Business Objects: The Next Step in Component Technology?*

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

Abstract

Component technology seems to be a promising approach towards more efficient software development by enabling application construction through "plug and play". However, the middleware supporting this approach is still complicated to use and distracts the attention of the component developer from the application domain to technical implementation issues. Business objects are intended to hide the complexities of middleware approaches and provide an easy to use environment for application developers. We conceptualize business object approaches by presenting a common model and survey some major players in the marketplace. We conclude by identifying implications of business objects on information systems engineering.

1 Introduction

Component-based software development is intended to facilitate and speed up application development. Reusable components are supposed to be plugged together in a distributed and inter-operable environment. Middleware systems such as CORBA, DCOM or Java RMI provide the necessary infrastructure for this purpose. Middleware systems offer a number of facilities important for information system development, including transactions, persistence and security. They are, however, rather complicated and difficult to use. It is desirable to hide the complexities from application developers and allow them to deal with concepts they know from their application domain rather than with implementation issues. An approach towards achieving this goal are business object facilities and common business objects.

In this paper we summarize a study that presents an overview and evaluates business object approaches that are currently be developed or are on the market already. We characterize business object facilities and common business objects and give a brief overview of six competing approaches, namely the Combined Submission to the OMG Business Object Domain Force (OMG approach), the SSA Business Object Facility (SSA approach), Sun's Enterprise JavaBeans (Sun EJB), IBM's San Francisco Framework (IBM SF), Microsoft ActiveX (MS ActiveX) and the SAP Business Framework (SAP BF). Finally, we conclude with some important implications of the business object approach for information systems engineering in order to encourage discussion on their advantages and disadvantages. A detailed description of our study focusing on an in-depth analysis and comparison of the approaches can be found in [2].

*This work was partially funded by the Austrian FWF (Fonds zur Förderung der wissenschaftlichen Forschung) under grant J1526-INF.

2 Business Object Concepts

In its 1996 Request for Proposal the OMG defined Common Business Objects and Business Object Facilities as follows [4]:

Common Business Objects (CBO): Objects representing those business semantics that can be shown to be common across most businesses.

Business Object Facility (BOF): The infrastructure (application architecture, services, etc...) required to support business objects operating as cooperative application components in a distributed object environment.

In the sense of these definitions, the Common Business Objects can be regarded as components providing core application functionality for a certain business domain. They are intended to build the kernel of applications tailored to the needs of a certain organisation. Business Object Facilities on the other hand represent the enabling technology for creating and executing business objects. They aim at hiding middleware complexities from application developers. The business object approach promises to realize the following achievements:

Simplicity: Business object developers can concentrate on business solutions. Infrastructural concerns, such as transactions or persistence are entirely transparent for them.

Reusability: A business object is able to support different applications. The business object facility supports the construction, integration and execution of such reusable components.

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

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

Heterogeneity: Business objects are 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 hides any heterogeneity from the developer.

Portability: Business object implementations are platform independent. BOFs 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 objects that reside in one BOF are able to communicate with business objects residing in other BOFs.

In order to understand and compare the various business object approaches, we developed a common model for describing their features and characteristics. It can be considered as a layered architecture. While the top three layers are ordered according to the level of abstraction they achieve from the underlying hardware and software infrastructure, the lowest layer deals with conceptual issues. Figure visualizes the model.

The **concept layer** is intended to provide the ”big picture” of an approach. It describes how an approach manages business objects and their interoperability at a high level of abstraction without technical detail.

The **infrastructure layer** is concerned with the underlying middleware of an approach. Two issues are addressed. The *basic technology* issue covers the infrastructure used for brokering messages. Examples are OMG/CORBA, Microsoft DCOM, or Java RMI. Furthermore it covers operating system platforms and programming languages supported. The *services* issue is concerned with system-level services a business object developer is provided with in order to create applications. Examples are naming, events, life cycle, persistence, transactions, concurrency control, relationships, externalisation, query, licensing, properties, time, security, object trader, and object collections.

The **facility layer** deals with business object facilities. According to our model, this layer contains five issues. First, we consider *interfaces*. On the one hand, clients or other business objects need to access a business object;

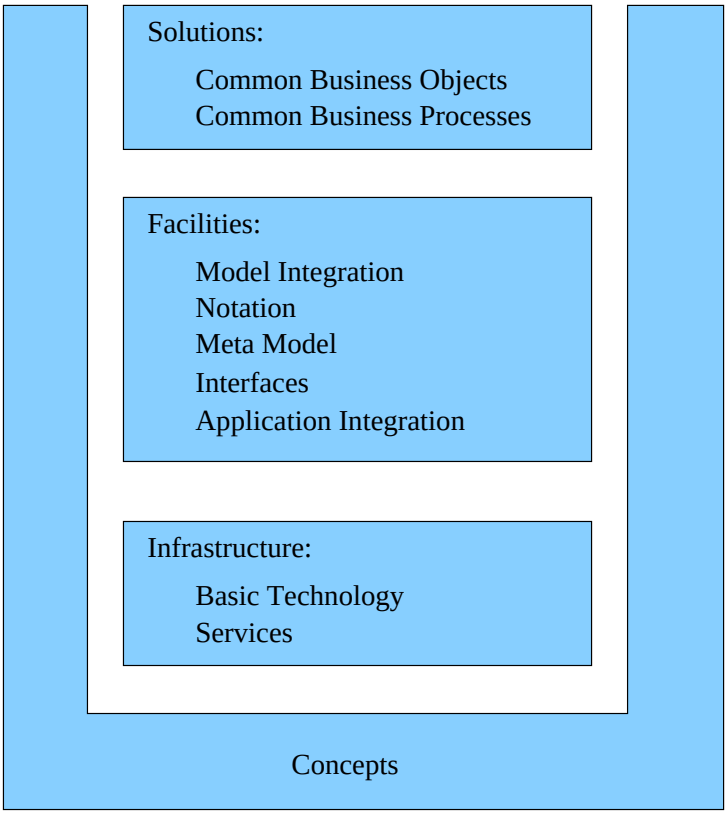

Figure 1: Common Model of Business Object Approaches

on the other hand, a business object has to access BOF services. Access to clients can for example be granted via static or dynamic CORBA interfaces or through RMI-compliant Java interfaces. Services can for example be made accessible by letting a business object class inherit from a system provided super-class. A very important feature of business objects is their ability to access legacy applications such as databases or application suites. The *application integration* issue focuses on the access mechanism rather than on the currently available set of accessible applications. Mechanisms for accessing legacy databases are for example schema mappers linking object attributes to tables in a relational database. For the development of applications based on business objects, it is important to provide modelling support. Two issues are concerned with modelling. First, modelling concepts are needed in terms of a *meta model*. It defines the constructs available for specifying business objects. Examples of such constructs are business object types, business system domains, or events. Beyond a meta model, a *notation* for representing business objects is needed. The representation has to cover the semantics of the business objects rather than a technical description of the interface specification. Last but not least, we evaluate the degree of *model integration* of the approach with business analysis models on the one hand and interface definition or implementation languages on the other hand. Integration with analysis models means that there should be a mapping between analysis model such as a UML specification and the business object model. On the other hand, a mapping between business object model and interface definition languages such as CORBA IDL or implementation languages such as Java is desirable.

The **solution layer** comprizes the highest layer of the comparison model. It is concerned with concrete implementations of business objects that can be used by a developer as a basis for a business application. Two issues are considered by our model. *Common business objects* on the one hand represent semantics common to a whole business domain. Their intend is to relief an application developer from implementing features that can be provided by pre-built components because of their generality. *Common business processes* on the other hand go even further than common business objects and provide a skeleton application for business domains. That means not only common objects are identified and implemented but also their interactions in order to perform a broader business task.

3 A Brief Survey of Existing Approaches

Combined Submission to the OMG Business Object Domain Force The OMG approach [5] provides a wide coverage of features of business object approaches as defined by the common model. It is based on a type manager for managing business objects. The basic technology is of course CORBA and five services are supported that are slightly different from the basic CORBA services. This approach also provides a meta model defining concepts for modelling business objects as well as a representation formalism. The Component Definition Language (CDL) is a textual language covering business object semantics. The approach integrates with UML as an analysis approach and CORBA IDL for interface definition. A set of common business objects for business financials, order management, and warehouse management is also provided. The specification is 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 it contains 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.

SSA Business Object Facility The SSA approach [7] relies on the concept of executable object (XO) as business object that are managed by the BOF. The basic technology is also CORBA. Six services are provided by the approach, which again differ from the basic CORBA services. Semantic data objects (SDOs) are used for wrapping legacy applications. The SSA approach only covers the infrastructure layer and provides the application integration and interface features from the facilities layer. It appears more compact and clear than the Combined OMG submission. SSA is currently implementing the approach within ESPRIT project OBOE.

Sun's Enterprise JavaBeans The EJB approach [8] is based on three components, namely Enterprise JavaBeans classes implementing business objects, Enterprise JavaBeans containers responsible for life-cycle management of the beans, and Enterprise JavaBeans servers for providing system-level functionality. Sun's Enterprise JavaBeans extend Java technology with component server capabilities based on Jave Remote Method Invocation (RMI). The approach covers the same range of features as the SSA approach. Services, however, are weak and need to be specified in more detail.

IBM's San Francisco Framework The IBM initiative [1] is based on Java technology but extends the basic RMI mechanisms. The services provided are based on the OMG specifications but simplified or extended where considered to be necessary. Furthermore Java features are included. Database applications can be integrated via schema mappers. San Francisco is well integrated with existing modelling approaches. Rational rose models can be directly translated into Java by the means of a code generator. Common business objects as well as common business processes are provided by the framework. 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.

Microsoft ActiveX Microsoft provides a set of component technologies marketed under the name ActiveX. However, business objects as defined in this paper are not explicitly addressed. The Microsoft Transaction Server [3] provides some basic facilities that might be of benefit for business objects and thus is included into this survey. It hosts ActiveX components and provides distribution, life-cycle, and security services for them. The base technology is COM/DCOM. MTS is available in Version 2.0 and needs a Windows NT Server for running. The services provided as well as the mechanism for integrating (Microsoft) legacy applications can compete with the other approaches. An advantage of the Microsoft technology is that it is already available and in use while many of the other approaches are still in a specification state.

SAP Business Framework An alternative approach to the ones introduced above is SAP's Business Framework [6]. It is not intended to support the development of business objects but provides a business object interface to SAP R/3. An object repository layer is established on top of R/3 providing access to non-SAP applications.

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 Conclusions

This paper was devoted to the next step in component technology represented by business object approaches. We developed a conceptual model consisting of four layers, namely concepts, infrastructure, facilities, and solutions. We furthermore provided a brief overview of six competing approaches. To conclude this paper, we want to point out some implications of business objects on information systems engineering as a basis for discussing pros and cons.

Business object approaches will have impact on the *development processes*. Applications will be built in a bottom-up style rather than top-down by plugging together various existing and newly developed business objects. The processes are more likely to follow a incremental approach rather than a waterfall model; applications can be assembled rapidly, enabling cross-checks with the users. Furthermore, development processes are supposed to be shorter. Executables can be reused instead of high level design documents or code.

Business application developers will need less implementation knowledge and can concentrate on the application domain. Services, such as transactions, persistence, security, life cycle management will be provided by the business object facility. "Intelligent IT users" can be integrated in the process as developers rather than users stating requirements that then have to be interpreted and realized by implementation specialists.

Development tools will need to be sophisticated in order to hide implementation complexities from application developers. This is a challenge for tool providers. Integrated toolkits covering all phases from analysis to implementation will be needed. In combination with business object approaches, appropriate development tools will make component technologies usable.

A *market for common business objects and processes* will be established. Applications will be assembled from bought as well as self-developed objects. The current "reinvention of the wheel" phenomenon can be avoided by making independent objects inter-operable and deployable in different environments.

Another interesting implication of business objects are enhanced *integration possibilities* with other information systems such as workflow systems. Current systems suffer from a lack of integration of application functionality. Based on component technology, workflow systems could evolve into "business operating systems" providing a graphical user interface as well as process support for application components.

References

[1] K. Bohrer. Middleware Isolates Business Logic. *Object Magazine*, November 1997.

[2] W. Emmerich, E. Ellmer, Birgit Osterholt, and Roberto Zicari.

[3] S. Hillier. *Microsoft Transaction Server Programming*. Microsoft Press, 1998. To appear.

[4] OMG, 492 Old Conneticut Path, Framingham, MA 01701-4568. *Common Facilities RFP-4: Common Business Objects and Business Object Facility*, January 1996.

[5] 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.

[6] SAP. Benefits of the Business Framework. `http://www.sap.com/bfw/media/pdf/50016302.pdf`, 1998.

[7] *OMG BODTF RFP-1 Submission - Business Object Facility*. 500 West Madison Avenue, Chicago, USA, November 1997.

[8] A. Thomas. Enterprise JavaBeans: Server Component Model for Java. Technical report, Patricia Seybold Group, 85, Devonshire Street, Boston Mass. 02109, DEC 1997.