

Business Object Facilities

A Comparative Analysis *

Wolfgang Emmerich, Ernst Ellmer
Dept. of Computer Science
University College London
London WC1E 6BT, UK
{w.emmerich | e.ellmer}@cs.ucl.ac.uk

Birgit Osterholt, Roberto Zicari
LogOn Technology Transfer
Burgweg 14
61573 Kronberg
{osterholt | zicari}@ltd.de

Abstract

Rapidly changing business processes require quick adaption of supporting information systems. Component technologies in general and business objects, in particular seem a promising approach. In this paper, we survey, analyse and compare six business object approaches. We develop a comparison model covering concepts, distribution infrastructure, object facilities and object solutions. We then use the model to analyse the Combined Submission to the OMG Business Object Domain Task Force, the SSA Business Object Facility, Sun's Enterprise JavaBeans, IBM's San Francisco, Microsoft ActiveX and the SAP Business Framework. Each of the approaches is described in terms of the model, which allows us to compare them analytically.

1 Introduction

Today's enterprises face highly competitive markets. They are forced to constantly adapt to new requirements, to revisit products, to change processes, to change suppliers and to establish new collaborations. Information systems supporting such enterprises have to be equally flexible. Rapid solution delivery is needed in response to continuous change. Users have to be given opportunities to dynamically adapt their working environment. This situation challenges the development of business information systems.

An approach towards achieving this goal is based on a principle learned from hardware production, the usage of specialised, independent, and encapsulated units, which are commonly referred to as *components* [16]. Components in business applications can be considered as *business objects*. They provide standardised and extensible functionality for a business domain. The vision is to construct business applications without much programming from a predefined and interoperable set of business objects. An implementation of such object plug-and-play requires a *business object facility* (BOF), that is a development infrastructure and a run-time environment for business objects.

*This work was partially funded by the Commission of the European Union in the ESPRIT Project Open Business Object Environment (OBOE) under grant number 23233. A full report providing a more detailed comparison is available from LogOn Technology Transfer.

A major goal of business objects is to foster creation of a market for business objects that can be bought off-the-shelf and be reused in many different settings. These *Common business objects* are defined as “*objects representing those business semantics that can be shown as common across most businesses*” [8]. They represent independent components with a defined behaviour that can be employed in different applications by simply plugging them in. Business objects have object-oriented features, such as identity, attributes, behaviour and relationships. There are, however, further requirements for business object facilities.

Simplicity: Business object developers want to concentrate on business solutions. Infrastructural concerns, such as transactions or persistence should be entirely transparent for them.

Reusability: A business object should be able to support applications in different business domains. The business object facility should facilitate the construction and execution of such reusable components.

Extensibility: Business objects should be extensible with mechanisms, such as inheritance and delegation. In this way, specialised solutions can be created quickly from common business objects.

Scalability: A key feature enabling scalability is distribution. Business objects should therefore be distributeable across different servers.

Heterogeneity: Business objects should be able to communicate with each other although they may reside on heterogeneous hardware and operating system platforms and may be written in different programming languages. The BOF should hide any heterogeneity from the developer.

Portability: Business object implementations should be platform independent. BOFs should hide the underlying operating system and legacy applications and make business objects portable between different deployment platforms.

Interoperability: There will be several BOF providers in the market. Business object that resides in one BOF should be able to communicate with business objects residing in other BOFs.

The contribution of this paper is an overview and an analytic comparison of business object approaches. In the next section, we extend the above requirements into a conceptual model for comparing business object approaches. In Section 3, we compare the combined submission to the OMG Business Object Request for Proposals, the SSA BOF, Sun’s Enterprise JavaBeans, IBM’s San Francisco project, Microsoft ActiveX and SAP’s business object framework. In Section 5, we conclude the paper by discussing how they influence the development of future business applications and software engineering in general.

2 A Model for Comparing Business Object Approaches

We use a common model for describing and comparing business object approaches. Our model can be considered as a layered architecture. It is shown in Table 1 and has four layers. While the top three layers are ordered according to the level of abstraction they achieve from the underlying hardware and software infrastructure, the bottom layer deals with concepts.

The *concepts layer* describes the concepts used by a BOF to manage business objects and their interoperability.

Layer	Issues
Business Object Solution	Common Business Processes
	Common Business Objects
Business Object Facility	Model Integration
	Representation Formalism
	Meta Model
	Interfaces
Distribution Infrastructure	Application Integration
	Services
Concepts	Basic Technology
	Conceptual Model

Table 1: A Comparison Model.

The *distribution infrastructure layer* is concerned with the middleware used by a BOF. We are interested in the *basic technology* used for communication between business objects and the platforms and languages supported. Moreover, we assess the support for basic *services*, such as naming, events, life cycle, persistence, transactions, concurrency control, relationships, externalisation, query, licensing, properties, time, security, object trader, and object collections.

In the *business object facility layer* we consider BOFs. We focus on available mechanisms for legacy *application integration*. We investigate the *interfaces* business objects provide for other business objects or clients, and how a business object can access BOF services. We review the concepts available in a business object *meta model* for specifying business objects and constructs a *representation formalism* provides for visualising the meta model concepts. Finally, we evaluate the degree of *integration* of the approach with business analysis models (such as UML), interface definition (such as OMG/IDL) and implementation languages (such as Java or C++).

The *business object solution layer* is concerned with concrete implementations of business objects that can be reused for a business application. We review *common business objects* that implement common functionality for many business domains and assess *common business processes* that provide skeletons for particular domains.

3 Comparative Analysis of Business Object Approaches

We analyse six business object approaches from leading object technology vendors and compare the features they provide. The descriptions of the approaches are structured according to the above model.

3.1 Combined OMG Business Object Submissions

The Combined Submission to the OMG Business Objects Request for Proposals consists of three complimentary submissions: The Business Object Component Architecture (BOCA) [10], an interoperability specification [12] and common business objects [11]. Together, they cover all issues of the model we introduced except for common business processes. Although the submissions have not yet been adopted two implementations are currently being developed by EDS and Data Access.

Conceptual Model The OMG approach introduces the notion of a *type manager* for managing business objects. It is an object representing all instances of a type and its sub-types. For each “managed” business object type a type manager will be included in each business system domain (BSD). A BSD is a distributed object system for a particular business domain. BSDs are interconnected by *adapters*. The type manager provides interfaces for object *life cycle* management such as creation, deleting, copying and moving. A type manager supports *queries*. They return a set of business objects meeting certain criteria. The type manager also facilitates *introspection*, that is run-time type information. *Relationship-based dependencies* facilitate triggering actions on objects when related objects change.

Basic Technology The OMG submission uses CORBA as distribution infrastructure. It is, therefore, open and platform independent. Business object interfaces are defined in CORBA IDL. For this reason, any programming language with IDL binding can be used for business application development. A restriction of the approach is that it requires a CORBA-compliant ORB as run-time support. Moreover, the approach still requires substantial programming. Developers have to implement business objects as CORBA server objects and clients have to use an IDL binding to a programming language in order to request functionality from a business object.

Services The approach defines five services, namely transaction, naming, security, event and collection. The transaction and the naming service are advanced versions of the CORBA Naming and Transaction services [9]. The *naming service* is extended to support a string based specification of naming context paths and to access and modify all names bound to a specific object reference. The *transaction service* is extended with three features. At the beginning of a transaction access to a shared session object is granted. This object manages the transaction context for the facility. Furthermore, two messages for deadlock detection are added and three transaction commit phases are defined (propagate, validate and synchronise).

Application Integration The approach gives no technical description on how to integrate legacy applications. The mechanism intended to provide interfaces to databases of application suites are *legacy wrapper objects*. They have limited functionality and must not fully comply with the business object specification. They do not provide notification services and type managers. Details are left to implementers of the facility.

Interfaces Basically, two interfaces have to be distinguished. One is provided by *BusinessObject*, the other is represented by *TypeManager*. *BusinessObject* specifies the interface an OMG business object has to provide. It inherits from the classes *IndependentObject* and *LifeCycleObject*. *BusinessObject* itself is the base class for all application specific objects. It exports generic methods for managing specific attributes, relationships and operations of those objects. The class *TypeManager* also inherits from *IndependentObject* and defines an interface for managing subclasses of *BusinessObject*. It defines an operation create for instantiating a new business object as well as objects to provide the features, such as queries and introspection described above.

Meta Model The Business Object Component Architecture (BOCA) of the submission defines a meta model for business objects. It introduces concepts, such as business object types, dependent types, features, business system domains, an event model. The BOCA meta model integrates these concepts with those of the CORBA Core meta model.

Representation Formalism The BOCA submission specifies a Component Definition Language (CDL) for representing business objects. CDL is a textual language covering enterprise object interfaces, structural relationships between enterprise objects and collective behaviour of communities of enterprise objects. CDL is designed to represent all concepts of the above meta model.

Model Integration UML [7] is proposed to be the analysis notation preceding a CDL business object design. BOCA is aligned as an UML extension. A mapping between the event-condition-action extension to UML and CDL exists. The CDL language contains CORBA IDL syntax and semantic and thus interface definition in IDL can easily be extracted from a CDL representation.

Common Business Objects Two sets of common business objects are provided by the OMG approach. One is intended to provide a baseline for business objects concerned with business financials, order management and warehouse management. The objects specified are *DDecimal*, *Description*, *Address*, *Time* and *InvolvedParty*. The second set defines people, places, resources and processes. It is intended for desktop managers, workflow systems and resource managers.

3.2 SSA Business Object Facility

We now explain the BOF of Software Systems Associates (SSA) [14]. In terms of our comparison model, SSA covers the distribution infrastructure layer and parts of the business facility layer, namely application integration, object management and interfaces. It is based on the concept of executable objects (XOs).

Conceptual Model An XO represents a component that is also a business object. The management of XOs is accomplished by a BOF. Each process or address space has one BOF. It consists of three major parts. A *BofRequestMgr* is responsible for message handling. It packages messages, delivers them and handles delivery errors. A *BofObjectToken* Pool represents a collection of *BofObjectTokens* each of which is a placeholder for an XO. A *BofBaseObject* Pool is a collection of *BofBaseObjects*. They provide the interfaces through which XOs can be messaged.

Basic Technology The basic technology for the SSA BOF is CORBA and therefore platform or language restrictions are linked to the availability of a CORBA environment for these platforms or languages.

Services The approach provides six services, namely events, externalisation, persistence, relationships, security, and transactions. SSA use CORBA services to some extent. Like the CORBA Event service, the SSA *event service* defines two roles, a supplier and a consumer. The *externalisation service* externalises semantic rich data based on SDOs (Semantic Data Objects). CORBA types are not sufficient for this purpose. The *life cycle service* is also not based on CORBA. It supports creation, deletion, copying, moving, activation and deactivation of business objects. In their specification SSA state to use the CORBA Persistent Object Service as a basis to implement the BOF *persistence service*. The CORBA Persistent Object Service, however, was never implemented and we assume that SSA will use the Persistent Object State service that will replace it. The XO *relationship service* is a simplified version of the complex and powerful CORBA Relationship Service, although it does not rely on it. The

use of the CORBA Security Service is encouraged. The SSA approach introduces a more powerful *transaction service* than CORBA but uses some of the features of the CORBA Transaction Service.

Application Integration The issue of integration of legacy applications is not particularly addressed by the SSA approach. It is stated that an XO could serve as an adapter to legacy applications using SDOs. SSA claims to have successfully used SDOs for wrapping legacy Unix and AS/400 code.

Interfaces Each XO is wrapped by an instance of *BofObject*. The class *BofObject* is a subclass of *BofBaseObject* and provides the interface to an XO. It contains an operation to deliver a request to the XO implementation. Each XO class is a sub-class of the class *Base* providing all services available within the framework. A class *BOF*—*System* has exactly one instance at runtime and provides a query message in order to get a reference to the various BOF manager objects such as *BofRequestMgr*. All messages are sent to an XO by getting its handle through the token and sending a request to the *BofRequestMgr*. Each request is represented by an instance of a Semantic Data Object.

3.3 Enterprise JavaBeans

Sun released version 0.9 of Enterprise Java Beans [15] in February 1998. EJB extends Java with a distribution infrastructure layer and parts of the BOF layer of the comparison model.

Conceptual Model The EJB architecture defines three components with different roles in the management of enterprise beans. An *Enterprise JavaBeans class* represents the definition and implementation for a business object. An *Enterprise JavaBeans container* is responsible for life-cycle management of enterprise beans. It allocates threads, initiates the component, and manages resources on behalf of the bean. Each container can host many beans of different types. An *Enterprise JavaBeans server* is responsible for providing system-level functionality, such as transactions, memory management, database connections, security enforcement.

Basic Technology EJB is based on Java. It uses Java RMI for communication between remote objects. Enterprise JavaBeans require a Java Virtual Machine (JVM). EJB is therefore confined to platforms with a JVM. Furthermore, enterprise beans have to be implemented in Java.

Services The EJB specification demands only a very basic set of services from complying implementations. These are persistence, transactions, exception handling, and security. *Persistence* is provided by means of entity beans, a special kind of enterprise beans that can be made persistent. The *transaction service* follows the CORBA Transaction service, but only supports flat transactions. For *security services*, EJB uses the standard Java security API defined by *java.security*. Other services such as naming (Java Naming and Directory Interface - JNDI) can be used as supported by standard Java APIs.

Application Integration No mechanism is defined for accessing legacy applications. The specification states that interfaces to external data sources can be provided by the EJB containers. Different containers are supposed to support different legacy applications.

Interfaces An enterprise bean is accessed through its *EJB Home* and *EJB Object* interfaces. The Home interface provides methods for creating and locating enterprise beans. It is defined by the enterprise bean and implemented by the container at deployment time. The Object interface provides access to a bean's business methods. The container creates an Object interface for each enterprise bean instance according to the definition provided by the bean class. Both Home and Object interfaces have to comply with RMI. For communication with the container, an enterprise bean must implement the *SessionBean* or *EntityBean* interface respectively. This interface contains methods for setting a bean's context, destroying, activating, or passivating it. The container on the other hand provides a *SessionContext* interface allowing a bean to get information about its Home and Object interfaces and current clients of its methods.

3.4 IBM San Francisco

The IBM San Francisco (SF) project [4] was launched in order to accelerate business application development. The version 1.0 was delivered in 1997. SF covers the basic distribution layer, parts of the BOF layer, and the complete business object solutions layer of our comparison model.

Conceptual Model The management and administration of SF business objects is accomplished through the *foundation layer* of the framework. It provides *Object Model Classes* for the basic structure of San Francisco objects and frameworks. Application classes inherit from the Object Model Classes. This enables a framework to call operations of business objects. The business object instances are managed during framework execution by a *Factory*. It includes functions to create and delete objects

Basic Technology San Francisco is based on Java technology and thus needs a JVM for running and restricts application development to the Java language. The basic communication technology employed is Java RMI, which is extended by San Francisco.

Services The services provided by SF are based on OMG definitions such as transaction service, collections, or persistence. They are simplified or extended where necessary. There are also some features included provided by Java. While this sounds very promising, a detailed examination of the services was not possible due to a lack of publicly available information.

Application Integration SF provides *schema mappers* that facilitate access of relational databases from San Francisco objects. A second area of integration is that with legacy code written in languages other than Java. This is said to currently being examined as well as the possibility to call San Francisco objects from external applications.

Interfaces Business objects inherit the functions provided by the San Francisco Framework from SF base classes. However, the technical details of the interfaces were not available for evaluation.

Model Integration Although San Francisco does not provide a meta model and a notation for business objects, the approach is well integrated with existing representation formalisms. The common business objects and processes described below are delivered as Rational Rose UML models. Thus,

an application developer can simply take the basic models and extend them as necessary for the special application. Moreover, a code generator is provided directly translating the UML models into Java code according to the San Francisco framework.

Common Business Objects The common business objects IBM submitted to the OMG as described above are based on San Francisco experiences. For this reason, they are also part of the San Francisco approach.

Common Business Processes The application domains addressed by San Francisco include business financials, order management and warehouse management. The initial toolkit contains a *General Ledger* framework, different *Common Business Objects*, and a *Base* infrastructure. The *AR/AP Ledger* framework provides basic functionality for recording and processing accounts receivable and accounts payable. A business task *Payment* is included as well as *Transfer Item*. The *General Ledger* framework, on which the *AR/AP Ledger* is based, supports basic ledger functionality. Examples are *Journaling* and *Closing*. The *Sales Order Management* framework provides functionality for managing quotations, sales orders, and sales order contracts. The *Purchase Order Management* framework focuses on creating and managing purchase orders and contracts. Both frameworks rely on a general *Order Management* framework. It provides an abstract model for order processing across several processes. The *Warehouse Management* framework, on which order management is based, supports warehouse logistics.

3.5 Microsoft ActiveX

Microsoft does not provide facilities explicitly addressing distributed business objects or common business objects as described above. However, Microsoft provides a large spectrum of technologies for component-based software development. They are based on COM and their marketing term is ActiveX. The latest product providing support for developing distributed interoperable components is Microsoft Transaction Server (MTS) [5], which we want to include into our survey. It covers the distribution infrastructure layer of the comparison model.

Conceptual Model The Microsoft Transaction Server defines five main concepts. *Base processes* represent clients like desktop application or internet browser requesting tasks from the server. *Application components* encapsulate the business logic in an ActiveX component. They are deployed in the MTS runtime environment. The *Transaction Server Executive* provides the “plumbing” service to application components, that is the server executive hosts components. *Resource dispensers* provide a mechanism to manage the shared state in a process like database connections. *Resource managers* are systems services managing durable data like MS SQL Server or persistent message queues, such as Microsoft’s Falcon.

Basic Technology Components managed and administrated by MTS have to comply with ActiveX requirements. Every language supporting COM can be used to write components. The communications mechanism thus is COM or DCOM. The platform is restricted to Windows NT 4.0, which is necessary to run the MTS.

Services Microsoft Transaction Server offers a small subset of the CORBA services. *Transactions* are supported by MTS through the Microsoft Distributed Transaction Co-ordinator (MS DTC). An *automated distributed security service* provides programmatic as well as declarative security features. *Life cycle* management is also provided, components can be activated and deactivated.

3.6 SAP Business Framework

SAP develops and distributes R/3, a standard business software package with more than 12,000 installations. R/3 covers a wide range of business domains such as financials, human resources, logistics, manufacturing. Though very broad, R/3 is not sufficient to meet very specific requirements. The SAP Business Framework [13] provides business object interfaces to its R/3 application suite. The SAP approach is different from the ones described above. It is not intended as a facility that supports the construction of business objects but it wraps R/3 using business objects. Then non-SAP applications can interoperate with R/3 using these business objects. The approach covers distribution infrastructure as well as parts of the business object facility and the business object solution layers.

Conceptual Model SAP introduces a *business object repository* based on R/3. The repository is a new layer in its application ecosystem. A *SAP business component* consists of business objects supporting and encapsulating certain business functionality of R/3. It provides a stable set of interfaces. Each component has its own life cycle concerning development, deployment and maintenance.

Basic Technology Access to SAP business objects can be obtained through COM/DCOM, CORBA or Java. The main restriction is that the business objects cannot be extended or adapted but only accessed as they are from external applications. To integrate business objects internally, R/3 relies on a technology called ALE (Application Link Enabling) representing a message broker within the R/3 business object repository.

Services Services for the business objects are provided implicitly by the R/3 system. However, when a business object is accessed from an external application, an integration of the operations on business object in the application transaction might be useful. Information on possible integration with external services such as transactions or security is not available.

Common Business Objects A large amount of business objects can be provided by the SAP approach, covering all SAP R/3 domains.

Common Business Processes Business components group business objects and implement business functionality. These components can be seen as common business processes. Again they can only be interfaced by external applications but not be extended and tailored.

4 Summary and Competitive Comparison

We have described six business object approaches based on a common conceptual model. Table 2 summarises our findings. In this table, - - means that a BOF does not support an element of our model, -

Criteria	OMG	SSA	EJB	SF	MS	SAP
Common Business Processes	--	--	--	+	--	+
Common Business Objects	0	--	--	+	--	++
Model Integration	+	--	--	++	--	--
Representation Formalism	0	--	--	--	--	--
Meta Model	+	--	--	--	--	--
Interfaces	+	+	+	?	--	--
Application Integration	-	-	--	0	0	+
Services	+	+	0	?	0	?
Basic Technology	++	+	0	0	0	+
Conceptual Model	+	+	+	?	+	+

Table 2: An Analytic Comparison

means that there is limited support, 0 means reasonable support, + means good support and ++ indicates excellent support.

4.1 Comparison based on Approach

The OMG approach provides a wide coverage of features of business object approaches as defined by the comparison model. The specification is based on CORBA and currently being implemented by several vendors. Its main advantage is that it is entirely open and covers all aspects from basic infrastructure to common business objects. However, the approach seems to be put together from various ideas and the three specifications are not well integrated. Moreover, the meta model seems rather complex and the practicability of the textual specification language CDL has still to be proven.

The SSA approach also relies on CORBA but only defines services and basic object management facilities. The approach appears more compact and clear than the Combined OMG submission. SSA are currently implementing the approach within ESPRIT project OBOE.

Sun's Enterprise JavaBeans extend Java technology with component server capabilities. The approach covers the same range of features as the SSA approach. Services, however, are weak and need to be specified in more detail. Moreover, EJB are bound to Java.

IBM's San Francisco is already available as Version 1.0 and provides all features except a modelling approach. Its strengths are undoubtedly the common business objects and processes that are supposed to provide about 40% of an application. Its integration of legacy applications, in particular relational databases, is better than in the other approaches.

Microsoft's Transaction Server does not explicitly support business objects but provides features for developing components. MTS is available in Version 2.0. The services provided as well as the mechanism for integrating (Microsoft) legacy applications can compete with the other approaches. The Microsoft approach however is based on COM, a proprietary object model.

SAP's Business Framework is not intended to support the development of business objects but provides a business object interface to SAP R/3. The approach therefore is especially strong in the area of common business objects and processes as well as application integration (with R/3). Another advantage is that the business objects can be accessed from CORBA, COM/DCOM and Java. The participation of

SAP business objects in global services, such as transactions remains to be demonstrated.

4.2 Comparison based on Functionality

The concepts layer shows differences due to the employment of different technology.

The distribution infrastructure layer is addressed by all approaches. The CORBA-based approaches provide richer services than their competitors because they can rely on existing CORBA services. Furthermore, they are language and platform independent. Enterprise JavaBeans and San Francisco are bound to Java as language and Microsoft Transaction Server is only available on Windows NT 4.0 Servers. SAP is the most powerful approach concerning access to business objects but most limited concerning their development.

The bottom features of the business object facility layer, namely application integration, and interfaces are addressed by all approaches. They provide similar features. OMG, SSA and EJB are still in a specification state. Modelling is only supported by the OMG approach. The approach, however, lacks integration with existing modelling techniques, such as UML and the OMG Meta Object Facility. San Francisco is well integrated with existing modelling approaches. Business object solutions are delivered as UML models and can be extended using Rational Rose. A code generator then creates Java code out of the UML model.

The business object solution layer is best addressed by SAP's approach but also limited in the sense that it is not extensible by a non-SAP application developer. The San Francisco approach also provides a reasonable set of predefined solutions.

5 Conclusions

The approaches we assessed are at different stages of their development and they are based on different technologies. Most approaches compared in this paper are not yet fully mature and their practicability still has to be proven. If they are successful, however, they could have serious impacts on how software is developed.

Business applications composed from business objects will be different from conventionally developed ones. Business applications will consist of heterogeneous and distributed business objects that encapsulate certain functionality. Business objects exist independently and autonomously. They are integrated into coherent software architectures by business object facilities.

Software processes that deploy business objects will be different, too. They will have to be more flexible and adaptable. In particular, they will be less geared towards the development of objects. They will favour deployment of business objects that were developed elsewhere. Future software processes will involve explicit make-or-buy decisions. These have to be supported by tasks that research the market for suitable objects and activities that evaluate candidate objects.

Acknowledgements

We would like to thank Peter Eeles and Paul Tanner for their useful comments on the report on which this paper is based.

References

- [1] V. R. Basili and H. D. Rombach. Support for comprehensive reuse. *Software Engineering Journal*, September 1991.
- [2] E. Berard. *Essays on Object-Oriented Software Engineering*. Prentice Hall, 1993.
- [3] G. S. Blair, J. J. Gallagher, D. Hutchinson, and D. Shepard. *Object-Oriented Languages, Systems and Applications*. Halsted Press, 1991.
- [4] K. Bohrer. Middleware Isolates Business Logic. *Object Magazine*, November 1997.
- [5] S. Hillier. *Microsoft Transaction Server Programming*. Microsoft Press, 1998. To appear.
- [6] C. W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2), June 1992.
- [7] P.-A. Muller. *Instant UML*. Wrox Press, 1997.
- [8] OMG. COMMON Facilities RFP-4: Common Business Objects and Business Object Facility. Technical Report TC Document CF/96-01-04, Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, JAN 1996.
- [9] *CORBA services: Common Object Services Specification, Revised Edition*. 492 Old Connecticut Path, Framingham, MA 01701, USA, March 1996.
- [10] OMG. Combined Business Object Facility: Business Object Component Architecture. Technical Report TC Document BOM/98-01-07, Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, JAN 1998.
- [11] OMG. Combined Business Object Facility: Common Business Objects. Technical Report TC Document BOM/98-01-06, Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, JAN 1998.
- [12] OMG. Combined Business Object Facility: Interoperability Specification. Technical Report TC Document BOM/98-02-03, Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, JAN 1998.
- [13] SAP. Benefits of the Business Framework. <http://www.sap.com/bfw/media/pdf/50016302.pdf>, 1998.
- [14] *OMG BODTF RFP-1 Submission - Business Object Facility*. 500 West Madison Avenue, Chicago, USA, November 1997.
- [15] A. Thomas. Enterprise JavaBeans: Server Component Model for Java. Technical report, Patricia Seybold Group, 85, Devonshire Street, Boston Mass. 02109, DEC 1997.
- [16] J. Udell. Componentware. *Byte*, May 1994.