INFORMATION ORGANIZER
A comprehensive view on reuse

Erik Gyllenswärd
Department of Computer Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden
Department of Industrial Information and Control Systems, Royal Institute of Technology, 100 44 Stockholm, Sweden
Compfab AB,Hylsvägen 4, 721 30 Västerås, Sweden, www.compfab.se
Email: erik.gyllensward@mdh.se

Mladen Kap
Department of Computer Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden
Compfab AB,Hylsvägen 4, 721 30 Västerås, Sweden, www.compfab.se
Email: mladen.kap@mdh.se

Rikard Land
Department of Computer Engineering, Mälardalen University, Box 883, 721 23 Västerås, Sweden
Email: rikard.land@mdh.se

Keywords: Reuse, integration, legacy systems, Business Object Model, software components, extensible, lifecycle support.

Abstract: Within one organization, there are often many conceptually related but technically separated information systems. Many of these are legacy systems representing enormous development efforts, and containing large amounts of data. The integration of these often requires extensive design modifications. Reusing applications "as is" with all the knowledge and data they represent would be a much more practical solution. This paper describes the Business Object Model, a model providing integration and reuse of existing applications and cross applications modelling capabilities and a Business Object Framework implementing the object model. We also present a product supporting the model and the framework, Information Organizer, and a number of design patterns that have been built on top of it to further decrease the amount of work needed to integrate legacy systems. We describe one such pattern in detail, a general mechanism for reusing relational databases.

1. INTRODUCTION

It is commonly believed that software reuse put into practice would solve many problems related to software development (Krueger 1992; Ambler 1998). There are many aspects of reuse: one can (at least in theory) reuse anything from mere concepts to data, information, program code, and executable components (see e.g. Krueger (1992)). However, in spite of the potential benefits of reuse, it has proved hard to put reuse into practice in a large scale. Related to reuse is the idea of integration – many organizations have a large number of legacy systems; an integration of these would provide great benefits by increasing the possibility to provide appropriate and related information in a timely manner. Is there any elegant solution to both of these problems – reuse and integration? We believe there is. In this paper we present a model for integrating existing applications, information and component reuse. The model is intended to cope with all aspects of an object and extensible enough to be used during the whole lifetime of a system.

The concept of reusing whole applications has been somewhat neglected in discussions of reuse. With this, we do not mean modifying applications to include new functionality, but rather to reusing whole applications, "as is", without need of access to source code, recompiling, reconfiguration or any

other modification whatsoever, much like "components" as defined by Szyperski (1998). If this is possible, integration is facilitated at a very low cost. Such attempts have been done (OMG; DCOM, 1996; DCOM, 1997) but have mostly been focused on debating the different competing standards for interoperability. Other attempts (IEC 1346-1; IT4, 1991; Krantz, L., 2000) focus more on information reuse and integration.

We have developed the *Business Object Model*, *BOM*, which defines a conceptual model for the integration of applications. To make BOM "come alive", it has been implemented in *Business Object Framework*, *BOF*. This implementation is the "core" of Information Organizer, a commercial product itself made possible through extensive reuse. Information Organizer has been used as the base for the implementation of several *application patterns*, such as a pattern for workflow applications and a pattern for database connection. For applications conforming to these patterns, it is possible to configure them to a particular organization's need with a minimum of effort.

We will thus cover three aspects of reuse throughout the paper: reuse of existing applications (through integration in a larger system), reuse of application patterns, and reuse to make the construction of Information Organizer possible.

We will use Information Organizer as a starting point and describe the features of the model and how it is realized in a framework in section 2; we then continue by describing the application patterns in section 3, and conclude with a discussion and a summary in sections 4 and 5.

2. THE MODEL AND THE FRAMEWORK

The *Business Object Model*, *BOM*, is a *model* which extend the concept of "directory enabled applications" (MSDN; Howes, T., 1997; King, R., 1999; Schwartz, R., 2000) with important capabilities for integration and modelling inspired by OMG (OMG) and IEC 1346-1 (IEC 1346-1). The Business Object Model, BOM, represents different entities of importance in a uniform way to the user. Business Object Model defines five central concepts: *objects*, *aspects*, *roles*, *relations* and *views*. *Objects* represent quite large grained entities such as issues, pumps or valves. An object can be described as an empty container; business logic is added in form of *aspects*. An object can play a number of *roles*, implemented by means of aspects. A *relation* connects objects, and finally, the concept of *views* provides a means to restrict access to a system and all its information.

While the Business Object Model is a conceptual model, the *Business Object Framework*, *BOF*, is a design environment provided to assist application programmers in building components and applications, and integrating existing applications. BOF thus provides an implementation of objects, aspects, relations, views and roles, as defined by BOM. It also contains tools for creating instances of these, finding them in a distributed environment and communicating with them. Business Object Framework can be described as a toolbox with a number of tools and software components common to different applications for effective reuse. The Business Object Framework is thus the *implementation* of BOM; it is based on Microsoft Active Directory and COM, and follows existing standards and de facto standards.

2.1 Business Object Model - BOM

We have designed Business Object Model to support cross-application integration and to be easily extensible. It supports integration through the means of *aspects*: different aspects can be associated with completely different systems. It is extensible in that an object can be extended with new aspects during its entire lifetime (without affecting other aspects of the object). It is important to understand that this model is independent of the manner in which the different external systems model their part of the entire activity; BOM resides "above" the systems it integrates – these systems need not be "BOM-enabled" in any way.

The five central concepts of the Business Object Model – objects, aspects, relations, roles, and views – are described in more detail below. Their relationships between these are also described in Figure 1.

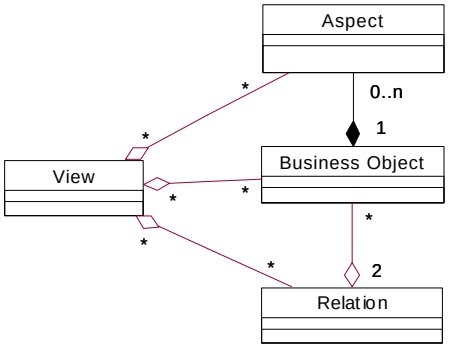

Figure 1: The relationships between the concepts.

Objects – The most central concept of BOM is the concept of *objects* (or *business objects*, to distinguish our notion from other uses of the term); these represent entities of interest in one or several applications. Examples of objects are issues, steps in a workflow, organizations, departments, or pumps and valves. An object usually contains very little, if any, information or implementation in itself. Rather, objects offer a uniform way to assemble related information through the concept of *aspects*.

Aspects – Instead of attempting to permit an object itself to represent all its behavior, part of its behavior is delegated to different *aspects*. This means that new aspects can be added to the object at any time during its entire life without necessarily affecting other aspects or the object itself. Objects and aspects offer the possibility of componentizing applications in a natural manner due to the fact that new business logic can be added to the object when the object is ready for a new role (roles are explained below). Aspects can either contain all business logic themselves or be used to associate existing applications or parts of existing applications with an object and thereby their reuse. For an issue, aspects could include mail, Excel sheets, PowerPoint presentations, reports, or video sequences; for objects in other domains, examples of aspects are process dialogs, CAD drawings, and invoices. It should be emphasized that both objects and aspects are complex entities, encapsulated into the system without applying any changes on the components themselves.

Relations – To be able to build a usable information system, objects can be *related* to other objects. The Business Object Model offers a relation model with both generic relations and typed relations i.e. relations with a strong semantic significance. New relation types can be defined in the system during its service life. Any number of relations can be associated with an object, and in this way both hierarchic structures and net structures can be built. New relation instances can be associated with an object at any time. This means that new relations can be associated with an object even if the object cannot utilize them, because the object is not aware of the relation and not implemented in such way that the relation can be used; however, these relations may be useful if an external user understands them and can interpret their semantics. With "external user" we mean both other applications and human users browsing through the information. Relations and aspects often occur together since aspects provide the semantics with which it is possible to interpret and utilize the relation. By extracting the relations and locating them outside the object, the architecture becomes adaptable in a changing world as new types of relation and instances can be added to the system, without affecting the existing functionality. This introduces a risk, however, since an object may assume that certain relations are present that has in fact been removed (without the object being informed).

Roles – The concept of *roles* is somewhat abstract, and must be seen in connection with objects, aspects, and relations. To take a simple example, a "person" object may play the role of a husband – to be able to play this role, it must have certain aspects, such as "being male" and "being grown-up". A more business-oriented example of a role is "to be participant in a workflow". A role can thus be said to define a certain function, or a set of capabilities, that can be offered by an object, and it is implemented by one or more aspects. A relation type associates two roles. A generic relation can associate any types of object as all objects are of the generic type.

Views – *Views* make possible the arrangement of objects, aspects and relations to limit the extent to

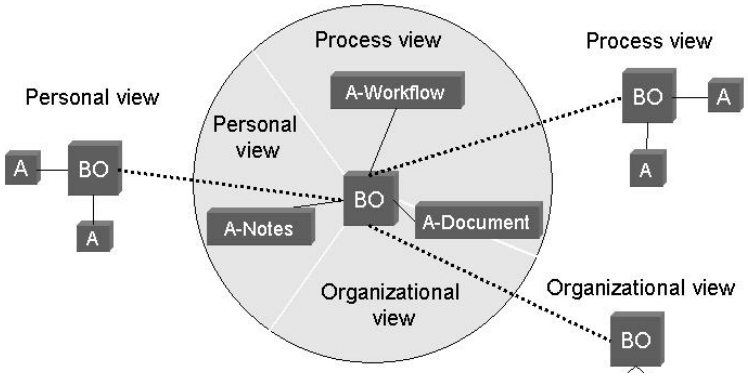

Figure 2: An issue with its aspects, relations, and views.

which they are accessible to different categories of users. This is necessary, partly because certain information is classified but also to reduce the volume of information presented to make it easier for the user to understand. Initially, a system most often contains a number of predefined views. A selected object will remain in focus if the user changes view – this is useful when a user finds an object in one view (such as his personal view) and changes to another (e.g. a process view, describing the object in the context of a workflow). The concept of views is very important when integrating different systems – a personal view would e.g. show all issues per individual, even if the issues originate from different issue management systems. Views can of course be added in the same dynamic manner as objects, aspects and relations.

Let us illustrate the relations between the five concepts using an example. In the center of Figure 2, there is a business object (BO) representing an issue in an issue-management system. With the circle we try to describe the visibility of the object in different views; in an issue-management system we can easily imagine the following views: a personal view showing all the persons dealing with issues and the issues for which they are responsible, a process view showing the issue's location in a workflow, and an organizational view describing the organization and all its employees. In the personal view, the issue and its relation to a user (its "owner") are visible; the object also has the aspect "A-Notes" indicating that personal notes has been added to it. To be able to participate in a workflow, the issue has been allocated the aspect "A-Workflow" and a relation to a workflow step; when the issue is processed, this relation will move to the next step (there are of course more steps visible in the process view than is shown in the figure). The organizational view shows how the organization is structured and, for each organizational unit such as a department, the issues associated with the department concerned. The aspect "A-Document" is placed on the white line to indicate that the document is visible in both the organizational view and the process view.

A user interested in how far in the workflow an issue has progressed can either browse through the process view to the issue of interest, or enter via another view, e.g. the personal view, find the object, select it and then change to the process view. The issue will then be in focus but visible in the process view, with the relevant relations and aspects.

2.2 Business Object Framework - BOF

Business Object Model is just a model, requiring considerable support in the form of tools and default implementations to be usable. Business Object Framework provides this support as a set of tools for building business objects. Some of these tools and functions are:

- A generic implementation of aspects, objects, views and relations.
- A configuration environment with tools and models for the simple creation of new instances of existing types and the easy configuration of new types.
- A development environment making it possible to programmatically add new components in the form of objects and aspects. The development environment of Business Object Framework is completely integrated with Microsoft Visual Studio, permitting the programming of objects and aspects, easily and in any of several well-known languages.
- There is also an API allowing dynamic creation of relations and views.
- A runtime environment making it possible to execute components locally on a client machine or centrally on one or more server machines. Business Object Framework also provides services for finding and calling components over both the Internet and an intranet.

BOF could be said to be the "core" of Information Organizer, because this is where BOM is implemented. In addition to this "engine", where the concepts of BOM are realized, Information Organizer includes other features, such as a user interface. The primary user interface for a user of the system is a standard browser. The system is largely based on the concept of "thin clients", even if "fat clients" are used with respect to certain functions and applications. An advantage with thin clients is that no code need be installed and maintained on the client machine. But if the system integrates legacy applications built without the Internet being taken into consideration, these applications must be installed in each client machine anyhow. The system also provides support for access to information via WAP.

2.3 Structuring and Search Mechanisms

The problem in large systems is not lack of information. The problem is often defective

mechanisms to keep related information together and ways to find accurate information when needed. This is one of the key problems we tried to solve with the Business Object Framework. That is why the framework provides three major ways to structure and search information.

- The first way is the most fundamental and is provided by the core of the framework and the object model. Information can, as we have described, be structured in form of objects in both multiple structures and multiple views. This can be used to create information models spanning several integrated applications.
- The Relational Database Connector is the second way to search for information. As good as every application has its own information model – i.e. internal structures. In case of database applications these structures are very often represented as database relations. The design pattern implemented in the Relational Database Connector provides a common way to follow these relations via an Internet-enabled user interface regardless of from which applications they originate.
- The third way is based on the concept of indexing. The basic idea is that information visible to a user can be indexed; usually it means that different kinds of files (such as documents) are indexed. Due to the fact that very much information is presented in the form of generated cards it is important that these cards can be indexed and in case of a hit the object represented by the card presented. By doing this, information in the system can be searched in the same familiar way as on the Internet; for example, keywords such as AND, OR and NEAR can be used.

Our experiences are that in large systems with huge number of objects the more static way to structure information is used to a less extent. The choice of structuring mechanism also depends on the nature of the application domain. In for example the automation industry some structures are of a quite static nature such as a structure representing the physical location of equipment. Thus are these structure quite familiar to people and they are used to follow for example the location structure to find a pump and all its aspects.

On the other hand when it comes to for example an issue management system people are very much influenced of the way information are structured and search for in a relational database. They are used to search for information in a variety of ways, which are impossible to foresee, and therefore more static structures cannot be used.

2.4 Integration

Aspects represent information included in the integrated system. The aspects can integrate information on different levels – at least three levels of integration can be identified: application level, business logic level and data level.

If the system is integrated on the application level, the application does not provide an API to its internal parts. When the application is referred to from an aspect, the application will be activated and the user will enter at the top level and is required to navigate to that part of the application at which the object (e.g. an issue) concerned is located.

To be able to integrate on the level of business logic the application must be componentized or provide an API permitting access to its different parts. I.e., when called, the application could itself receive a number of input parameters describing the part in which the user is actually interested and with the help of this information, navigate to the part concerned. The input parameters very much depend on the application to be integrated and are often stored in the aspect instance. The aspect can be seen as a gateway in between the framework and the integrated application. The complexity of the aspect implementation very much depends on the level of integration but also which kind of application to be integrated. If the application is COM based it is very likely to be easier because the framework itself is COM based. To manipulate the data, in this case an issue, the application's own dialogs are used i.e. its own business logic. For the user, a modular/component-based picture of the integrated application would be presented even if it is not implemented in a component-based manner.

Integration at data level means that data is accessed directly without invoking the business logic (code), which the integrated system itself makes available for the presentation, and processing of data. In many applications, this is an appropriate level of integration. It can be used to present information from many different systems but to change data, system dialogs already available should be used. The Relational Database Connector, described in section 3, is an example of a component providing support for the integration of applications on this level. By using the connector information stored in a relational database can easily integrated. If data is stored in some other data source a specific connector for that particular data source must be implemented. In practice, this level has been found to be very useful as a rapid integration can be performed and Business Object Framework features (such as access control) can be applied to each row

in the database because they are represented as Business Objects.

Integration at data level is most often a suitable level of ambition at which to begin. The level of ambition can be raised subsequently and integration can then be performed on the business logic level.

3. APPLICATION PATTERNS - ONE WAY OF REUSE

A design pattern is a solution to a problem that occurs over and over again (Gamma et al, 1995; Buschmann et al, 1996). We have identified three major *application patterns* and implemented these in Information Organizer, using Business Object Framework and the concepts defined by the Business Object Model. With an application pattern, we mean a solution to a problem that occurs in many applications, such as a "workflow" pattern. In our case a pattern is implemented as a number of objects, aspects, and relations. These patterns present a number of benefits: first, they are common to many applications and can thus be used in many contexts, and secondly, application boundaries are crossed. Moreover, due to the modular model of BOM, several patterns can be applied simultaneously; any object can be extended with the aspects implementing a pattern. We have used a number of patterns in practice when developing a document and issue-management system (Arch Issue).

3.1 Patterns Implemented

The following three major application patterns have been implemented.

Business Process Support, BPS, provides support when building workflow applications, such as issue-management systems. This pattern is applicable when the items handled by the system flows between steps or phases, such as in a system implementing the review process of a scientific paper. Such systems are relatively easily built using the implementation of objects, aspects etc. that makes up this pattern. Worth to note is that BPS provides workflow functionality extending beyond application limits.

Document Management Support, DMS, supports management and generation of documents over the Internet, using templates and information from objects associated with the document. The template's "hot spots" are filled dynamically with information from business objects. One use of this pattern would be generation of reports on the history of an issue: dates of completion and names of people associated with different workflow steps would be filled in dynamically.

Relational Database Connector, RDC, provides a function by means of which, with the assistance of XML, external relation databases can be defined and imported. To import a database means that all the database objects are represented in Information Organizer but the data itself remains in the database. The RDC also provides support for building dialogues, which can present information from one or more data sources, and support for simple navigation between different lines in a database. All such navigation is performed with the help of URL's. In an imported database, all rows are represented as Business Object Framework objects which in turn means that they acquire all the properties which characterize a Business Object Framework object, such as strong security, the ability to keep all aspects of an object together. One feature worth to note is that security on row level can be obtained since Information Organizer represents each row in the database by an object, and the security properties can be set on each object independently.

The rest of section 3 discusses the Relational Database Connector in more detail.

3.2 Relational Database Connector

Many database applications have very little business logic and provide some kind of standard mechanism for accessing data directly (usually SQL). From the integration perspective, a viable solution is thus to provide such a generic front-end "connector" as we have done; it understands the target application's data (relational database concepts in this case) and provides components capable of encapsulating data from external databases for management, navigation, access and manipulation purposes. Since such a connector has no business logic whatsoever, it is unable to replace the original application entirely, but according to our work it can usually provide 60 to 80% percent of the original application functionality without any extension. The business logic of an application is however less often restricting "reads", and more often of the kind restricting how data can be modified or added. If an application contains much such logic, it is still possible to integrate database access but only permit reads. Such read-only integration can be of great benefit, if use cases including only reads are more common than use cases including writes.

Since the connector is generic, it is highly reusable because it can solve integration problems for many target applications with similar problems. An additional benefit is that the RDC components are fully integrated into the framework and can thus offer a much broader range of functions than that of the original application.

3.3 Description Files in XML

To define which parts of a database should be represented by objects in Active Directory, and how to present and interact with the database data, a number of XML description files are used. For each table in the database, three XML files have to be defined.

- The first one is mainly used to describe the table's columns and their data types. For each column it is possible to define whether it is editable or not and if a new data item has to be initiated or not. Related tables can also be described; for example, in a file describing a "decision", information is provided in form of keys to be able to find a way back to the correct issue. And in the "issue" object, file information is provided to be able to present the owner of, or all documents belonging to the issue.
- The second file defines how information can be presented in "summary cards", and describes available predefined queries. Whenever a row is selected in table, the data is presented according to the specification in the file. The XML file can of course be edited, and thus the summary card's appearance is modified. This approach provides an easy way to configure displays for different tables within a database, but also to present information originating from different database systems in a homogeneous manner. These summary cards are also the foundation to provide a powerful and common search mechanism for different information systems, integrated in Information Organizer.
- The third file provides means to map to the language of your choice.

The business logic using the description files are implemented as a number of Active Server Pages and COM objects.

4. DISCUSSION

The following describes some of the lessons learned from practical experience gained from the development of Information Organizer (Information Organizer) and the document and issue management system Arch Issue (Arch Issue).

4.1 Reuse

The overall and certainly the most important lesson learned is that reuse can be highly profitable. For organizations with limited resources undertaking relatively ambitious development projects, it is the only viable - and therefore practically mandatory - approach. With a very limited investment, Compfab (Compfab) was able to build a functionally comprehensive framework for its intended purpose, which in addition is secure, scalable, and reliable. This would not have been possible without total commitment to the reuse of not only platform components, but also architectural and design patterns, as well as "best practices" known for the platform.

We chose a set of standard products integrating e.g. Internet access and security. These not only provide a runtime and design-time environment but also a large number of components and knowledge of how to build user interface components. The word "build" was intentionally used to emphasize that a significant part of the development time was spent in learning the full capabilities and impacts of existing technologies and components on functionality and features targeted in the resulting framework. Development of custom functions for the framework actually occupied a smaller part of the total project time. Our impression is that this is one of the main reasons why verbal commitments to component-based development often fall short in practice.

4.2 Practical Experience

Information Organizer is currently used for developing an issue-management system (Arch Issue), and therefore our practical experience of using Information Organizer, and the concepts of BOM, is somewhat limited.

However, experience from the application of the framework to real world problems only reinforced most of the conclusions arrived at from experience from the development of the framework itself. In general, integrating modern, well-componentized applications is easy and straightforward, provided the application is designed to run on the same platform at which the framework is targeted (or provides "proxies" for accessing it when running on other platforms).

Integrating monolithic applications with poor or no defined application programming interfaces is difficult and cumbersome – sometimes to such a

degree that the original motivation for integrating such applications becomes highly questionable. For example, if there is an order management application which encapsulates orders, customers, responsible personnel etc into well defined components, and another invoice management application which is monolithic and provides access to its logical parts only through the proprietary user interface, there is no way to automate management of relations between logically related objects in these two applications, even at the user interface level. Unfortunately, many database-centred applications existing today are precisely of that kind. However, since many of these applications have very little business logic but provide a SQL interface for data access, the Relational Database Connector is a simple but very useful means to integrate database applications in Information Organizer.

5. SUMMARY

Reuse by integration of applications and information and reuse based on component-based development are two equally important ways to improve software development. Information Organizer emphasizes this and provides an object model, a framework and a number of components to encourage the building of integrated solutions. By taking the concept of "directory enabled applications" defined by Microsoft further by adding a number of important properties defined in standards such as IEC 1346-1 (defining the concept of aspects which relates all relevant information to an object), OMG (defining a powerful relation model) and IT4 (defining a way to build integrated industrial applications), we have achieved a strong and powerful environment based on a standard concept to build integrated systems. The total commitment to reuse not only platform components, but also architectural and design patterns and known "best practices" for the platform has been vital to the success of building not only the product itself but also components and applications based on it.

We have thus covered three aspects of reuse. First, with Information Organizer, implementing the concepts of BOM, it is possible to reuse whole applications, not originally intended for reuse. The level of integration can be chosen somewhat: either on user interface level or data level (using Relational Database Connector). Second, using the BOM concepts, we have implemented generic, i.e. reusable, application patterns. Third, we also described shortly how reuse of existing technologies made Information Organizer possible.

In the future, we will explore how different categories of users react to an integrated approach to different separate applications. How does the system respond to extremely large data quantities? How well does it support the maintenance of relationships when the original data sources changes? The "loose" coupling between objects and applications may prove to give rise to maintenance and consistency problems.

REFERENCES

Ambler, S., 1998. A Realistic Look at Object-Oriented Reuse. In *Software Development Magazine*, January 1998. URL: http://www.sdmagazine.com.

Arch Issue. URL: http://www.compfab.se.

Buschmann, F., 1996. *Pattern-Oriented Software Architecture, A System of Patterns*. John Wiley & Sons. Chichester

Compfab. URL: http://www.compfab.se.

DCOM, 1996. Technical Overview, MSDN Library. URL: http://msdn.microsoft.com.

DCOM, 1997. A Business Overview, MSDN Library. URL: http://msdn.microsoft.com.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. *Design Patterns - Elements of Reusable Object Oriented software*. Addison-Wesley. Upper Saddle River, NJ.

Howes, T., 1997. Ldap : Programming Directory-Enabled Applications With Lightweight Directory Access Protocol, MacMillan Technology Series.

IEC 1346-1, Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations, IEC 1346-1, First edition, 1996-03, International Electrotechnical Commission

Information Organizer. URL: http://www.compfab.se.

IT4, 1991. *Knowledge-Based Real-Time Control Systems, Phase II*, Studentlitteratur, Lund, Sweden.

King, R., 1999. *Mastering Active Directory*. Network Press.

Schwartz, R., 2000. *Windows 2000® Active Directory Survival Guide*, Wiley, England.

Krantz, L. 2000, ABB Industrial IT: The next way of thinking, http://www.abb.com .

Krueger C. W., 1992. Software reuse. In *ACM Computing Surveys*, volume 24, issue 2, pp. 131-183.

MSDN (Microsoft Developer Network). URL: http://msdn.microsoft.com.

OMG (Object Management Group). URL: http://www.omg.org.

Szyperski C., 1998. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley. Harlow.