BOOSTER*Process*

A Software Development Process Model Integrating Business Object Technology and UML

Axel Korthaus and Stefan Kuhlins

University of Mannheim
Department of Management Information Systems III
D–68131 Mannheim
Germany
{korthaus|kuhlins}@wifo3.uni-mannheim.de
http://www.wifo.uni-mannheim.de/

Abstract. This paper describes a UML-based process model (called BOOSTER*Process*) for system development founded on business object technology. It integrates business and software engineering aspects, describes the specific modeling activities needed for business and software system modeling and connects the various UML diagrams, particularly taking into consideration the requirements of business objects and their component character. It propagates a multi-level approach, starting with use case, activity and class modeling at the organizational level, and then shifting to analysis and design of business applications.

1 Promises of UML and Business Object Technology

Nowadays, new component technologies start to emerge rapidly as a successor of object-oriented ideas which have eventually become the mainstream in software industry today. They build upon the most successful concepts of object-orientation, but go further, e.g. by defining larger-grained units of reuse compared with conventional objects in object technology. These component technologies, e.g. OCX/ActiveX [15] or (Enterprise) Java Beans [23], are no longer limited to the field of GUI components, but begin focusing on the implementation of business concepts and business logic. One of the most interesting developments in this area are OMG's standardization efforts for so-called *Common Business Objects* (CBOs) and a *Business Object Facility* (BOF) [17] on which we will concentrate in this paper as a representative of those technologies. While CBOs are "objects representing those business semantics that can be shown to be common across most businesses", the BOF is "the infrastructure (application architecture, services, etc.) required to support business objects operating as

* Accepted for First International Workshop on the Unified Modeling Language ≪UML≫'98: Beyond the Notation, Mulhouse, France, June 3–4, 1998

cooperative application components in a distributed object environment." [17]. According to the CBO Working Group of the OMG Business Object Domain Task Force (BODTF), business objects capture "information about a real world's (business) concept, operations on that concept, constraints on those operations, and relationships between that concept and other business concepts. [...] a business application can be specified in terms of interactions among a configuration of implemented business objects." [18] This definition reflects the object-oriented foundation of business object technology. The vision of CBOs as both design-time and run-time constructs comprises, above all, the goals of interoperability of business object components, including the possibility of ad-hoc integration (i.e. "plug-and-play"), and simplicity of design, implementation, configuration and deployment, so that an average developer is helped to build business object systems easily [17,22].

The CBO vision aims at a marketplace for standardized "off-the-shelf" business objects which are easily integrated with other business objects through the BOF and are able to interact with each other in order to perform some business function, even if the collaboration was not planned or foreseen by their developers (for ad-hoc-integration cf. Corba Component Initiative [19]). The eventual achievement of these goals would bring software engineering a significant step closer to meeting the increased requirements on modern information systems development.

For the purpose of modeling business and software systems with business objects, a suitable object-oriented analysis and design (OOA&D) method is needed [14]. Object-oriented modeling is another area where the OMG has sought standardization and has been successful recently through the adoption of the Unified Modeling Language (UML) in Nov. 1997 [20]. UML combines common concepts of some earlier analysis and design methods, enhances this set with additional modeling concepts meeting the requirements of current modeling tasks, and defines notational symbols for those concepts. As a general purpose language, UML is designed to model a wide range of different types of systems, from purely technical (non-software) through software to business systems.

In contrast to its predecessors, UML is merely a modeling language, not a complete methodology, because there is no specification of a particular software development process included in the standard with recommendations of how to deal with the UML elements. The structure of a UML-based process for modeling systems is strongly dependent on the kind of system under development (e.g. business/software/technical system) as well as on other determinants (e.g. project size). Process definitions have to state which techniques are appropriate at various levels of detail during development, which deliverables have to be produced, who should produce them, and which inspections, standards, metrics, and tests should be used to control quality and certify system correctness [1,13]. A UML-based process for systems development founded on the business object paradigm must span the whole range from business engineering (using business object models) to application engineering (with an emphasis on reusing pre-built business object software components) and business object component engineer-

ing (in order to build new business objects). In this paper, we will describe the basics of a process model (BOOSTER*Process) which meets these requirements.

BOOSTER*Process is part of a project called BOOSTER, which has just been launched at the University of Mannheim. BOOSTER is an acronym for "*B*usiness *O*bject *O*riented *S*oftware *T*echnology for *E*nterprise *R*eengineering". BOOSTER will examine topics around object-oriented business and software engineering, OMG business object and infrastructure standards, architecture, analysis, design, and implementation of distributed business object systems, component technologies (e.g. Enterprise Java Beans [23]), business object frameworks (e.g. IBM San Francisco [9]) etc.

2 Multi-Level UML-Based Business Systems Development

According to [7], processes must be viewed from four aspects: *process context*, *process user*, *process steps*, and *process evaluation*. In the following subsections, we will concentrate on the *process steps* as well as on the foundations and basic principles of BOOSTER*Process, which describe the activities to be taken and the UML elements to be applied during the process.

Although UML does not include a specific process, its designers had certain basic process principles in mind which are derived from the most popular existing methodologies – above all Booch '93 [4], OMT [21], and OOSE/Objectory [11]. The UML documentation [20] mentions that processes using UML should be *use-case-driven*, *architecture-centric*, *iterative*, and *incremental*. BOOSTER*Process sticks to these basic principles which are common practice in object-oriented development, although the concept of use cases is not over-emphasized.

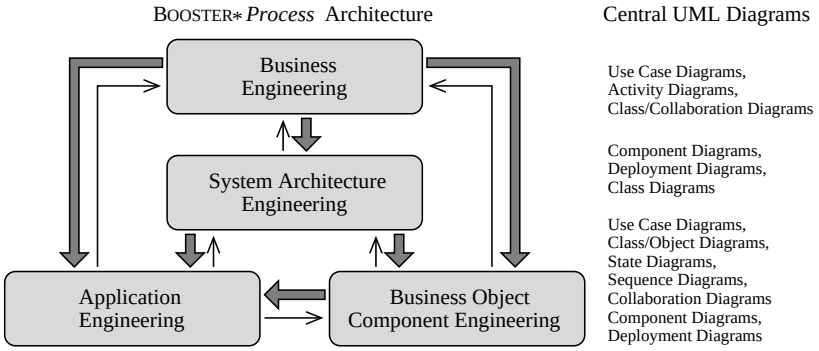

Fig. 1. Process architecture of BOOSTER*Process

The most important foundation of BOOSTER*Process is a multi-level approach to business object system development, which is represented by a process

architecture. This multi-level process architecture defines a framework for the activities which have to be performed. The macro process constituted by the architecture levels of BOOSTER*Process* is broken down into micro processes, which roughly reflect the well-known activities of requirements gathering, object-oriented analysis, design, implementation, and test. The macro process architecture of BOOSTER*Process* is shown in Fig. 1. As can be seen from the figure, the levels business engineering, system architecture engineering, application engineering, and business object component engineering are distinguished. Each of these levels is described by a micro process (cf. Subsect. 2.1-2.4). The different kinds of arrows indicate more or less significant directions of information flow and express the principle of iterations and increments, which can be found even in the macro process depicted in Fig. 1. On the right side, those UML diagrams are listed that are most important for the respective engineering activities.

While business and software engineering activities have different viewpoints and different levels of abstraction, they are not independent of each other. Modeling business goals and processes is the basis for deriving requirements on the information systems needed to support the business. New information system technologies, on the other hand, influence the way how the business processes are to be shaped to provide the best customer value. Therefore, it is very important to integrate these different viewpoints within a comprehensive process, using the same underlying technologies, i.e. UML modeling and business object concepts (see e.g. Taylor's ideas [24] of "convergent engineering").

System architecture engineering serves for designing a stable system architecture as a basis for the development of a number of related applications. Fortunately, the BOF RFP [17] already suggests a basic system architecture for OMG business object systems (see Subsect. 2.2). This basic layered architecture has to be refined by company-specific enhancements.

Application engineering is an activity resulting in a new software application for the organization. Its most important characteristic is the reuse of pre-existing business objects, at the modeling level as well as at the software component level. Business object component engineering is the process of designing and implementing new business object components for reuse. This activity may be independent of a concrete application engineering process. Similar distinctions between these two kinds of processes can be found in several approaches: [1], for example, distinguish between solution projects and component projects, while [13] speak of application system engineering and component system engineering.

The normal course of activities begins with a business engineering process as the starting point, followed by a system architecture engineering process. When the system architecture is defined, several application engineering processes will be performed, concurrently with several business object component engineering processes, which are a result of the requirements generated by the application engineering processes, or which independently produce components for future needs, in the sense of a domain engineering activity. In the following subsections, the individual levels of BOOSTER*Process* are described in detail.

2.1 Business Engineering

A key characteristic of BOOSTER*Process* is that it starts at the enterprise level with a business (re-)engineering activity. Part of a successful business (re-)engineering activity is the modeling of the business with its goals, rules, resources, actions, workflow etc. UML provides a number of diagrams which are very useful for this purpose, namely use case diagrams, class diagrams, and activity diagrams (cf. [14]). [12] describe a process for object-oriented business engineering, which nearly exclusively builds on use cases for modeling business processes. BOOSTER*Process* takes up those ideas, but supplements the use case models with activity diagrams and high-level class models. The basic steps in business engineering follow the pattern of those described by [13], but are adapted to the needs of BOOSTER*Process*:

- *Formulate a business vision*: Define the rationale and the goals of the BPR activity, discuss it with managers and employees, consider new technologies which might be helpful to improve the business, formulate objectives and high-level descriptions of future business processes.
- *Reverse engineer the existing business*: Build use case models, class models, and activity models of the existing business structures and processes to be improved in order to facilitate a detailed problem identification and analysis. Transform perceived business concepts into suitable business object types, i.e. entity, process and event business objects.
- *Forward engineer and implement the new business*: Produce a detailed description of the new processes and the internal organization of the business in the form of new versions of the use case, class and activity models. Identify suitable business objects and map them to standardized Common Business Objects and existing domain specific business objects as early as possible in the process. Identify areas of operation which can be supported by business information systems. Implement the new business incrementally and develop the associated software systems.

In the context of business engineering, use cases appear as business use cases. Business use cases model sequences of work steps performed in a business system which produce a result of perceived and measurable value to business actors. Business actors are roles that people or external systems in the environment play in relation to the business. The business use cases, which model business processes, should be detailed with the help of high-level class and collaboration models, expressing the internal realization of the business processes by workers with appropriate competencies and a number of business objects the workers work with. In order to facilitate the distinction between UML models at the business and the software level, UML will be adjusted appropriately for business engineering activities in BOOSTER*Process* by making use of suitable enhancement techniques such as stereotypes, tagged values and constraints, similar to the UML Extension for Business Modeling which is part of the UML documentation [20]. Furthermore, we recommend defining new stereotypes to be able to express a taxonomy of special kinds of business objects (see above). All models

produced during business engineering should be arranged in a top-level package labeled with the stereotype ≪business system≫.

The realization of use cases should not only be modeled by class and collaboration diagrams, but also by activity diagrams, which are very suitable for expressing workflow and parallel activities. Activity diagrams are similar to conventional approaches to modeling business processes (e.g. Event Driven Process Chains, see [16], thus making additional modeling techniques apart from UML superfluous. Like use cases, activity diagrams are not object-oriented in nature and thus render the mapping to object-oriented concepts more difficult. Activity diagrams can help identify activities in the business processes that can be executed or supported by information systems. This is where the transition to application engineering takes place. A rule of thumb regarding the mapping from the business models to models of the information systems could be that each business use case, described by an activity diagram, might be supported by and mapped to several information system use cases. Some of the internal workers identified and even some of the business actors might become actors in the information system use cases. Information system use cases might correspond with process business objects, and some of the business objects identified in the high-level class models might be represented by entity business object packages in the information system models.

To enable the extensive reuse of existing business object components, their integration must be considered as early in the system life cycle as possible. Therefore, existing business object specifications must be matched with the business concepts identified during business engineering. What is needed in order to support this is the standardization of documentation and specification techniques for business object components. The BOCA (Business Object Component Architecture) submission [6] to the CBO/BOF RFP [17] represented the first step in this direction, because it comprised a Component Definition Language (CDL), which was designed to rigorously specify business object components. Unfortunately, the work had to be stopped because of technical difficulties, but at the moment of writing, new Business Object RFPs are being prepared which will probably continue the work on BOCA and CDL (so we will refer to the current BOCA proposal in this paper). For convenience and to be able to express more of the semantics, CDL specifications should be supplemented by suitable UML models describing interfaces, structure, and behavior of the components.

While the standardization of a business object specification technique is a basic requirement, the standardization of CBOs has the additional advantage that even business terminology and semantics are clearly defined, too. Using these standardized semantics and integrating the UML diagrams associated with existing business objects with the models of the system under development, it should be possible to begin reuse activities already at the business engineering level. The earlier the mapping to existing business object components occurs, the better chances are of quick information systems implementation through assembly of existing software components. Thus, business modeling activities

should be performed with strict adherence to standardized CBO terminology and semantics where possible.

2.2 System Architecture Engineering

The intrinsic complexity of modern large-scale information systems can be managed best with a good software architecture, which, for example, enables parallel development activities. The goal of architectural modeling is to define a robust framework within which applications and subsystems such as business object components may be developed. The system architecture provides the context for defining how applications and business object components interact with one another to perform the needed business functions. As a common base from which all project teams work, a good architecture increases the reusability on system development projects.

For applications built from business object components, the basic architecture required will be part of the OMG standard. Figure 2 shows the proposed architecture for business objects. Integrated in OMG's Object Management Architecture (OMA), which includes a reference model for distributed object computing and defines OMG's objectives and terminology, the Business Object Facility (BOF), based on CORBA, provides the infrastructure for CBOs, domain specific business objects, and enterprise specific business objects. This is the basic layered architecture on which all business object systems will be based. The software is organized in layers according to this layered architecture. Objects and components in lower levels are more general than those in higher levels and encapsulate technical details about transactions, persistence etc. with which the developer and user of higher-level components does not want to be concerned. The application engineering process uses the different kinds of business objects to assemble them into software applications. The business object component engineering process produces business objects fitting the architecture of Fig. 2.

The generic architecture for business objects must be enhanced by an enterprise-specific, change-tolerant application systems architecture, which defines subsystems (using UML packages and components) and defines clear interfaces to reduce communication overhead and to allow graceful system evolution over time. It should be decided which parts of the system are most stable and which will change frequently in order to arrive at a good subsystem organization.

UML provides sufficient modeling capabilities to clearly distinguish between the logical and the physical architecture of the system. While the logical architecture is expressed in the form of class diagrams, for the most part containing only packages, interfaces, and dependency relationships, the physical architecture is modeled by component and deployment diagrams (cf. Fig. 1). UML packages on the logical level and components on the physical level are very important during architectural modeling, because they allow the partitioning and control of the overall software structure. At the logical level, both legacy systems that must be wrapped to fit into the architecture and large-grained business objects identified during business modeling are modeled as packages. Clear interfaces between

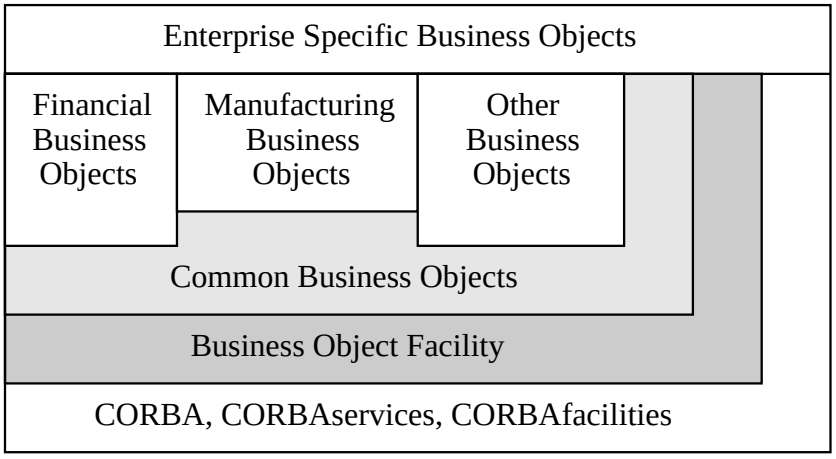

Fig. 2. Architecture for business objects [17]

these packages have to be defined, so that the packages can be allocated to different teams. Apart from class and component diagrams, deployment diagrams can be useful in structuring the physical architecture and in initial consideration of the run-time distribution of the components already known.

The iterative and incremental micro process of system architecture engineering comprises the following activities:

- *Capturing requirements*: Using the results of business engineering as input and doing further research (e.g. interviews etc.), the global needs and expectations of the (internal) customers and end users must be gathered and modeled, often with the help of use case diagrams; furthermore, non-functional requirements have to be analyzed in terms of an overall quality plan;
- *Perform global analysis*: Through examination of the requirements, candidate applications and business object components should be identified and modeled as packages. Domain analysis activities as well as feedback from previous application engineering projects about needs for reusable business object components can contribute to the results;
- *Architectural design*: As much as possible of the overall architecture should be specified on a design level. This includes the precise definition of facades (see Subsect. 2.4) and interfaces of the applications and business object components to be developed, the legacy systems to be integrated, and the technology components for lower system levels (e.g. ORBs, BOF). For this purpose, it might be necessary to begin a more detailed behavioral modeling involving the use of UML interaction diagrams (not mentioned in Fig. 1).
- *Implementation and test of the layered architecture*: At this point the packages identified have to be implemented (if not already in existence), which involves application engineering and business object component engineering. Finally, the system functionality has to be tested on a global level.

In analogy to the business engineering level, UML should be adjusted appropriately for software engineering based on business object technology. This means that a suitable version of UML has to be defined to meet the given needs. Provided that the approach of the BOCA proposal [6] is followed up, business object systems will have to be specified in CDL (similar to IDL specifications of CORBA objects). The BOCA metamodel would thus become a design target for the UML models, so that an appropriate UML extension must be defined to facilitate the mapping between UML and CDL.

2.3 Application Engineering

Application engineering is the process during which single business applications are implemented that directly serve the business by supporting the business processes. Since the vision of OMG business objects comprises a considerable simplification of developing business information systems, the goals of application engineering in BOOSTER*Process* are to develop solutions quickly, but on a sound, evolving architectural base, thus producing applications that confer early user benefits at minimum costs and leverage existing legacy systems where possible, while maintainability is retained. These goals are sometimes subsumed under the term Rapid Architectural Application Development (RAAD), as opposed to Rapid Application Development (RAD), where no models are produced and no system architecture is designed [1].

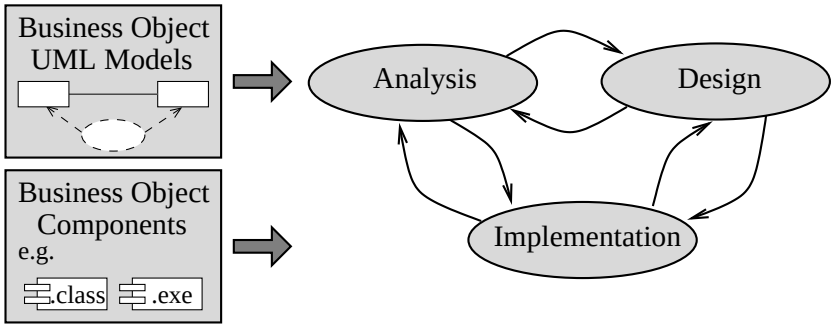

Fig. 3. Reuse-oriented micro process for application engineering

Application engineering is very much like conventional software engineering approaches described in literature, extended by aspects of reusing existing business objects. The micro process to be followed is composed of the classical activities of object-oriented analysis, design, implementation, and test, performed iteratively and concurrently in part and involving the complete set of UML diagrams to express structure, behavior, and algorithms of the application system. On the right side of Fig. 3, these activities are shown (except for testing) according to the baseball model of object-oriented software engineering described

in [5]. The left side shows how the process uses a combination of model-based reuse and component-based reuse. During analysis and design, the developers permanently assess the possibilities of reusing existing business object specifications (the types of UML diagrams and modeling elements used for this purpose are described in Subsect. 2.4). If appropriate specifications can be found and integrated in the modeling process, this will lead to reuse and assembly of the corresponding business object components during implementation.

Analysis starts with the definition of use cases and actors, who will interact with the application. As already stated, these requirements can be partially derived from the results of business modeling. More information has to be uncovered in cooperation with the customers and end users to specify the different usage scenarios of the application. If possible, existing use case descriptions belonging to large-grained business object components should be retrieved and used. The second step is to build an analysis model, which should be independent of the technical details of the specific implementation environment. The analysis model shows structural and behavioral aspects of the problem domain. Therefore, class and object diagrams, collaboration diagrams, sequence diagrams, and state diagrams are produced. There are several heuristics about how to identify the modeling elements in this phase, starting from the specified requirements. The realization of the use cases, for example, should be shown by sequence diagrams and collaboration diagrams. The UML notation for patterns can be of help here, and available business patterns stemming from business object documentations should be searched and integrated at this point in preparation for component reuse. During design, technical details are added and the models are adjusted to fit the concrete conditions of implementation. Component diagrams, which contain representations of the runtime business object components, and deployment diagrams are added to the set of models. In order to prepare for the implementation in the BOF environment, the business objects have to be specified in CDL at this point. Probably, future UML CASE tools will provide features for transforming UML models into textual CDL specifications. In the implementation phase, those parts of the system that could not be assembled by pre-existing components have to be implemented, existing business object components have to be customized and some glue code may have to be written.

2.4 Business Object Component Engineering

Business object component engineering is the process which delivers common, domain specific or enterprise specific business objects. Provided that a marketplace for business objects evolves, non-software enterprises will primarily have to deal with the production of their own enterprise specific business objects, while more generic business objects will be purchased from component suppliers. Delivery of business object components is iterative and incremental, no less than application delivery, but the level of rigor and detail is much greater, since the components have to have a high level of quality in order to be reused frequently.

If business object component engineering were restricted to delivering runtime components, reuse would be very hard. Instead, the components must be

documented in a way suitable to facilitate understanding and retrieval by reusers to allow reuse in early phases of system development, before too many design decisions have been made that cannot be matched with the available components in the implementation phase.

While textual CDL specifications are the most rigorous and system-specific tool for documentation, they need to be supplemented by UML models that can be built into the software models during application engineering. It is not sufficient to model the software units as UML components with clearly defined interfaces, because more information about the semantics is needed for reuse. Typical usage patterns, modeled as collaborations, allowed sequences of messages, represented by state diagrams and illustrated by exemplary sequence diagrams, and the business concepts implemented, modeled with the help of class diagrams, can ensure the usability of the components. What can be seen and used by a reuser of the component is called the external design of the business object component. It does not necessarily need to reveal the internal implementation of the component. The internal implementation again builds upon the complete set of UML diagrams. Only those internal aspects of a business object component that must be known to be able to reuse it should be exported via facades (see [8]), which represent a simplified model of the component and reveal only those parts that need to be directly visible and understood by the application developer.

Input to the business object component engineering process are the requirements, models and documents produced during business engineering, potential domain analysis activities and, primarily, the needs of application systems under development. A new business object component finally has to be modeled as a UML package that contains the implementation files as well as the documentation and usage guidelines.

3 Conclusion

In this paper, we have described basic elements of a new business and software system development process model, which uses the UML as its modeling language and focuses on the concepts of OMG business object technology (but could easily be adapted to other business component technologies, e.g. Enterprise Java Beans [23]). We have tried to emphasize the necessity of an integrated approach to business and software engineering, building on a clearly defined and stable underlying business object system architecture in order to provide for ease of system enhancement and modification and to allow the seamless integration of business object components conforming to OMG standards, and we have suggested the usability of activity diagrams for business process modeling.

Booster*Process will have to evolve and to be adapted to the emerging and still changing business object technology. Possibilities of tool support for the process have to be considered explicitly and the role of business object specification has to be clarified. Furthermore, the relationships to important non-OMG standards such as RM-ODP [10] have to be examined in order to provide conformance. It appears that the specialized architecture of Booster*Process can

easily be mapped to the more general viewpoints architecture of RM-ODP. An important element of our future work will be the evaluation of BOOSTER*Process* in the context of a number of software development projects.

References

1. Allen, P. and Frost, S. (1998): Unravelling the Unified Modeling Language. Application Development Advisor, SIGS Publications, Jan.
2. Atkinson, C. (1997): Adapting the Fusion Process. Object Mag., Nov., 32–39.
3. Boehm, B.W. (1976): Software Engineering. IEEE Transactions on Computers 25 (12), 1226–1241.
4. Booch, G. (1994): Object-Oriented Analysis and Design with Applications. 2nd ed. The Benjamin/Cummins Publishing Company, Redwood City, CA.
5. Coad, P. and Nicola, J. (1993): Object-Oriented Programming. Yourdon Press, Englewood Cliffs, New Jersey.
6. CBOF (1998): Combined Business Object Facility – Business Object Component Architecture (BOCA) Proposal. OMG Business Object Domain Task Force BODTF-RFP 1 Submission. Rev. 1.1. OMG Doc. bom/98-01-07.
7. Eriksson, H.-E. and Penker, M. (1998): UML-Toolkit. Wiley Computer Publishing, New York.
8. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994): Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts.
9. IBM (1998): IBM San Francisco. IBM Inc. `http://www.ibm.com/Java/Sanfrancisco/` (May 1998).
10. ISO/IEC (1995): Reference Model of Open Distributed Processing. ISO/IEC 10746-1 – 10746-4. `http://www.iso.ch/` (May 1998)
11. Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992): Object-Oriented Software Engineering – A Use Case Driven Approach. Addison-Wesley, Wokingham, England.
12. Jacobson, I., Ericsson, M. and Jacobson, A. (1995): The Object Advantage – Business Process Reengineering with Object Technology. Addison-Wesley, Wokingham, England.
13. Jacobson, I., Griss, M. and Jonsson, P. (1997): Software Reuse – Architecture, Process and Organization for Business Success. Addison Wesley Longman, Harlow, England.
14. Korthaus, A. (1998): Using UML for Business Object Based Systems Modeling. In: Schader, M. and Korthaus, A. (eds.): The Unified Modeling Language – Technical Aspects and Applications, Physica, Heidelberg (1998), 220–237.
15. Microsoft (1998): Microsoft COM Homepage. `http://www.microsoft.com/com/` (May 1998).
16. Nüttgens, M., Feld, T., and Zimmermann, V. (1998): Business Process Modeling with EPC and UML – Transformation or Integration? In: Schader, M. and Korthaus, A. (eds.): The Unified Modeling Language – Technical Aspects and Applications, Physica, Heidelberg (1998), 250–261.
17. OMG (1996): Common Business Objects and Business Object Facility. Common Facilities RFP-4. Object Management Group. OMG Doc. cf/96-01-04.
18. OMG (1997): Business Object DTF – Common Business Objects. Object Management Group. OMG Doc. bom/97-11-11 Version 1.3.

19. OMG (1997): CORBA Component Model RFP. Request for Proposal. Object Management Group. OMG Doc. orbos/96-06-12.
20. OMG (1997): The Unified Modeling Language. Vers. 1.1, 1 Sept. 1997, Docu. Set, Rational Software Corp. et al., OMG Doc. ad/97-08-03 – ad/97-08-08.
21. Rumbaugh, J., Blaha, M., Remerlani, W., Eddy, F., and Lorensen, W. (1991): Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NJ.
22. Sims, O. (1994): Business Objects – Delivering Cooperative Objects for Client-Server. IBM McGraw-Hill series. McGraw-Hill Book Company, London.
23. Sun (1998): Enterprise Java Beans 1.0 Specification. Sun Microsystems Inc. `http://java.sun.com/products/ejb/` (May 1998).
24. Taylor, D. (1995): Business Engineering with Object Technology. Wiley Computer Publishing, New York.