Characterization of Early Aspects Approaches

Jethro Bakker, Bedir Tekinerdoğan, Mehmet Akşit

Department of Computer Science,
University of Twente
P.O. Box 217 7500 AE Enschede, The Netherlands

ABSTRACT

Research at the early stages of aspectual software development is in a progressing state and some interesting results have been published. This paper presents an overview of various selected approaches and compares these using criteria derived from a software development framework. The characterization provides a set of open issues that are important for tackling the problems in the identification, specification and evaluation of early aspects.

1. Introduction

Aspect-Oriented Software Development (AOSD) is an emerging technology that provides explicit means to model concerns that tend to crosscut multiple system components. Although, initially the focus was merely on aspects at the programming level, recently a considerable amount of research has been carried out to identify and model aspects in the early phases of software development life cycle. Despite a common and obviously stable notion of aspect at the programming level, the notion of aspect in the early phases, also called early aspects, is not consolidated yet. In our opinion this is not the result of a shallow understanding of the concept of early aspects, but rather the lack of a common basis for discussion around the concepts that have been explored so far.

This paper aims to provide an overview of the different techniques around early aspects that could be used for guiding the research activities. In particular the goal of the paper is as follows:

- *Individual evaluation of the prominent approaches*

Considerable research has been done on the identification and specification of aspects at the requirements analysis and architecture design phases. It is important to understand the underlying concepts of the approaches along with their strong and weak points within an evaluation framework. We have selected the approaches that appear in previous Early-Aspect workshops and we did not try to provide a complete overview of all Early Aspect approaches.

- *Compare the approaches*

Once the approaches are analyzed individually, they have to be compared to determine how much they overlap or complement each other. This may also help the "method designers" to adopt the useful features of each approach.

- *Define the open research issues*

Besides understanding the essential concepts of the state-of-the-art technology, it is also necessary to identify their problems.

The remaining of the paper is organized as follows: In section 2 we discuss the methodological framework for software development. Section 3 will list the prominent approaches that we have identified from the literature and analyze these with respect to the criteria that have been derived from the framework. Section 4 will provide a discussion on the open problems. Finally, section 5 will conclude the paper.

2. Methodological Framework for Software Development

Figure 1 provides a methodological framework for software development [18] which will be used to derive the basic criteria for comparing the various early aspect approaches.

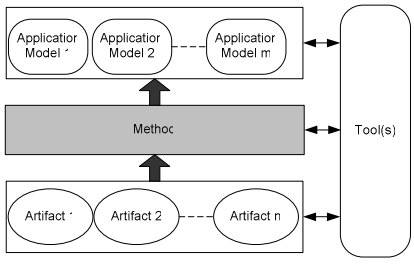

Figure 1. *Methodological framework for Software Development*

The framework consists of four layers. The top layer represents the application models being developed. For example, a UML class model can be considered as an application model. The method layer includes process descriptions, notations, rules and hints to build the application model by composing the basic

modeling/computation artifacts. For example, RUP can be considered as an example of an object-oriented method. Typical examples of artifacts are classes, operations and variables. The supporting layer provides tools, like dedicated compilers and CASE tools.

The framework shown in Figure 1 can be repeated for each phase in the lifecycle. For example, in the requirements analysis phase, the set of artifacts can be scenarios and use cases, the method is a requirement analysis method, and the application model is a model for the given set of requirements. In Figure 1, the dependencies among the phases are not shown. A method generally needs the application model defined in the previous phase. For example, the requirements analysis method can be utilized by the design method in the next refinement phase.

In this lifecycle framework, the phases in the life cycle can be linked together through the required artifacts and defined models. Models are in fact composed artifacts. This implies that given an artifact we should be able to trace the subsequent artifacts.

Based on this framework, we will use the following criteria for analyzing the early aspect approaches:

1. ***Phase***: What part of the life cycle does the approach explicitly consider? Examples are requirements analysis, domain analysis and/or architecture.
2. ***Goal***: What is the goal of the approach? Examples are identification, specification, and/or evaluation of aspects.
3. ***Artifacts***: What are the (descriptions of) the adopted artifacts that are used for the identification and/or specification of early aspects?
4. ***Heuristics rules and Process***: Does the approach use explicitly defined heuristic rules or processes to identify and specify the early aspects?
5. ***Application Model***: What kinds of application models/artifacts are delivered?
6. ***Tool Support***: Is there any tool support?
7. ***Traceability support****:* In case the approach supports more than one phase, are there means to map/trace the artifacts/models? .

3. Analysis of the Approaches

Based on the criteria defined in the previous section, in this section, an analysis of the following early aspects approaches is presented:

- Theme/Doc [4]
- Modularisation and composition of aspectual requirements [11]
- Integrating the NFR framework in a RE model [6]
- Identifying aspects using architectural reasoning [5]
- Aspect-oriented generative approaches [9][10]
- Aspectual software architecture analysis method [16]
- Concern modeling in the Concern Manipulation Environment [12]

We believe that this is a representative list of early aspects approaches that can be found in the public media.

3.1 Theme/Doc

Phase: Requirements analysis

Goal: Identifying and specifying early aspects.

Artifacts: textual requirements, actions, entities and themes. Two kinds of themes are distinguished: base themes and crosscutting themes.

Process and Heuristics Rules: Theme/Doc consists of three sub-processes: Identification of actions and entities form the textual requirements, categorization of actions into themes, identification of crosscutting themes. Each sub process includes heuristic rules. The decision whether a theme is crosscutting is explicitly defined, whereas the rules for identifying actions and entities and the decision which actions are major enough to be a theme remains implicit ("highly intuitive").

Application Model: Theme/Doc provides views that expose which behaviors are co-located in requirements. The *major action view* is used to graphically display the relation between requirements and actions. The *clipped action view* is used to represent the crosscutting themes. Besides of these views the *theme view* is used to plan the design and modeling of the identified themes. Theme views differ from action views in that they do not only show requirements and actions, but also show key elements of the system that will need to be considered for each theme design.

Tool Support: the Theme/Doc tool is currently under development. The tool takes textual requirements, concerns, concern-groups, entities, attachments, associations and postponements as input. This tool creates several views and supports the identification and specification of crosscutting concerns at the requirements analysis phase.

Traceability Support: Theme/Doc is focused on the requirements analysis phase only. There is no traceability support to the domain analysis or architecture design phase.

3.2 Modularisation and composition of aspectual requirements (AORE)

Phase: Requirements analysis

Goal: Identification of crosscutting concerns at the requirements analysis level, specifying and evaluating these concerns.

Artifacts: concerns, stakeholder requirements, a matrix relating concerns and stakeholder requirements, composition rules and a contribution matrix.

Process and Heuristics Rules: AORE provides the following seven process steps that are performed sequentially: identify & specify concerns and stakeholder requirements, identify course grained concern / stakeholders requirements relationships, identify candidate aspects, define composition rules for a candidate aspect and the stakeholder requirements, identify and resolve conflicts among candidate aspects, redefine requirements and specify aspect dimensions [11].

Although, the process is well-defined we can not say the same with respect to the heuristic rules. Some heuristic rules are explicit, such as, the identification of candidate aspects, but many other rules remain implicit.

Application Model: The result of the process is a set of candidate aspects, which are concerns that crosscut multiple requirements.

Tool Support: there is a tool called Aspectual Requirements Composition and Decision (ARCADE). This tool is currently under development. The tool makes it possible to define the viewpoint requirements, aspectual requirements and composition rules using pre-defined templates [11]

Traceability Support: AORE supports identifying and specifying aspects in the requirements analysis phase, but there is no traceability support to other phases of the life cycle.

3.3 Integrating the NFR framework in a RE model

Phase: Requirements Analysis

Goal: Identifying crosscutting concerns.

Artifacts: documentation of stakeholders, catalogues, templates, concerns, and match points. The templates are used to specify a concern. A match point specifies which crosscutting concerns should be composed with a given concern. Composition rules define the order in which the concerns will be applied in a particular match point.

Process and Heuristics Rules: The INFR approach consists of four tasks: identify concerns, specify concerns, identify crosscutting concerns and compose concerns.

Application Model: candidate aspects, which are concerns that crosscut multiple requirements.

Tool Support: currently there is no tool support for this approach available.

Traceability Support: The INFR approach focuses on the requirements analysis phase, and contains no traceability support to other phases of the early software development life cycle.

3.4 Identifying aspects using architectural reasoning

Phase: Architecture Design

Goal: The identifying aspects using architectural reasoning (IAAR) approach [5] tries to derive a software architecture from its quality attribute requirements via insights gained from the quality attribute models created using architectural tactics.

Artifacts: quality attribute requirements, architecture.

Process and Heuristics Rules: contains architectural tactics and those tactics lead to (candidate) architectural aspects. An architectural tactic is a means of satisfying a response measure of a quality attribute scenario by manipulating some aspect of a quality attribute model through architectural design decisions. Architectural join points are well-defined points in the specification of the software architecture. Architectural point cuts are means of referring to collections of architectural join points. An architectural advice is a specification of transformations to perform at architectural join points. Architectural aspects are architectural views consisting of architectural point cuts and architectural advices.

Architectural tactics lead to architectural aspects. But how to identify the architectural tactics? The first process step is to formulate quality attribute requirements according to a scenario format. The second process step is to construct a quality attribute model. The third process step is to convey information among reasoning frameworks. The fourth step in the process is to satisfy conflicting constraints. The last step is the interaction between the design method and the architect. [5]

Application Model: the identified candidate aspects form the application model.

Tool Support: there is currently no tool support available for this approach.

Traceability Support: this approach focuses on the architecture design phase, and there is no traceability support to other phases of the early software development life cycle.

3.5 Aspect-Oriented Generative Approaches

The approach (AOGA) [9, 10] is focused on the Multi-Agent Systems (MAS) domain. The concepts introduced are general and not limited to the MAS domain.

Phase: Domain analysis and Architecture design

Goal: identification and specification of domain and architectural aspects. With the use of aspects modularization of MAS features could be accomplished.

Artifacts: domain concepts represent the artifacts in the domain analysis phase. In the first layer of the architecture

design phase, components and aspectual component with interfaces are the artifacts.

Process and Heuristics Rules: the process is composed as follows, first a domain-specific language (DSL), called Agent-DSL is used to collect and model both orthogonal and crosscutting features of a software agent. Secondly, an AO architecture is designed to model a family of software agents. The architecture is centered on the definition of aspectual components to modularize the crosscutting agent features at the architectural level of abstraction. In the last step a code generator that maps abstractions of the Agent-DSL to specific compositions of objects and aspects in the agent architecture is used.

Application Model: contains two major application models: feature models and component models. Feature models are represented in a domain specific language called Agent-DSL. Agent-DSL uses XML-Schema to represent the features. In the architecture design phase, the aSide modeling language is used to represent the software architecture.

Tool Support: The tool *agent architecture generator* is implemented as an Eclipse plug-in. The tool can read the agent description of the Agent-DSL. The plug-in can generate the classes that represent the elements of a XML-Schema.

Traceability Support: there is a direct trace link between a crosscutting feature in domain analysis and an aspectual component in domain design. [9] So, there is traceability support available from the domain application model to the architecture application model.

3.6 Aspectual Software Architecture Analysis Method (ASAAM)

Phase: Architecture design

Goal: the aim of this approach is to explicitly identify and specify architectural aspects early in the software life cycle. [16]

Artifacts: scenarios and architectural components. There are three types of scenarios: direct, indirect and aspectual. A direct scenario can directly be performed. Indirect scenarios require a change of a component. And aspectual scenarios can either be direct or indirect and are scattered across multiple components. There are four types of components: cohesive component, ill-defined component, tangled component, composite component,

Process and Heuristics Rules: The process consists of five steps: candidate architecture development, development scenario's, individual scenario evaluation and aspect identification, scenario interaction assessment & component identification and refactoring the architecture.[16]

Application Model: a set of architectural aspects that can be used to refactor the architecture. Architecture components are specified using the stereotype: architecture. This is an abstract notation for candidate aspects.

Tool Support: ASAAM has been implemented as an Eclipse plug-in in the tool environment ASAAM-T [15].

Traceability Support: this approach focuses on the architecture design phase, and there is no trace support to other life cycle phases.

3.7 Concern modeling in the Concern Manipulation Environment

Phase: the ConMan approach abstracts from the software life cycle and therefore this model can be used in every phase of the software life cycle.

Goal: ConMan is intended both as a tool for concern modeling and as a platform for developing alternative concern-modeling approaches.

Artifacts: the concern space consists of: units, concerns, relationships and constraints. Units are concern model elements whose purpose is to represent, in the concern model, the work products of software development. In the concern space, each unit refers to an artifact, which represents the details of the software. In ConMan new kinds of artifacts, possibly involving languages not previously handled can be introduced. This includes software (such as requirements, design or programming) languages as well as artifacts like directories, jar files, XML and other documents.

Process and Heuristics Rules: there is no process support and there are no heuristic rules.

Application Model: ConMan manages a concern space, which models a body of software, and concerns of interest within it. The body of software might be a single project, system or component, or a larger body, such as an Eclipse working set workspace.

Tool Support: the Concern Manipulation Environment (CME) tool offers a set of components to create and compose concerns. Currently, the CME supports identifying and modeling concerns at the detailed design level. However, in the future the tool can be used for concern identification and modeling in all phases of the software life cycle.

Traceability Support: there is no explicit traceability support because this approach abstracts from the software life cycle.

4. Evaluation

Table 1 depicts the analysis of the early aspect approaches presented in the previous sections. Using the table data, we derive the following observations:

1. No aspect identification in the Domain Analysis phase

Most of the approaches seem to be focused on either the requirements analysis or the architecture design phases. There is no support yet for aspect identification at the domain analysis phase [2]. Domain analysis aims to identify, capture and organize domain knowledge about the problem domain [3], and as such provides the concepts from the solution domain perspective. Requirements analysis provides the concerns from the problem domain perspective. In fact identification of aspects at the requirements analysis and domain analysis could support each other.

2. Strong focus on Aspect identification

There seems to be a strong focus on identification of aspects in all the approaches. Although, several approaches have been provided to model aspects at the detailed design level, modeling early aspects has not been investigated in detail. We also observe that evaluation of aspects is largely missing. In [11] a trade-off analysis and prioritization of conflicting aspects is presented. Besides analyzing aspects with respect to each other, obviously there should be also a broader evaluation in approaches to assess whether the appropriate aspects are identified, and whether the aspects have been correctly specified.

3. No uniform concern model

Although the notion of concern seems to be the common theme for all approaches, the adopted artifacts for identifying aspects and the adopted concern models are different. Even the approaches defined for the same phase in the lifecycle seem to have differences. For example, the Theme/Doc approach identifies concerns but specifies no properties of the identified concerns but the INFR approach uses a template to describe the concern properties. We can further observe that the definition of the term aspect in several approaches is very similar to the definition of aspects in programming languages [7]. In addition, different approaches apply concepts that are similar to the concepts of *pointcut, joinpoint* and *advice*. However, it is not clear whether these concepts are the best means for coping with crosscutting concerns at the early phases of the software development. For example, what does the notion of *advice* mean for requirements analysis, domain analysis and architecture design?

4. Heuristic rules differ per approach

Although most of the approaches define an explicit process, the adopted heuristics are generally implicitly defined or depend on the intuition of the software engineer. This does not only reduces the understandability of the approaches, but also makes the interpretation of the methods subjective.

5. No stable tool support

It seems that the tools defined for the identification, specification or evaluation of early aspects are not mature yet. Nevertheless, there are extensive efforts for developing tools, for example for Theme/Doc, AORE, INFR and ASAAM approach.

6. No traceability support

Most of the approaches have a strong focus on a dedicated phase of the software life cycle. As such, none of the approaches offers support for traceability between the phases. For providing traceability the mapping from the concerns in one phase to the other should be well-defined. This could be supported by adopting a uniform concern model. ConMan provides a meta-model that can be used to model concerns in every phase of the software life cycle.

5. Conclusion

During the last years several approaches have been presented for coping with aspects at the early stages of the software development life cycle. In this paper we have provided an characterization of selected approaches and compared these with respect to the criteria derived from a methodological framework for software development. The characterization provides a systematic perspective on the different approaches and resulted in the identification of several open problems that are important to identify, specify and evaluate aspects at the requirements analysis, domain analysis and architecture design level. It appears that the focus is merely on identification of aspects during the requirements analysis and architecture design. Further, a uniform concern model for representing aspects in the early stages is missing. Although, the different approaches provide a systematic process, the heuristic rules that are applied are not always explicit.

In our future work we will also consider other early aspect approaches. We will focus on these open issues and especially the identification of aspects during the domain analysis phase. Further, we think that for coping with aspects along the software development life cycle it is important to provide traceability among the aspects in the different phases.

Acknowledgement

This work is supported by the European Network of Excellence on AOSD project.

	Life Cycle	Goal	Artifacts	Heuristic Rules	Application models	Tool Support	Traceability Support
Theme/doc	RA	Ident.	Textual requirements, actions, entities and themes	Action views, rule support and process support for identifying aspects	Theme view	Under Dev.	-
AORE	RA	Ident. Spec.	Concerns, stakeholder requirements, matrix relating concerns and stakeholder requirements and a contribution matrix	Composition rules, process support for identifying and specifying aspects	Candidate aspects	Under Dev.	-
INFR	RA	Ident.	Documentation, catalogues, templates, concerns and match points	Process support for identifying aspects, heuristic rule and composition rules	Candidate aspects, Model of the system	Under Dev.	-
AOGA	DA Arch	Ident.	Domain concepts (Architectural) Components, feature models	Process support for specifying aspects	Architecture model	Feature model XML editor, Agent Architecture Generator Code generation	Yes, direct mapping of crosscutting features
AOAR	Arch.	Ident.	Quality attribute requirements	Process support for identifying candidate aspects	Candidate aspects	-	-
ASAAM	Arch.	Ident.	Scenario's	Heuristic rules and process support for identifying aspects; Explicitly specified	Architectural Aspects	ASAAM-T	-
ConMan	Abstracts	Spec. of Aspects	Concerns, relationships, constraints, units	-	Concern model	CME	-

Table 1. *Evaluation of the selected early aspects approaches*

6. References

[1] J. Araujo, A. Moreira, I. Brito, and A. Rashid. Aspect-oriented requirements with uml. Workshop: Aspect-oriented modeling with UML, 2002.

[2] M.Aksit & L, Bergmans. *Aspect-Oriented Domain Analysis*, tutorial notes, 2nd Aspect-Oriented Software Development Conference, Boston, USA, March 2003.

[3] G. Arrango. *Domain Analysis Methods*. In Software Reusability, Schäfer, R. Prieto-Díaz, and M. Matsumoto (Eds.), Ellis Horwood, New York, New York, pp. 17-49, 1994.

[4] E. Baniassad and S. Clarke. Finding aspects in requirements with theme/doc. 2004.

[5] L. Bass, M. Klein, and L. Northrop. Identifying aspects using architectural reasoning. 2004.

[6] I. Brito and A. Moreira. Integrating the NFR framework in a RE model. In Proceedings of the 3rd Workshop on Early Aspects, 3rd International Conference on Aspect-Oriented Software Development, March 2004.

[7] T. Elrad, R. Fillman, & A. Bader. *Aspect-Oriented Programming*. Communication of the ACM, Vol. 44, No. 10, October 2001.

[8] W. Harrison, H. Ossher, S. M. Sutton Jr., and P. Tarr. Concern Modeling in the Concern Manipulation Environment. IBM Research Report RC23344. IBM Thomas J. Watson Research Center. Yorktown Heights, NY. September 2004.

[9] U. Kulesza, A. Garcia, and C. Lucena. Generating aspect-oriented agent architectures. In Proceedings of the 3rd Workshop on Early Aspects, 3rd International Conference on Aspect-Oriented Software Development, March 2004.

[10] U. Kulesza, A. Garcia, and C. Lucena. Towards a method for the development of aspect-oriented generative approaches. 2004.

[11] A. Rashid, A. Moreira, and J. Araujo. Modularisation and composition of aspectual requirements. In Proceedings of the 2nd international conference on Aspect-oriented software development, pages 11–20. ACM Press, 2003.

[12] S. Sutton & I. Rouvellou. *Modeling of Software Concerns in Cosmos*, in: Proceedings of First Aspect-Oriented Software Development Conference, pp. 127-134, Enschede, The Netherlands, 2003.

[13] S. M. Sutton and P. Tarr. Aspect-oriented design needs concern modeling. 2002.

[14] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton. N degrees of separation: Multidimensional separation of concerns. 1999.

[15] B. Tekinerdogan & F. Scholten. ASAAM-T: A tool environment for identifying architectural aspects. Demo at AOSD 2005 conference, Chicago, 2005.

[16] B. Tekinerdogan. ASAAM: Aspectual software architecture analysis method. In WICSA 4th Working IEEE/IFIP Conference on Software Architecture, pages 5–14. IEEE, 2004.

[17] B. Tekinerdogan, A. Moreira, J. Araujo, P. Clements. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design: Workshop Report AOSD 2004, TR-CTIT-04-44, 119 pp, University of Twente, Dept. of Computer Science, October 2004,

[18] B. Tekinerdogan, M. Saeki, P. van den Broek, G. Sunyé, & P. Hruby. Automating Object-Oriented Software Development Methods. in: A. Frohner, Object-Oriented Technology, ECOOP 2001 Workshop Reader, Lecture Notes in Computer Science. Vol. 2323, Springer Verlag, 2002.