Linking Goals to Aspects*

Charles Zhang and Hans-Arno Jacobsen
Department of Electrical and Computer Engineering
and Department of Computer Science
University of Toronto
{czhang,jacobsen}@eecg.toronto.edu

Yijun Yu
Department of Computer Science
University of Toronto
yijun@cs.toronto.edu

ABSTRACT

In RE models such as goal-oriented models, a complex system is directly described in terms of its purposes, which makes its functionality much easier to understand and to reason as compared to code-level implementations. Part of the difficulty in maintaining a stronger correspondence between requirements and code is possibly due to the sufficient modularization capabilities of traditional architectures where many functionalities do not exist in distinct modular entities. This paper reports on an investigation of *how* and *where* some distinct design requirements lead to crosscutting concerns when decomposed into code in goal models such as KAOS. We begin by matching our past experience in aspect discovery at the code level with a detailed requirements modeling of the same architecture in KAOS. The discovered patterns are validated in an independent project where the requirements modeling and the aspect identification are separately conducted. We observe that satisfying OR-decomposed subgoals in the KAOS model typically leads to tangled implementations, and agents responsible for multiple OR-refined goals should be implemented in the aspect-oriented manner.

Keywords

Aspect Oriented Programming, Goal-Oriented Requirement Engineering

1. INTRODUCTION

In complex software systems, it is typically a challenging task to maintain direct correspondence between distinct features or characteristics at the requirements level and code or modules at the implementation level. For example, in [17], we have shown that many features of middleware platforms do not exist in modular forms. This is a problem of "losing design in code"[1], a result of the difficulty in maintaining modularity over rapid evolutions, extensions and expansions. The modularization capability of traditional languages and architectures is becoming insufficient in accommodating the magnitude of the diversification in both feature and application domains. This is of costly consequences in achieving adaptability, configurability, customizability, and maintenance. A new school of thoughts aiming at alleviating this problem focuses on improving the modularization capabilities of traditional languages by offering new types of modules such as features [11], aspects [7], subjects [6], and composition filters [2]. A common characteristic of these new techniques is that they are capable of breaking the modularity boundaries of conventional languages and offering a finer modularity to express different views of the target system. However, these techniques are elaborated more as mechanisms than methodologies. For aspect-oriented software development (AOSD), it is also important to study what kind of correspondence exists between requirement specifications and traditional architectures. These correspondences serve as valuable advices in effectively adopting AOSD into the requirement engineering process.

In search for this correspondence, we focus on discovering patterns in the organization of concepts or requirements that govern the purpose of software construction. These patterns can very likely lead to code tangling in the final implementation, We make the assumption that, as in the case of object oriented analysis, aspect orientation occurs later in the architecture and implementation stages after requirements are specified. Therefore, instead of proposing requirement engineering (RE) treatments directly mapped to aspects, we examine existing well-established RE models and identify patterns in these models that could be better designed and implemented using aspect-oriented programming. The RE model of choice is the knowledge acquisition in the automated specification framework (KAOS) [5], we carry out our analysis in the following stages:

1. Aspect identification: We first start from the code level and, through the technique of aspect mining, identify application features existing in a crosscutting fashion.
2. Requirements modeling: We then perform a consolidated modeling of the application requirements in terms of KAOS concepts.
3. Pattern discovery : Through the comparison between the goal decomposition and the actual code decomposition, we try to identify the connection patterns in the goal decomposition graph which will give rise to

*MSRG Technical Communication, University of Toronto
[1]Gregor Kiczales URL:`http://www.cs.ubc.ca/gregor`

aspects. This then helps us to link the existence of aspects to the properties and the interactions of requirements.

4. Cross validation: The patterns discovered in one RE model are validated in another application context where the application domain is dramatically different, the requirement models are independently developed, and aspect identification is separately carried out.

Following this methodology, we first leverage our extensive knowledge [16, 17, 18] about implementation-level aspects in middleware systems and map these aspects to the goal decomposition of middleware design requirements. An experimental requirement modeling is then conducted to consolidate a large variety of common middleware functionalities. Our preliminary comparison of the requirement model against implementation-level aspects reveals that supporting the **OR** relationships between *goals* usually cause violation of cohesion in modules and, hence, should be dealt with AOSD concepts at the implementation level. The **OR** relationship means that satisfying a particular design goal is equivalent to satisfying either of its many possible refinements. The **Agent** entity, which is responsible for executing the goals, is likely to contain aspect functionality in the code if it is responsible for multiple OR-related goals. This pattern is then validated in an e-commerce application, *Mediashop*, where KAOS methodology is previously used and aspects are independently identified.

The rest of the article is presented as follows: Section 2 gives a brief introduction of middleware, the RE modeling language, KAOS, used in this analysis, and the e-commerce application used for cross-validation. Section 3 discusses how patterns in the KAOS model are identified in analyzing aspect orientation. Section 4 reports the results of matching these patterns in the Mediashop application. Section 5 discusses related work.

2. BACKGROUND

Middleware

The term "middleware" has various interpretations. In the context this discussion, we focus on middleware that facilitates the development of distributed systems in a heterogeneous networking environment. Middleware can be categorized depending on its identity coupling and temporal coupling characteristics among participants who request services (clients) and ones who provide computing services (servers). Common middleware technologies include DCOM, CORBA, J2EE and Web Service.

KAOS

The knowledge acquisition in automated specification framework (KAOS) [5] is a methodology for obtaining, explaining and justifying design requirements. The essence of KAOS consists of a set of entities representing concepts in deriving design requirements from goals. A goal is an objective of the composite system which can either be decomposed into subgoals or achieved through some responsibility of an agent. A goal can be achieved in the equivalence of achieving all or either of its multiple subgoals. Therefore, a goal can be further decomposed or refined into more specific goals in either AND or OR relationships with each other. We will see more examples of KAOS diagrams in the next section.

Mediashop

The *Mediashop* is an example e-commerce application initially used in [3] and modeled with goal models in [15] where the relationship between goal models and implementation-level aspects are also studied. It implements an online store "selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, and the like." [3] For more details of this application, please refer to [3, 15].

3. ASPECTS IN KAOS MODEL

We believe that functionalities manifested as aspects at the implementation level have patterns or characteristics when specified as requirement concepts either in words or in visual shapes. Our approach is to first reverse-engineer existing systems with requirements modeling and to compare requirement models against our extensive knowledge about implementation-level aspects. We then discover patterns in requirement specifications and validate these patterns in independent requirement models where aspect identification is separately conducted. This and the next sections describe our approach in detail.

3.1 KAOS modeling of middleware

The design requirements of middleware are extraordinarily complex to model even in our preliminary attempt. We map middleware requirements to commonly supported features extracted from a number of middleware design documents [8, 14, 9] and synthesized with our own knowledge of middleware functionalities. These features are categorized as to support five essential goals of middleware systems including identity binding, temporal binding, inter-program communication, and others. Figure 1 shows a glimpse of the consolidated view of the KAOS model for achieving part of the enumerated goals. The entire model, although still simplistic and abbreviating, consists of over 200 concepts, while only implementing 6 out of 12 high level goals, if we count the five essential middleware goals, and their immediate sub-goals as high level goals. This degree of complexity will almost certainly lead to unmanageability at the level of architecture and code.

3.2 Architectural and code level aspects

Plain code inspection of middleware implementations reveals a similar problem as described earlier: a single architecture often supports multiple distinct functionalities which are alternative solutions to the same problem. Each functionality is often not cleanly modularized and difficult to identify, configure, and maintain. We categorize this phenomenon as concern crosscutting. Concern crosscutting is an inherent phenomenon in legacy middleware implementations. In [16], we have observed that over 50% of all classes in three different mature middleware implementations are crosscut by a certain concern, or, equivalently speaking, an aspect. Concern crosscutting occurs at two levels: architectural level where aspects exist in parts of class compositions, i.e., properties and methods, and code level where aspects

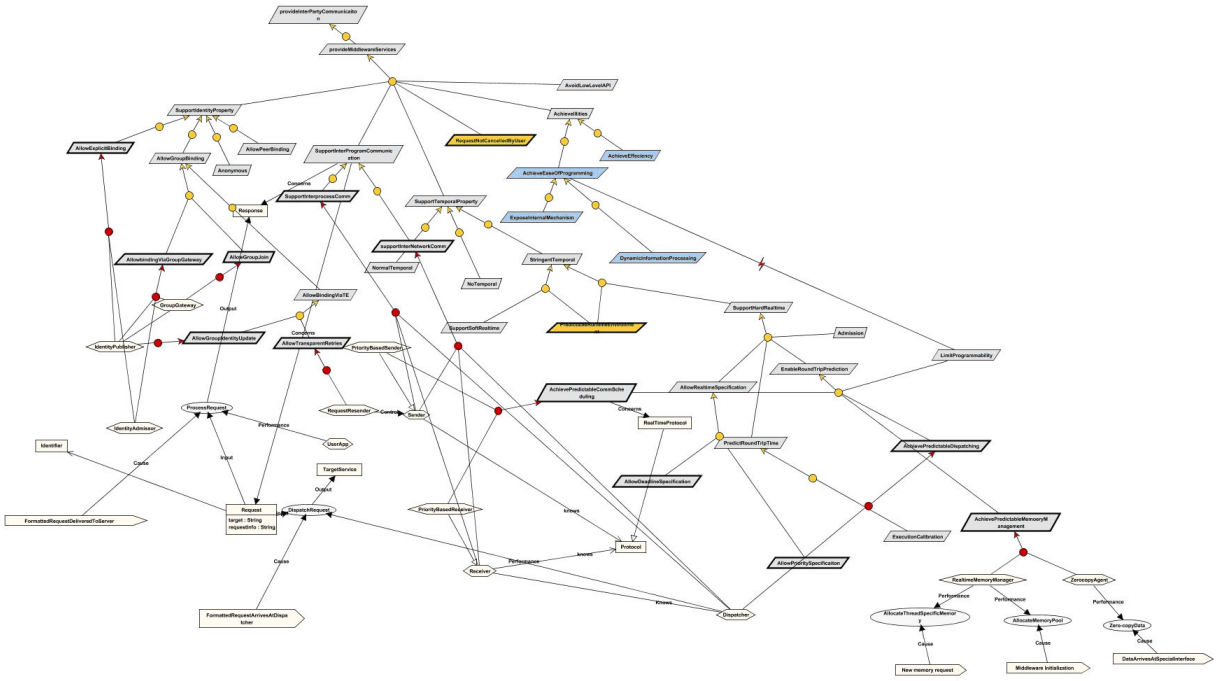
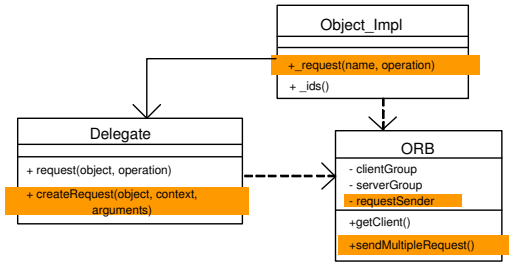


Figure 1: Consolidated requirements modeling in KAOS

exist in interactions among classes, i.e., in implementations of class methods.

Figure 2 (A) illustrates an example of crosscutting at the architectural level. In this simple class hierarchy of a group of CORBA classes, the highlighted attributes and operations are part of the support for the dynamic composition of remote CORBA requests, whereas typical CORBA remote requests are composed statically. The functionality of dynamic request composition, although semantically independent from the static request composition, does not exist in separate modules but rather spreads across the main middleware class hierarchy. Crosscutting also happens at the code level. Figure 2 (B) shows crosscutting among multiple functionalities in an actual ORBacus code snippet. In this short piece of code, three concerns are present: portable interceptors (PI), which allow the normal call flow of the ORB to be intercepted and altered; oneway, which supports the asynchronous communication style; and dynamic programming interface (DII), which supports dynamic request composition at the client side. This high degree of tangling undoubtedly makes all three functionalities harder to understand, to change and to configure.

A:



B:

```
public Downcall createPIDIIDowncall(String op,
 boolean resp,
 //arguments omitted) throws FailureException{
  com.ooc.OCI.ProfileInfoHolder profile =
  new com.ooc.OCI.ProfileInfoHolder();
  Client client = getClientProfilePair(profile);
  Assert._OB_assert(client != null);

  if(!policies_.interceptor)
   return new Downcall(orbInstance_, client,
         profile.value, policies_, op, resp);

   PIManager piManager = orbInstance_.getPIManager();
   if(piManager.haveClientInterceptors()){
    return new PIDIIDowncall(orbInstance_, client,
           profile.value, policies_,
           op, resp, IOR_, origIOR_, piManager,
           args, result, exceptions);
   }
   else{
    return new Downcall(orbInstance_, client,
        profile.value, policies_, op, resp);
   }
}
```

Oneway

Validation

Interceptor

Figure 2: A.Architectural-level crosscutting B. Code-level crosscutting

3.3 Mapping OR-Reduction to Aspects

The OR relationship in the KAOS model entails that there are alternative ways of achieving the same goal. This is an un-dictating view of how system functionality should be reasoned. In contrast, traditional construction paradigms of complex systems are far less flexible in terms of implementing alternative decompositions. Aspect oriented programming pays more respect to the multiview problem and

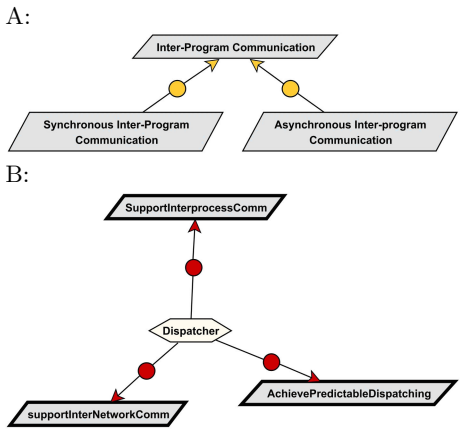


Figure 3: A. OR decomposition of middleware goals B. Agent responsible for multiple goals

defines concern crosscutting as "In general, whenever two properties being programmed must compose differently and yet be coordinated, we say that they cross-cut each other." [7] The referred "two properties" are the ease of reuse and the efficiency of memory use. They are two alternatives for implementing the same system, which appeals to different stakeholders: a system architect for the former and an end-user for the latter. We think this view is very similar to the concept of the OR-reduction of goals in the KAOS model. In addition, a goal in KAOS is a collectively achieved functionality by multiple agents. This conceptually corresponds to the scattering problem which is a sufficient condition for the use of aspects.

The inspection of the aspect analysis in existing middleware architecture corresponds to this conjecture. Our previous aspect analysis work [17] has shown that a number of features within the monolithic middleware core architecture are not simultaneously modularized in code and orthogonal to other middleware functionality in semantics. For instance, one of the essential middleware design requirements, the support of inter-program communication, can be satisfied by two alternatives – supporting communication between programs either over the network or across address spaces. In KAOS, this is expressed in terms of **ORed** *sub-goal* decompositions or *goal refinements* as illustrated in Figure 3. In practice, it is actually common to implement both alternatives simultaneously to support different types of user applications. Our aspect analysis has shown that the support for inter-process communication is an aspect in the presence of supporting inter-network communications. Similarly, **ORed** goal decomposition also happens when supporting communication styles in both synchronous and asynchronous ways and when providing both static and dynamic programming styles. These properties are proven to be better modularized through aspects [18].

Therefore, we state the following pattern:

A goal involved in an OR-relationship is likely to cause concern scattering and should be treated in the aspect-oriented way.

From the previous conjecture, alternative subgoals give rise to the use of aspects. It is also interesting to know how effectively the KAOS models can predict places in the architecture where AOP can be actually applied. We think the KAOS concept of *agent* has very close correspondence to architectural entities represented by classes because an *agent* represents an autonomous computing unit which encapsulates certain functionalities and carries clear responsibilities. Agents involved in OR relationships must simultaneously handle different responsibilities. Therefore, it is hard to achieve coherent architectures to implement these agents due to dramatic differences of supporting multiple alternatives of the same parent goal. For example, the responsibility diagram generated by Objectiver[2] in Figure 3(B) shows that the `Dispatcher` agent, which is initially incepted to support inter-network communications, supports multiple goals (requirements) at the end of the modeling process. The inspection of the code shows that the corresponding architectural entities of `Dispatcher` supports both inter-network and inter-process communication in a tangled fashion. Same phenomenon can be observed on other agents such as `Sender` and `Receiver`. Their requirement-level interactions with multiple alternative goal refinements well explain the tangled realizations of these agents at the code-level.

Therefore, we state the following pattern:

Agents which are simultaneously responsible for multiple subgoals of certain goals should be decomposed in an aspect-oriented manner.

4. CROSS VALIDATION

Patterns identified using reverse engineering are likely to be subject to the influence of implementation choices of particular applications. Therefore, to further validate these patterns, we examine the goal model of a completely different type of application obtained in an independent project reported in [15].

The media shop is an e-commerce application for selling products online. This application has been a subject of study by goal-oriented requirements engineering [4]. In the open-source community, osCommerce[3] is a product of such an application. Since we did not have its original requirements engineering documents, a reverse engineering process was conducted on the code artifacts to identify its functions (hard goals) and quality attributes (soft goals). Functions are a result of domain-specific decomposition of the media shop system, such has *ShoppingCart*, *InventoryReport Administration*, etc. Quality attributes are a result of domain-independent concerns for the system, such as *Usability, Responsiveness, Integrity, Security*, etc. They must be correlated in order to satisfy the quality attributes through non-functional requirements on top of the functional requirements. In other words, the quality must be built or weaved into the end-product. These correlations, observed by our case study [15], are identified as the weaving of aspects.

More specifically, the candidate aspects found through an analysis of the reverse engineered goal model are given in [15].

[2]Objectiver. URL:`http://www.objectiver.com`
[3]osCommerce URL:`http://www.oscommerce.com/`

In order to cross-validate our claim that OR goals are the source of aspects, we explain the goal analysis for the functionality of each aspect found in [15].

1. **Security:** *HTTPS/SSL check[protocol]* and *Password-Protection[access]* are two identified security aspects. They are used as optional tasks in the requirements: to enable or disable high security. In a goal model, such optional goals are represented by OR rules.

2. **Responsiveness versus Integrity:** *SessionCookies* and *DatabaseTransactions* are two ways to implement session persistence for the *ShoppingCart*, *ProductInfo*, *AccountManagement* and *ReportGeneration*. *SessionCookie* is more responsive than *DatabaseTransactions* for transactions as frequent as *ShoppingCart*; on the other hand, *DatabaseTransaction* delivers higher integrity for *AccountManagement*, *ReportGeneration* and *ProductInfo*. Therefore, a trade-off was made in the design of the media shop such that *ShoppingCart* is implemented through session cookies and the other three tasks are implemented through database transactions. These trade-offs involve a preference elcitation to pick the proper subgoal of an OR decomposition rule according to the quality concern.

3. **Usability:** The natural language used in the media shop has a large impact on the usability of the system since the clients of the system are human beings, rather than softbots. *Customization[Language]* is thus a highly crosscutting aspect that involves almost all subsystems of the media shop that output natural language strings to the clients. In osCommerce, such an aspect has at least four implementations, one for English, German, French, and Portuguese respectively. Thus, the language of choice is an example of the OR subgoals. Another aspect *Similar[Look and Feel]* is another operationalization of the usability, which reduces the memorizability burden for the end-users. In particular, the *Infobox[templates]*, *Similar[fonts and colors]*, *Common [navigation bars]* all contribute to the goal. Although implemented in the media shop as mandatory parts by the designers, they can be detached from the implementation as aspects and weaved into another system to improve its reusability. Moreover, different themes and plugins in osCommerce can replace the look and feel, without affecting other parts of the system. Therefore, we consider them also a result of OR rules.

In short, all the identified aspects in the media shop case study involve some form of the OR rule, such as options, choices, trade-offs or substitutable parts of the system. Of course, such an observation is not a coincidence. Aspects are inherently associated with OR rules, because they crosscut different parts of the system to implement non-functional requirements. The functional parts at the join point still satisfy their parent goals regardless of the weaving. On the other hand, quality attributes must be affected by the introduction of aspects.

5. RELATED WORK

The work in [15] describes an approach of using patterns of goal decomposition graph to discover aspects in early requirements. In this work, candidate aspects are identified if a goal is being intensively dependent upon by a number of other goals. A web application is then analyzed to verify whether goal aspects correspond to actual aspects in code. Our approach is inspired by this work and complements it by matching requirement engineering models with our empirical aspect analysis accumulated from our past experience. We contribute another explanation of why and where aspects would exist deriving from the requirements model.

Rashid *et al.* [12] presents a similar approach to [15] by assessing the relationships between concerns and requirements in the aspect oriented requirements engineering framework (AORE). If a particular concern is present in multiple requirements, it is labeled as a candidate aspect. Baniassad *et al.* [1] has proposed Theme/Doc, a methodology for extracting features, termed "themes", from the texts of requirements. Aspect identification is carried out by a tool through checking if any requirement is associated with more than one theme. Rosenhainer [13] proposes a related aspect identification technique which uses information retrieval methods to examine requirement specification texts. For each requirement, patterns in its detailed explanations are identified to locate scattered descriptions of other requirements. This type of scattered requirement specification likely leads to scattering in the implementation.

Navarro *et al.* [10] proposes the addition of a new entity in the meta-model of goal-oriented requirement engineering to directly represent crosscutting concerns. The identification of such entities in specific application contexts is not discussed in detail.

6. CONCLUSION

Requirement models are the very first artifacts generated in the software engineering process. In terms of aspect oriented software development, many new requirement acquisition models have been proposed to directly model crosscutting concerns. We believe that many well-established requirement modeling methodologies are sufficient in capturing the intent of the software system. Concern crosscutting at the code level originates from the insufficient capability of conventional modularization techniques. Therefore, in this work, we aim at discovering patterns in existing requirements models that likely lead to tangled implementation in the code. Our task starts from a simplified and less formal requirement modeling of middleware using the KAOS goal-oriented requirements modeling framework. These design goals and requirements are extracted from several research works and author's own experience. By matching the requirements model with our knowledge of code-level crosscutting concerns, we observe that the goals in OR-relationships are likely to give rise to aspects, and the architectural entities corresponding to Agents handling multiple "ORed" goals are likely to suffer from tangled implementations. We then verify these patterns using an independently developed goal model for a different application. The aspect identification in this application confirms that crosscutting features can be traced to OR-decomposed goals in the model.

Our goal modeling has only leveraged part of the power of

the modeling framework as none of the definitions in the presented model employs formal logic descriptions. The KAOS model, for example, use linear temporal logic to formally describe concepts. Tropos [4] framework uses probabilistic Bayesian networks to conduct quantitative reasoning. Formal definitions allow verification of inconsistencies and inference of new interactions in achieving middleware goals. And perhaps new insights regarding how aspects can be discovered or applied can come into light. This is an undoubtedly interesting future exercise. Our aspect analysis so far has been conducted from a retrospective. To further validate the idea, the modeling should be conducted independent of the aspect knowledge, and the aspect analysis should involve creating new systems. It is interesting to see how such exercises further illustrating a more direct correspondence between requirements and implementations can be achieved by aspect oriented programming

7. REFERENCES

[1] E. Baniassad and S. Clarke. Finding aspects in requirements with theme/doc. In Early Aspects Workshop 2004, held in conjunction with the 3rd International Conference on Aspect-Oriented Software Development, March 22-26, 2004, Lancaster, UK.

[2] Lodewijk Bergmans and Mehmet Aksits. Composing crosscutting concerns using composition filters. *Commun. ACM*, 44(10):51–57, 2001.

[3] Jaelson Castro, Manuel Kolp, and John Mylopoulos. A requirements-driven development methodology. *LNCS*, 2068:108–??, 2001.

[4] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems*, 27(6):365–389, 2002.

[5] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. pages 3–50, 1993.

[6] William Harrison and Harold Ossher. Subject-oriented programming: a critique of pure objects. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 411–428. ACM Press, 1993.

[7] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[8] Anna Liu. Gathering middleware requirements. In *ICOIN '01: Proceedings of the The 15th International Conference on Information Networking*, page 81. IEEE Computer Society, 2001.

[9] Victor B. Lortz, Kang G. Shin, and Jinho Kim. Mdarts: A multiprocessor database architecture for hard real-time systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):621–644, 2000.

[10] E. Navarro, P. Letelier, and I. Ramos. Goals and quality characteristics: Separating concerns. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, to be held in conjunction with ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), October 24-28, 2004, Vancouver, Canada.

[11] Christian Prehofer. Feature-oriented programming: A fresh look at objects. In *ECOOP*, volume 1241 of *Lecture Notes in Computer Science*, page 419 ff, 1997.

[12] Awais Rashid, Ana Moreira, and Jo ao Araújo. Modularisation and composition of aspectual requirements. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 11–20. ACM Press, 2003.

[13] L. Rosenhainer. Identifying Crosscutting Concerns in Requirements Specifications. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, to be held in conjunction with ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), October 24-28, 2004, Vancouver, Canada.

[14] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The design of the tao real-time object request broker. *Computer Communications*, 21(4), April 1998.

[15] Yijun Yu, Julio Cesar Sampaio do Prado Leite, and John Mylopoulos. From goals to aspects: Discovering aspects from requirements goal models. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International (RE'04)*, pages 38–47. IEEE Computer Society, 2004.

[16] Charles Zhang and Hans-Arno Jacobsen. Quantifying Aspects in Middleware Platforms. In *2nd International Conference on Aspect Oriented Systems and Design*, pages 130–139, Boston, MA, March 2003.

[17] Charles Zhang and Hans-Arno Jacobsen. Refactoring Middleware with Aspects. *IEEE Transactions on Parallel and Distributed Systems*, 14(11):1058–1073, November 2003.

[18] Charles Zhang and Hans-Arno Jacobsen. Resolving Feature Convolution in Middleware Systems. In *Proceedings of the 19th ACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and Applications*, September 2004.