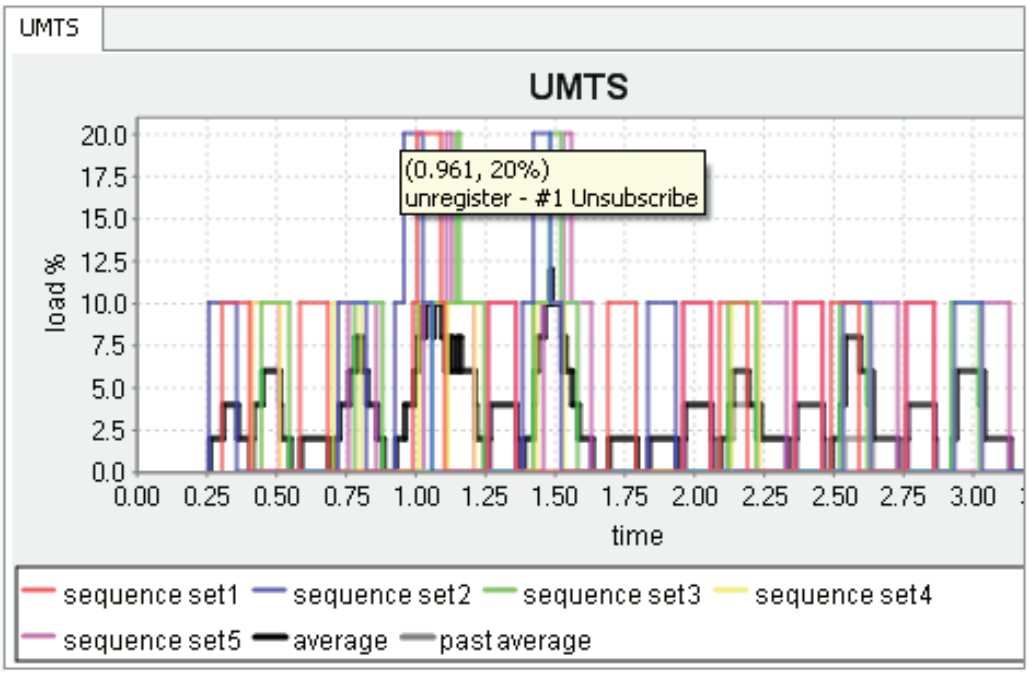



Figure 73 Sample output for a simulation run

Another feature of model simulation is incremental simulation. That is, I managed to realize a concept that allows the architect to change the design, while the simulation updates the simulation results at the same time. Thus, the designer is always aware of the impact of change, and the feedback for design changes is almost immediate.

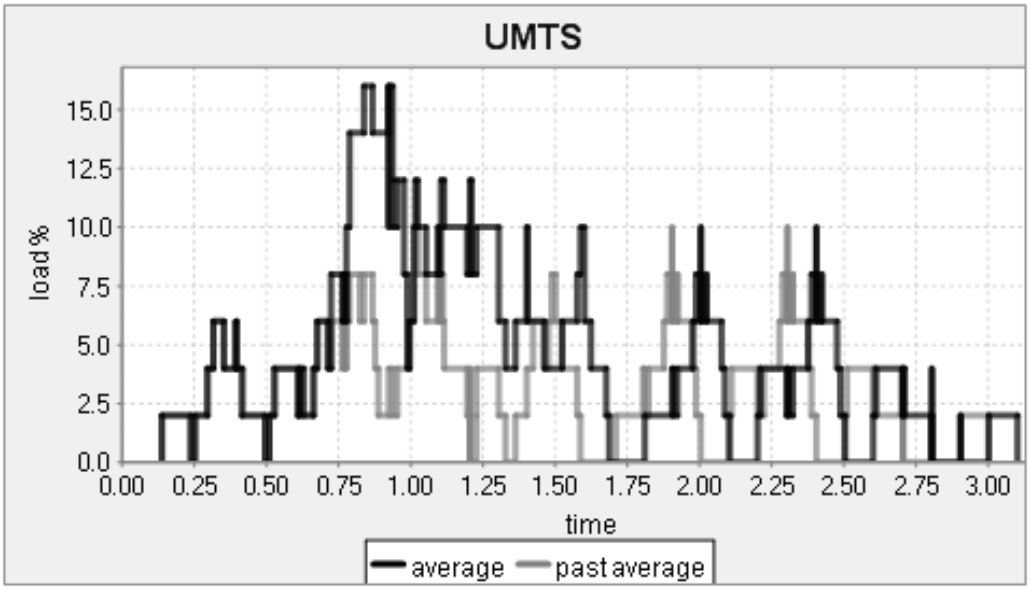



Figure 74 Showing the differences between the current simulation and the previous ones

# 7 Validation

*"If the facts don't fit the theory, change the facts."*
*Albert Einstein*

## 7.1 Overview

The goal of the approach as defined in Chapter 4 is to realize separation of concerns at the architectural level using aspect-oriented concepts. The effect of separating cross-cutting concerns from core design artifacts is reduced effort for performing changes on architectural models. To that end, the hypotheses at the problem level, that is, hypotheses stating cause-effect relations regarding scalability issues in architecture design in practice, are mapped to solution-level hypotheses. On the solution level, we assume that aspect-oriented modeling is effective in realizing separation of concerns in such a way that scalability issues at the problem level are positively impacted.

In this chapter, we show the results of a controlled experiment conducted in the context of a practical course held at the University of Kaiserslautern in the summer term of 2009. Since the technical goal is to separate the solutions for non-functional requirements from design artifacts, we refined the solution hypothesis as defined in Chapter 1 accordingly.

The experiment aimed at assessing the impact of the aspect-oriented architectural approach on the efficiency of model creation, analysis, and evolution, as described in Chapter 4.

The experiment was set up in such a way that two groups performed the same tasks on different architectural models. However, the architectural models differed only in terms of the level of separation. In other words, one group worked on an architectural model that realized certain design decisions in an integrated fashion, whereas the other group worked on an architectural model structured according to the aspect-oriented metamodel as described in Chapter 3.

The experiment revealed that the tasks that had to be performed significantly differed in terms of effort used. Besides, the correctness of the resulting models differed as well from one group to the other.

A second experiment was conducted for checking the effectiveness of model checking. For this purpose, we designed a reference model in order to assess the performance of the heuristics for generating fluents and assertions. We compared the number of interactions detected by the automatically generated assertions and fluents with the reference solution. Basically, we checked if our approach returned as many interactions as the manual model checking approach. In addition, we checked for false positives, that is, we checked if the fully automated approach produces results that are not valid interactions.

## 7.2 Hypotheses

As defined in Chapter 1, the solution-level hypothesis is defined as:

**$H_S$**: Aspect-oriented concepts can be used for efficiently manifesting and changing design decisions in the realm of inter-component communications.

The solution hypothesis expresses that the design activities based on connector models are significantly improved in terms of efficiency through aspect-oriented separation of concerns. On the other hand, the hypothesis implicitly postulates that the aspect-oriented separation of concerns yields architectural models that are at least as correct as those built using traditional approaches, that is, approaches that do not apply advanced principles of separation of concerns.

Next, the terms correctness and efficiency are defined in the context of my solution hypothesis.

**Definition: Correctness**

Correctness is defined as the degree to which an architectural model conforms to fact or specification.

In other words, an architectural model is correct if there are no faults or errors in the model regarding its specification. In the context of connector design, an error could be a wrong sequence of messages, missing messages, or too many messages at a particular place in the model.

**Definition: Efficiency**

Efficiency is defined as the number of architectural elements that can be processed per time unit. Processing of architectural elements either refers to identification, modification, or validation.

Given this argumentation, we refine the hypothesis according to the following sub-hypotheses.

**$H_{S1}$**: **Creating** architectural models using aspect-oriented concepts is significantly more efficient than standard (integrated) models.

This hypothesis expresses the expectation that significant performance increases will be achieved during model creation. The comparison is relative to traditional approaches that do not explicitly separate cross-cutting concerns from architectural elements. The improvement factor depends on the number of artifacts to be considered during modeling.

**$H_{S2}$**: **Creating** architectural models using aspect-oriented concepts is generally at least as correct as doing so with standard (integrated) models.

Besides efficiency, the proposed approach should yield models that are correct. Therefore, the hypothesis states that the resulting architectural models are at least as correct as in the case of traditional approaches.

**$H_{S3}$**: **Changing** Aspect-oriented model-based architectures is, on average, at least as correct as with standard (integrated) models.

This hypothesis manifests the activity of model changes as mentioned in the solution hypothesis **$H_{S.}$**. Here we refine a model change into two distinct steps that are observable in isolation from one another: identification of the model elements to be changed, and the change of the model elements themselves.

**$H_{S4}$**: **Changes** to the aspect-oriented models are, on average, significantly more efficient than changes to standard (integrated) models.

## 7.3 Controlled Experiment

The goal of the controlled experiment was to investigate the validity of the solution hypotheses **$H_{S3}$** and **$H_{S4}$** as defined in the previous section. Concerning hypothesis **$H_{S1}$**, we refer to the tool-supported generation of architectural models in terms of connectors. That is, since architectural model creation is simply a matter of generating connector designs from given composition specifications, the increase in the proclaimed efficiency is obvious. On the other hand, **$H_{S2}$** remains to be checked for validity. However, checking the correctness of the generated resulting models is not subject of the experiment presented, since such a check can be performed by applying model checking techniques that do not require any experimental setup.

Hence, regarding the solution hypotheses, the focus of the experiment presented in this chapter is on the efficiency as well as on the effectiveness (or correctness) of **changing** architectural connector designs.

In the next section, the design of the experiment will be explained in more detail.

### 7.3.1 Experiment Design

The experiment design entailed "experience" as an independent variable, and the respective modeling approaches including distinct example models as factors. The groups followed a 2x2 factorial design [WRH00], and the tasks performed by one group on one example model were switched for the respective control group. We compared the aspect-oriented modeling approach as defined in Chapter 5 with an approach that does not explicitly separate solutions to cross-cutting concerns from architectural base models. The dependent variables in this experiment are correctness and efficiency.

The experiment design is reflected in the following statement:

Analyze a model-based architecture design

- for the **purpose** of evaluating the impact of AO modeling
- with respect to the **correctness and efficiency** of changing architectural models
- from the point of view of the architect
- in the context of an experiment with M.Sc. and B.Sc. students

The experiment was designed in such a way that a number of subjects were supposed to perform a number of activities on a prepared architectural design.

### 7.3.2 Experiment Execution

The experiment was conducted in the context of a practical course at the University of Kaiserslautern. In terms of preparation, the subjects received training on the method during an exercise and, in addition, all subjects got a set of guidelines describing the process of aspect-oriented modeling.

The initial group assignment was random, based on the sequence in which the subjects entered the room for experimentation. Regarding the environ-

ment in which the experiment was conducted, every student used IBM Rational Software Architect [IBM10] as the modeling tool.

The time frame for the experiment was basically not strictly limited. However, the target time for performing all defined tasks was about two hours. During this time, all subjects had to perform tasks on the given models that could be measured in terms of time and model properties. Here, we give a sample task description representing a typical task as defined by the experiment.

> "Change the Logging-Tactic in such a way that a second log-role is introduced that is triggered in addition to the already existing log-role (redundant logging). Apply this change to **all** sequence diagrams of Increment1A!"

Each group executed two runs of similar tasks over different examples. After a subject finished all tasks belonging to a group, a new workspace was loaded with the prepared models.

Data was collected using questionnaires for pre-briefing and de-briefing the subjects. In addition, every subject was supposed to note the exact time when a particular task started and finished. In order to check the quality of the resulting task performances, the changed models were stored and inspected afterwards.

### 7.3.3 Analysis and Interpretation

In this section, we will evaluate the data collected during the experiment.

For evaluating how efficiently the tasks were performed, we used the data collected in terms of time stamps. For evaluating correctness, we compared the answers given by the participants with a reference solution. If the participants had to change the architectural model at hand, we checked the changed architectural models for correctness as defined in Section 7.2.

For testing the hypotheses stated in Section 7.2, we applied the Wilcoxon-Ranksum test [MW47].

First, we analyzed the efficiency achieved by the subjects in terms of model changes. As defined earlier, we partitioned "change" into two distinct steps, namely, identification and modification.

### Efficiency of Identification

The null-hypothesis for testing the efficiency of identifying model elements is defined as:

**$H_{0\ \text{Change efficiency}}$**: The identification of model elements in AO models is, on average, as efficient as in integrated models.

The alternative hypothesis can be formulated as:

**$H_{1\ \text{Change efficiency}}$**: The identification of model elements in AO models is, on average, significantly more efficient.

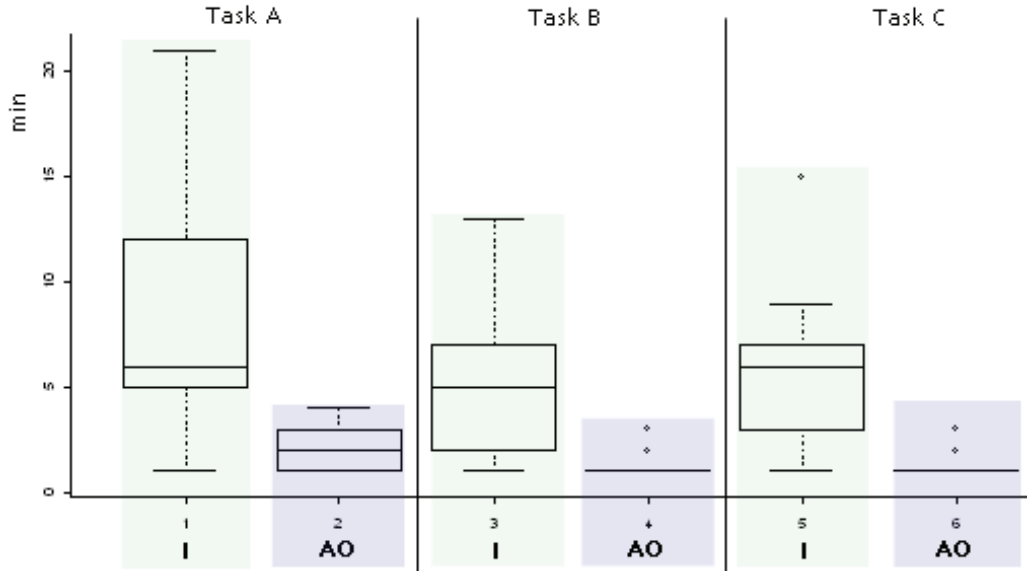



Figure 75 Efficiency of identification

The results of the identification tasks are shown in Figure 75. As can be seen, all tasks could be performed significantly faster with aspect-oriented design models than with integrated architectural models. On average, all identification tasks performed on the aspect-oriented designs were at least three times as efficient as those performed in the control group.

### Efficiency of Modification

The null-hypothesis for testing the efficiency of modifying AO models is defined as:

**$H_{0\ \text{Change efficiency}}$**: The **modification** of AO models is, on average, as efficient as in integrated models.

The alternative hypothesis can be formulated as:

**$H_{1\ \text{Change efficiency}}$**: The **modification** of AO models is, on average, significantly more efficient.

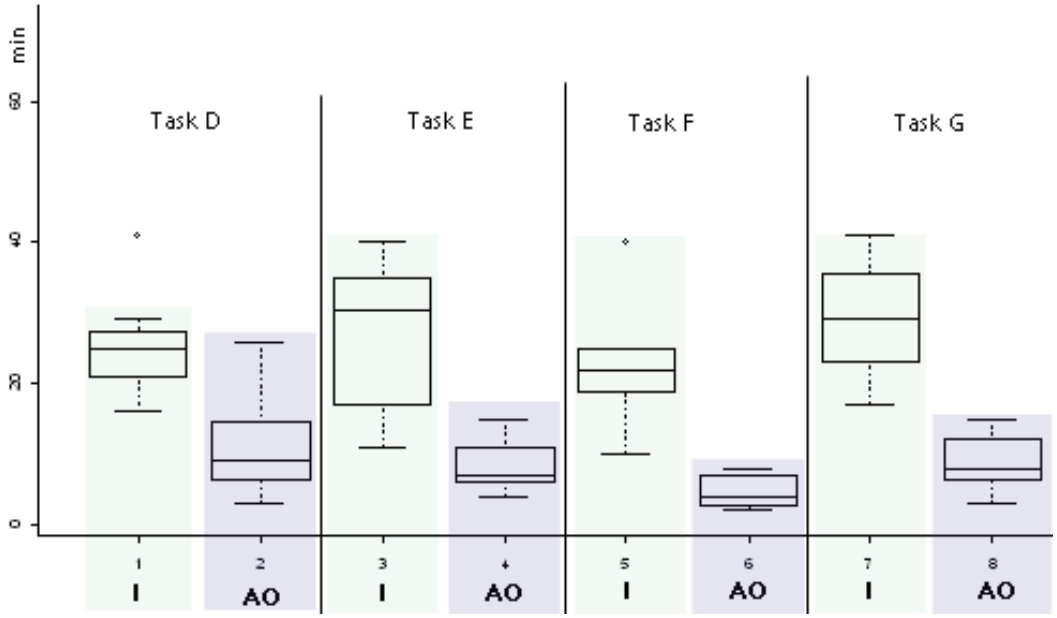



Figure 76 Efficiency of modification

In case of modification tasks, we again see a significant performance increase in the case of aspect-oriented architectural models. All tasks that required architectural models to be changed, showed the aspect-oriented design approach to be more efficient by a factor of three on average.

Table 4 shows the concrete results of the tasks performed for testing the efficiency increase. Looking at the p-values of the respective Wilcoxon-rank sum tests, all stated null-hypotheses can be rejected, which leads to the acceptance of the alternative hypotheses.

| | Task | Wilcoxon-rank sum test (p-value) | Accept/Reject [if p<0.05] |
|---|---|---|---|
| **Identification** | Task A (I) vs Task A (AO) | 0.000084 | Reject H0 |
| | Task B (I) vs Task B (AO) | 0.00047 | Reject H0 |
| | Task C (I) vs Task C (AO) | 0.00016 | Reject H0 |
| **Modification** | Task D (I) vs Task D (AO) | 0.01262 | Reject H0 |
| | Task E (AO) vs Task E (I) | 0.008133 | Reject H0 |
| | Task F (I) vs Task F (AO) | 0.002093 | Reject H0 |
| | Task G (AO) vs Task G (I) | 0.0005828 | Reject H0 |

Table 4 Statistical analysis - efficiency of model changes

Consequently, the experiment has shown that

**Changes** to the aspect-oriented models are, on average, significantly more efficient than changes to standard (integrated) models.

### Correctness of Identification

The null-hypothesis for testing the correctness of identifying model elements is defined as:

$\mathbf{H_{0\ Change\ effectiveness}}$: The identification of model elements in AO models yields, on average, less correct results than model elements identified in integrated models.

The alternative hypothesis can be formulated as:

$\mathbf{H_{0\ Change\ effectiveness}}$: The identification of model elements in AO models yields, on average, results that are at least as correct as elements identified in integrated models.

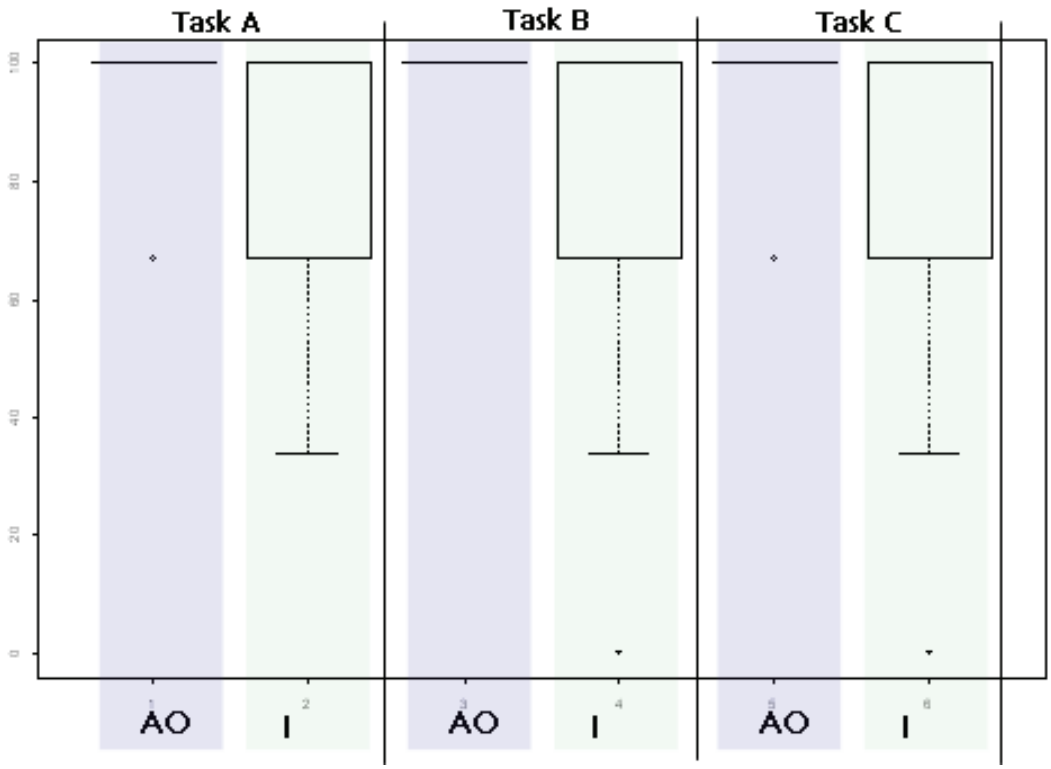



Figure 77 Correctness of identification

The correctness of the results produced by performing the modification tasks revealed differences that were not that obvious as in the case of efficiency. However, as can be seen in Figure 77, on average all results produced based on the aspect-oriented designs were completely correct. (There is one outlier, though.) In the case of the integrated models, more results were less correct than the aspect-oriented designs.

### Correctness of Modification

The null-hypothesis for testing the correctness of modifying AO models is defined as:

| |
|---|
| $\mathbf{H_{0\ Change\ effectiveness}}$: The modification of AO models yields, on average, architectural models that are less correct than if integrated models are used. |

The alternative hypothesis can be formulated as:

| |
|---|
| $\mathbf{H_{0\ Change\ effectiveness}}$: The modification of AO models yields, on average, architectural models that are at least as correct as if integrated models are used. |

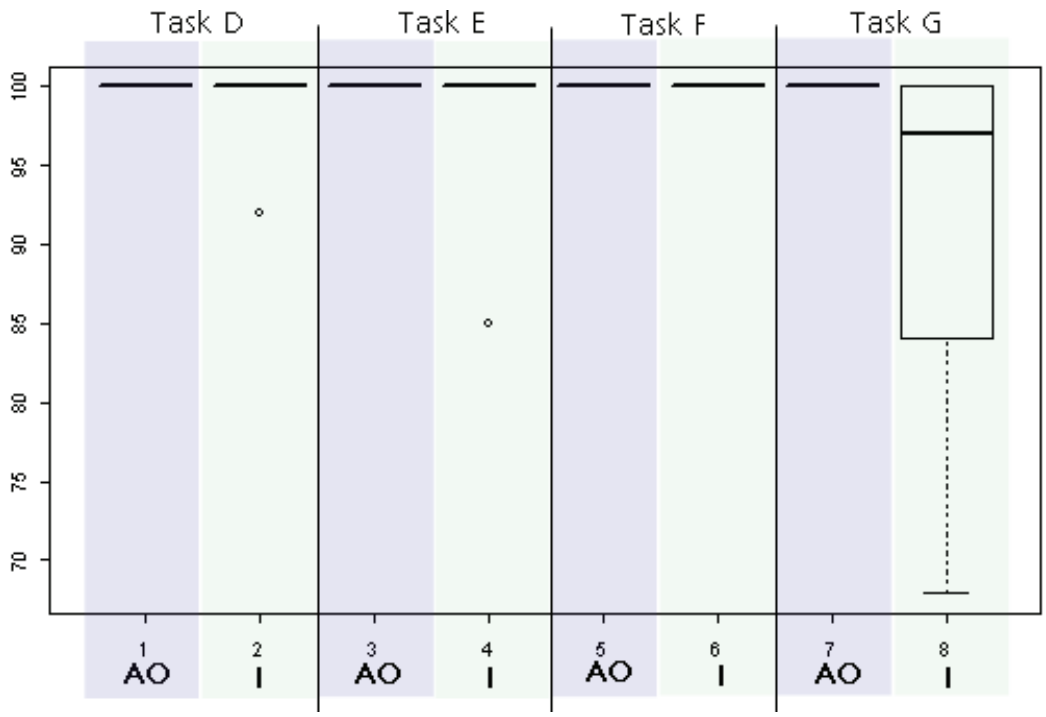



Figure 78 Correctness of modification

The results of the modification tasks show that tasks D, E, and F, are not significantly distinct from one another. Only task G has a remarkable difference in correctness of the results when both approaches were compared. This phenomenon is reflected in the statistical analysis, since in the case of modification tasks, there are no significant differences between the group that performed the tasks on the aspect-oriented designs and the control group. However, regarding correctness of the identification tasks, significant differences were seen in favor of aspect-oriented designs.

| | Task | Wilcoxon-rank sum test (p-value) | Accept/Reject [if p<0.05] |
|---|---|---|---|
| **Identification** | Task A (I) vs Task A (AO) | 0.005247 | Reject H0 |
| | Task B (I) vs Task B (AO) | 0.01749 | Reject H0 |
| | Task C (I) vs Task C (AO) | 0.005275 | Reject H0 |
| **Modification** | Task D (I) vs Task D (AO) | 0.3914 | No Reject of H0 |
| | Task E (AO) vs Task E (I) | 0.3545 | No Reject of H0 |
| | Task F (I) vs Task F (AO) | ->1 | No Reject of H0 |
| | Task G (AO) vs Task G (I) | 0.03072 | Reject H0 |

Table 5 Statistical analysis - correctness of model changes

In sum, the statistical analyses regarding the correctness of the results produced by each group only revealed significant differences in the identification tasks.

Concerning the null-hypothesis as defined for the modification task, we claimed that the aspect-oriented design yields architectural models that are at least as correct as models produced by the control group. Since this null hypothesis cannot be rejected by just showing a non-significant difference in both groups, we do not have statistical evidence that the aspect-oriented approach yields different (more correct) results than those obtained by the group performing the tasks on the integrated models. However, even though statistically there are no significant differences, we see that aspect-oriented design, on average, did not produce results that were less correct than the results produced in the control group.

### 7.3.4 Threats to Validity

According to [WRH00], threats to validity exist in four dimensions: internal, external, construct, and conclusion validity

**Internal validity** is defined as the degree to which conclusions can be drawn regarding the effect of the independent variable (e.g., usefulness and ease of use) on the dependent variables.

As described by the experiment design, we chose two treatments for two factors in order to prevent internal validity risks as far as possible. Half of the participants performed the tasks on the aspect-oriented models first, while the other half applied the standard approach first. For the second run of the experiment, the sequence was reversed. By having two treatments of the method as well as the example systems and a change of groups after the first run, we most likely smoothed out learning effects. This is mainly due to the models being different and the tasks requiring the subject to work with the models in detail.

In terms of maturation, there were indicators that the subjects tended to get bored. In one case we could observe that some answers to the questions were just guessed, based on a pattern that we included. However, the pattern was designed in such a way that we could identify the cases where subjects just guessed answers based on previous results.

In terms of instrumentation, some task descriptions turned out to be ambiguous. Due to this, additional clarification was necessary. In one case, the results were partially useless since the time stamps were applied incorrectly. Moreover, the tasks as used in the experiment most likely do not represent the full spectrum of change tasks in the context of architectural design.

**Construct validity** is defined as the degree to which the variables used accurately reflect the constructs of interest.

The goal of this experiment was to measure the efficiency and correctness of tasks performed on architectural design models. Efficiency can be objectively measured by the time units captured. Correctness can be checked by inspecting the results and comparing them to a reference solution. In addition, we have the possibility to apply formal model checking techniques for analyzing the correctness of the results. However, model checking itself can be regarded as an indirect measure only. The reason is quite obvious: Using model checking, we can check if a design is correct, that is, if a design does not violate any formally specified rules. Model checking cannot be used to check whether a design is “good”.

**Conclusion validity** is defined as the degree to which the results of the research are statistically significant.

For the efficiency hypotheses, we could statistically prove a significant difference between the groups. For effectiveness, the difference between the groups could not be tested statistically though.

**External validity** is defined as the degree to which the results of the research can be generalized.

The experiment population were bachelor and master students of computer science. Since the intended population for applying the aspect-oriented design approach are software architects with real-world industry experience, the population selection constitutes a threat to external validity.

The example models used during the experiment were basically hypothetical toy models. They do not represent a realistic software system's architecture. In other words, the examples provided did not reflect the complexity and scalability issues found in real-world projects.

The timeframe available for the experiment was limited. Since in practice, the timeframe might have an influence on the results produced by either one of the methods, this is considered a threat to external validity.

We used a realistic modeling tool that is applied in practice as well. IBM Rational Software Architect is a commercial tool providing support for architecture design activities for practitioners. In that sense, the experiment tooling infrastructure was realistic.

### 7.3.5 Experiment Conclusions

The main goal of the experiment was to investigate the impact of aspect-oriented architecture design on the ability to efficiently react to change. We defined a set of tasks that represent change tasks to be performed on a given architectural model. Then two groups alternatingly performed the same set of tasks on different models. We expected to gain some insights regarding the effects of aspect-oriented modeling at the architectural level in terms of its effectiveness in realizing separation of concerns.

The main solution-level hypothesis was defined as:

| |
|---|
| $\mathbf{H_S}$: Aspect-oriented concepts can be used for efficiently creating, changing, and evolving design decisions in the realm of component communications. |

The results of the experiment showed that there is a positive effect of the aspect-oriented design approach on the efficiency of changing architectural models.

However, we strive for replicating this experiment under a given set of context factors such as:

- Long-term project

  We aim at conducting a longer-term experiment in the context of a realistic architecture project.

- Realistic architectural model

  Since the architectural models used during the experiment were toy models, we aim at applying the approach to realistic architectural models as they are being developed in industry.

- Realistic change scenarios to be performed on the model

  Besides simplistic example models representing architectural design decisions, we would like to replicate the experiment with realistic change scenarios.

- Population should be professional architects

  The population of the experiment was comprised of students only. Since there might be a significant difference to professional architects, we would like to perform a replication of the experiment with industrial professionals.

- Ideally, there should be two teams that independently develop the same system (not realistic)

  In a perfect case, we would be able to have two distinct teams that would be able to develop the same architecture in parallel. However, since this is not realistic for industry-relevant products, we do not believe that we will get the chance to replicate the experiment under such a condition.

Even though these context factors did not apply to the experiment we conducted in the context of a practical course at the University of Kaiserslautern, we have clear indications of the positive effect of the aspect-oriented design of architectural connectors on the efficiency of model creations as well as the correctness of the resulting architectural models. We believe that the aspect-oriented approach as described in this thesis will bring significant benefit to architecture-centric development in practice.

## 7.4 Composition Validation

For evaluating our model weaving approach in terms of correctness, we designed a reference model in order to assess the performance of the generated heuristics for checking the compositional correctness of the generated models. To that end, we compared the number of interactions detected by the automatically generated assertions and fluents with the reference solution. Basically, we checked if our approach returned as many interactions as the manual model checking approach. In addition, we checked for false positives, that is, we checked whether the fully automated approach produces results that are not valid interactions.

In the following, the tool setting and the models involved in the validation activities are sketched.

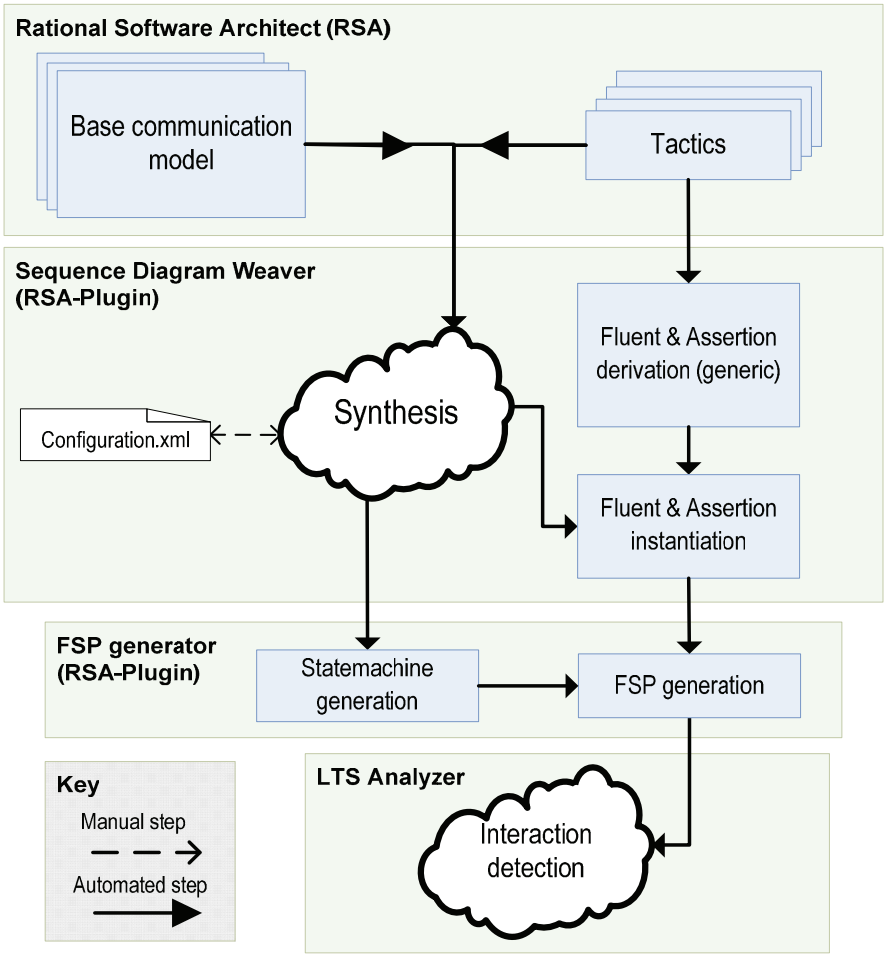



Figure 79 Tool setting for the validation of composition correctness

As explained in Chapter 6, the weaving configuration dialog allows for checking potential problems of the current configuration by testing for syntactical collisions before the actual weaving takes place. After that,

the generic fluents are derived from the weaving specification. They are instantiated after the actual synthesis, since by then we know about the concrete join points. An additional plug-in (FSP generator) is in charge of generating a state machine from the synthesized sequence diagrams. In combination with the fluents and assertions derived during the synthesis step, we are able to generate an FSP representation of the connector. This is taken as input for the LTS analyzer in order to check the semantic properties of the synthesized connector design.

As indicated before, we used a reference connector model to check our approach with respect to completeness and correctness. In order to get an idea of the complexity of the reference model, we used the following metrics:

**#Tactics [#interactionsPerSequence]**:= Describes the number of different tactics applied to the same connector design. Each tactic indicates its own complexity by the number of interactions specified by that particular tactic.

**#BaseSequences [#interactionsPerSequence]**:= Describes the complexity of the base communication model in terms of interactions per sequence specified by that connector.

**#Join points**:**=** Describes the number of places in the base communication model affected by the tactics to be applied.

**#DirectDependencies**:= For definition, see Section 5.4.1.

**#IndirectDependencies**:= For definition, see Section 5.4.1.

Our reference model exhibits the following values with respect to the metrics defined above:

#Tactics: 5 (T1: Billing[2], T2: Authentication[6], T3: Authorization[6], T4: Monitoring[3], T5: Heartbeat[3])

#BaseSequences: 6 (S1: register[2], unregister[2], update[3], privRegister[2], privUnregister[2], privUpdate[3])

#Join points: 12 (for the configuration used)

The complexity indicators for the reference model show that the base communication model is quite simple, since we only have three base sequences with two or three interactions each. Hence, the number of join points in the base communication model is quite low, too. However, the results show that even in a simple connector model, the interaction detection is far from trivial. In total, we generated seven fluents (with up to 12 messages each) and seven assertions. With these fluents and assertions, we were able to detect all semantic conflicts as described in the

reference solution. In the example case, we had no collisions since the two around-advices did not remove any join points used by other tactics.

| | T1 | T2 | T3 | T4 | T5 |
|---|---|---|---|---|---|
| **T1** | | 6 | 6 | 12 | 12 |
| **T2** | 6 | | 6 | 6 | 6 |
| **T3** | 6 | 6 | | 6 | 6 |
| **T4** | 12 | 6 | 6 | | 12 |
| **T5** | 12 | 6 | 6 | 12 | |

Table 6 Dependency matrix for the selected set of tactics

However, since we specified strict assertions for the authentication as well as for the authorization tactics, we encountered two semantic conflicts. This was due to the fact that the weaving sequence was set up in such a way that the monitoring and heartbeat tactics tore the security-related tactics apart. We could resolve this by rearranging the weaving order. However, one semantic conflict still remained (false positive). We had to loosen the strict assertion for authentication, since we had to allow that authorization tactic to be executed between the authentication and the mutually shared join points.

| | Reference | Automated |
|---|---|---|
| **Collisions (Pre-synthesis)** | 0 | 0 |
| **Conflicts (Post-synthesis)** | 2 | 2 |
| **False Positives** | 0 | 1 |
| **Missed** | 0 | 0 |

Table 7 Results of the automatic composition validation

# 8 Summary and Outlook

*"The best way to predict the future is to invent it."*
*Alan Kay*

## 8.1 Results and Contributions

In this chapter, we summarize the main results of the research conducted, describe the limitations and open issues of the solutions presented, and sketch a number of potential next steps for future research efforts.

In short, the main results presented in this thesis comprise a formalized aspect-oriented architectural model that can be effectively and efficiently utilized for architecture design, and mitigate scalability issues as stated in Chapter 1. The aspect-oriented architecture model comes with a method that supports the design process with sophisticated extensions of a commercial architecture design tool. The aspect-oriented architecture model, the design method, and the tool support are fundamental ingredients for efficiently implementing the principle separation of concerns at the level of architectures.

In order to present the specific contributions of this thesis, we partition them into four categories, namely: formalization, methodical, tool-related, and validation.

### 8.1.1 Aspect-oriented Architecture Model

In the literature, there exist solutions that formalize partial aspects of architecture design; however, a conceptualized model interrelating entities that are of importance in the context of the problems motivating this thesis is missing. In order to find solutions to the practical problems as described in the previous sections, we needed to map the practical issues to a conceptualized model that could be taken as a starting point for building a solution. That is, in a first step we needed to build a model that formalizes and interrelates concepts within the realm of architecture design.

**Contribution: Formalization of Architecture**

In terms of formalization, the contribution comprises an architectural meta-model describing elements and relationships relevant for architecture design. The meta-model formalizes existing knowledge and current state-of-the-art perceptions of architectures. To that end, I defined a meta-model relating all facets of architecture that are of relevance based on a consolidation of existing formalizations as described by current state-of-the-art research. The conceptualized model was the starting point for defining specific extensions that address the architectural issues regarding the improvement of separation of cross-cutting concerns.

**Contribution: Utilization of Aspect-orientation at the Architectural Level**

The second step of formalization was concerned with model building at the solution level. Since I aim at leveraging aspect-oriented concepts at the architectural level, I needed to overcome a number of current shortcomings existing in the realm of aspect-oriented modeling in general and aspect-oriented architectures in particular.

**Contribution: Controlled Modification of UML Sequence Diagrams**

Looking at the problem statement regarding cross-cutting concerns at the level of inter-component communication, I found that an appropriate solution needs to provide a means for expressing cross-cutting concerns at the architectural level in the form of behavioral models.

**Contribution: Integration of Cross-Cutting Solutions**

I found that existing aspect-oriented design solutions at the architectural level were realized as extensions to architectural description languages (ADLs) [GCB06]; however, "model compilers" that produce an integrated solution from the aspect-oriented architecture descriptions are missing. In this thesis, I present a solution that is able to compile a number of crosscutting concerns into a consistent architecture description.

**Contribution: Composition Specifications on Type- and Instance Levels**

Existing approaches are only appropriate for working on the type-level of model elements rather than on the instance-level. This shortcoming is relevant for two reasons: First, architectural element types defined after the creation of concrete models cannot be easily propagated to existing instances. This is particularly important in the context of improving scalability of teams by increasing parallelism. Second, the spectrum for defining specific aspects is limited. Either the types are extended or mod-

ified, or a new type needs to be defined that is appropriate for the particular context. The solution presented in this thesis comes with a means for specifying aspects to operate on the type- as well as on the instance-level of the base model elements. We can propagate new or changed element types into the model and achieve a finer-grained spectrum between the worlds of types and instances. In addition, we leverage meta-model relationships as defined in the architectural meta-model in order to be able to provide a powerful means for efficiently selecting huge portions of a model.

**Contribution: Treatment of Compositional Interactions**

Compositional interactions denote the situation when aspects interfere with one another within the base model. Since aspects potentially operate on the same base model elements, there is a high potential that two or more aspects change the model in such a way that the resulting aspectual composition does not reflect the intended model properties. Regarding state of the art we found that there is only limited support for detecting, resolving, or preventing aspect interactions at the modelling level.

### 8.1.2 Design Method

Since the architectural model is formalized, the correctness of generated model compositions can be based on formal model checking approaches such as linear-time temporal logic in combination with labeled transition systems. Concerning the model checking approach, we derived a set of heuristics that can be used for automatically generating formal specifications of properties suitable for being model checked. That is, we formalized the notions of interactions at the modeling level, extending existing work in the area of aspect interaction detection. In addition, we derived algorithms for automatically generating formal property specifications in terms of fluents and assertions that can be used for model checking resulting designs using the Labeled Transition Systems Analyzer.

In general, any formalization of real-world phenomena in terms of a model also requires a process that shows how to use the model for solving real problems at hand. However, as can be seen in the literature, existing aspect-oriented modeling approaches usually do not provide methodical guidance.

**Contribution: Methodical Guidance**

In terms of model evolution, aspect-oriented approaches usually suffer from the fact that they need to keep track of changes of the base model in order to keep the composition specification consistent. This problem is

also referred to as *pointcut fragility* [CGB09]. In fact, there are no approaches that provide solutions to that issue at the modeling level.

In terms of methodical contributions, we defined a method that guides the architect through the process of architecture design, focusing on connector design in particular. Since the weaving itself can be considered the bottleneck of the approach's scalability, subsequent challenges needed to be resolved.

**Contribution: Evolution of Aspect-oriented Models**

The method comes with a set of guidelines for tackling problems along the way of designing. That is, we provide guidelines for evolving the model composition specification in the face of change by giving concrete guidance regarding identified change classes. In addition to the guidelines, we provide a reflective model that allows for checking base model changes against pointcut specifications.

### 8.1.3 Tool Support

In terms of tool support, we created a set of plugins that realize a number of design steps that are automated and necessary for applying the approach in realistic settings. Basically, we automated all formalized steps as defined in the aspect-oriented architecture model. In particular, we support the composition specification, comprised of the pointcut definition, the actual implementation of the weaving algorithms, the generation of fluents and assertions for model checking, and the generation of a simulation model. The pointcut definition steps are supported by a graphical, interactive selection of join points within the designed connector models, as well as by the automatic generation of pointcut expressions according to the pointcut definition language defined in Section 4.3.4.

**Contribution: Using the UML for Increasing Industry Acceptance**

One important drawback of ADLs is that, in general, they have not found adoption in practice [WH05]. The solution provided by this thesis uses the UML, which can be considered a de-facto industry standard.

**Contribution: Efficient Creation of Weaving Specifications**

An important aspect of engineering solutions regarding aspect-oriented modeling is the specification of the composition. Usually, a so-called pointcut definition language is used for specifying the model composition. Since these languages are based on a formal syntax, pointcut expressions are hard to create and to evolve in the face of change. In this thesis, we provide a concept for interactively selecting the model ele-

ments that are considered the places for modifying the respective behavior. Based on that selection, we are able to automatically generate a formal pointcut expression that can be used by the model composition algorithm.

**Contribution: Weaving of Aspect-oriented Models**

Another challenge regarding the engineering of aspect-oriented model creation is related to the actual model weaving. We found that almost no approach currently supports effective weaving of models. Most approaches defer the actual weaving to the programming level. [SSK06]. In this thesis, we provide an implementation of algorithms that actually realizes the weaving of models.

**Contribution: Automatic Generation of Fluents and Assertions for Model Checking**

The foundation for checking compositional properties is fluent linear temporal logic. That is, we utilize the concepts in the realm of labeled transition systems [GM03] in order to check the resulting models for time-varying properties. In practice, however, the specification of a formal model describing a labeled transition system including formal assertions is a manual task and error-prone. We defined a set of heuristics that allow for automatically deriving formal specifications of mode properties in terms of fluents and assertions. Besides the fact that we can generate a labeled transition system out of the interaction specifications defined by sequence diagrams, we are also able to hide the formal complexity from the user, which is positive in three ways: 1. Automatic generation tremendously improves the efficiency of model checking. 2. The designer does not need to be familiar with formal model checking techniques. 3. The automatic generation is less, if at all, error-prone.

### 8.1.4 Validation

In the literature, there is only little evidence regarding the effectiveness and efficiency of aspect-oriented concepts in the realm of model-based development in general and model-based architectures in particular.

**Contribution: Evidence regarding the Effectiveness and Efficiency of Aspect-oriented Approaches in Model-based Development**

In terms of validation, we performed a controlled experiment that proved the efficiency increase achieved by the aspect-oriented approach presented in this thesis. On the solution level, we validated a set of hypotheses derived from the general solution hypothesis stated in Chapter 1. We showed that, under the given context parameters, the solution hypothesis is to be accepted in the sense that the aspect-oriented sepa-

ration of concerns significantly increases the efficiency of connector design.

## 8.2 Limitations

The limitations of the solutions presented in this thesis are induced by the fundamental assumptions that the approach is based on, as well as by the effectiveness and applicability of the solutions in the context of architecture design tasks.

### 8.2.1 Aspect-oriented Architectural Model

A major assumption in the context of the aspect-oriented architectural meta-model and the related aspect-oriented extensions is that we tackle non-functional requirements that can be traced to architectural connectors only. Hence, the aspect-oriented approach as proposed in this thesis addresses only a subset of potential non-functional requirements that might be imposed on an architecture. Even though we assume a number of relevant quality attributes to be traceable to the communication properties of a system (e.g., availability, safety, performance, reliability), we stress the fact that the operationalization of the respecitve quality attribute plays an important role. One hypothesis at that point is that all quality attributes that can be observed during runtime have an impact on or are impacted by the communication properties of the system. However, the degree of impact most likely depends on the respective operationalization of the quality attribute.

In addition, we focused on homogenous crosscutting concerns only. In that case, we assume that the aspect applies to multiple places of the architectural model in the same way. Some crosscutting concerns, however, are heterogenous by definition. For instance, aspects that impact different kinds of views such as deployment, team allocation, and component connectors at the same time with different solutions portions within all of the views.

### 8.2.2 Design Method

In terms of interaction detection, we are able to detect semantic interactions that can be measured based on message sequences. The automatic generation approach of fluents and assertions is currently geared towards simple “yes/no” evaluations only. That is, currently we are able to generate fluents and assertions that check whether a tactic is interrupted by any other tactic or not. We do not consider a finer-grained evaluation in terms of specific parts of a tactic needing to be interrupt-free, for instance.

In terms of design, we currently have a limitation based on the assumption that that we do not reflect changes on the woven model itself back to the composites, namely the base model and the aspects. One way to address this problem in the context of subsequent modeling activities that change the woven models maybe to introduce so-called protected regions of the woven models. The protected region would not be modified in the event of subsequent weavings.

### 8.2.3 Tool Support

In terms of tool support, we need to consider two things: first, the tooling itself in terms of its capabilities, and second, the scalability of the tooling in terms of teams that might be working on different parts of the same architectural model at the same time. Regarding the tool support in terms of its capabilities, we currently face limitations in its interoperability. Since lots of realistic development environments already have a tool infrastructure, it is necessary to be able to interface with the already existent models and tools. Currently, the tool setting is sort of isolated. Some model exchange capabilities are provided by the tool; however, this will hamper the efficient design loop we aim at.

## 8.3 Future Work

In the context of the work presented in this thesis, there are several areas that exhibit interesting questions to be addressed by future research efforts.

### 8.3.1 Aspect-oriented Model Compositions

In the area of model composition, I propose extending the generation capabilities in terms of fluents and assertions that can be model-checked. I suggest extending the rules that can be used for automatically deriving compositional properties right from the weaving specification. That is, the properties should be defined on lower levels of granularity such that the fluents and assertions are not necessarily too strict, or too loose.

The algorithm for generating the respective pointcut specifications lends itself for the improvement of pointcut generation. We believe that heuristics can be defined to make the pointcut generation process significantly more efficient.

In terms of the quality attributes that we are addressing, we aim at extending the approach in such a way that solutions to development-time

requirements such as maintainability or portability can also be modularized.

### 8.3.2 Model-based Development

In terms of model-based prediction of system properties, we see high potential in the systematic integration of concepts defined by the Model-driven Architecture (MDA) [MDA], [BG01] initiative driven by the OMG. Since we are able to efficiently generate high-level specifications for systems, the approach could leverage model transformations in order to create a connection to more platform-specific models. Using platform-specific models makes the prediction more accurate in terms of the derivation from real products that adhere to a given simulated architecture. In the realm of embedded systems, we would go for the integration of technology-specific models that describe the communication behavior of particular bus technologies. Using model transformations, trade-off decisions are supported by realistic impact estimations. A vision at that point would be to close the loop between model-based simulations and measured real-world systems. Closing this loop implies a potential for automatically self-adapting simulation models by connecting real-world entities to the system.

During the development of the approach, we found an interesting application scenario of the model generation techniques presented in this thesis. Since we have a powerful mechanism for replacing huge portions of sequence-based interaction specifications, we can use the very same techniques for generating refined models from a given compositional specification. That is, by replacing a single message (or a set of messages) with sequences that are much more detailed, we are able to generate lower-level design models based on architectural abstractions. We believe this capability can be utilized for pushing the envelope in model-driven software development. This is related to the idea of stratification as described in [AK03]. Since the goal of stratification is to create a set of architectural views that express cross-cutting concerns revealed on a particular level of abstraction, we believe that the approach presented in this thesis can be used for realizing this idea efficiently.

### 8.3.3 Model-based Simulations

The current simulation approach comes with evaluation models for performance only. Currently, we are able to simulate communication needs given a particular network topology and component structures including the connector models. In the future, additional evaluation models for other properties that can be simulated need to be developed (e.g., availability, reliability, etc.)

Especially the way of influencing the simulation model during runtime offers great potential for cross-impact analyses. For instance, by specifying faulty behavior at the modeling level, we would be able to simulate the effectiveness of particular architectural tactics in the realm of safety. Such kinds of simulations would exceed the kinds of simulations based on the plain run of predefined sequences. Such simulations would be particularly useful for combining different evaluation models on the same set of simulation runs. We could imagine using the power of UML 2.1 constructs like fragments for specifying sophisticated simulation using conditional executions.

### 8.3.4 Extended Validations

Regarding the validation of the aspect-oriented design approach, we strive for conducting more experiments to check different aspects of the solution hypotheses. Since the experimentation potential heavily depends on the tooling infrastructure, we would combine tool improvements with the design of more complex experiments, isolating specific variables, and gaining deeper insight into the potential of impacting scalability issues using aspect-oriented separation of concerns.

Besides validating the solution hypotheses in more detail, we propose conducting a number of case studies in industrial contexts in order to see what influencing factors might hamper the efficiency of the approach in reality. It is particularly interesting to see to which extend teams can be decoupled from each other in realistic project settings.

## 8.4 Concluding Remarks

The goal of this thesis was to improve architecture design regarding its scalability depending on the number of non-functional concerns at hand. In practice, however, there exists a multitude of external influencing factors that are beyond of scope of the architect's influence, and might therefore be the ones that make architects feel lost in practice. However, the contributions of this thesis provide a solid basis for coping with these real-world challenges by offering effective improvements of daily engineering activities. By delivering one piece of a solution towards scalable engineering practices, we are convinced that architects are getting one step closer to the vision of being able to efficiently react to changes in requirements, contextual parameters, or managerial strategies.

# References


[AEB03] Aldawud, O.; Elrad, T. & Bader, A. (2003), UML Profile for Aspect-Oriented Software Development, *in* 'The Third International Workshop on Aspect Oriented Modeling'.

[All97] Allen, R. J. (1997), 'A Formal Approach to Software Architecture'. PhD thesis.

[AG97] Allen, R. & Garlan, D. (1997), 'A formal basis for architectural connection', *ACM Transactions on Software Engineering and Methodology*.

[ARO03] America, P.; Rommes, E. & Obbink, J. H. (2003), Multi-view Variation Modeling for Scenario Analysis, *in* 'PFE', pp. 44-65.

[AK03] Atkinson, C. & Kuhne, T. (2003), 'Aspect-Oriented Development with Stratified Frameworks', *IEEE Softw.* 20(1), 81--89.

[ATL10] Atlas Project. The Atlas Transformation Language (ATL). From http://www.eclipse.org/m2m/atl/

[BB01] Bachmann, F. & Bass, L. (2001), Introduction to the Attribute Driven Design Method, *in* 'In 23rd International Conference on Software Engineering', pp. 745--746.

[BE07] Bakre, S. & Elrad, T. (2007), Scenario based resolution of aspect interactions with aspect interaction charts, *in* 'AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling', ACM, New York, NY, USA, pp. 1--6.

[BGL04] Barra, E., Genova,G., Llorens, J. (2004). An approach to Aspect Modelling with UML 2.0. In *Proc. of 5th Aspect-Oriented Modeling Workshop (UML'04), Lisbon, Portugal*.

[Bas93] Basili, V. R. (1993), The Experimental Paradigm in Software Engineering, *in* 'Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions', Springer-Verlag, London, UK, pp. 3--12.

[BCG+06] Batista, T.; Chavez, C.; Garcia, A.; Rashid, A.; Sant'Anna, C.; Kulesza, U. & Filho, F. C. (2006), Reflections on architectural connection: seven issues on aspects and ADLs, *in* 'EA '06: Proceedings of the 2006 international workshop on Early aspects at ICSE', ACM, New York, NY, USA, pp. 3--10.

[BC05] Berg, K. V. D. & Conejero, J. M. (2005), A conceptual formalization of crosscutting in AOSD, *in* 'In Desarrollo de Software Orientado a Aspectos.

[BG01] Bézivin, J. & Gerbé, O. (2001), Towards a Precise Definition of the OMG/MDA Framework, *in* 'ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering', IEEE Computer Society, Washington, DC, USA, pp. 273.

[Boo07] Booch, G. (2007), 'The Irrelevance of Architecture', *IEEE Softw.* 24(3), 10--11.

[Bos00] Bosch, J. (2000), *Design and Use of Software Architectures – Adopting and evolving a product-line approach*, Addison-Wesley, Reading, MA, USA.

[BH06] Boucké, N. & Holvoet, T. (2006), Relating architectural views with architectural concerns, *in* 'EA '06: Proceedings of the 2006 international workshop on Early aspects at ICSE', ACM, New York, NY, USA, pp. 11--18.

[BGH07] Boucké, N.; Garcia, A. & Holvoet, T. (2007), Composing Structural Views in xADL, *in* 'Proceedings of the 10th international conference on Early aspects', Springer-Verlag, Berlin, Heidelberg, pp. 115--138.

[Caz06] Cazzola, W. (2006), Semantic Join Point Models: Motivations, Notions and Requirements, *in* 'In Proceedings of the Software Engineering Properties of Languages and Aspect Technologies Workshop (SPLAT'06'.

[CB05] Clarke, S. & Baniassad, E. (2005), *Aspect-Oriented Analysis and Design: The Theme Approach (Addison-Wesley Object Technology Series)*, Addison-Wesley Professional.

[Cle96] Clements, P. C. (1996), A Survey of Architecture Description Languages, *in* 'IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design', IEEE Computer Society, Washington, DC, USA, pp. 16.

[CGB+02] Clements, P.; Garlan, D.; Bass, L.; Stafford, J.; Nord, R.; Ivers, J. & Little, R. (2002), *Documenting Software Architectures: Views and Beyond*, Pearson Education.

[CKK01] Clements, P.; Kazman, R. & Klein, M. (2001), *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley.

[CK03] Clements, P. & Kazman, R. (2003), *Software Architecture in Practice*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[CN02] Clements, P. C. & Northrop, L. (2002), *Software Product Lines: Practices and Patterns*, Addison-Wesley.

[CGB09] Chavez, C.; Garcia, A.; Batista, T.; Oliveira, M.; Sant'Anna, C. & Rashid, A. (2009), Composing architectural aspects based on style semantics, *in* 'AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development', ACM, New York, NY, USA, pp. 111--122.

[CRS+05] Chitchyan, R., Rashid, A., Sawyer, P., Garcia, A., Alarcon, M.P., Bakker, J., Tekinerdogan, B., Clarke, S., Jackson, A. (2005), Survey of Aspect-Oriented Analysis and Design Approaches, Technical Report, AOSD-Europe-ULANC-9: AOSD-Europe.

[CRF+06] Cuesta, C. E.; Romay, M. P.; de la Fuente, P. & Barrio-Solórzano, M. (2006), 'Coordination as an Architectural Aspect', *Electron. Notes Theor. Comput. Sci.* 154(1), 25--41.

[CPF05] Cuesta, C. E.; Romay, M.; de la Fuente, P.; Barrio-Solorzano, M. (2005). Architectural aspects of architectural aspects. In 2nd European Workshop on Software Architecture (EWSA), volume LNCS 3527, pages 247–262.

[Des10] Desmo-J Simulation Framework (2010). From http://desmoj.sourceforge.net/

[Dij82] Dijkstra, E. W. (1982), *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag.

[DoD04] DoD Architecture Framework Working Group. (2004). DoD Architecture Framework, Version 1.0, United States Department of Defense.

[DFS02] Douence, R.; Fradet, P. & Südholt, M. (2002), A framework for the detection and resolution of aspect interactions, *in* , Springer-Verlag, , pp. 173--188.

[DFS04] Douence, R.; Fradet, P. & Südholt, M. (2004), Composition, reuse and interaction analysis of stateful aspects, *in* 'AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development', ACM, New York, NY, USA, pp. 141--150.

[EKK+10] Eisenbarth, M.; Keuler, T.; Knodel, J.; Naab, M. & Rost, D. (2010), 'Fraunhofer DSSA'(IESE-Report No. 035.10/E), Technical report, Fraunhofer IESE.

[EAB05] Elrad, T.; Aldawud, O. & Bader, A. (2005), Expressing Aspects Using UML Behavioral and Structural Diagrams, pp. 459-478.

[FF00] Filman, R. E. & Friedman, D. P. (2000), 'Aspect-Oriented Programming is Quantification and Obliviousness', RIACS.

[FBL02] Filman, R. E.; Barrett, S.; Lee, D. D. & Linden, T. (2002), 'Inserting ilities by controlling communications', *Commun. ACM* 45(1), 116--122.

[FEC04] Filman, R., Elrad, T., Clarke, S., & Aksit, M. (2004), *Aspect-oriented software development*, Addison-Wesley Professional.

[FRG+04] France, R.; Ray, I.; Georg, G. & Ghosh, S. (2004), Aspect-oriented approach to early design modelling, *in* 'IEE Proceedings - Software', pp. 173--185.

[FS06] Fuentes, L., Sanchez, P. (2006) Elaborating UML 2.0 Profiles for AO Design. In *Proc. of the 8th International Workshop on Aspect-Oriented Modeling (AOSD'06), Bonn, Germany.*

[GHJ+95] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1995), *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Professional.

[GCB06] Garcia, R.; Chavez, C.; Batista, T.; Kulesza, U.; Rashid, A. & Lucena, C. (2006), On the Modular Representation of Architectural Aspects, *in* 'In Proc. of the European Workshop on Software Architecture'.

[GAO95] Garlan, D.; Allen, R. & Ockerbloom, J. (1995), 'Architectural Mismatch: Why Reuse Is So Hard', *IEEE Software* 12, 17-26.

[GS94] Garlan, D., Shaw, M. (1994), An Introduction to Software Architecture, *in* Advances in Software Engineering and Knowledge Engineering, Volume 1: World Scientific Publishing Company.

[GM03] Giannakopoulou, D. & Magee, J. (2003), 'Fluent model checking for event-based systems', *SIGSOFT Softw. Eng. Notes* 28(5), 257--266.

[GK06] Goldman, M., Katz, S., Modular Generic Verification of LTL Properties for Aspects, Foundations of Aspect-oriented Languages Workshop, AOSD, 2006, Bonn.

[GK07] Goldman, M. & Katz, S. (2007), MAVEN: modular aspect verification, *in* 'TACAS'07: Proceedings of the 13th international conference on Tools and algorithms for the construction and analysis of systems', Springer-Verlag, Berlin, Heidelberg, pp. 308--322.

[HHK00] Hilsdale, E.; Hugunin, J.; Kersten, M.; Kiczales, G.; Lopes, C. & Palm, J. (2000), AspectJ: the language and support tools, *in* 'OOPSLA '00: Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum)', ACM, New York, NY, USA, pp. 163.

[HKN+07] Hofmeister, C.; Kruchten, P.; Nord, R. L.; Obbink, H.; Ran, A. & America, P. (2007), 'A general model of software architecture design derived from five industrial approaches', *J. Syst. Softw.* 80(1), 106--126.

[HNS00] Hofmeister, C.; Nord, R. & Soni, D. (2000), *Applied software architecture*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[IBM10] IBM (2010). Rational Software Architect (RSA). From http://www-01.ibm.com/software/awdtools/architect/swarchitect/

[IEEE1471] IEEE-Std.1471. (2000). *ANSI/IEEE Std 1471-2000 – Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE, New York, October 2000.

[ISO98] ISO/IEC (1998). Information Technology – Open Distributed Processing-Reference Model, ISO/IEC, 10746.

[ISO01] ISO/IEC 9126-1: (2001). Software engineering, Product quality, Part 1: Quality model.

[JUN10] JUnit (2010). From http://www.junit.org/

[KRK10] Kabanov, A., Rombach, D. (Supervisor), Keuler, T. (Supervisor), 2010, Graphical Selection of Joinpoints for Weaving of Architectural Models, Bachelor Thesis.

[Kan03] Kande, M. M., A concern-oriented approach to software architecture, in Computer Science, vol. PhD. Lausanne,Switzerland: Swiss Federal Institute ofTechnology (EPFL), 2003.

[KMB+06] Kellens, A.; Mens, K.; Brichau, J. & Gybels, K. (2006), Managing the Evolution of Aspect-Oriented Software with Model-based Pointcuts, *in* 'In Proceedings of the European Conference on Object-Oriented Programming (ECOOP', Springer-Verlag, , pp. 501--525.

[KMU08] Keuler, T.; Muthig, D. & Uchida, T. (2008), Efficient Quality Impact Analyses for Iterative Architecture Construction, *in* 'WICSA '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)', IEEE Computer Society, Washington, DC, USA, pp. 19--28.

[KW09] Keuler, T.; Webel, C. (2009), Interaction-sensitive Synthesis of Architectural Tactics in Connector Designs, Workshop on Software Architecture Design, WICSA/ECSA 2009.

[KLM+97] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.-M. & Irwin, J. (1997), Aspect-oriented programming, *in* Mehmet Akşit & Satoshi Matsuoka, ed., 'ECOOP'97" Object-Oriented Programming ', Springer-Verlag, Berlin/Heidelberg, pp. 220--242.

[KRK09] Kornev, Y., Rombach, D. (Supervisor), Keuler, T. (Supervisor). (2009), Style-based Pointcut Definition for Weaving of Architectural Tactics, Diploma thesis.

[KTG+06] Krechetov, I.; Tekinerdogan, B.; Garcia, A.; Chavez, C. & Kulesza, U. (2006), Towards an Integrated Aspect-Oriented Modeling Approach for, *in* 'Software Architecture Design. 8th Workshop on Aspect-Oriented Modelling (AOM.06), AOSD.06'.

[KFG04] Krishnamurthi, S.; Fisler, K. & Greenberg, M. (2004), Verifying aspect advice modularly, *in* 'SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering', ACM, New York, NY, USA, pp. 137--146.

[Kru95] Kruchten, P. (1995), 'The 4+1 View Model of Architecture', *IEEE Software* 12, 42-50.

[Kru03] Kruchten, P. (2003), *The Rational Unified Process: An Introduction*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[LKM+05] Letier, E.; Kramer, J.; Magee, J. & Uchitel, S. (2005), Fluent temporal logic for discrete-time in event-based models, *in* 'In Proceedings of the 10th European Software Engineering Conference', ACM Press, , pp. 70--79.

[LWF01] Lopes, A.; Wermelinger, M. & Fiadeiro, J. L. (2001), A Compositional Approach to Connector Construction, *in* 'WADT '01: Selected papers from the 15th International Workshop on Recent Trends in Algebraic Development Techniques', Springer-Verlag, London, UK, pp. 201--220.

[LWF03] Lopes, A.; Wermelinger, M. & Fiadeiro, J. L. (2003), 'Higher-order architectural connectors', *ACM Trans. Softw. Eng. Methodol.* 12(1), 64--104.

[MW47] Mann, H. B. & Whitney, D. R. (1947), 'On a test of whether one of two random variables is stochastically larger than the other', *Annals of Mathematical Statistics*.

[MKD02] Masuhara, H.; Kiczales, G. & Dutchyn, C. (2002), 'Compilation Semantics of Aspect-Oriented Programs'.

[MEH01] Maier, M. W.; Emery, D. E. & Hilliard, R. (2001), 'Software Architecture: Introducing IEEE Standard 1471.', *IEEE Computer* 34(4), 107-109.

[MT00] Medvidovic, N., & Taylor, R. N. (2000). *A Classification and Comparison Framework for Software Architecture Description Languages*. IEEE Transactions on Software Engineering, 26(1), 70-93.

[MMP00] Mehta, N. R.; Medvidovic, N. & Phadke, S. (2000), Towards a taxonomy of software connectors, *in* 'ICSE '00: Proceedings of the 22nd international conference on Software engineering', ACM, New York, NY, USA, pp. 178--187.

[MS99] Miller, R. & Shanahan, M. (1999), 'The Event Calculus in Classical Logic – Alternative Axiomatisations'.

[MV07] Mostefaoui, F. & Vachon, J. (2007), Verification of Aspect-UML models using alloy, *in* 'AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling', ACM, New York, NY, USA, pp. 41--48.

[MDA] MDA Guide Version 1.0.1. from http://www.omg.org/docs/omg/03-06-01.pdf

[NPM+02] Navasa, A., Perez, M.A., Murillo, J.M., Hernandez, J. (2002), Aspect-Oriented Software Architecture: a Structural Perspective, *in* Proceedings of the Aspect-Oriented Software Development, pp. 1554-1558.

[NPM09] Navasa, A.; Pérez-Toledano, M. A. & Murillo, J. M. (2009), 'An ADL dealing with aspects at software architecture stage', *Inf. Softw. Technol.* 51(2), 306--324.

[Par72] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules (Vol. 15, pp. 1053-1058): ACM.

[PAC+05] Perez, J.; Ali, N.; Cars, J. A. & Ramos, I. (2005), Dynamic Evolution in Aspect-Oriented Architectural Models, *in* 'EWSA', pp. 59-76.

[Per06] Perez, J. PRISMA: Aspect-Oriented Software Architectures. (2006) PhD thesis, Department of Information Systems and Computation, Polytechnic University of Valencia.

[PAC+06] Perez, J.; Ali, N.; Carsi, J. A.; Ramos, I. (2006). Designing Software Architectures with an Aspect-Oriented Architecture Description Language. In I. Gorton, G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. A. Szyperski, and K. C. Wallnau, editors, Component-Based Software Engineering, volume 4063 of Lecture Notes in Computer Science, pages 123–138, V¨asteras, Sweden, 2006. Springer Verlag.

[PW92] Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture (Vol. 17, pp. 40-52): ACM.

[PFT03] Pinto, M.; Fuentes, L. & Troya, J. M. (2003), DAOP-ADL: an architecture description language for dynamic component and aspect-based development, *in* 'GPCE '03: Proceedings of the 2nd international conference on Generative programming and component engineering', Springer-Verlag New York, Inc., New York, NY, USA, pp. 118--137.

[PF07] Pinto, M., Fuentes, L. (2007) AO-ADL: An ADL for describing aspect-oriented architectures. In *Early Aspect Workshop at AOSD 2007*.

[Ran00] Ran, A., 2000. ARES Conceptual Framework for Software Architecture. In: Jazayeri, M., Ran, A., van der Linden, F. (Eds.), Software Architecture for Product Families Principles and Practice. Addison-Wesley, Boston, pp. 1–29.

[RTT04] Reina, A., Torres, J., Toro, M. (2004) Separating Concerns by Means of UML-pro¯les and Metamodels in PIMs. In *Proc. of the 5th Aspect-Oriented Modeling Workshop (UML'04), Lisbon, Portugal*.

[RGG01] Roessler, F.; Geppert, B. & Gotzhein, R. (2001), Collaboration-Based Design of SDL Systems, *in* 'SDL '01: Proceedings of the 10th International SDL Forum Copenhagen on Meeting UML', Springer-Verlag, London, UK, pp. 72--89.

[RW05] Rozanski, N. & Woods, E. (2005), *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, Addison-Wesley Professional.

[STJ06] Sanen, F.; Truyen, E. & Joosen, W. (2006), Classifying And Documenting Aspect Interactions, *in* 'Proceedings of the Fifth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software', pp. 23--26.

[SSK06] Schauerhuber, A.; Schwinger, W.; Kapsammer, E.; Retschitzegger, W. & Wimmer, M. (2006), 'A Survey on Aspect-Oriented Modeling Approaches'.

[Sha94] Shaw, M. (1994), Procedure Calls Are the Assembly Language of Software Interconnection:, *in* , Springer-Verlag, , pp. 17--32.

[Sha96] Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc.

[Som01] Sommerville, I. (2001). *Software engineering (6th ed.)*. Addison-Wesley Longman Publishing Co., Inc.

[SK04] Spitznagel, B. & Koopman, P. (2004), 'Compositional Transformation of Software Connectors'.

[SHU02a] Stein, D.; Hanenberg, S. & Unland, R. (2002), A UML-based aspect-oriented design notation for AspectJ, *in* 'AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development', ACM, New York, NY, USA, pp. 106--112.

[SHU02b] Stein, D.; Hanenberg, S. & Unland, R. (2002), Designing Aspect-Oriented Crosscutting in UML, *in* 'In AOSD-UML Workshop at AOSD '02'.

[SHU02c] Stein, D.; Hanenberg, S. & Unl, R. (2002), On representing join points in the UML, *in* 'Workshop on Aspect-Oriented Modeling with UML'.

[SHU06] Stein, D.; Hanenberg, S. & Unland, R. (2006), Expressing different conceptual models of join point selections in aspect-oriented design, *in* 'AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development', ACM, New York, NY, USA, pp. 15--26.

[SKB03] Stoerzer, M.; Krinke, J.; Breu, S. & Passau, U. (2003), 'Trace Analysis for Aspect Application'.

[Sub10] Subversion (2010). From http://subversion.tigris.org/

[Szy98] Szyperski, C. (1998), *Component software: beyond object-oriented programming*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[THO00] Tarr, P.; Harrison, W.; Ossher, H.; Finkelstein, A.; Nuseibeh, B. & Perry, D. (2000), Workshop on multi-dimensional separation of concerns in software engineering (workshop session), *in* 'ICSE '00: Proceedings of the 22nd international conference on Software engineering', ACM, New York, NY, USA, pp. 809--810.

[THS99] Tarr, P., Ossher, H., Harrison, W., Sutton, S.M. (1999), N Degrees of Separation: Multi-Dimensional Separation of Concerns, *in* Proceedings of the 21st International Conference on Software Engineering (ICSE'99), pp.107-119.

[TMD09] Taylor, R. N.; Medvidovic, N. & Dashofy, E. M. (2009), *Software Architecture: Foundations, Theory and Practice*, Addison-Wesley.

[TMA+04] Tekinerdogan, B., Moreira, A., Araujo, J., Clements, P. (2004), Workshop report of Early Aspects at AOSD.

[ToG03] The Open Group. (2003). TOGAF (The Open Group Archtiecture Framework) Version 8.1, "Enterprise Edition", The Open Group.

[TA05] Tyree, J. & Akerman, A. (2005), 'Architecture Decisions: Demystifying Architecture', *IEEE Softw.* 22(2), 19--27.

[UML08] UML. (2008). *Unified Modeling Language*. From http://www.omg.org/technology/documents/formal/spem.htm

[WL99] Weiss, D. M. & Lai, R., ed. (1999), *Software Product Line Engineering: A Family-Based Software Development Process*, Addison-Wesley.

[WJ08] Whittle, J. & Jayaraman, P. (2008), 'MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation', 16--27.

[WS00] Whittle, J. & Schumann, J. (2000), Generating statechart designs from scenarios, *in* 'ICSE '00: Proceedings of the 22nd international conference on Software engineering', ACM, New York, NY, USA, pp. 314--323.

[WRH00] Wohlin, C.; Runeson, P.; Hцst, M.; Ohlsson, M. C.; Regnell, B. & Wesslén, A. (2000), *Experimentation in software engineering: an introduction*, Kluwer Academic Publishers, Norwell, MA, USA.

[WH05] Woods, E. & Hilliard, R. (2005), Architecture Description Languages in Practice Session Report, *in* 'WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture', IEEE Computer Society, Washington, DC, USA, pp. 243--246.

[Zac87] Zachman, J. A. (1987). *A framework for information systems architecture*. IBM Systems Journal, 26(3), 276-292.

[ZHZ02] Zakaria, A., Hosny, H., Zeid, A. (2002) A UML Extension for Modeling Aspect-Oriented Systems. In *Proc. of the 2nd International Workshop on Aspect-Oriented Modeling with UML (UML'02)*.

# Appendix A: Experimentation Material

This appendix contains the materials used for the experiment as described in Chapter 7.

Pre briefing questionnaire

ID: _____

**Studies**
(e.g. Bachelor, Master): _____ Semester: __

**Experience**

What is your experience with software architecture?

| None | Little | Average | Substantial | Professional |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

What is your experience with the UML 2.0?

| None | Little | Average | Substantial | Professional |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

What is your experience with the Eclipse IDE?

| None | Little | Average | Substantial | Professional |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

What is your experience with the IBM Rational Architect/Modeler?

| None | Little | Average | Substantial | Professional |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

**Motivation**

Estimate how motivated you are to perform well in the experiment:

| Not | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Figure 80 Pre-briefing Questionnaire

Task description

ID: _____

Group: A

1. **Task**

**Starting time:** _____

Open the "**Increment1A**" project

a. How many messages of type <<*publisherCommunication*>> can be found in this project? (**Remark**: synchronous messages have a response-part, do not count such response-parts)

_____

[End time: _____]

b. How many tactics have been explicitly defined in the *Increment1A-Tactics.emx* project?

_____

[End time: _____]

c. How many distinct tactics can be found in the sequence diagrams of *Increment1A? (Do not count tactics from Increment1A-Tactics project!)*

_____

[End time: _____]

2. **Task:**

**Starting time:** _____

Change the Logging-Tactic in such a way that a second log-role is introduced that is triggered in addition to the already existing log-role (redundant logging). Apply this change to **all** sequence diagrams of Increment1A!

[**Tool-hint**: In order to connect a lifeline within a fragment (e.g. alt-fragment) with a newly created lifeline in the same diagram, drag the corner of the fragment over the lifeline and select the lifeline to be covered by the fragment]

_____

[End time: _____]

Figure 81 Task Description - Group A (1/3)

**3. Task**

**Starting time: _____**

Open the "**Increment2A**" project

a. How many messages of type <<*serverCall* >> can be found in this project?

_____

[End time: .....]

b. How many tactics have been explicitly defined in the *Increment2A-Tactics.emx* project?

_____

[End time: .....]

c. How many distinct tactics can be found in the sequence diagrams of *Increment2A? (Do not count tactics from Increment2A-Tactics project!)*

_____

[End time: .....]

**4. Task:**

**Starting time: _____**

Open the "**Increment2A-Tactics**" project

Change the LoadBalancing-Tactic as defined in the *Increment2A-Tactics* project in such a way that a second role is introduced for being triggerd by a cache-operation in order to store the latest resource selected

[End time: .....]

**5. Task**

Open the "**Increment3A**" project

a. How many messages of type <<*filterCommunication*>> can be found in this project?

_____

[End time: .....]

b. How many tactics have been explicitly defined in the *Increment3A-Tactics.emx* project?

_____

Figure 82 Task Description - Group A (2/3)

[End time: _____]

c. How many distinct tactics can be found in the sequence diagrams of *Increment3A? (Do not count tactics from Increment3A-Tactics project!)*

_____

[End time: _____]

**6. Task:**

Change the Encryption-Tactic in such a way that a second cryptoEngine is introduced that is triggered in addition to the already existing one (sequential encryption). Apply this change to **all** sequence diagrams of Increment3A!

[Time: _____ min.]

**7. Task**

**Starting time: _____**

Open the "**Increment4A-Tactics**" project

Change the Authorization-Tactic as defined in the *Increment4A-Tactics* in such a way that all <<secureAccess>> authorize messages are encrypted according to the encryption-tactic

_____

[End time: _____]

Figure 83 Task Description - Group A (3/3)

Task description

ID: _____

Group: B

**1. Task**

**Starting time: _____**

Open the "**Increment1B**" project

a. How many messages of type <<*publisherCommunication*>> can be found in this project? (**Remark**: synchronous messages have a response-part, do not count such response-parts)

_____

[End time: .....]

b. How many tactics have been explicitly defined in the *Increment1B-Tactics.emx* project?

_____

[End time: .....]

c. How many distinct tactics can be found in the sequence diagrams of *Increment1B? (Do not count tactics from Increment1B-Tactics project!)*

_____

[End time: .....]

**2. Task:**

**Starting time: _____**

Open the "**Increment1B-Tactics**" project

Change the Logging-Tactic as defined in *Increment1B-Tactics* in such a way that a second log-role is introduced that is triggered in addition to the already existing log-role (redundant logging).

_____

[End time: .....]

**3. Task**

**Starting time: _____**

Open the "**Increment2B**" project

a. How many messages of type <<*serverCall* >> can be found in this project?

Figure 84 Task Description - Group B (1/3)

b. How many tactics have been explicitly defined in the *Increment2B-Tactics.emx* project?

_____

[End time: .....]

c. How many distinct tactics can be found in the sequence diagrams of *Increment2B? (Do not count tactics from Increment2B-Tactics project!)*

_____

[End time: .....]

**4. Task:**

**Starting time:** _____

Change the LoadBalancing-Tactic in such a way that a second role is introduced for being triggerd by a cache-operation in order to store the latest resource selected. Apply this change to **all** sequence diagrams of Increment2B!

[**Tool-hint**: In order to connect a lifeline within a fragment (e.g. alt-fragment) with a newly created lifeline in the same diagram, drag the corner of the fragment over the lifeline and select the lifeline to be covered by the fragment]

[End time: .....]

**5. Task**

Open the "**Increment3B**" project

a. How many messages of type *<<filterCommunication>>* can be found in this project?

_____

[End time: .....]

b. How many tactics have been explicitly defined in the *Increment3B-Tactics.emx* project?

_____

[End time: .....]

Figure 85 Task Description - Group B (2/3)

c. How many distinct tactics can be found in the sequence diagrams of *Increment3B? (Do not count tactics from Increment3B-Tactics project!)*

_____

[End time: _____]

**6. Task:**

Open the "**Increment3B-Tactics**" project

Change the Encryption-Tactic as defined in *Increment3B-Tactics* in such a way that a second cryptoEngine is introduced that is triggered in addition to the already existing one (sequential encryption).

[Time: _____ min.]

**7. Task**

**Starting time: _____**

Open the "**Increment4B**" project

Change the Authorization-Tactic in such a way that all *<<secureAccess>> authorize* messages are encrypted according to the encryption-tactic. Apply this change to **all** sequence diagrams of Increment4B!

_____

[End time: _____]

Figure 86 Task Description - Group B (3/3)

Debriefing questionnaire

ID: _____

**Performance Estimation**

Estimate how well you understood what you were asked for:

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Estimate the quality of your results (correctly answered questions) (in %):

| 0-20 | 21-40 | 41-60 | 61-80 | 81-100 |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

**Tool assessment**

How easy did you find the provided tools to use?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

**Experiment assessment**

Was there enough time given for the experiment?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

If you could not complete the tasks, please specify why (tick one):

- Not enough time
- Did not understand the task
- No functionality found in the tool for solving the task
- Other (please explain)

Any additional comments?

Thank you very much!

Figure 87 Debriefing questionnaire

Validation Questionnaire

ID: _____

*Please tick a number for every question. You can also add a short explaining comment.*

*Usability:*

1. **How well do the modeling elements of RSA express architectural behavior?**
   a. Is the coordination behavior of connectors appropriately illustrated?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

   b. How suitable is it to partially design communication behavior that can be composed to a full behavior design later on?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

2. **To what extend do the tools support architecture design decisions (w.r.t. connectors & tactics)?**
   a. Do the names of the tools explain the functions of these tools?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

   b. In how far do the different perspectives of the tools support the decision making process?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

*Effort reduction:*

3. **Does the approach support incremental development?**
   a. Does the effort for adding new design decisions depend on decisions already made?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

   b. Is the effort for changing earlier design decisions depending on the number of decision already made?

| Not at all | Poorly | Fairly | Well | Highly |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

Figure 88 Validation questionnaire

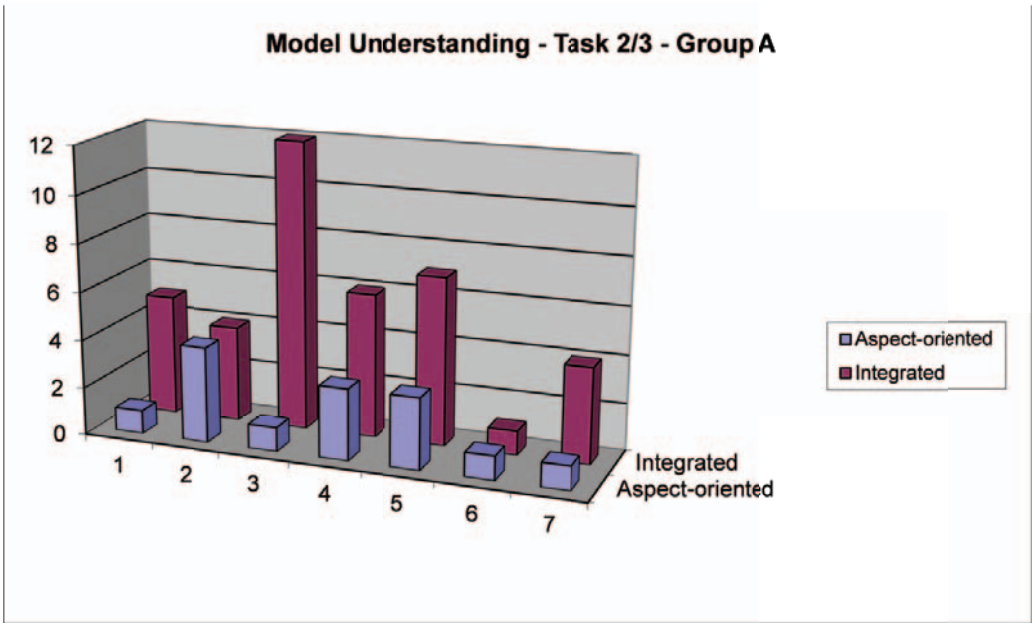



Figure 89 Efficiency increase between AO and Integrated Modeling in btw. two tasks

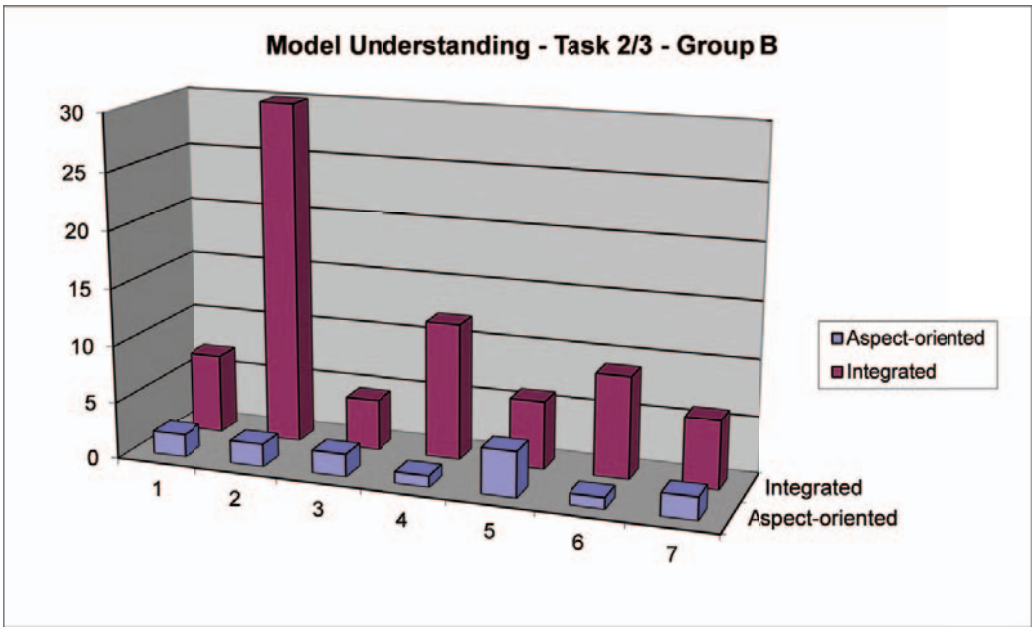



Figure 90 Efficiency increase between AO and Integrated Modeling in btw. two tasks

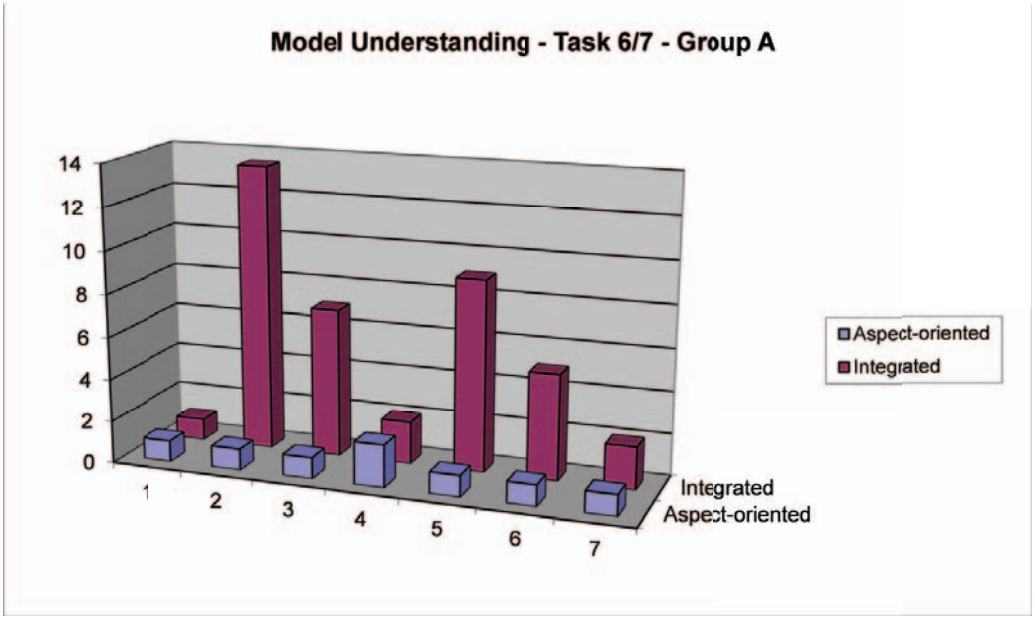



Figure 91 Efficiency increase between AO and Integrated Modeling in btw. two tasks

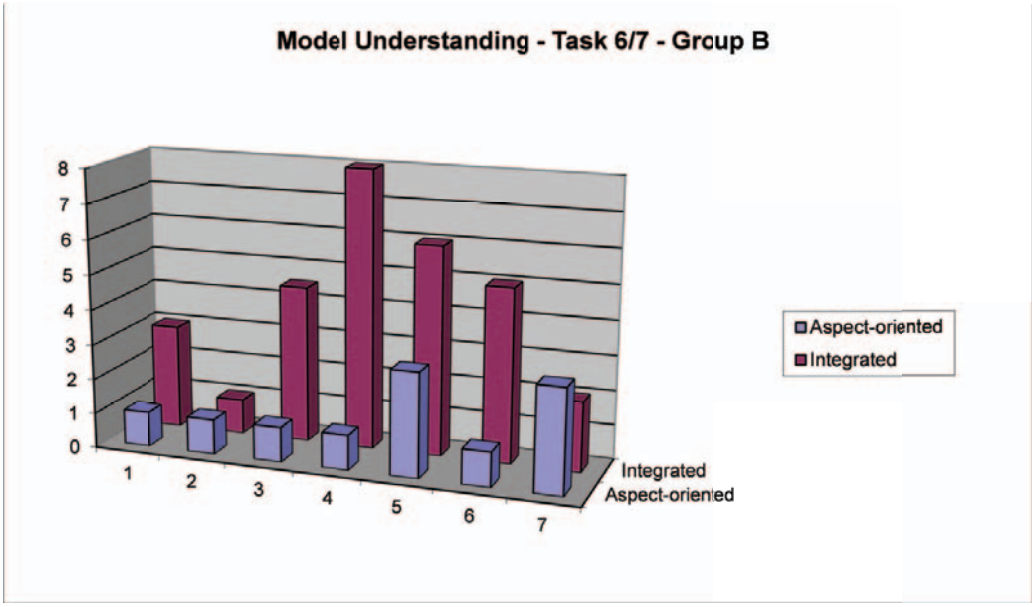



Figure 92 Efficiency increase between AO and Integrated Modeling in btw. two tasks

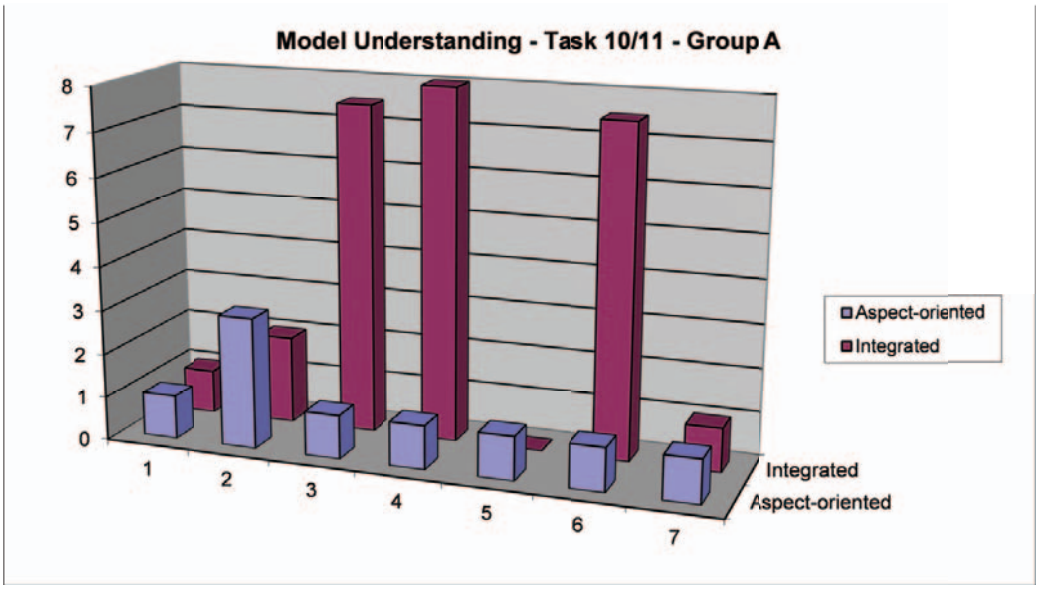



Figure 93 Efficiency increase between AO and Integrated Modeling in btw. two tasks

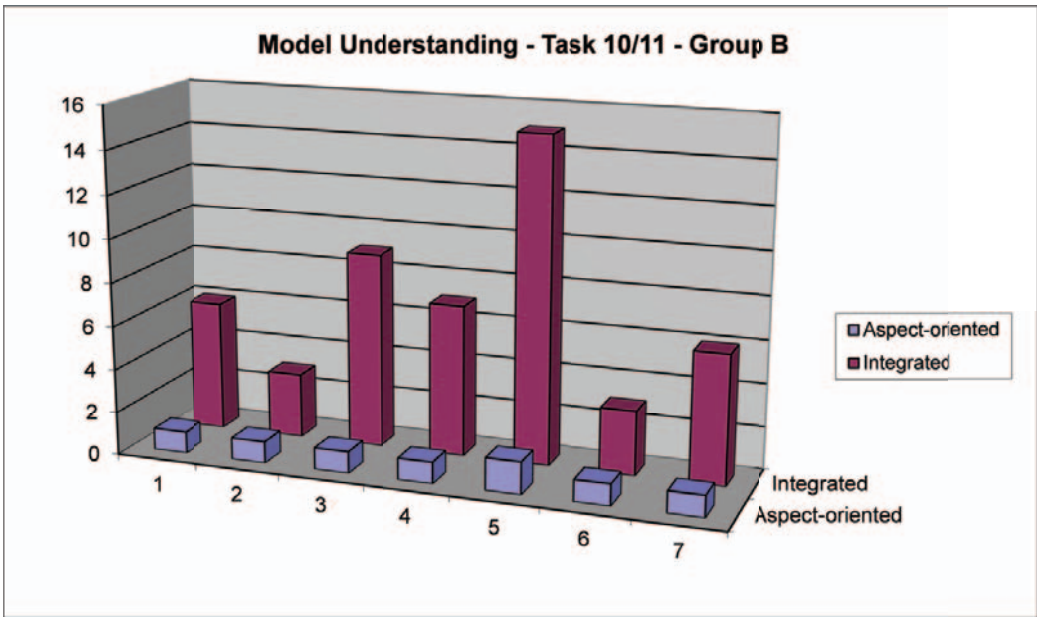



Figure 94 Efficiency increase between AO and Integrated Modeling in btw. two tasks

| | | Task 1 | | |
|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Time[min.** |
| | ID2_2 | 6 | 100 | 7 |
| | ID7_2 | 6 | 100 | 5 |
| | ID2_1 | 6 | 100 | 5 |
| | ID5_1 | 6 | 100 | 5 |
| | ID10_1 | 6 | 100 | 5 |
| | ID7_1 | 6 | 100 | 6 |
| | ID4_1 | 6 | 100 | 4 |
| | | | | |
| **Avg** | | | 100 | **5,285714** |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 6 | 100 | 6 |
| | ID1_1 | 5 | 83,33333 | 3 |
| | ID9_1 | 6 | 100 | 5 |
| | ID3_1 | 6 | 100 | 7 |
| | ID1_2 | 6 | 100 | 8 |
| | ID6_2 | 6 | 100 | 3 |
| | ID3_1 | 6 | 100 | 4 |

Figure 95 Results Task 1

| | | Task 2 | | |
|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Time[min.** |
| | ID2_2 | 3 | 100 | **1** |
| | ID7_2 | 3 | 100 | **4** |
| | ID2_1 | 3 | 100 | **1** |
| | ID5_1 | 3 | 100 | **3** |
| | ID10_1 | 3 | 100 | **3** |
| | ID7_1 | 3 | 100 | **1** |
| | ID4_1 | 3 | 100 | **1** |
| | | | | |
| **Avg** | | | 100 | **2** |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 3 | 100 | **2** |
| | ID1_1 | 3 | 100 | **2** |
| | ID9_1 | 3 | 100 | **2** |
| | ID3_1 | 3 | 100 | **1** |
| | ID1_2 | 2 | 66,66667 | **4** |
| | ID6_2 | 3 | 100 | **1** |
| | ID3_1 | 3 | 100 | **2** |

Figure 96 Results Task 2

| | | Task 3 | | | |
|---|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Weighted-Efficiency** | **Time[m** |
| | ID2_2 | 3 | 100 | **5** | 5 |
| | ID7_2 | 3 | 100 | **4** | 4 |
| | ID2_1 | 3 | 100 | **12** | 12 |
| | ID5_1 | 3 | 100 | **6** | 6 |
| | ID10_1 | 1 | 33,33333 | **21** | 7 |
| | ID7_1 | 3 | 100 | **1** | 1 |
| | ID4_1 | 2 | 66,66667 | **6** | 4 |
| | | | | | |
| **Avg** | | | 85,71429 | | **5,571429** |
| | | | | | |
| | | | | | |
| Group B | | | | | |
| | ID6_1 | 3 | 100 | **7** | 7 |
| | ID1_1 | 2 | 66,66667 | **30** | **20** |
| | ID9_1 | 4 | 66,66667 | **4,5** | 3 |
| | ID3_1 | 4 | 66,66667 | **12** | 8 |
| | ID1_2 | 2 | 66,66667 | **6** | 4 |
| | ID6_2 | 1 | 33,33333 | **9** | 3 |
| | ID3_1 | 1 | 33,33333 | **6** | 2 |

Figure 97 Results Task 3

| | | Task 4 | | |
|---|---|---|---|---|
| Group A | | Correct | Time[min.] | Complete |
| | ID2_2 | 100 | 26 | 100 |
| | ID7_2 | 100 | 25 | 100 |
| | ID2_1 | 100 | 20 | 100 |
| | ID5_1 | 100 | 16 | 100 |
| | ID10_1 | 100 | 22 | 100 |
| | ID7_1 | 100 | 41 | 100 |
| | ID4_1 | 100 | 29 | 84 |
| | | | | |
| **Avg** | | | **25,57143** | |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 100 | 15 | 100 |
| | ID1_1 | 100 | 6 | 100 |
| | ID9_1 | 100 | 14 | 100 |
| | ID3_1 | 100 | 3 | 100 |
| | ID1_2 | 100 | 26 | 100 |
| | ID6_2 | 100 | 7 | 100 |
| | ID3_1 | 100 | 9 | 100 |

Figure 98 Results Task 4

| | | Task 5 | | |
|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Time[min.** |
| | ID2_2 | 10 | 100 | 3 |
| | ID7_2 | 10 | 100 | 5 |
| | ID2_1 | 10 | 100 | 2 |
| | ID5_1 | 10 | 100 | 1 |
| | ID10_1 | 10 | 100 | 10 |
| | ID7_1 | 10 | 100 | 2 |
| | ID4_1 | 20 | 0 | 2 |
| | | | | |
| **Avg** | | | 85,71429 | **3,571429** |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 10 | 100 | 3 |
| | ID1_1 | 9 | 90 | 3 |
| | ID9_1 | 10 | 100 | 5 |
| | ID3_1 | 10 | 100 | 4 |
| | ID1_2 | 10 | 100 | 4 |
| | ID6_2 | 10 | 100 | 4 |
| | ID3_1 | 20 | 0 | 1 |

Figure 99 Results Task 5

| | | Task 6 | | |
|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Time[min.** |
| | ID2_2 | 3 | 100 | **1** |
| | ID7_2 | 3 | 100 | **1** |
| | ID2_1 | 3 | 100 | **1** |
| | ID5_1 | 3 | 100 | **2** |
| | ID10_1 | 3 | 100 | **1** |
| | ID7_1 | 3 | 100 | **1** |
| | ID4_1 | 3 | 100 | **1** |
| | | | | |
| **Avg** | | | 100 | 1,142857 |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 3 | 100 | **1** |
| | ID1_1 | 3 | 100 | **1** |
| | ID9_1 | 3 | 100 | **1** |
| | ID3_1 | 3 | 100 | **1** |
| | ID1_2 | 3 | 100 | **3** |
| | ID6_2 | 3 | 100 | **1** |
| | ID3_1 | 3 | 100 | **3** |

Figure 100 Results Task 6

| | | Task 7 | | | |
|---|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Weighted** | **Time[min.** |
| | ID2_2 | 3 | 100 | **1** | 1 |
| | ID7_2 | 4 | 66,66667 | **13,5** | 9 |
| | ID2_1 | 6 | 0 | **7** | 7 |
| | ID5_1 | 3 | 100 | **2** | 2 |
| | ID10_1 | 1 | 33,33333 | **9** | 3 |
| | ID7_1 | 3 | 100 | **5** | 5 |
| | ID4_1 | 3 | 100 | **2** | 2 |
| | | | | | |
| **Avg** | | | 71,42857 | **5,642857** | 4,142857 |
| | | | | | |
| | | | | | |
| Group B | | | | | |
| | ID6_1 | 3 | 100 | **3** | 3 |
| | ID1_1 | 3 | 100 | **1** | 1 |
| | ID9_1 | 4 | 66,66667 | **4,5** | 3 |
| | ID3_1 | 3 | 100 | **8** | 8 |
| | ID1_2 | 2 | 66,66667 | **6** | 4 |
| | ID6_2 | 3 | 100 | **5** | 5 |
| | ID3_1 | 3 | 100 | **2** | 2 |

Figure 101 Results Task 7

| | | Task 8 | | |
|---|---|---|---|---|
| Group A | | Correct | Time[min.] | Complete |
| | ID2_2 | 100 | 6 | 100 |
| | ID7_2 | 100 | 4 | 100 |
| | ID2_1 | 100 | 10 | 100 |
| | ID5_1 | 100 | 12 | 100 |
| | ID10_1 | 100 | 6 | 100 |
| | ID7_1 | 100 | 7 | 100 |
| | ID4_1 | 100 | 15 | 100 |
| | | | | |
| **Avg** | | | **8,571429** | |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 100 | 30 | 100 |
| | ID1_1 | 100 | 17 | 100 |
| | ID9_1 | 100 | 31 | 100 |
| | ID3_1 | 0 | ? | 0 |
| | ID1_2 | 70 | 40 | 100 |
| | ID6_2 | 100 | 11 | 100 |
| | ID3_1 | 100 | 35 | 100 |

Figure 102 Results Task 8

| | | Task 9 | | |
|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Time[min.** |
| | ID2_2 | 20 | 100 | 6 |
| | ID7_2 | 8 | 40 | 6 |
| | ID2_1 | 18 | 90 | 4 |
| | ID5_1 | 20 | 100 | 3 |
| | ID10_1 | 20 | 100 | 4 |
| | ID7_1 | 20 | 100 | 3 |
| | ID4_1 | 11 | 55 | 4 |
| | | | | |
| **Avg** | | | 83,57143 | 4,285714 |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 20 | 100 | 7 |
| | ID1_1 | 11 | 55 | 3 |
| | ID9_1 | 20 | 100 | 6 |
| | ID3_1 | 20 | 100 | 5 |
| | ID1_2 | 19 | 95 | 5 |
| | ID6_2 | 20 | 100 | 3 |
| | ID3_1 | 20 | 100 | 3 |

Figure 103 Results Task 9

| | | Task 10 | | | |
|---|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Weighted Efficiacy** | **Time[min.** |
| | ID2_2 | 3 | 100 | **1** | 1 |
| | ID7_2 | 3 | 100 | **3** | 3 |
| | ID2_1 | 3 | 100 | **1** | 1 |
| | ID5_1 | 3 | 100 | **1** | 1 |
| | ID10_1 | 3 | 100 | **1** | 1 |
| | ID7_1 | 3 | 100 | **1** | 1 |
| | ID4_1 | 3 | 100 | **1** | 1 |
| | | | | | |
| **Avg** | | | 100 | | **1,285714** |
| | | | | | |
| | | | | | |
| Group B | | | | | |
| | ID6_1 | 3 | 100 | **1** | 1 |
| | ID1_1 | 3 | 100 | **1** | 1 |
| | ID9_1 | 3 | 100 | **1** | 1 |
| | ID3_1 | 3 | 100 | **1** | 1 |
| | ID1_2 | 2 | 66,66667 | **1,5** | 1 |
| | ID6_2 | 3 | 100 | **1** | 1 |
| | ID3_1 | 3 | 100 | **1** | 1 |

Figure 104 Results Task 10

| | | Task 11 | | | |
|---|---|---|---|---|---|
| Group A | | Answer | **Correct** | **Weighted Efficiacy** | **Time[min.** |
| | ID2_2 | 3 | 100 | **1** | 1 |
| | ID7_2 | 3 | 100 | **2** | 2 |
| | ID2_1 | 4 | 66,66667 | **7,5** | 5 |
| | ID5_1 | 3 | 100 | **8** | 8 |
| | ID10_1 | 0 | 0 | **0** | 1 |
| | ID7_1 | 4 | 66,66667 | **7,5** | 5 |
| | ID4_1 | 3 | 100 | **1** | 1 |
| | | | | | |
| **Avg** | | | 76,19048 | | **3,285714** |
| | | | | | |
| | | | | | |
| Group B | | | | | |
| | ID6_1 | 4 | 66,66667 | **6** | 4 |
| | ID1_1 | 1 | 33,33333 | **3** | 1 |
| | ID9_1 | 4 | 66,66667 | **9** | 6 |
| | ID3_1 | 3 | 100 | **7** | 7 |
| | ID1_2 | 1 | 33,33333 | **15** | 5 |
| | ID6_2 | 3 | 100 | **3** | 3 |
| | ID3_1 | 2 | 66,66667 | **6** | 4 |

Figure 105 Results Task 11

| | | Task 12 | | |
|---|---|---|---|---|
| Group A | | Correct | Time[min.] | Complete |
| | ID2_2 | 100 | 17 | 100 |
| | ID7_2 | 100 | 25 | 100 |
| | ID2_1 | 100 | 10 | 100 |
| | ID5_1 | 100 | 22 | 100 |
| | ID10_1 | 100 | 21 | 100 |
| | ID7_1 | 100 | 25 | 100 |
| | ID4_1 | 100 | 40 | 100 |
| | | | | |
| **Avg** | | | **22,85714** | |
| | | | | |
| | | | | |
| Group B | | | | |
| | ID6_1 | 100 | 2 | 100 |
| | ID1_1 | 100 | 7 | 100 |
| | ID9_1 | 100 | 4 | 100 |
| | ID3_1 | 100 | 7 | 100 |
| | ID1_2 | 100 | 8 | 100 |
| | ID6_2 | 100 | 2 | 100 |
| | ID3_1 | 100 | 3 | 100 |

Figure 106 Results Task 12

| | | Task 13 | | | |
|---|---|---|---|---|---|
| Group A | | Correct | Time[min.] | | Complete |
| | ID2_2 | 100 | 8 | | 100 |
| | ID7_2 | 100 | 14 | | 100 |
| | ID2_1 | 100 | 6 | | 100 |
| | ID5_1 | 100 | 3 | | 100 |
| | ID10_1 | 100 | 7 | | 100 |
| | ID7_1 | 100 | 10 | | 100 |
| | ID4_1 | 100 | 15 | | 100 |
| | | | | | |
| **Avg** | | | **9,166667** | | |
| | | | | | |
| | | | | | |
| Group B | | | | Weighted | |
| | ID6_1 | 100 | 15 | **26,78571** | 56 |
| | ID1_1 | 100 | 20 | **20** | 100 |
| | ID9_1 | 100 | 29 | **29** | 100 |
| | ID3_1 | 100 | 17 | **17,89474** | 95 |
| | ID1_2 | 100 | 34 | **91,89189** | 37 |
| | ID6_2 | 100 | 41 | **41** | 100 |
| | ID3_1 | 81 | 30 | **30** | 100 |

Figure 107 Results Task 13

# Lebenslauf

## Persönliche Daten

| | |
|---|---|
| **Name** | Thorsten Keuler |
| **Anschrift** | Mannheimerstr. 25<br>67655 Kaiserslautern |
| **Geburtsdatum und -ort** | 07.12.1978 in Zweibrücken |
| **Familienstand** | Ledig |

## Werdegang

| | |
|---|---|
| **1985 - 1988** | Grundschule, Beuren |
| **1989 - 1998** | Gymnasium, Hermeskeil (Abitur) |
| **1998 - 1999** | Zivildienst, Aach |
| **1999 - 2004** | Studium der angewandten Informatik, Universität Kaiserslautern (Diplom) |
| **seit 2004** | Wissenschaftlicher Mitarbeiter am Fraunhofer IESE, Kaiserslautern |

Kaiserslautern, den 26.11.2010

# PhD Theses in Experimental Software Engineering

**Volume 1** **Oliver Laitenberger** (2000), *Cost-Effective Detection of Software Defects Through Perspective-based Inspections*

**Volume 2** **Christian Bunse** (2000), *Pattern-Based Refinement and Translation of Object-Oriented Models to Code*

**Volume 3** **Andreas Birk** (2000), *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering*

**Volume 4** **Carsten Tautz** (2000), *Customizing Software Engineering Experience Management Systems to Organizational Needs*

**Volume 5** **Erik Kamsties** (2001), *Surfacing Ambiguity in Natural Language Requirements*

**Volume 6** **Christiane Differding** (2001), *Adaptive Measurement Plans for Software Development*

**Volume 7** **Isabella Wieczorek** (2001), *Improved Software Cost Estimation A Robust and Interpretable Modeling Method and a Comprehensive Empirical Investigation*

**Volume 8** **Dietmar Pfahl** (2001), *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*

**Volume 9** **Antje von Knethen** (2001), *Change-Oriented Requirements Traceability Support for Evolution of Embedded Systems*

**Volume 10** **Jürgen Münch** (2001), *Muster-basierte Erstellung von Software-Projektplänen*

**Volume 11** **Dirk Muthig** (2002), *A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*

**Volume 12** **Klaus Schmid** (2003), *Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines*

**Volume 13** **Jörg Zettel** (2003), *Anpassbare Methodenassistenz in CASE-Werkzeugen*

**Volume 14** **Ulrike Becker-Kornstaedt** (2004), *Prospect: a Method for Systematic Elicitation of Software Processes*

**Volume 15** **Joachim Bayer** (2004), *View-Based Software Documentation*

**Volume 16** **Markus Nick** (2005), *Experience Maintenance through Closed-Loop Feedback*

**Volume 17** **Jean-François Girard** (2005), *ADORE-AR: Software Architecture Reconstruction with Partitioning and Clustering*

**Volume 18** **Ramin Tavakoli Kolagari** (2006), *Requirements Engineering für Software-Produktlinien eingebetteter, technischer Systeme*

**Volume 19** **Dirk Hamann** (2006), *Towards an Integrated Approach for Software Process Improvement: Combining Software Process Assessment and Software Process Modeling*

**Volume 20** **Bernd Freimut** (2006), *MAGIC: A Hybrid Modeling Approach for Optimizing Inspection Cost-Effectiveness*

**Volume 21** **Mark Müller** (2006), *Analyzing Software Quality Assurance Strategies through Simulation. Development and Empirical Validation of a Simulation Model in an Industrial Software Product Line Organization*

**Volume 22** **Holger Diekmann** (2008), *Software Resource Consumption Engineering for Mass Produced Embedded System Families*

**Volume 23** **Adam Trendowicz** (2008), *Software Effort Estimation with Well-Founded Causal Models*

**Volume 24** **Jens Heidrich** (2008), *Goal-oriented Quantitative Software Project Control*

**Volume 25** **Alexis Ocampo** (2008), *The REMIS Approach to Rationale-based Support for Process Model Evolution*

**Volume 26** **Marcus Trapp** (2008), *Generating User Interfaces for Ambient Intelligence Systems; Introducing Client Types as Adaptation Factor*

**Volume 27** **Christian Denger** (2009), *SafeSpection – A Framework for Systematization and Customization of Software Hazard Identification by Applying Inspection Concepts*

**Volume 28** **Andreas Jedlitschka** (2009), *An Empirical Model of Software Managers' Information Needs for Software Engineering Technology Selection A Framework to Support Experimentally-based Software Engineering Technology Selection*

**Volume 29** **Eric Ras** (2009), *Learning Spaces: Automatic Context-Aware Enrichment of Software Engineering Experience*

**Volume 30** **Isabel John** (2009), *Pattern-based Documentation Analysis for Software Product Lines*

**Volume 31** **Martín Soto** (2009), *The DeltaProcess Approach to Systematic Software Process Change Management*

**Volume 32** **Ove Armbrust** (2010), *The SCOPE Approach for Scoping Software Processes*

**Volume 33** **Thorsten Keuler** (2010), *An Aspect-Oriented Approach for Improving Architecture Design Efficiency*

Software Engineering has become one of the major foci of Computer Science research in Kaiserslautern, Germany. Both the University of Kaiserslautern's Computer Science Department and the Fraunhofer Institute for Experimental Software Engineering (IESE) conduct research that subscribes to the development of complex software applications based on engineering principles. This requires system and process models for managing complexity, methods and techniques for ensuring product and process quality, and scalable formal methods for modeling and simulating system behavior. To understand the potential and limitations of these technologies, experiments need to be conducted for quantitative and qualitative evaluation and improvement. This line of software engineering research, which is based on the experimental scientific paradigm, is referred to as 'Experimental Software Engineering'.
In this series, we publish PhD theses from the Fraunhofer Institute for Experimental Software Engineering (IESE) and from the Software Engineering Research Groups of the Computer Science Department at the University of Kaiserslautern. PhD theses that originate elsewhere can be included, if accepted by the Editorial Board.



ISBN 978-3-8396-0225-6


Fraunhofer IESE

Technische Universität Kaiserslautern
AG Software Engineering

PhD Theses in Experimental Software Engineering