

Separation of Concerns and Roles in the Object Coordination Nets Approach

Holger Giese
AG Softwaretechnik, Universität Paderborn
Warburger Str. 100, 33098 Paderborn
hg@upb.de

Abstract

Cross-cutting aspects and role-based modeling are two complementary approaches to achieve separation of concerns. The Object Coordination Nets (OCoN) approach for distributed system design and architecture is used to study both in a common environment. The OCoN approach emphasize the contract principle in form of role-based modeling. The different provided roles of a subsystem are represented by distinct contracts to enable their separation during design. The support for secure run-time binding of contracts further permit to consider the case of more dynamic and open systems. W.r.t. the OCoN approach the underlying concepts and the achieved degree of separation for role-based modeling and in the domain of distributed and open systems are described. The integration of dynamic service lookup to support also cross-cutting aspects and the limitations of such a procedure are further addressed.

1. Introduction

The classical notion of *separation of concerns* [25] is one of the fundamental principles to break complexity in software engineering. The common notion of modules with interfaces leads to a *dominant dimension of separation* [30] and thus, depending upon design decisions, other design aspects become *cross-cutting aspects* [18]. *Multi-dimensional separation of concerns* [30, 15] has therefore been proposed to support the separated handling of different aspects even when one dominant decomposition is given.

The main idea is to independently develop suitable functionalities of the system and to automatically *weave* them together later on. However, the individual aspects — at least in the original proposals — are required to be orthogonal to each other. For this reason, the proposals mainly pertains to crossover functionality like, for instance, persistence or error handling. Moreover, weaving is carried out on the level of program code in contrast to design as addressed in this paper.

The idea of a separation of concerns on an architectural level is supported by most modern models of the software development process, e.g. the *Rational Unified Process* [19]. However, while these models generally

agree to consider software architecture and design as a multidimensional combination of various specific views on the system, a well-defined formal foundation as well as a methodology for the composition of independently developed, non-orthogonal aspects is currently missing.

Traditional object-oriented design methods and notations [3, 28, 16] and the UML [24] standard are appropriate and specialized to support programming language-based development. A seamless transition from the design to the implementation cannot be achieved for even simple concurrent or distributed applications, because the composition of behavioral aspects is assumed to be orthogonal. For distributed systems however the phenomena of concurrency and asynchronous communication and their semantical impact cannot be ignored.

A suitable separation between the objects of a design is often missing, which leads to unexpected dependencies and results in several problems during implementation. A lack of separation becomes even a bigger drawback when the system evolves and the size of the system grows or when subsystems are distributed using middleware approaches such as CORBA or DCOM. For such systems, a higher emphasis upon a sufficient software architecture including a stronger need for separation is therefore essential.

Specification of component behavior based on commitments and assumptions is often referred to as *design by contract* [21] and leads to a better separation between components of the system. Additionally, *role-based modeling* as presented in [27] supports several, dedicated views on a system or component which may lead to suitable contracts. Consequently, recent work [17] suggests to realize role-based models with AOP. This fundamental relation is also exploited in the OCoN approach on the level of system design.

2. The OCoN Approach

The OCoN approach reflects this additional requirement by putting a strong emphasis upon the basic software engineering ideas of modularization and separation and further by applying the design by contract idea of Meyer [21]. The separation of independently served contracts as in the case of *role-based modeling* [27] in conjunction with multiple interfaces [32] for different concerns further improve separation. The combination

of the usually simple syntactical interface notion with a behavioral protocol builds a synchronization contract which permits reduction of harmful implementation dependencies to a great extent. These behavioral contracts form a suitable mechanism to describe architectural aspects using the *connector* idea of Allen and Garlan [2]. Hence, we extend the object subtyping notion by means of the behavior protocol. The OCoN formalism carefully distinguishes shared from exclusive contracts and makes an improved subtyping notion for these cases feasible. The provided visual notation permits the construction of subtype contracts by a form of graphical composition.

The basic mechanisms used in the OCoN approach are those of the currently widely used object-oriented analysis and design [3, 28, 16]. In order to introduce as few new notations as possible, the different kinds of diagrams provided by the UML [24] are used wherever suitable. When dealing with static structure this works fine. As discussed in [10], however, the UML notations for behavior modeling are not particularly mature with respect to distributed systems. The OCoN approach overcomes such drawbacks of the UML by means of providing appropriate visual formalisms based on Petri nets [4] for behavioral descriptions which then are seamlessly integrated with the concepts of object-orientation, concurrency aspects and resource coordination in distributed systems [12]. Moreover, the very basis of the approach places focus upon a contract-based system design [11, 8].

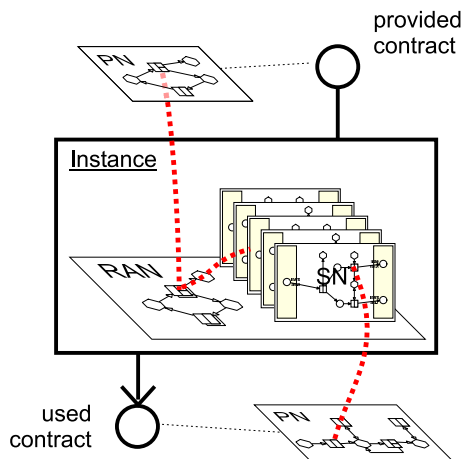


Figure 1. The structural concept for a class or subsystem

The OCoN formalism provides three specific net forms to describe contract protocols and the instance behavior. Figure 1 presents the relation between the different nets. Protocol nets (PN) are used to describe the guaranteed or assumed behavior of contracts. So called service nets (SN) describe behavior for a specific task right like a method with its own thread of control in a programming language. An instance wide unique resource allocation net (RAN) is used to describe the over-

all instance activities like request acceptance as well as the allocation of needed resources for the request processing including the creation of related service nets for a request. For sake of conciseness we further omit any details of the behavior description with nets and denote them only using their abbreviations to focus on the overall concepts for separation.

3. Distributed System Engineering

The specific problem of software engineering for distributed systems and its treatment has been the main goal of the OCoN approach. Waldo et al. [31] have identified that concurrency, resource management, the phenomenon of partial failures and performance are the essential aspects which a distributed system design is required to cover. When an open system is further intended, the resulting problems have also to be suitably managed.

The main concept to address the highlighted problems within the OCoN approach are executable design models. It is to be noted that even for the core UML, a restricted attempt exists to define and standardize action semantics [1]. The attempt to define an operational semantics for the constructive UML behavior description mechanisms state diagram (statechart) and *activity diagram* are the correct step, while the considered behavioral specification techniques do not support any useful notion for contracts or behavior abstraction and thus the resulting extended specifications using the action language result in mere implementation. The OCoN behavior description in contrast supports the simulation for even incompletely specified systems due to its explicit design for behavioral abstraction.

While concurrency and resource management are aspects served by the underlying Petri net formalism, the carefully ensured executable abstract design model is required to consider performance at this early stage of the process. For the phenomena of partial faults in contrast, we have to refer to the intended underlying middleware or restrict its choice accordingly.

The beneficial mechanisms identified during the design process do to some extent determine both architecture and suitable middleware solutions. If in this manner, the dimensions of suitable software architectures are initially explored, the choice of a middleware product has at least to ensure that the applied or assumed principles, for example, reliable messaging, distributed transactions or associative access are also provided by that middleware. This choice has to take into account the restrictions of specific middleware products, while the middleware standards aim to support a complete list of features and thus do not result in any restriction.

When, in contrast, a specific middleware product has been selected beforehand design, the restrictions of such have to be considered during design. *Design for middleware* [7] which explores suitable architectural alternatives for fixed middleware would therefore be more ap-

appropriate. The OCoN approach does not support the embedding of middleware or language-specific constructs into the design itself so as to ensure that the design remains a conceptual model and not a direct implementation. Restrictions enforced by a specific middleware solution are usually rather general. The fact that reliable messaging or distributed transactions are supported does not directly affect the availability of behavioral elements, but rather influences the failure semantics of a design. The restriction cannot therefore be expressed in direct OCoN language terms and instead results in adjusted error semantics.

4. Roles and Separation

The supported role-based modeling allow the separation of functional aspects by means of contracts. The different roles a specific subsystem can play within a design can be described by distinct contracts and are therefore separated from each other. During further design activities these distinct contracts can be employed as required while keeping them separated where possible. Thus the overall design complexity is reduced in contrast to a handling which considers the whole class instead of its role specific contract during system construction.

The majority of approaches for multi-dimensional separation of concerns function at the implementation level and current approaches for the design level, as for example, subject-oriented design [6], are restricted to simple strategies such as merge or select. Commonly only non-functional aspects are addressed and relevant notions of synchronization and protocols for distributed systems are not considered. A single OCoN component can provide multiple independent operating contracts which further permit the separation of different functional aspects. Due to synchronization, resource sharing and semantical dependencies between different served contracts in such an implementation type, the related conflicts and dependencies have to be addressed. In the OCoN approach this is achieved in an explicit fashion by taking the needed internal coordination and use of imported contracts into account.

It is to be noted, that in contrast to the common techniques for aspect weaving which assume the automatic weaving of orthogonal aspects the presented approach considers non-orthogonal aspects and therefore the weaving becomes an important design activity that cannot be totally automated when conflicts have to be resolved in a semantical correct manner. The resulting demand for manual aspect weaving for each component involved in multiple architectural aspects seems cumbersome on the first sight, but the weaving can be done locally for each component and the approach separates aspects globally for the overall design. Therefore, the proposed approach permits to reduce the overall complexity of a design while only the design of each single component might become more complex.

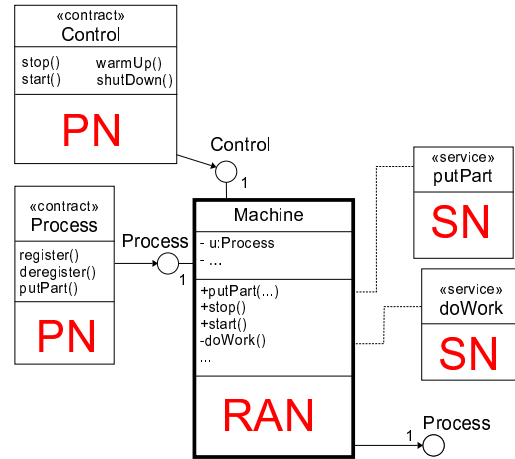


Figure 2. Role composition via an executable model

The resulting scenario is visualized in Figure 2. A simplified version of a production planning and control system of [13] is considered. Products are passing through multiple successive stages which are equipped with machines. Modern production equipment offers a variety of interfaces for accessing different functionalities like, for instance, configuration of the machine, monitoring of relevant parameters and control of the actual working process. In the design we consider the control and processing *aspects* of the application domain in isolation using distinct roles for a machine. The OCoN approach further explicitly address the combination of these partial specifications into an overall machine behavior by means of an executable design model describing the required coordination.

Even when during the beginning of a design the internal needed usage of resource should be omitted, the composition of a set of contracts for a known set of constraints can be studied. This permits the separation of architectural views w.r.t. the given constraints. If no constraints exist, the resulting external behavior for a component realizing two contract protocols is simply the parallel product, but when the protocol states are related to each other by constraints, only a subset of this product protocol is still correct. By abstracting from explicit states and operations using classifications we can further specify the constraints on a conceptual level. Changing one aspect and its contract does then not effect other aspects. In joint work with Vilbig [13] a corresponding general approach for the separation of functional aspects at the level of a software architecture has been developed.

The structural situation of the described approach is depicted in Figure 3. A set of rather abstract constraints together with the provided contracts is analyzed to detect conflicts which will hold for every possible implementation. The constructed most general realization can be further used as starting point for the implementation.

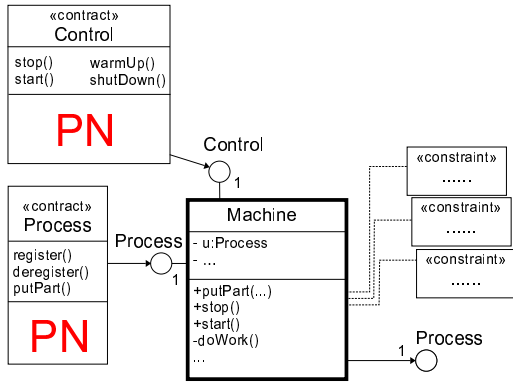


Figure 3. Analysis of role composition for a set of constraints

If in contrast relevant dependencies between provided and used contracts exists also the necessary information for correct component composition is required. The notion of *typed component systems* (TYCS) [8] does further address this issue. In contrast to the explicit behavior specification of object-oriented systems as achieved with the OCoN approach, the components types preserve a great degree of freedom concerning component implementation by abstract from such details but remain sufficient to exclude all general behavioral anomalies. The TYCS approach goes one step further and provides a component notion with black-box abstraction which still ensures the necessary knowledge to control their application and composition.

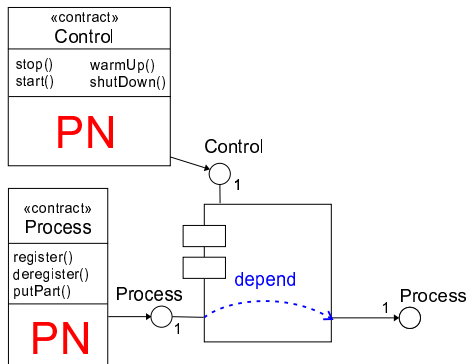


Figure 4. Role composition for black-box components

For our machine example a possible design is visualized in Figure 4. While the processing may depend on the used contracts, the control contract has to be served independent of pending requests.

5. Aspects and Weaving

Open systems and run-time binding of services are fundamental requirements bigger distributed systems

bring into the game. Temporary cooperation of independent developed components that do not share any organizational background will therefore become mandatory in the future. Classical class-based designed systems as well as programming language-based approaches for advanced separation by means of weavers will however fail under such conditions. During system design and implementation no fixed boundary or common code basis can be assumed and therefore developed techniques have to be adjusted.

A first fundamental change will be that a system has to operate in multiple possible contexts instead of a fixed static one. Therefore, weaving has to be adjusted accordingly. For predefined interception-points there already exists the technique of run-time service lookup [29] which can be used to do the weaving "on the fly".

The OCoN approach supports efficient run-time contract matching which includes protocol conformance [9]. In combination with explicit context dependencies in form of required context capabilities that are needed to obtain policies or services for well understood orthogonal aspects like tracing, security or failure handling the available technology can provide advanced separation of concerns. However, pre-defined interfaces for service lookup are needed to enable their integration.

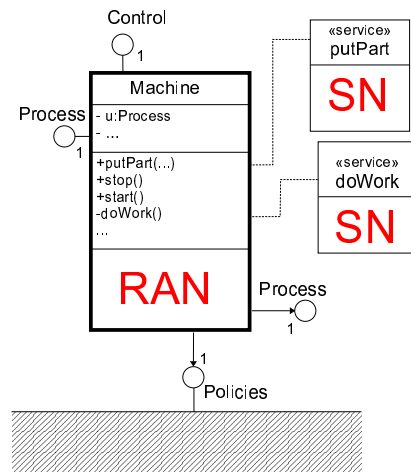


Figure 5. Weaving by service lookup for a known contract type

In Figure 5 the resulting situation is visualized when the machine processing depends on a given context dependent set of policies. For known variations such as material dependent processing parameters the explicit architectural solution at design time is possible.

The need for such context capabilities can be further reduced when default policies and solutions operating locally for a single component are given, however, it is often useful to make the context dependencies explicit and demand them as prerequisite for running a component. We propose therefore to model the component context rather explicitly and determine the interception-points

for the aspects a component should support at design and implementation time.

Another common approach to address this issue is *configuration* at time of deployment as realized by the EJB [20] component approach. Here, for example, the persistence aspect can be determined when installing a component. Note, however, that again the pre-defined component life-cycle of a EJB component ensures that a well-defined interface exists. Instead of specific modules realizing a particular aspect, the application server permit only to chose from a predefined list of options.

The presented solutions are however not able to add arbitrary aspect. Instead, only such aspects the component design and implementation is aware of can be supported. In contrast middleware interceptors [23], e.g., for thread management issues, can be used the realize the commonly used concept to extend a class by adding code when enter and leave one of its method in a generic way. However, the scope of such attempts is restricted to a single executable connected to a specific middleware and therefore does not cover flexible open systems.

For components and system modules of different origin and constructed with rather distinct engineering culture the question whether such generic solutions such as interceptors are suitable means to realize aspect weaving at run-time is rather critical. While it is attractive w.r.t. system maintenance and code compactness to transfer the benefits of programming language-based advanced separation of concerns concepts to this level, we have to consider also that the risk of unexpected interoperability problems and their impact on system reliability is nevertheless more serious in a distributed system.

6. Design and Methods

As a design notation, the OCoN approach was originally intended to support any design style, but we observe that responsibility-driven design of Wirfs-Brock et al. [32], as well as role-based modeling approaches such as *OOram* of Reenskaug [27] fit best. In contrast to data-driven design, the responsibility-driven design attempt to avoid both *centralized* and *overly distributed* designs, by instead attempt to build systems where behavior and data are well distributed and tasks suitably delegated. Role and responsibility-driven design therefore results in a suitable design where *coordinators* take care of their responsibilities by coordinating other instances, while details are delegated to subordinated coordinators. This includes reactive as well as proactive behavior which can be particularly well described by using the OCoN approach. By further emphasizing the coordinator role stereotype and avoiding *controller* stereotypes, a suitable design style for distributed systems is achieved. This design style is also appropriate for variability and adaptability, because so-called *hot spots* [26] can be located in specific coordinated parts. Their change will only effect other parts when the changes are so drastic that overall coordination requires adjustment.

A related paradigm is agent-based modeling. An *agent* is characterized by its reactive and proactive nature, its autonomy with respect to state and behavior and is situated in some environment. Wooldridge et al. [33] note that agent-based design and techniques are suitable at a coarse-grain view. The OCoN approach permits integration of agent technology into the software architecture at just the appropriate design level. In contrast to the contract notion of Meyer [21], agent contracts emphasize the autonomy of the contract partners and assume an agreement-oriented style, while programming language contracts tend to describe a form of delegation in which the serving side is strictly controlled by the client side. The OCoN contract notion due to its inherent non-determinism further permit description of agreement-oriented and strict controlling contracts.

Support for behavioral contracts becomes particularly advantageous when a specific design style with emphasis upon separation, for example, component-based design is used. In doing so, it also harmonizes better with methods which support and emphasize modularity. The OCoN contract concept provides a solid ground to establish system interconnections in open systems, but in supporting component-based systems besides the contract notion, the components and their composition require specific handling so as to ensure suitable design.

7. Conclusion

Software engineering depends upon different aspects including the available technology for describing artifacts alongside the process itself. Brooks [5] analyzed the problem of large software projects and concluded that languages cannot provide an improvement of magnitude until they tackle the *essence* of development. Harel later predicted [14] that a suitable visual language and executable model combined with model analysis and code synthesis can improve situation when smoothly integrated and applied where appropriate, arguing for the statechart approach in reactive systems. Today, *software process improvements* and other meta-level activities are often seen as favorable to technical solutions such as a visual design language, but "management isn't everything" [22] and technological improvements such as advanced separation of concerns are also promising. The OCoN approach exploit its support for non-orthogonal functional aspect separation to simplify the design of distributed systems by providing a visual language with abstract executable model. It permits to reduce the overall complexity of a design while only the complexity of each single component might be increased.

For the coordination aspect emphasized in the OCoN approach, the described separation of functional aspects is achieved using method signatures and contracts. The possible synthesis of the coordination code does further simplifies implementation of this aspect. Coordination is therefore lifted to the design level rather than that of implementation as a result of the identified overall impact

on building a suitable software architecture and design with respect to interaction and modularity.

The provided notion of separation by means of distinct contracts supports to exploit non-orthogonal separation. The clients benefit from the less complex contract and are separated from the overall class complexity, while the class or subsystem internal design has to find a realization fulfilling all the provided contracts using only the assumed ones. Weaving orthogonal aspects usually ensure the existence of a realization. In contrast for the case of functional aspects as considered in this paper contradicting contracts may result in unresolvable conflicts. Therefore, the OCoN approach propose to consider the explicit composition at the design level to exclude problems when implementing the system. Also further extensions for early detection of conflicts [13] and component-based systems [8] have been presented.

For orthogonal aspects the application of explicit context modeling and dynamic service lookup has been proposed as solution within the OCoN approach. While suitable for the case where well-defined interfaces for the aspect integration seem feasible, and whether it is appropriate to support the integration of aspects the component designer was not aware at design and implementation time remains an open question for distributed systems.

References

- [1] Action Semantics Consortium. *Response to OMG RFP ad/98-11-01 Action Semantics for the UML*, Feb. 2001. Version 19a.
- [2] R. Allen and D. Garlan. A Formal Basis for Architectural Connections. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.
- [3] G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, Menlo Park CA, 1993. (Second Edition).
- [4] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models (part I)/Applications (part II)*, volume 254/255 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1987.
- [5] F. P. Brooks, Jr. No Silver Bulet: Essence and Accidents on Software Engineering. *IEEE Computer*, 20(4):10–19, 1987.
- [6] S. Clarke, W. Harrison, H. Ossher, and P. Tarr. Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. In *Confernece on Object-Oriented Programming, Systems, Languages, and Applications, November 1-5, 1999, Denver, Colorado, USA*, pages 325–339, 1999.
- [7] W. Emmerich. Software Engineering and Middleware: A Roadmap. In A. Finkelstein, editor, *22nd International Conference on Software Engineering: The Future of Software Engineering*, pages 119–129, 2000.
- [8] H. Giese. Contract-based Component System Design. In J. Ralph H. Sprague, editor, *Thirty-Third Annual Hawaii International Conference on System Sciences (HICSS-33)*, Maui, Hawaii, USA. IEEE Press, Jan. 2000.
- [9] H. Giese. Object Coordination Nets 3.0: Synchronization Behavior Typing for Contracts. TechReport 2/01-I, University Münster, Computer Science, Distributed Systems Group, Feb. 2001.
- [10] H. Giese, J. Graf, and G. Wirtz. Closing the Gap Between Object-Oriented Modeling of Structure and Behavior. In R. France and B. Rumpe, editors, *UML'99 - The Second International Conference on The Unified Modeling Language Fort Collins, Colorado, USA*, volume 1723 of *Lecture Notes in Computer Science*, pages 534–549. Springer Verlag, Oct. 1999.
- [11] H. Giese, J. Graf, and G. Wirtz. Contract-based Coordination of Distributed Object Systems. In H. R. Arabnia, editor, *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Las Vegas, Nevada, July 1999.
- [12] H. Giese, J. Graf, and G. Wirtz. Seamless Visual Object-Oriented Behavior Modeling for Distributed Software Systems. In *IEEE Symposium On Visual Languages, Tokyo, Japan*. IEEE Press, Sept. 1999.
- [13] H. Giese and A. Vilbig. Towards Aspect-oriented Design and Architecture. In *15th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications October 15-19, 2000, Minneapolis, Minnesota, USA. Workshop: Advanced Separation of Concerns, Monday, 16 October, 2000*.
- [14] D. Harel. Biting the Silver Bulet: Towards a Brighter Future for System Development. *IEEE Computer*, 25(1):8–20, 1992.
- [15] R. Hilliard. Aspects, Concerns, Subjects, Views, ... In *First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems (at OOPSLA '99)*, 1999.
- [16] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1996. (Revised Printing).
- [17] E. A. Kendall. Role Model Designs and Implementations with Aspect-oriented Programming. In *Confernece on Object-Oriented Programming, Systems, Languages, and Applications, November 1-5, 1999, Denver, Colorado, USA*, pages 353–369, 1999.
- [18] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. Techreport, Feb. 1997. XEROX PARC Technical Report, SPL97-008 P9710042.
- [19] P. Kruchten. *The Rational Unified Process - an Introduction*. Addison Wesley, 1999.
- [20] V. Matena and M. Hapner. *Enterprise JavaBeans™ Specification*. Sun Microsystems, Dec. 1999. Version 1.1.
- [21] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997. 2nd edition.
- [22] B. Meyer. Every Little Bit Counts: Toward More Reliable Software. *IEEE Computer*, 32(11):131–133, Nov. 1999.
- [23] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. Using Interceptors to Enhance CORba. *IEEE Computer*, 32(7):62–66, July 1999.
- [24] Object Management Group. *OMG Unified Modeling Language Specification, Version 1.3*, June 1999. OMG document ad/99-06-08.
- [25] D. L. Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.
- [26] W. Pree. *Design Patterns for Object-Oriented Software Development*. ACM Press, 1995.
- [27] T. Reenskaug, P. Wold, and O. A. Lehene. *Working with Objects: The OOram Software Engineering Method*. Addison-Wesley/Manning, 1996.
- [28] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenzen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
- [29] Sun Microsystems. *Jini Specification*, Jan. 1999. Revision 1.0.
- [30] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 1999 international conference on Software engineering May 16 - 22, 1999, Los Angeles, CA USA*, pages 107–119, 1999.
- [31] J. Waldo, G. Wyant, A. Wollrath, and S. Kendal. A Note on Distributed Computing. Techreport, Sun Microsystems Laboratories, Nov. 1994. TR-94-29.
- [32] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice Hall, 1990.
- [33] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the third annual conference on Autonomous Agents May 1 - 5, 1999, Seattle, WA USA*, pages 69–76, 1999.