Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design

Workshop Report,

Lancaster, 21 March, 2004

Bedir Tekinerdoğan
University of Twente
Dept. of Computer Science,
Software Engineering,
TRESE, The Netherlands
bedir@cs.utwente.nl

Ana Moreira
Universidade Nova de Lisboa, Dept. Informática, Faculdade de Ciências e Tecnologia, Portugal,
amm@di.fct.unl.pt

João Araújo
Universidade Nova de Lisboa, Dept. Informática, Faculdade de Ciências e Tecnologia, Portugal
ja@di.fct.unl.pt

Paul Clements
Software Engineering Institute, Carnegie Mellon University, USA
clements@sei.cmu.edu

Abstract

This paper reports on the third Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design Workshop, which has been held in Lancaster, UK, on March 21, 2004. The workshop included a presentation session and working sessions in which the particular topics on early aspects were discussed. The primary goal of the workshop was to focus on challenges to defining methodical software development processes for aspects from early on in the software life cycle and explore the potential of proposed methods and techniques to scale up to industrial applications.

1. Introduction

Conventional aspect-oriented software development (AOSD) approaches have mainly focused on identifying the aspects at the programming level and less attention has been taken on the impact of crosscutting concerns at the early phases of the software development. Current requirements engineering and architecture design approaches, on the other hand, have not explicitly addressed the crosscutting nature of some requirements. The combination of these two issues –the importance of crosscutting concerns at programming level and the impact in the whole system of the decisions made during the early development phases – led to the creation of the Early Aspects research topic in 2002 (www.early-aspect.net). Early aspects are defined as concerns in the early life cycle phases which cannot be localized and tend to be scattered over multiple early phase modules.

Obviously, the early software development phases, including requirements analysis, domain analysis and architecture design, actually set the early design decisions and as such impact the whole system. Therefore, if early aspects are not handled properly, they will, similarly to aspects at programming level, lead to serious maintenance and evolution problems.

This paper reports on the results of the workshop on *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design* which was held on March 21, 2004 in Lancaster, UK. This workshop aimed at supporting the cross-fertilization of ideas in requirements engineering, software architecture design and aspect-oriented software development. It continued the work started at the first and second editions of this workshop held in conjunction with AOSD'2002 and AOSD'2003, respectively.

The outline of the paper is as follows. Section 2 sets Early Aspect within the context of AOSD. Section 3 provides an overview of the topics covered by the workshop. Section 4 presents the

workshop papers. Section 5 shows the workshop program. Section 6 talks about the workshop discussions and results. Section 7 lists the workshop participants. Finally Section 8 presents the conclusions of the workshop.

2. Early Aspects in the context of AOSD

Early Aspects focus on aspects at a higher abstraction level than programming or even design. To make the explicit distinction we categorize aspects as *early aspects*, and *intermediate aspects*, and *late aspects* (see Figure 1).

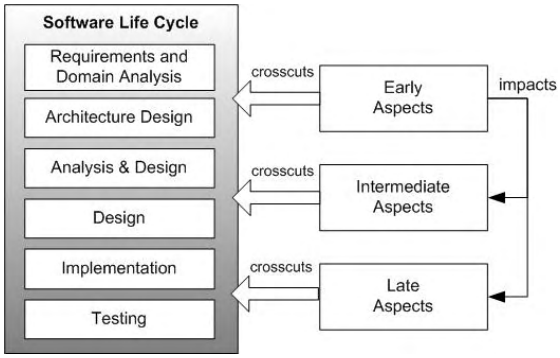

Figure 1. *Relation between Early Aspects, Intermediate Aspects and Late Aspects*

The impact of early aspects on the software development life cycle manifests itself in two ways. First of all, early aspects are crosscutting concerns that are identified in the early phases of the software development life cycle, including requirements analysis, domain analysis and architecture design. Secondly, early aspects also impact aspects in the subsequent phases. Many early aspects identified in the early phases will ripple through the other phases as well. Other early aspects might be specific to the early phases, and crosscut only the specific modules at the early phases. Likewise, it may well be that new aspects will appear as we progress in the software life cycle.

3. Topics

Topics of interest for the workshop included in particular: aspect-oriented requirements engineering, aspect-oriented domain engineering, mapping between aspect-oriented requirements, domain analysis and architecture, aspect-oriented architecture design, tool support and automation for aspect-orientation. A set of open questions for each of these topics, is listed below.

Aspect-oriented requirements engineering

- How to identify aspects at the requirements level?
- How to model aspects at the requirements level?
- How to integrate and compose aspects with other modelling mechanisms, such as goals, viewpoints and use cases, and establish trade-offs?
- How to trace requirements level aspects through later development stages and during re-engineering?
- How to validate aspects identified at the requirements level?

Aspect-Oriented domain engineering

- What are the criteria for domain aspect decomposition?
- How can we derive aspects from domain knowledge?
- How can we abstract and generalize domain aspects for reuse?
- What are the composition relations between domain aspects?
- How to represent domain aspects?

Mapping between aspect-oriented requirements, domain analysis and architecture

- Should the mapping be formal or informal?
- To what is a requirements concern mapped onto?
- What are the language' features required to support a mapping?
- What is the benefit ratio of mapping/coding? What are the pros and cons of mapping in the first place?

Aspect-oriented architecture design

- How to support evolution in the architecture using aspects?
- How to reason about architectures and aspects to know that the architecture is a good one (trade-offs between aspects)?
- How to model the architecture to take aspects into account?
- When designing an architecture, how and when to identify aspects?
- How to set the scope for a software product line architecture using aspects

- Aspects in the Model-Driven Architecture approach

Tool support and automation for aspect-orientation

- Which tools are there to support aspect-oriented development?
- Formalisms and notations for specifying aspects
- What formalisms can be used at early software development stages?

4. Workshop Papers

The workshop received 17 submissions. Table 1 contains a list of the 14 papers accepted for the workshop.

Table 1. List of accepted papers

Title of the paper	Authors
Finding Aspects in Requirements with Theme/Doc	E. Baniassad, S. Clarke
Identifying Aspects using Architectural Reasoning	L. Bass, M. Klein, L. Northrop
Facets of Concerns	C. Bogdan
Integrating the NFR framework in a RE model	I. Brito, A. Moreira
Tracing Aspects in Goal driven Requirements of Process Control Systems	I. El-Maddah, T. Maibaum
Aspect-Orientation from Design to Code	I. Groher, T. Baumgarth
Problems, Subproblems and Concerns	M. Jackson
Aspect-Oriented Context Modeling for Embedded Systems	T. Kishi, N. Noda
Generating Aspect-Oriented Architectures	U. Kulesza, A. Garcia, C. Lucena
Concerned about Separation	H. Mili, A. Elkharraz, H. Mcheick
Refining Feature Driven Development - A Methodology for Early Aspects	J. Pang, L. Blair
On imperfection in information as an "early" crosscutting concern and its mapping to aspect-oriented design	M. Sicilia, E. Garcia
Separation of Crosscutting Concerns from Requirements to Design: Adapting the Use Case Driven Approach	G. Sousa, S. Soares, P. Borba, J. Castro
Modeling Pointcuts	D. Stein, S. Hanenberg, R. Unland

We value the interaction between the participants and the results of the working groups. For this reason used the morning session for a limited number of short presentations and the afternoon was reserved for discussions and overall conclusions. Table 3 contains the 6 papers we have chosen for presentation.

5. Program

The program of the workshop is illustrated in Table 3. The program consisted of two sessions:

1. **Presentation Session**, in which selected papers were presented.
2. **Discussion Session**, in which selected topics on early aspects were discussed.

Table 3. Program of the workshop

8:45-9:00	**Introduction to workshop**
09:00-10:30	**Presentation Session** 1. Finding Aspects in Requirements with Theme/Doc, E. Baniassad, S. Clarke 2. Integrating the NFR framework in a RE model, I. Brito, A. Moreira 3. Tracing aspects in goal driven requirements of process control systems I. El-Maddah, T. Maibaum 4. Generating Aspect-Oriented Agent Architectures, U. Kulezsa, A. Garcia, C. Lucena 5. Identifying Aspects Using Architectural Reasoning, L. Bass, M. Klein, L. Northrop 6. Problems, Subproblems and Concerns, M. Jackson
10:30-11:00	**Morning break**
11:00-14:30	**Discussion Session I** – *Key Problems and Motivations*
12:30-14:00	***Lunch***
14:00-14:30	**Plenary Session:** *Presenting Fundamental Problems + Plenary discussions*
14:00-15:30	**Discussion Session II** - Setting the Research Agenda
15:30-16:00	*Afternoon break*
16:00-17:00	**Discussion Session II Cnt'd** - Setting the Research Agenda
17:00-17:30	***Plenary Session:*** *Presenting the Research Agenda for next years*

5.1 Presentation Sessions

The presentation session consisted of six paper presentations, each of which was presented in 15 minutes. The presented papers and the short

description of these, taken from the original papers, is as follows:

Problems, Subproblems and Concerns
M. Jackson
This position paper sketches how problems may be understood from a perspective based on problem frames. Problem analysis from this perspective reveals structural issues in a clearer light. It leads to a need for composition, both in the problem world and in the solution world. The goals of aspect technology would be clarified by such analysis, and the aspect technology may in turn offer some power in understanding and implementing the compositions.

Finding Aspects in Requirements with Theme/Doc
E. Baniassad, S. Clarke
To identify aspects early in the software lifecycle developers need support for aspect identification and analysis in requirements documentation. To address this, we have devised the Theme/Doc approach for viewing the relationships between behaviours in a requirements document to identify and isolate aspects in the requirements. This paper describes the approach, and illustrates it with a case study and analysis.

Integrating the NFR Framework in a RE Model
I. Brito, A. Moreira
This paper presents a model to handle advanced separation of concerns during requirements engineering. It builds on our work already produced on advanced separation of concerns for requirements engineering by adding two main ideas: (i) the integration of catalogues to help identifying and specifying concerns and (ii) improve the composition rules by informally defining some new operators.

Tracing Aspects in Goal Driven Requirements of Process Control Systems
El-Maddah, T. Maibaum
Goal driven requirements analysis methods are well suited to trace the different aspects of software applications to the early design level. This paper illustrates how to use the GOPCSD tool in developing aspect-based process control applications. The GOPCSD tool adopts goal driven requirements analysis concepts and has been adapted from the KAOS method to address process control systems.

Generating Aspect-Oriented Agent Architectures
U. Kulezsa, A. Garcia, C. Lucena
In this paper we define a domain specific language (DSL) that permits us to model orthogonal and crosscutting agent features. The agent features are then expressed in an aspect-oriented architecture. The implementation of the generative approach encompasses: (i) XML technologies to specify the DSL; (ii) Java and AspectJ programming languages to implement a concrete version of our aspect-oriented agent architecture; and (iii) a code generator, implemented as an Eclipse plugin.

Identifying Aspects Using Architectural Reasoning
L. Bass, M. Klein, L. Northrop
Architectural aspects are candidate aspects to be carried through detailed design and implementation. We set the stage by introducing some new terminology. We begin with a small set of quality requirements for an example system, present a software architecture that satisfies those requirements, and highlight the architectural tactics at work in that architecture. We then identify architectural aspects and their constituent architectural advice, pointcuts, and join points.

5.2 Discussion Sessions

We have deliberatively adopted a very short presentation session to provide more opportunity for discussion.

For the discussion sessions the participants were separated in four sub-groups.

- *Requirements Engineering*, which focused on aspects in requirements engineering
- *Domain Engineering/Application Domain*, focusing on aspects in domain engineering
- *Software Architecture Design*, focusing on aspects in architecture design
- *Specification of Early Aspects*, focusing on defining appropriate notations for early aspects.

Note that these sub-groups were not totally independent and there is some overlap. We did not consider this as a problem, nor did we experienced it later on as a problem. Each sub-group provided some interesting results that we will describe in Section 6.

The discussion session was split in two sub-sessions. The first section, *Key Problems and Motivations* focused on defining the context of the selected topics and the identification of the important problems. The session, *Setting the Research Agenda*, focused on defining the important research problems derived from the first problem.

The two sub-sessions were separated by an intermediate plenary session. The reason for this was to provide a chance to pollinate the ideas and to get feedback from the participants of the other groups.

The tasks for the first session *Key Problems and Motivations* were the following:

- Define a **Mind Map**, providing a conceptual representation of the selected topic based on the input from the participants in the group.
- Identify the Fundamental Problems from the Mind Map.

A mindmap is based on the concept of visual thinking of cognitive science. It is a way of organizing and sharing knowledge. Mindmaps are often developed by a brainstorm session and aim to reflect the common ideas on the domain.

A mindmapping activity starts with writing the main topic into the center of the map, and later on main ideas are linked to the main topic. Each main idea on its turn can have branches describing the detail ideas.

Within the first session each sub-group defined a mindmap of the selected topics. The result of the mindmaps are shown in Figure 2, Figure 5, and Figure 6.

In the second session, *Setting the Research Agenda*, the mindmaps were used to identify the most fundamental problems. The problems were described in the following format:

- **Problem Description,** describing the problem shortly
- **Motivation**, motivating the problem
- **Example**, describing an example

Given the mindmaps and the problem descriptions we aimed at providing a concrete research agenda. The following sections will elaborate on the results of the subgroups.

6. Results of Discussion Sessions

In the following we will describe the results of the workshop for each sub group.

6.1 Requirements Engineering

The mindmap for the topic Requirements Engineering is shown in Figure 52. There are many issues to be discussed and investigated: from identification to validation of aspectual requirements, from composition to traceability of aspectual requirements.

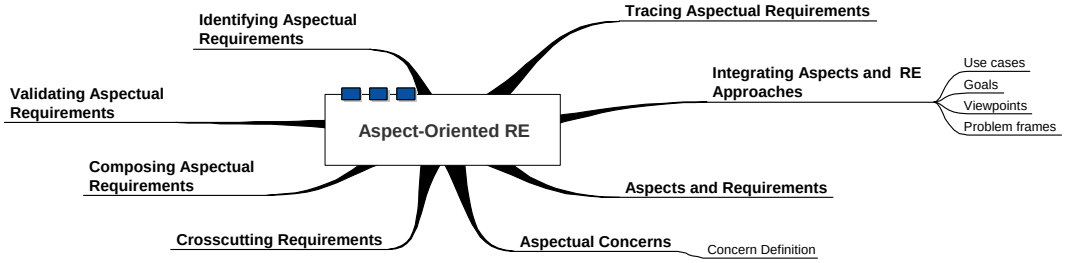

Figure 2. Mindmap for Aspect-Oriented Requirements Engineering

From the list of topics identified in our mindmap the majority of group members decided that we should first concentrate the discussions on understanding the basic concepts behind aspect-oriented requirements engineering: concerns, crosscutting, and aspectual requirements.

Unfortunately, the discussion could not progress to the other topics of the mindmap, because of lack of time.

6.1.1 Problems identified

The following set of questions was intensively discussed, and some answers were proposed.

What is the definition of a concern?

- Is a concern a requirement?
- Is a concern what a stakeholder thinks is important?
- Is a concern anything the developer must consider for the system to be successful?
- Is a concern a user expectation?
- Is a concern anything a developer must consider for the system to be successful?
- Is a concern ANYTHING? (!).

Not having an agreed definition provokes in general communication problems. However, such a definition cannot be rigid, i.e., it is important to have an agreed and flexible one. Moreover, to have a definition is helpful to identify and relate concerns. Our agreed possible definitions are listed below:

- Concern is a desired observable property;
- Concern is a feature;
- Concern is some responsibility;
- Concern is a sub-problem.

What is crosscutting?

- Crosscutting concerns are domain specific concepts that do not fit into an object abstraction.
- A concern is crosscutting when it, or part of it, contributes to multiple concerns.
- Crosscutting arises when two abstractions relate.

What is an aspect?

- It is an artifact to address a requirement.
- It is a situation where N concerns interact.
- It is a crosscutting concern.
- It is a requirement that comes from different points of view.

Why is the lack of a common terminology a problem?

- Because we need to know what we are talking about;
- We need to know how to prepare for design;
- Different versions of “aspect” may be useful in different domains.

6.1.2 Research agenda for the topics discussed

The following points were identified as interesting research topics for the near future:

- Decomposition of concerns;
- Traceability and completeness;
- Appropriate concern representations;
- How to represent specific kinds of subject matter: is there a specialization needed?
- Composition of crosscutting concerns;
- Resolution of conflicts that emerge during composition.

6.2 Domain Engineering and Aspects

(Contribution from the subgroup consisting of: Hafedh Mili, Robert Laney, Bashar Nuseibah, Jianxiong Pang, and Peter Sawyer)

Aspect-oriented software development (AOSD) methods propose new software modularization boundaries that provide additional opportunities for reuse and easier maintenance. Domain engineering is concerned with the development of software artifacts that are reusable across applications within the same application domain. Our group looked at the potential synergies between domain engineering and AOSD. We argue that AOSD is an enabling technology for domain-engineering and propose a number of research directions for the field.

6.2.1 AOSD: an enabling technology for domain engineering

Domain engineering may be defined as the process of developing software artifacts that are reusable across an application *domain.* Domain engineering differs from application engineering in terms of

intent, *process*, and *product*. The **intent** is to develop reusable components. The **product** of domain engineering is a set of development artifacts that are (should be) reusable by design. Those artifacts may not be concrete enough to be used within an application as is; they often require more or less well-defined tailoring mechanisms to make them usable for a specific application. A recurring challenge in domain engineering is to develop general components that are as concrete as the technology of the day permits [Mili et al., 2001].

Figure 3 shows the trade-off between concreteness (usability) and generality (usefulness). New software development techniques enable us to trade less and less applicability for a given "concreteness", and vice-versa. Indeed, reusable components tend to consist of a fixed part, that can be reused as-is, and a variable part, which is often to be supplied by the component user. New modularisation and abstraction techniques typically help us reduce the size of the variable part. For example, by separating interface from implementation, objects enable us to write client code that is not dependent on service implementation.

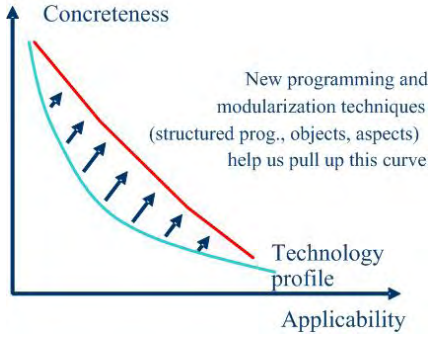

Figure 3. *Trade-off between generality (usefulness) and concreteness (usability)*

Consider the case of two modules M1 and M2 where M1 implements "Grading and logging" and M2 implements "Grading and tracing". Because logging and tracing may crosscut a number classes, those classes will be reusable. If, on the other hand, we are able to separate M1 into two artifacts, one for "Grading" and one for "logging", and M2 into two artifacts, one for "Grading" and one for "tracing", we realize that M1 and M2 will share the "Grading" artifact. Figure 4 illustrates this point. We thus argue that AOSD is an enabling technology for domain engineering.

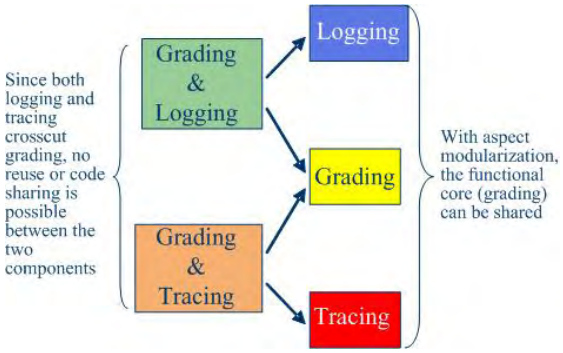

Figure 4. *AOSD supports greater reuse thanks to finer-grained variabilities*

6.2.2 AOSD for domain engineering: a research agenda

Having identified AOSD as an enabling technology for domain engineering (DE), DE will naturally benefit from general advances in aspect-oriented theory and tools and techniques. However, we see more interesting synergies. One promising area of research appeared to us to deal with what we might call *domain engineering of aspects*, described below.

A number of participants have recognized that while user requirements are good sources of concerns, a lot of concerns that pertain to a software system are often implicit. Such is the case with most non-functional requirements, including security, logging, error handling, and the like. Such concerns are business domain independent, and it pays to identify them and to build, if possible, the corresponding artifacts. Domain engineers can reuse such concerns and the corresponding artifacts in their domain models and components, and focus instead on the *functional* concerns of the domain. We discuss the relevant issues in some detail.

6.2.3 Identify common concerns

The goal here is twofold:

1) identify or *catalogue* of common non-functional concerns, and
2) identify the common interactions between such concerns, where applicable.

Neither task is trivial. When building the catalog of concerns, we should strive for completeness, but more importantly, for clear boundaries between the various concerns. For example, traceability and

logging are related. A financial decision support system (e.g. mortage application handler) needs traceability for legal reasons to be able to justify the system's decision. Traceability can be achieved in part through logging, but logging may provide useless output (of non decision-making functions) and miss out on some important decision critical information. The distinction has to be drawn and clarified. *Feature models* in the *Feature-Oriented Domain Analysis* (FODA) [Kang et al., 1990] do a good job of organizing "features" (or concerns[1]) in feature trees, embodying feature containment (the parent feature includes its children features, as in "security" implies "authentication" and "encryption"), feature selection (two alternative features, as in "logging" mapping to either "local logging" or "remote logging"), and dependencies across feature tree branches. An example of the latter is the case where our example grading system needs to run in either local mode or client-server mode: remote logging may only be possible with the client server architecture.

There could be other dependencies besides the "excludes" (between "local logging" and "remote logging") and "requires" relationships ("remote logging" requires "client-server architecture).

6.2.4 Identify relationships between concerns and development artifacts

Some of our concerns may map nicely to identifiable development artifacts while others won't. If we are building an on-line auctioning system for eGolf, and eGold is concerned about throughput (e.g. 10,000 transactions/second for any given item), there is no single artefact, regardless of shape, aspect or color, that will address this concern. However, before we give up on a concern completely, we have to look at its subconcerns: it may be the case that *sub-concerns* may map more easily to artifacts.

"Mappable" concerns may map to classes or methods, and yet others may map to subjects, aspects — in the AspectJ sense [Kiczales et al., 1997] — or composition filters [Aksit & Bergman, 1992]. One the many challenges here is to recognize that not all concerns are aspectual — they don't all look like nails to the aspect hammer. Domain engineers should explore alternative implementations of concerns.

6.2.5 Identify interaction patterns between the artifacts and see how they map back to interactions between concerns

In section 2.2, we identified some dependencies between concerns. Some of these dependencies may translate into interaction patterns between the corresponding artifacts. For example, if a concern *C1* implies another concern *C2*, it means that we cannot incorporate (include, compose or weave) the aspect that embodies *C1*—call it *A1*. Without incorporating the aspect that embodies *C2*—call it *A2*. More complex interactions may occur. We may refer to this case as *essential interaction* between aspects.

There may also be cases where the concerns do not interact, but where the corresponding aspects do. This is a case of *accidental interaction*, which would normally be symptomatic of poor aspect (artifact) design or of a shortcoming of the implementation technolog;y. Either way, such interaction patterns have to be identified, and potential resolutions developed.

6.2.6 Discussion

This is a preliminary investigation into the possible synergies between AOSD and domain engineering that focussed on the reuse of non-functional concerns across application domains. Clearly, domain engineering will also benefit from advances in aspect-oriented implemenation techniques as well as advances in concern identification and aspect modeling to handle domain-specific functional requirements.

6.3 Software Architecture

The mindmap for the topic software architecture is shown in Figure 5.

Software architectures include the early design decisions and embody the overall structures that impact the quality of the whole system. It is generally accepted that architecture design should support the required software system qualities. As shown in Figure 5, *Quality Concerns* forms obviously a key issue in Software Architecture.

[1] For simplicity, we equate feature with concern.

For ensuring the quality factors the common assumption is that identifying the fundamental concerns for architecture design is necessary (*Decomposition)*. A number of approaches have been introduced to derive the fundamental architectural abstractions. The abstractions can be derived from the solution domain or the requirements (*Functional Concerns*).

Although the architecture design approaches vary in deriving architectural abstractions they share the common idea that architectural abstractions should represent the relevant concerns of the system. This implies that for identifying the right architectural abstractions, a thorough understanding and an appropriate application of the separation of principle is necessary.

A solid architecture design heavily depends on and represent solution domain knowledge (*Architectural Knowledge)*. This is derived from the domain knowledge and provided by the architectural patterns.

Composing the architectural concerns is one of the challenging tasks (*Composition)*. In general, architectural concerns can be combined in different ways and it is important to derive the appropriate composition with respect to the quality requirements.

Once the architectural abstractions are identified it is necessary that the architecture is appropriately specified (*Specification*). This can be done visually or textually as in the case of architecture description languages.

The conventional approaches have mainly focused on separating the concerns that fit nicely into architectural components. Unfortunately, less focus has been given on concerns that cannot be captured in single components and tend to crosscut several components.

Crosscutting concerns need to be identified, specified and evaluated at the architecture design level. Software architectures are generally documented using architectural views. The key question here is how to specify aspects in the views. Another issue to investigate is how the crosscutting relates to the views. Can aspects be totally captured in single views, if not how to cope with the crosscutting over different views?

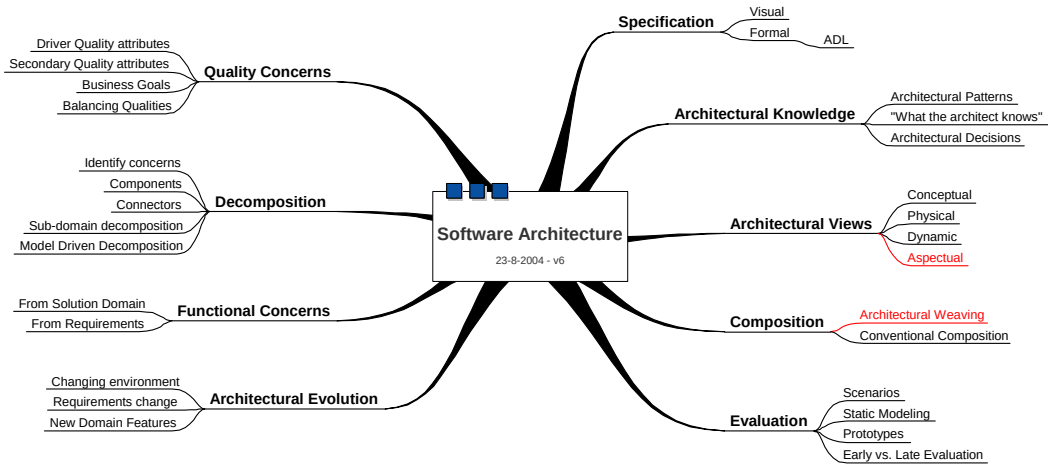

Figure 5. *Mindmap of software architecture topic*

6.4 Specification of Early Aspects

The mindmap for *Specification of Early Aspects* is shown in Figure 6. It should be noted that specification of aspects plays also a role in the other subgroups. During the presentation of the ideas we could also identify some recurring ideas. Specification of early aspects refers to the specification of aspects during the architecture design, requirements analysis and domain analysis phases. Before specifying early aspects it is necessary to identify the aspects first, which is done in the requirements analysis and architecture design phases.

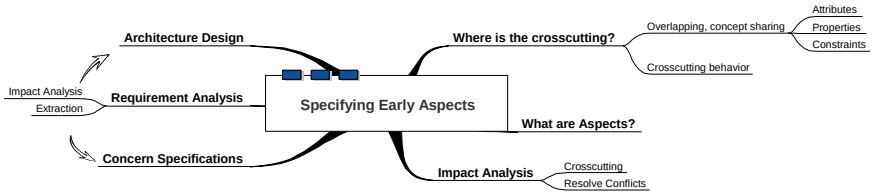

Figure 6. *Mindmap for Specification of Early Aspects*

7. Participants in the Workshop

The participants in this edition of the Early Aspects workshop is listed in Table 2.

Table 2. Participants to the workshop

Participant	Affiliation
1. João Araújo	Univ. Nova de Lisboa, Portugal
2. Elisa Baniassad	Trinity College, Ireland
3. Isabel Brito	Inst. Politécnico de Beja, Portugal
4. Gary Chastek	Software Engineering Institute, US
5. Siobhan Clarke	Trinity College, Ireland
6. Islam El-Maddah	King's College, UK
7. Iris Groher	Siemens AG, Germany
8. Charles Haley	Open University, UK
9. Michael Jackson	Open University, UK
10. Shmuel Katz	Israel Inst. of Technology, Israel
11. Uirá Kulesza	PUC-Rio, Brazil
12. Robin Laney	Open University, UK
13. Marius Marin	Delft Univ. Technology, Holland
14. Hafedh Mili	University of Quebec, Canada
15. Ana Moreira	Univ. Nova de Lisboa, Portugal
16. Bashar Nuseibeh	Open University, UK
17. Jianxiong Pang	Lancaster University, UK
18. Awais Rashid	Lancaster University, UK
19. Pete Sawyer	Lancaster University, UK
20. Miguel-Angel Sicilia	University of Alcalá, Spain
21. Sergio Soares	Federal Univ. Pernambuco, Brazil
22. Daniel Speicher	University of Bonn, Germany
23. Dominik Stein	University of Essen, Germany
24. Stan Sutton	T. J. Watson Research Center, US
25. Bedir Tekinerdogan	Univ. of Twente, Holland

8. Conclusions

The results of the workshop show that the early aspects topic is still in its infancy but progressing. The goal of the workshop was primarily not to find solutions but first to identify the right questions to shape the research of the early aspects topics. During the workshop a number of key research areas have been identified to scope and consolidate the area of early aspects.

Currently we can state that the scope of the early aspects domain is defined to a large extent. This workshop showed that several ideas are recurring with respect to the previous early aspects workshops. In particular there seems now to be an agreement that *early aspects* refers to the aspects that can be identified during the requirements analysis, domain analysis and architecture design phases. This means that aspects during the detailed design are not counted as early aspects. In this report we have termed the aspects at the detailed design as *intermediate-aspects.* Aspects which refer to the crosscutting concerns at the implementation phase, testing and maintenance phases are termed as *late aspects*.

Since early aspects refers to the crosscutting concerns during the requirements analysis, domain analysis and architecture design phases, the research is also focused on these three phases. So far, in general, the research on early aspects appeared to proceed separately and independently in each of these phases. It appears, however, that the early aspects in the three phases are not independent and directly impact each other. A concern such as *synchronization* that is identified during the requirements analysis phases requires the modeling of it during the architecture design. During the domain analysis some aspects might be identified which were overlooked during the requirements analysis phases. There is certainly a relation among the concerns but so far the parity and the semantics of the relations among the concerns in the early phases are not completely clear yet and more research is required to crystallize the concepts.

This workshop and the previous workshop have shown that early aspects exist and that they need to be handled with care to provide better maintainable software. In the future, we expect that the questions addressed in this workshop will be solved gradually.

9. Acknowledgements

We would like to thank the participants to the workshops and the reviewers of the workshop papers.

10. References

[Aksit et al., 1992] M. Aksit, L. Bergmans, and L. Vural, “An object-oriented language-database integration model: the composition filters approach”, *in Proc. of ECOOP 92*, Springer Verlag 1992.

[Domain, 1999] *Domain engineering: A model-based approach*, technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1999. http://www.sei.cmu.edu/domain-engineering.

[Harrison & Ossher, 1993] W. Harrison & H.Ossher, “Subject-oriented programming: a critique of pure objects,” *in Proc. OOPSLA’93*, pp. 411-428.

[Kang *et al.* 1990] Kang, K., S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA): Feasibility Study" CMU/SEI-90-TR-021. www.sei.cmu.edu/publications/documents/90.reports/ 90.tr.021.html

[Kiczales et al., 1997] G. Kiczales, J. Lamping, C. Lopez, “Aspect-Oriented Programming,” in *Proc. ECOOP’*97.

[Mili et al., 2001] H. Mili, A. Mili, S. Yacoub & E. Addy, *Reuse-Based Software Engineering*, Addison-Wesley, 2001

[Mili et al., 2004] H. Mili & A. Elkharraz, “Aspect composition: problems and some solutions,” LATECE Technical Report LAT-3-4-04, 12 pages, March 2004.