BUSINESS OBJECTS:

FROM DEFINITION TO APPLICATI

Mohammad Abolhassani

TR 4022 S

Stellingen

behorende bij het proefschrift

BUSINESS OBJECTS:
FROM DEFINITION TO APPLICATION

Mohammad Abolhassani

1. Het onderbrengen van objecten in vijf hoofdcategorieën geïntroduceerd in dit proefschrift is een duidelijke en welgedefinieerde categorisatie van objecten in informatiesystemen.
2. Het definiëren van Business Objects op basis van de in de eerste stelling genoemde categorisatie levert een degelijk uitgangspunt voor een universeel aanvaardbare definitie van Business Objects.
3. Het toewijzen van elke (macro-)functionaliteit van (business) softwaresystemen aan een afzonderlijk(e) deel (laag) heeft voordelen voor het modelleren en realiseren van deze systemen. De lagen kunnen betrekking hebben op de User Interface, Presentation Logic, Business Logic, Information Logic en Data Source.
4. Standaardisatie is een hoofdkenmerk en voordeel van Business Objects vergeleken met andere (objectgeoriënteerde) methoden voor het modelleren van (business) systemen en het realiseren van (business) softwaresystemen.
5. Het ontdekken van de analogie tussen verschillende gebieden van de wetenschap en technologie is een interessante en behulpzame ervaring.
6. Wetenschap gaat over het kennen van de ”werkelijke wereld” en technologie gaat over het veranderen van de ”werkelijke wereld” in de ”ideale wereld”. Hoewel, er is maar één “werkelijke wereld” maar er zijn veel “ideale werelden”.
7. Op basis van de bestaande prestaties van wetenschap en technologie zou de wereld er veel beter uit kunnen zien.
8. Mensen zijn geen, en horen niet behandeld te worden als, ”business objects”.

Deze stellingen worden verdedigbaar geacht en zijn als zodanig goedgekeurd door de promotor: Prof.dr.ir. J. L. G. Dietz.

Propositions

attached to the thesis

BUSINESS OBJECTS: FROM DEFINITION TO APPLICATION

Mohammad Abolhassani

1. Dividing objects into the five main categories introduced in this dissertation is a clear and well-defined categorization of objects in information systems.
2. Defining Business Objects based on the in the first proposition mentioned categorization provides a solid basis for a globally acceptable definition of this concept.
3. Assigning each (macro) functionality of (business) software systems to a separate part (layer) leads to advantages in modelling and realizing these systems. The layers can be related to the User Interface, Presentation Logic, Business Logic, Information Logic and Data Source.
4. Standardization is a main characteristic and advantage of Business Objects compared to other (Object-Oriented) approaches in modelling (business) systems and realizing (business) software systems.
5. Discovering the analogy between different areas of science and technology is an interesting and helpful experience.
6. Science is about knowing the "real world" and technology is about changing the "real world" to the "ideal world". However, there is only one "real world" but there are many "ideal worlds".
7. On the basis of the existing achievements in science and technology, the world could be a much better place.
8. People are not, and should not be treated as, "business objects".

These propositions are considered defendable and as such have been approved by the supervisor: Prof.dr.ir. J. L. G. Dietz.

4 02I

BUSINESS OBJECTS:

FROM DEFINITION TO APPLICATION

788589

[illegible]

TR 4022

BUSINESS OBJECTS:

FROM DEFINITION TO APPLICATION

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 11 maart 2003 om 16:00 uur
door *Mohammad ABOLHASSANI*
informatica ingenieur
geboren te Teheran, Iran

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. J. L. G. Dietz

ISBN 90-9016609-2

Printed in The Netherlands by Sieca Repro BV, Delft.

To my dear parents
Ebrahim and Mahin-Banou

Contents

Chapter 1

INTRODUCTION

Basic research is what I am doing when I don't know what I am doing.

[Wernher von Braun]

1.1 THE RATIONALE

Around the mid-seventies, Object-Oriented technology began to be used for modelling systems and realizing software systems, and since the beginning of the nineties, it has found widespread acceptance. Subsequently, over the last years, the term "Business Objects (BOs)" frequently appeared in the literature of information systems, software engineering and the related fields. This term is used to represent different ideas and is associated with various topics. As such, basic knowledge about the core concept that is (or should be) represented by this term is advantageous (or even requisite). This knowledge should indicate and clarify the role of the concept of Business Objects in information systems and its value for software engineering. The goal of this research work is to provide and present such knowledge.

In order to provide such knowledge, one should study all different aspects related to the concept of Business Objects and discuss these aspects together with the relationship they have with one another. Only in this way the (supposed) added value of Business Objects for software engineering in realizing information systems can be comprehended and realized. Therefore, the main purpose of this research work has been: providing a comprehensive and integrated image of Business Objects.

As such, this research work aims to identify the core concept of Business Objects by elaborating on various of its aspects and providing a logical coherence between them.

However, to achieve the mentioned goal one has to deal with the aspects concerned at an abstract level. This is necessary in order to be able to clarify the connections between those aspects, while guaranteeing general validity of the discussed issues. Moreover, the resulting knowledge is useful for studying literature that deals with specific issues.

1.2 THE RESEARCH QUESTIONS

In order to provide a comprehensive and integrated view on Business Objects, we should discuss and deal with the aspects concerned in an organized manner with respect to the (supposed) added value of this concept for software engineering in realizing information systems. Therefore, we formulate the fundamental research question as follows:

How can the concept of Business Objects contribute to the improvement of the field of information systems engineering?

Accordingly, the research should be concentrated on the concept of Business Objects based on its relationship with development and maintenance of software systems. It should be investigated if by applying Business Objects we can improve system development and reduce system maintenance efforts.

The way of achieving the research goal can be further organized by formulating a number of key questions which can be derived from the main research question. The research sub-questions should, on the one hand, address the main issues discussed in the literature and, on the other hand, address those issues that are not discussed (adequately).

Therefore, in accordance with its rationale, this research work focuses on answering the following questions:

What are Business Objects?

The definition of the concept of Business Objects should serve as the starting point of the work, and hence should form the basis of any discussion about any aspect of this concept. Therefore, this definition should show the role of Business Objects in information systems adequately, unambiguously and expressively.

Where is the place of Business Objects?

Defined based on their role in information systems, the place of Business Objects with respect to the existing architectures should be determined. The basic types of architectures should be surveyed and the location of Business Objects with respect to them should be assessed.

How can Business Objects be created?

Obviously Business Objects are, in the first place, "objects". As such, for the creation of Business Objects we can rely on the existing methods in the area of modelling, design and implementation of "objects". However, the creation of Business Objects as a specific kind of "objects" may not expose the specific features of this concept adequately. Therefore, for the creation of Business Objects we should also take their specific traits into consideration. The main aspects of the creation of Business Objects should be recognized.

How can Business Objects be used?

The use of Business Objects with respect to their location and creation should be discussed. The main issues of using Business Objects should be recognized and the solutions should be elaborated on.

What are the resources and tools for Business Objects?

The relationship of the concept of Business Objects, as established in this research, with the existing resources and tools should be assessed. This means that we should study a number of representative resources and tools.

How can Business Objects be applied?

The application of Business Objects should be demonstrated through representative examples. The examples should show how Business Objects can be applied in practice, and how they can be taken advantage of for specific and general goals.

1.3 THE METHOD

Before trying to find the answers to the above questions, we give the following hypotheses:

- A globally acceptable definition of Business Objects can be formulated.
- Business Objects can be advantageous for realizing information systems.
- The concept of Business Objects can form the central point for a particular discipline in software engineering. This discipline can be formed around a group of architectural and constructional concepts and aspects concerned with information systems.

In order to accomplish the research goal, formulated by the research questions, and to test the hypotheses, we should apply the appropriate method(s). There are several methods for this purpose, such as literature study, case study and field experiment. [Yin 1994] calls these methods "research strategies", where each strategy represents a different way of collecting and analysing (empirical) evidence, follows its own logic and possesses peculiar advantages and disadvantages. The type of the research question is one of the main factors that distinguish the research strategies.

Due to the nature of this research topic, expressed by the research main question and sub-questions, both literature study and case study are the appropriate methods.

1.4 THE OUTLINE

The structure of this thesis is based on the research questions mentioned above. Since the questions are formulated according to the research goal, in order to provide a comprehensive and integrated view on Business Objects, their answers represent the research work appropriately. Furthermore, as the questions are ordered according to the inherent logic of the research subject, the order of the chapters follows the order of the questions.

Accordingly, the outline of this thesis is as follows:

- Chapter 2 aims to answer the question "What are Business Objects?" by discussing the existing definitions of Business Objects and their shortcomings. It supplies a new definition and outlines the main advantages of Business Objects.
- Chapter 3 aims to answer the question "Where is the place of Business Objects?" by discussing the existing architectures, the position of Business Objects and their relationship to the concept of components.
- Chapter 4 aims to answer the question "How can Business Objects be created?" by discussing the main aspects of the creation of Business Objects that includes devising and building Business Objects, as well as standardization and categorization of Business Objects.
- Chapter 5 aims to answer the question "How can Business Objects be used?" by discussing the main issues of using Business Objects based on an architectural viewpoint.
- Chapter 6 aims to answer the question "What are the resources and tools for Business Objects?" by assessing the architectural and constructional issues of representative enabling technologies and system environments for Business Objects.
- Chapters 7, 8 and 9 aim to answer the question "How can Business Objects be applied?".
  - Chapter 7 presents a case study carried out within the framework of the realization of an information system in a company.
  - Chapter 8 discusses the use and advantages of Business Objects for the health-care domain and the banking domain.
  - Chapter 9 discusses how Business Objects can be used for simulation and system integration.
- Chapter 10 draws conclusions on the basis of the issues discussed in the preceding chapters.

Chapter 2

DEFINING BUSINESS OBJECTS

Every now and then the computer industry gets swept up in a wave of enthusiasm for some new Silver Bullet that's apparently going to solve everyone's problems overnight. Actually, these days the wild surges of millennial euphoria seem to come at annual intervals. Usually the technology in question actually is a step forward, able to solve real problems better or faster than was possible before. However, as word spreads about the power of the new technique some people will inevitably try to apply it to the wrong problems. It's a bit like the enthusiasm for microwave ovens when they first became cheap enough for anyone to buy; one could buy microwave cookbooks explaining how to use them to cook everything from a complete Christmas dinner to a soufflé. Fortunately, after a while sanity returned, and people now use microwaves for what they're best at, and go back to making toast in the toaster or roasting the turkey in the oven, just as they always did, because they're the best tools for the job.

[CORBA and XML; conflict or cooperation?, Andrew Watson, Object Management Group]

In the last years, the term "Business Objects" frequently turned up in the literature of information systems, software engineering and the related fields. In general, this term represents applying Object-Oriented technology in modelling business systems and realizing business software systems (in distributed environments). However, a brief look at the literature shows that there are many different definitions for this term, with many differences and even contradictions, and a clear and widely accepted definition of Business Objects is missing. This shortcoming forms a barrier to research about, as well as to the use of Business Objects. This chapter addresses this shortcoming and aims to define Business Objects in an unambiguous and descriptive way.

The chapter begins with a discussion about the concept of "object", using a few basic definitions. This is followed by a categorization of objects based on their roles in information systems. Subsequently, the concept of "Business Objects" is discussed. First a number of representative statements about the definition of this concept and its relationship with other object categories are surveyed, and their shortcomings are addressed. Then the concept of Business Objects is defined, and its main advantages are pointed out[1].

2.1 OBJECTS

The term "object" represents a computational[2] entity which is a combination of data (attributes) and procedures (operations) that may have associations (relationships) with other ones, and can represent an entity of a universe of discourse. The Object Management Group (OMG)[3] [OMG] defines "object" as follows [OMG 1997-1]:

> *An object is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.*
>
> *[An object is] a combination of a state and a set of methods that explicitly embodies an abstraction characterized by the behaviour of relevant requests.*
>
> *An object models a real-world entity and is implemented as a computational entity that encapsulates state and operations (inter-*

[1]The background of this chapter can be found in [Abolhassani 1999].

[2]Here "compute" denotes its generic meaning in the area of computer science and technology.

[3]The Object Management Group (OMG), formed in 1989, is an institute that aims to develop standards for a platform-independent technological infrastructure for the Object-Oriented programming paradigm.

> *nally implemented as data and methods) and responds to requests for services.*
>
> *A basic characteristic of an object is its distinct object identity, which is immutable, persists for as long as the object exists and is independent of the object's properties or behaviour.*

In the area of database systems the definitions of an object are basically the same, although it is more transparent to reflect the structure of the entities of the universe of discourse. The Object Data Management Group (ODMG)[4] [ODMG] states [Cattel and Barry 1997]:

> *The state of an object is defined by the values it carries for a set of properties. These properties can be attributes of the object itself or relationships between the object and one or more other objects. Typically the values of an object's properties can change over time.*
>
> *The behaviour of an object is defined by the set of operations that can be executed on or by the object. Operations may have a list of input and output parameters, each with a specified type. Each operation may also return a typed result.*

As the above-mentioned definitions describe different aspects of the concept of "object" and do not contradict each other, with respect to our discussion they are all acceptable.

Furthermore, there are widely recognized advantages associated with the utilization of objects[5]. The three main characteristics of objects, which are commonly considered as the major advantages of working with objects, are:

- Encapsulation: To hide the (implementation) details of objects.
- Inheritance: To found objects on more general objects.
- Polymorphism: To respond to the same requests in different ways by (different) objects.

[4]The Object Data Management Group (ODMG), formed in 1991, is an institute that aims to develop standards for an object data model, as well as object database languages.

[5]A discussion of these advantages is beyond the scope of this work.

2.2 OBJECT CATEGORIES

Objects can be considered as building blocks for constructing information systems[6]. By determining the major roles that objects can/should play in information systems and by dividing objects into different categories based on those roles we can recognize and define different types of objects, including Business Objects.

In general, one can apply objects in information systems in order to:

- Reflect semantics of real-world concepts.
- Interact with end-users.
- Provide complementary facilities and required services.
- Make connection with other constituents of the system.
- Provide programming constructs.

Accordingly, we divide objects into five main categories, defined as follows:

Reflection Objects (ROs) are those objects that reflect semantics of real-world concepts. They are related to the most evident exploitation of the concept of "object" and are often meant thereby, by default. Besides, they are the only objects that come into play during the analysis process. For example "person", "purchase" and "reception" belong to this category.

Interaction Objects (IOs) are those objects that deal with interaction with end-users. They are concerned with presenting data to, and obtaining data from end-users. For example "button", "box" and "icon", as well as "table", "form" and "report" belong to this category.

Completion Objects (COs) are those objects that provide complementary facilities and required services. For example objects

[6]An information system is a system which manipulates data and normally serves to collect, store, process and exchange or distribute data to users within or between enterprises or to people within wider society [Sim 1996].

that supply facilities and services offered by operating systems, database management systems and object request brokers belong to this category.

CoNnection Objects (NOs) are those objects that connect constituents of the system, including objects, to each other. For example the objects that take care of conversions, transformations and communications belong to this category.

ConStruction Objects (SOs) are those objects that provide programming constructs. They are offered by programming languages and environments. For example "real", "integer" and "character" belong to this category.

These categories make a clear distinction between the different tasks that objects can carry out in information systems. Therefore, they can help us to prevent confusions and contradictions in defining specific object types and assigning objects to specific groups[7].

2.3 BUSINESS OBJECTS

Although Object-Oriented technology has been used for various disciplines for a long time, it was not applied for "business" until recently [Casanave 1995] and [Marshall 1997]. In fact, the term "business" in itself has a wide and generic meaning, and many have the habit of referring to almost every activity in the world using this term. For instance [Cummins 1996] notes:

> *What do we mean by "business"? Generally the intent is to comprehend any kind of enterprise including academic, governmental or charitable. The name has marketing value and should not be narrowly interpreted.*

Anyhow, the use of Object-Oriented technology for realizing "business systems" has been growing very fast. By virtue of this growth, the use of the term "Business Objects" has also grown rapidly. It seems that there is a

[7]In practice, however, an object may have characteristics of more than one category. Familiar examples of this are objects that deal with both real-world concepts and user interface issues.

tendency to refer to any "object" with the term "Business Object". Consequently, there is much confusion about this concept. Besides, not everybody is familiar with the term. [Hung et al. 1998] presents an interesting survey in this respect.

2.3.1 LITERATURE

Of the different definitions of Business Objects, the definitions presented by OMG are more (generally) accepted than the others. The OMG Business Object Management Special Interest Group (BOMSIG)[8] defines Business Objects as follows [OMG 1995]:

> *A Business Object is a representation of a thing active in the business domain, including at least its business name and definition, attributes, behaviour, relationships and constraints. A Business Object may represent, for example, a person, place or concept. The representation may be in a natural language, a modelling language or a programming language.*

Later on, the OMG Business Objects Domain Task Force (BODTF)[9], pointed to the following categories in addition to Business Objects [OMG 1997-2]:

> *Application Objects (computer simulated representations of real-world objects) are presented to end-users as objects that can be manipulated in a way that is similar to the manipulation of the real-world objects.*
>
> *The Application Objects part of the architecture represents those application objects performing specific tasks for users. One application is typically built from a large number of basic object classes, partly specific for the application, partly from the set of Common*

[8]The Business Object Management Special Interest Group (BOMSIG) of OMG was in charge of dealing with issues concerning Business Objects.

[9]The Business Objects Domain Task Force (BODTF) of OMG replaced BOMSIG in January 1996. The role of BODTF is to deal with a wide range of architecture and business engineering issues, including development and deployment of Business Objects, as well as the cooperation and interoperation among Business Objects.

> *Facilities. New classes of application objects can be built by modification of existing classes (inheritance) as provided by Object Services. The multi-object class approach to application development leads to combine and configure their applications.*
>
> *Common functionality in different applications (such as storage and retrieval of objects, mailing of objects, printing of objects, creation and deletion of objects, or help and computer-based training) is realized by common shared objects leading to a uniform and consistent user interface.*
>
> *The Common Facilities component provides a set of generic application functions that can be configured to the requirements of a specific configuration. Examples are printing facilities, database facilities and electronic mail facilities.*

However, these categories do not help in finding the position of Business Objects in information systems. Besides, the Application Objects can embrace different categories, like those mentioned in the following categorization [Shelton 1995]:

> *Business Objects are objects that represent a person, place, thing or concept in the business domain. They package business procedures, policy and controls around business data. Business Objects serve as a storage place for business policy and data, holding together in a coherent unit the right business policy with the right data and ensuring that data is only used in a manner semantically consistent with the business intent...*
>
> *Technology objects represent a programming or technology concept, and thus are the building blocks of applications and implemented Business Objects. They are components of the information systems and applications environment. Examples of technology objects include GUI components like windows and push buttons, programming constructs like integer and string classes and application frameworks...*
>
> *Application objects are programs which present information and manage interaction with human users, process information and produce reports. They are solutions to specific business problems,*

and take the form of control panels, which operate [on] a specific set of Business Objects to perform a specific task. Examples include: Order Entry, Quarterly Report, Reservation and Ticketing ...

This categorization considers Business Objects as an object category in itself, and is (almost) in agreement with the following statements [Casanave 1995]:

Business Objects can be distinguished from programming objects such as arrays and I/O channels or from user interface objects such as buttons and windows. Business Objects can also be distinguished from system objects such as your word-processing program...

The difference between the above-mentioned technology and application object categories is not very clear. Furthermore, to be of any use, the term Business Objects should refer to a more specific concept, and therefore Business Objects should not be considered as a category of objects at the general level.

Later, BODTF defined Business Objects as follows [OMG 1999]:

A Business Object [is] an object which represents a corresponding entity in the real world of the business. A Business Object has an identity that corresponds to the identity of the real-world entity. Business Objects are (usually) persistent and recoverable. Operations on Business Objects are in a transactional context that assures concurrency control and supports commit and rollback operations.

Except being identifiable, transactional and persistent, Business Objects can also have state, attributes, operations, relationships and events, according to BODTF.

The above-mentioned definitions emphasize the structural aspects of Business Objects and do not provide a clear distinction between Business Objects and other objects. The following statements refer to the role of Business Objects as representing elements of enterprise, but emphasize their role as application building blocks [Cummins 1996]:

> *A Business Object is an object used to represent elements of the enterprise as described here. This is not a definition that always provides a clear distinction.*
>
> *What is important technically, is that a Business Object is an abstraction that hides the technical details of implementing the computer representation of a problem and allows the application developer to focus on the business problem to be solved.*

Finally, in the literature much confusion also arises from the term "Business Objects" itself. Some mix up the term "Business Objects" with other terms, such as "Core Business Objects" and "Common Business Objects", others use the terms "Business Objects", "Process Business Objects" and "Entity Business Objects" interchangeably, or use different names for the same concept, such as "Task-Process Objects" and "Common Business Processes" for "Process Business Objects".

As Business Objects are supposed to provide a suitable means for communication with (the expert) people in different (business) domains by using familiar terms [Casanave 1995], the term Business Objects itself should not be a source of such confusion.

All in all, we can conclude that a clear, strict and generally accepted definition of Business Objects is missing. Therefore, in order to supply a basis for our discussion about different aspects of this concept, we should invent a new definition.

2.3.2 DEFINITION

The categorization of objects and the definition of Reflection Objects, given in {2.2}, can offer a good basis for an appropriate definition of Business Objects.

As mentioned before, Reflection Objects are those objects that reflect the semantics of real-world concepts. They neither deal with user interface issues, nor provide any general-purpose functionality. They can be built based on (other) Reflection Objects and Construction Objects, can use services provided by Completion Objects and can participate in relationships with (other) Reflection Objects and Interaction Objects.

Consequently, Business Objects can be considered as those Reflection Objects that should serve as a means for modelling (business) systems and for realizing (business) software systems based on specified rules and conventions. Accordingly, Business Objects should give (a necessary and sufficient description of) the functionalities, properties, associations and features of business facets, in a clear and unambiguous manner.

Furthermore, Business Objects can be related to a particular (business) domain, cross (business) domain boundaries and be concerned with different interoperating (business) domains or be involved in all (business) domains.

In order to be able to provide any added value for modelling (business) systems and realizing (business) software systems, Business Objects should represent standards and guarantee openness of systems and interoperability among systems.

Business Objects can be devised using any medium, such as human language, modelling language and programming language, and can be built using any programming code, language and platform.

Finally, Business Objects can be defined as follows:

Business Objects (BOs) are those Reflection Objects that represent standard real-world (business) concepts, are related to different (business) domains and give (a necessary and sufficient description of) the functionalities, properties, associations and features of those concepts, in a clear and unambiguous manner, for modelling (business) systems and realizing (business) software systems.

2.3.3 MAIN ADVANTAGES

In general, Business Objects can offer a comprehensive and disciplined means for modelling (business) systems and realizing (business) software systems:

- Comprehensive, because they (should) cover all of the (main) real-world (business) concepts of the domain concerned.
- Disciplined, because they are based on the concept of "object" and hence can result in organized (software) elements.

With respect to the modelling of (business) systems, Business Objects can (potentially) offer an efficient and effective means for:

- Concentrating on the essential real-world (business) concepts, and leaving out of consideration any other aspect.
- Communication between people, including software and (business) domain experts, through the use of the same terminology and business jargon.

With respect to the realizing of (business) software systems, Business Objects can (potentially) offer an efficient and effective basis for:

- Using different user interfaces.
- Using different data sources.
- Interoperation between different systems and platforms in distributed and heterogeneous environments.
- Utilization of off-the-shelf components.

Other (potential) advantages of using Business Objects are that they provide reusability, flexibility, extendibility and scalability, and prevent redundancy, improve maintainability and offer a global view on the whole system.

Accordingly, as standardized Relation Objects, Business Objects do not only provide the advantages of Object-Oriented technology, but they can also enhance and extend these advantages for modelling (business) systems and realizing (business) software systems.

Chapter 3

LOCATING BUSINESS OBJECTS

Form ever follows function.

[Louis Henri Sullivan]

The previous chapter defined Business Objects based on the role they (should) play in information systems. To make this role more specific, we should determine the place of Business Objects in the existing architectures. Therefore, we should survey the basic types of architectures and should assess the location of Business Objects with respect to them.

Broadly speaking, there are two basic types of architectures, Monolithic and Client/Server. The first section describes these architectures. The Three-Layer Architecture can be considered as a particular type of the Client/Server architecture. The second section describes this architecture and its main characteristics. The Five-Layer Architecture extends the Three-Layer Architecture in order to deal with its shortcomings. The third section explains this architecture and its main features. The Multi-Tier Architecture is described in the fourth section. Subsequently, the fifth section indicates the position of Business Objects with respect to the Five-Layer Architecture, along with other objects that can be applied for each layer. Finally, the sixth section first describes the concept of "component" and then discusses the relationship between the concept of Business Objects and this concept, and consequently defines the concept of Business Object Components.

3.1 BASIC ARCHITECTURES

3.1.1 MONOLITHIC

In this architecture, the whole system consists of a single, autonomous and independent body, possibly divided into different interrelated parts. There is no distinguishable part of the system that can present or represent a specific (macro) functionality with a clear interface to the (other parts of the) system, so that it can be (re-)used in more than one system.

Although this architecture can lead to efficient systems, any modification to any part/functionality of the system may require and result in intrusive changes to the whole system.

3.1.2 CLIENT/SERVER

In this architecture, the whole system is divided into two main parts. Each part is concerned with one or more specific (macro) functionality. There is a clear interface between the two parts, and one of the parts, Client, uses the functionality of the other, Server. In this way, the functionality of the Server can be used for more than one Client.

This architecture can lead to efficient systems, and modifications to one of the two parts of the system do not affect the other part.

3.2 THREE-LAYER ARCHITECTURE

In general, business software systems can be considered to consist of three main parts that each has a (macro) functionality:

- User Interface
- Business Logic
- Data Source

Where the User Interface (UI) takes care of interaction with user, the Data Source (DS) stores and retrieves data and Business Logic is defined as follows:

Business Logic (BL) is the collection of real-world business concepts and their manipulation according to business rules and constraints.

In fact, Business Logic can be observed as that part of a Client/Server architecture that completes the data model in such a manner that a comprehensive and realistic representation of the real world is established. Accordingly, systems based on the Client/Server architecture can belong to one of the following categories:

- Fat Client/Server: Client consists of User Interface and Business Logic, and Server consists of Data Source.
- Semi-Fat Client/Semi-Fat Server: Client consists of User Interface and (part of) Business Logic, and Server consists of (part of) Business Logic and Data Source.
- Client/Fat Server: Client consists of User Interface, and Server consists of Business Logic and Data Source.

While the Fat Client/Server category limits the code reuse capacity for maintaining applications with different user interfaces, the Client/Fat Server category limits the usage of logic constructs and complex data types, like arrays and pointers, and reduces the Data Source performance by checking rules.

Placing each of the three above-mentioned parts having a different (macro) functionality at a separate location (*layer*) leads to the Three-Layer Architecture (TLA) [Orfali and Harkey 1997], shown in figure 3.1.

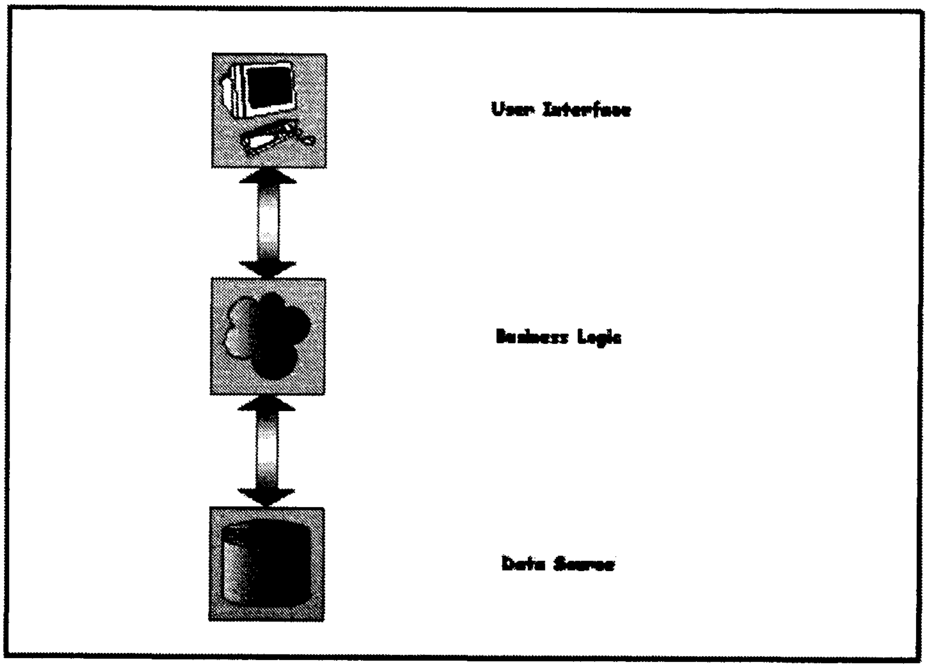

Figure 3.1: Three-Layer Architecture.

This architecture is a particular type of the Client/Server architecture, where the User Interface is a client for the Business Logic, and the Business Logic is, in turn, a client for the Data Source. Accordingly, those who look at each separate layer as a *tier* refer to this architecture as the Three-Tier Client/Server architecture[Edwards 1997].

Furthermore, in order to provide the whole functionality required for maintaining complete systems, one needs additional facilities and services. For Object-Oriented systems these facilities and services include management of life cycle, transaction, state and session, as well as security, authorization and load balancing. These can be taken care of by Control Logic, defined as follows:

Control Logic is a collection of facilities and services, including management of life cycle, transaction, state and session, as well as security, authorization and load balancing. It offers the complementary functionality of the system.

Consequently, System Logic can be defined as follows:

System Logic is the composition of Business Logic and Control Logic.

3.3 FIVE-LAYER ARCHITECTURE

It is not easy to realize user-friendly user interfaces when the Business Logic layer is located remotely and has no knowledge about user interfaces of the User Interface layer. Nor is it easy to maintain Business Logic when the Data Source layer consists of different data sources.

In fact, there are certain gaps between the layers of the Three-Layer Architecture. After all, this fact stems from the very nature of this architecture, where a clear distinction between the three layers is assumed.

If the definition of the Three-Layer Architecture in general and that of Business Logic in particular is applied rigorously, some (basic) functionalities cannot be carried out[1], including:

- Reading/writing data from/to different data sources.
- Checking data sent to/got from Business Logic.
- Extracting and acquiring the desired (subset) of data by combining and splitting data, etc.
- Converting and transforming data.

However, as the borders between the layers are not always considered to be as strict as they should be, and, in practice, some of the tasks concerned are assigned to Control Logic, the existence of gaps between the layers has not attracted much attention. Nevertheless, attempts to deal with this problem and to fill these gaps have resulted in architectures that are (mainly) based on the Three-Layer Architecture.

[1]Certain general functionalities, such as maintaining communications and caching data, can be taken care of by the environments embracing each layer.

For instance, [Schmid et al. 1998] suggests that the field-related consistency checks are a responsibility of the "presentation layer" (User Interface layer), because the user should rapidly get feedback when he has entered a wrong input field, and [Fowler 1997] suggests an "application logic" tier (layer) in addition to the presentation tier (User Interface layer) and domain tier (Business Logic layer). It describes the application logic tier as:

> *A selection and simplification of the domain model for an application. Contains no user interface code but provides a set of facades of the domain tier for the user interface.*

Furthermore, another motive for thinking up a new architecture can be the desire to improve the development and performance of systems: one may wish to facilitate the design of the User Interface layer, or provide a fast response to user input.

To comply with the principles of the Three-Layer-Architecture, provide the missing functionalities, and improve the development and performance while maintaining robustness, one can use the Five-Layer Architecture (FLA), shown in figure 3.2. This architecture consists of the following layers:

- User Interface
- Presentation Logic
- Business Logic
- Information Logic
- Data Source

Where Presentation Logic and Information Logic are defined as follows:

Presentation Logic (PL) is the collection of functionalities for interoperation with Business Logic, including reading, writing, extracting and checking data and converting data to the desired form.

Information Logic (FL) is the collection of functionalities for interoperation with different data sources, including reading, writing,

extracting and checking data and converting data to the desired form.

Note that, although the checking of data can be considered as applying business rules and constraints, there is no contradiction between the definition of Business Logic and the definitions of Presentation Logic and Information Logic. Presenting and maintaining business concepts, rules and constraints is the only role of Business Logic, while Presentation Logic and Information Logic can apply (a subset of) business rules and constraints as a (small) portion of their task.

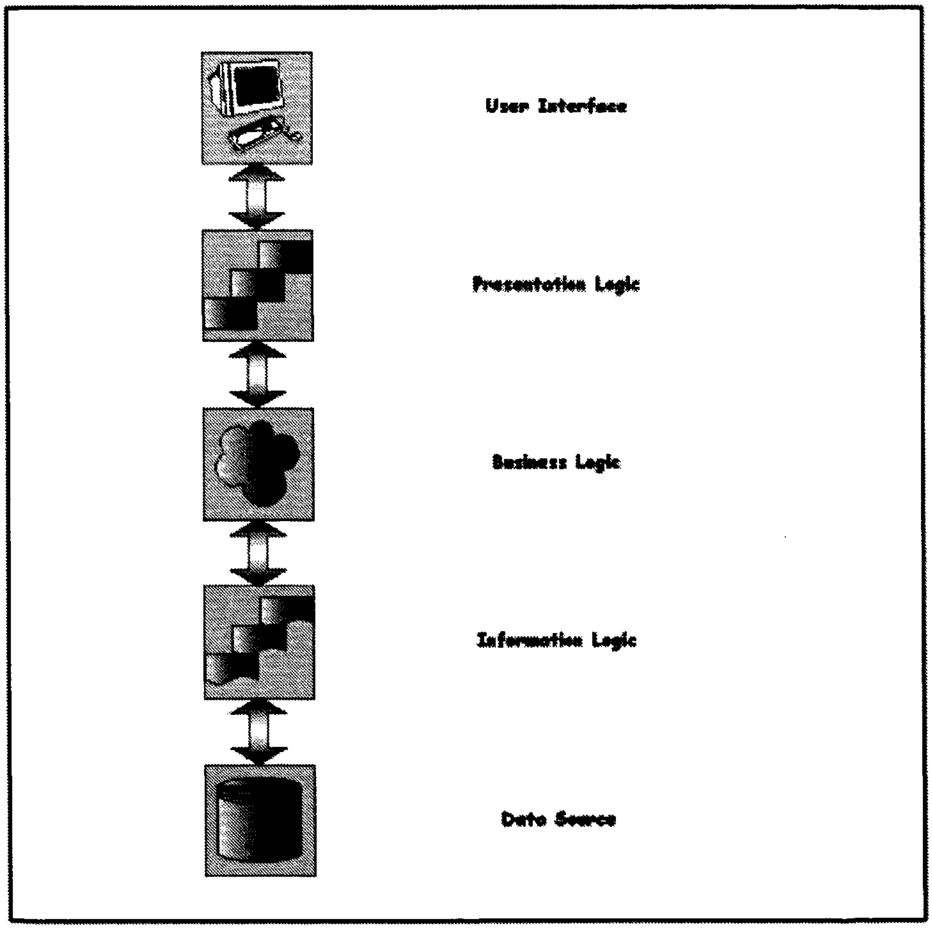

Figure 3.2: Five-Layer Architecture.

Furthermore, Core Logic can be defined as follows:

Core Logic (CL) consists of Presentation Logic, Business Logic and Information Logic.

Confusing Business Logic with Core Logic is a common mistake. Moreover, in this respect, the following points are noteworthy:

- Presentation Logic can be used for a particular group of applications based on the Five-Layer Architecture or similar architectures.
- Business Logic can be of general use for systems based on the Five-Layer Architecture or similar architectures.
- Information Logic can be of general use for systems based on the Five-Layer Architecture or similar architectures, as well as for other architectures.

Consequently, System Logic can be (re-)defined as follows:

System Logic is the composition of Core Logic and Control Logic.

The common general architecture of computer systems can be considered as an analogy for the Five-Layer Architecture. From this point of view, the major parts of a computer system and their corresponding parts in the Five-Layer Architecture are as follows:

- Applications <=> User Interface
- Operating System Shell <=> Presentation Logic
- Operating System Kernel <=> Business Logic
- Basic Input Output System (BIOS) <=> Information Logic
- Devices <=> Data Source

3.4 MULTI-TIER ARCHITECTURE

The layers of a Five-Layer Architecture, as well as those of a Three-Layer Architecture and its different versions, can exist at separate locations. Moreover, each layer in itself can consist of different parts divided over different locations. In this case, the architecture concerned can also be referred to as the Multi-Tier Architecture (MTA).

However, the term Multi-Tier Architecture can also be used for referring to different kinds of architectures that include at least three layers in general.

3.5 POSITION OF BUSINESS OBJECTS

Business Objects can indeed be used for systems based on Monolithic and Client/Server architectures. They can form the basis of the Business Logic (macro) functionality, albeit interwoven with other functionalities.

By separating the whole functionality of a system into loosely coupled layers and tiers, architectures based on multiple layers and tiers, that is, Multi-Tier Architectures, can offer a more suitable basis for distributed and heterogeneous environments, and can make system development and deployment more efficient and effective. Accordingly, they can provide a more appropriate basis for deploying the concept of Business Objects.

However, for the sake of completeness and clearness, the Five-Layer Architecture is the preferred basis for describing the location of Business Objects.

In the Five-Layer Architecture, the Business Logic layer can be based on an information model. In order to control the rules and constraints that govern the manipulation of data related to the business in question, this model should be able to reflect the reality of the business in question. Besides, interaction with the user, as well as storage and retrieval of data should be left out of the consideration of the Business Logic layer. Since we defined Reflection Objects and accordingly Business Objects in such a way that they only reflect real-world concepts, and leave any other (macro-)functionality to other system constituents, Business Objects can be the proper means for realizing the Business Logic layer.

Furthermore, in order to provide a system based on the Five-Layer Architecture, we can use Object-Oriented technology to provide objects, in addition to Business Objects, for other parts of the system. These objects include Presentation Objects and Information Objects defined as follows:

Presentation Objects (POs) are those Connection Objects that take care of interoperation between Interaction Objects and Business

Objects by providing the required functionality, including reading, writing, extracting and checking data and converting data to the desired form.

InFormation Objects (FOs) are those Connection Objects that take care of interoperation between Business Objects and data sources by providing the required functionality, including reading, writing, extracting and checking data and converting data to the desired form.

In addition, in case we use an Object-Oriented database as Data Source, we can call the corresponding objects - which can belong to the Reflection Objects or Connection Objects categories - Data Objects (DOs).

Devising these conventions, that is, defining other object categories than Business Objects for the other parts of system, can provide us with better means, in terms of clearness and robustness, to discuss different parts and aspects of system and their relationship with Business Objects.

In this way, a system based on the Five-Layer Architecture can consist of the following objects:

- Interaction Objects for User Interface
- Presentation Objects for Presentation Logic
- Business Objects for Business Logic
- Information Objects for Information Logic
- Data Objects for Data Source

where Completion Objects maintain Control Logic.

Figure 3.3 shows a simplified example of applying these conventions for the concept "Person".

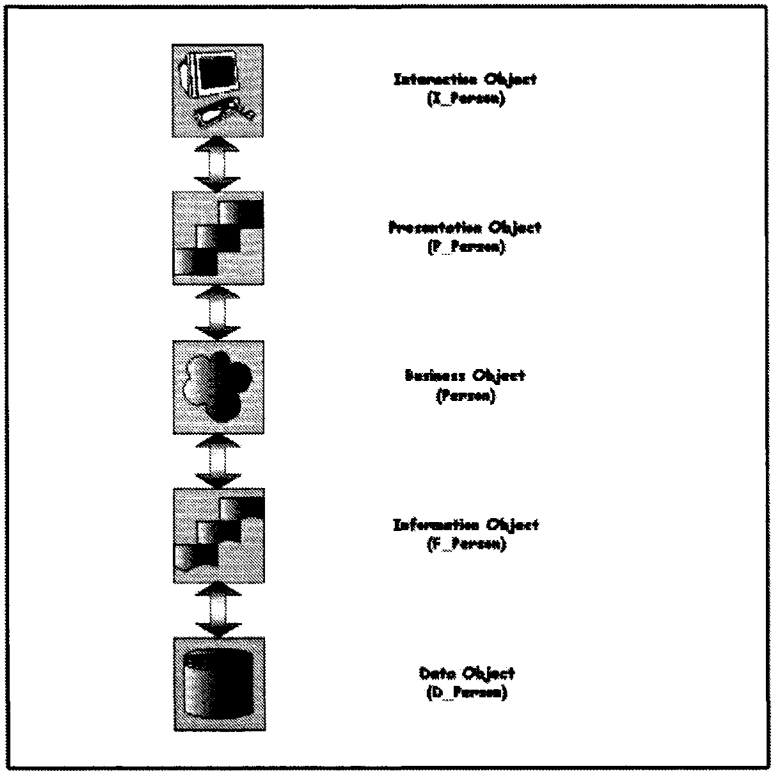

Figure 3.3: A simplified example of applying the system object conventions to the concept "Person".

Obviously, objects, including Business Objects, can be distributed over different locations in order to be able to take advantage of different software and hardware resources for storage, processing, communications, etc., and to deal with issues such as performance, authorization and security.

3.6 BUSINESS OBJECTS AND COMPONENTS

3.6.1 COMPONENTS

There is a lot of discussion about, and there are many definitions of, the concept of Components, such as the definitions presented by [Orfali et al. 1996], [Orfali et al. 1999], [Heineman and Councill 2001] and [Stacey 2001]. An interesting description of components that focuses on their role in software systems is the one of [Orfali et al. 1999], which draws an analogy with the role of Integrated Circuits (ICs) in hardware systems:

> *A component is what Brad Cox calls a "software IC". "Application frameworks" are boards, or containers, into which we plug these components. The object bus provides the back plane. Families of software ICs that play together are called "suites". You should be able to purchase your software ICs, or components, through standard part catalogues.*

Having given an idea what components are, we can now give a definition of components that is appropriate for our discussion about their relationship to Business Objects:

A Component is a reusable and replaceable software element with a known and clear functionality and interface. The interface must specify the outputs and services provided by the component, as well as the inputs and services required by the component.

Furthermore, a component can be:

- Atomic or composite, that is, consist of other components as sub-components.
- Fine-grained or coarse-grained.
- Opaque or transparent.
- Dedicated to specific environments or customisable through configuration specifications that deal with their development environments, as well as their execution environments[2].

Applying Object-Oriented technology and providing components as objects is common practice. In fact, the concept of objects in the (distributed) Object-Oriented technology is closely related to the concept of Components.

[2]The environment within which the components run is called "container". Container can be a (sub) system or an application program that takes care of different management issues involved.

Accordingly, there is a substantial and ongoing discussion about the relationship between components and objects [Szyperski 1998].

In general, the use of components can promote standardization. Components can be used as building blocks for realizing software systems and enable the notion of software plug and play. Their characteristics make them suitable for distributed environments and evolving systems. In addition, components can ease interoperability between different systems and offer the possibility of integration of legacy systems.

In particular, components are suitable for realizing systems based on the Five-Layer Architecture. Each layer can then be considered as a (coarse-grained) component or system part, which consists of (fine-grained) components.

3.6.2 BUSINESS OBJECT COMPONENTS

As an essential part of the Five-Layer Architecture, Business Logic can also be built using components. Correspondingly, Business Objects can also be embodied in components.

By getting the characteristics of the concepts of both Business Objects and components, we can define the concept of Business Object Components as follows:

Business Object Components (BOCs) are reusable and replaceable software elements with a known and clear functionality and interface. Their interface specifies the outputs and services provided by them, as well as the inputs and services required by them. They represent standard real-world (business) concepts, related to different (business) domains, and give the functionalities and features of those concepts for realizing (business) software systems.

The use of Business Object Components can, among others, have the following advantages for the realization of software systems:

- Reusability as the existing elements can be deployed for new applications.
- Flexibility during the design, development and deployment, as we deal with well-defined elements.

- Scalability as well-defined elements can be added or removed with no or little effect on the other parts.
- Openness through a well-defined functionality and an interface that is based on standards.

However, the choice between Business Objects and Business Object Components depends on the resources, possibilities and requirements. In general, Business Objects are more flexible and suitable for use in limited environments, while Business Object Components are more efficient and suitable for use in broad environments.

Chapter 4

CREATING BUSINESS OBJECTS

Before the first useful IC was built, engineers had had decades to discover the useful patterns that crop up in system after electronic system. In software, by analogy, we must make sure that the classes that we develop are based on sound, robust, handy abstractions. Classes such as "Customer" and the loveable, furry old "Stack" are likely to receive standing ovations; classes such as "Egabragrettu" are more likely to be dumped at the edge of town.

[Fundamentals of Object-Oriented Design in UML, M. Page-Jones, Addison-Wesley]

Business Objects should be created in accordance with their intended characteristics and advantages, as well as their location with respect to information systems; only then they can help us in modelling (business) systems and realizing (business) software systems efficiently and effectively. The first three sections discuss the basic aspects of creating Business Objects.

The first section mentions the main issues that should be taken into consideration when we create Business Objects. The second section discusses how we can devise Business Objects. It presents an algorithm that can be applied for this purpose. The third section discusses how we can build Business Objects. It elaborates on building the (devised) Business Objects as objects and components, and introduces the concepts of Business Object Blocks, Business Core Components and Business Core Blocks.

Furthermore, to benefit fully from Business Objects in order to model (business) systems and realize (business) software systems, one should take some additional aspects into consideration as well. These aspects have to do with standardization and categorization of Business Objects.

Standardization is one of the main characteristics of Business Objects. In fact, it is one of the main distinguishing features of Business Objects, and one of the advantages of this concept over the other (Object-Oriented) approaches for realizing software systems. The forth section discusses standardization of Business Objects.

Categorization is a beneficial practice, which is applied in many fields of science and technology. It can be applied to the concept of Business Objects as well. The fifth section discusses categorization and its relevance to Business Objects. In this way, the benefits of categorization for the realization of software systems using Business Objects are mentioned, the basic categories of Business Objects are defined and some example criteria for further categorization of Business Objects are suggested.

4.1 MAIN ISSUES

Obviously, the differences and contradictions between the definitions of Business Objects have consequences for their creation. As a result, in the literature, taking care of interaction with the user, as well as storage and retrieval of data are also considered as functions that should be accomplished by Business Objects. For instance, [Bonar 1997] assigns tasks such as initialisation of underlying persistent store, backup, restore, update and recovery to Business Objects.

Furthermore, although, in principle, Business Objects should be created in order to reflect real-world concepts, this does not mean that all real-world concepts should be (directly) embodied in Business Objects. In fact, some real-world concepts cannot, should not or may not need to be reflected (by Business Objects) for different reasons.

In addition, when creating Business Objects, one must take into account factors such as complexity, flexibility and efficiency as well.

In reflecting the real-world concepts, one should try to restrict the complexity of the structure of, as well as the semantics represented by, Business Objects.

One way to decrease the complexity is leaving details to other constituents (objects) than Business Objects. Besides, as the (business) concepts of the real world are continuously changing, and systems based on Business Objects should be able to adapt to the changes, Business Objects should be flexible. One way to increase the flexibility is reducing the dependencies between the concepts. For instance, [Herzum and Sims 1998] advocates using the concept of "minimal dependencies" not only at run-time but also throughout the development. As such, extracting the associations (relationships) from the related concepts and treating them as separate concepts can be helpful. Moreover, reducing the volume of exchanged data and information can reduce the complexity, improve flexibility and result in efficiency.

In brief, while creating Business Objects one should find the answers to the following questions:

- To what extent should the real-world concepts be reflected?
- Which of those real-world concepts should be reflected by Business Objects?
- To which abstraction levels should those Business Objects belong?

4.2 DEVISING BUSINESS OBJECTS

In the first place, Business Objects should be the "right" means for modelling (business) systems. That is, they should be efficient and effective means for concentrating on the essential real-world (business) concepts, and a means for communication between the people involved. Moreover, the devised Business Objects should offer the "right" means for providing the information model for Business Logic.

For instance, [Eeles and Sims 1998] emphasizes the role of Business Objects as a concept that does not (merely) represent a software constituent:

> *A Business Object is generally seen as being a "concept", not a lump of software. As such, it's useful in the requirements-capture and analysis stages of development because they're undertaken with the business domain in mind.*

Business Objects can be devised using the existing approaches for Object-Oriented analysis and design, such as those presented by [Rumbaugh 1991], [Jacobson et al. 1992], [Booch 1994], [Cook and Daniels 1994], [Larman 1998] and [Oestereich 1999]. However, as Business Objects are at a higher conceptual level than objects in the general sense, additional directives can/should be adopted for devising them. There are a number of specific directives for devising Business Objects in the literature, such as those presented by [Hertha et al. 1995], [Ramackers and Clegg 1995], [Sutherland 1995], [Hung and Patel 1997] and [Eeles and Sims 1998]. Although these approaches are based on different definitions of Business Objects, rule of thumb, such as the directives suggested in [Sullo 1994] can always be applied. Accordingly, it can be assumed that:

- Nouns should be represented by "objects", which stand for concepts, including people, places, things, documents, etc.
- Adjectives should be represented by "attributes", which stand for entities and descriptive characteristics.
- Verbs should be represented by "operations", which stand for processes, functions and actions.

In addition, to devise Business Objects, one can apply the following algorithm. This algorithm looks at the (main) real-world (business) concepts as belonging to one of the three categories: Entity, Process and Event[1].

1. Define the domain as accurately as possible. The area of the domain concerned can be as big as a whole business domain, or as small as a specific unit of a section of a department of a specific branch of a business domain.
2. Study the domain as exhaustively as possible using literature and available documents.
3. Find the people responsible for or (highly) involved in the domain.
4. Provide a list of the people (mentioned in 3).

[1]Categorization of Business Objects is discussed in {4.5}.

5. Interview the people on the list (mentioned in 4).
6. Try to acquire as much domain knowledge as possible through the interviews.
7. Edit the interviews and study them carefully.
8. Underline the key concepts.
9. Provide the interviewed (and other involved) people with the edited interview results, and get feedback.
10. If the results are (still) not satisfactory, go to 5.
11. If new people are found or introduced in the interviews, add them to the list and go to 5.
12. Make a list for each category, i.e. Event, Process and Entity.
13. Try to assign each concept to one of the three categories according to its rough characteristics.
14. Discuss each Event and summarize the related information.
15. Find out which Events can/should be merged and which Events can should be split.
16. Find out the hierarchies in the structure of the Events.
17. Find out the associations (relationship, inheritance, etc.) within Events. Think of chains of events, time orders, etc.
18. Based on each Event provide a scenario including different Events, Processes and Entities.
19. Discuss each Process and summarize the related information.
20. Find out which Processes can/should be merged and which Processes can/should be split.
21. Find out the hierarchies in the structure of the Processes.

22. Find out the associations (relationship, inheritance, etc.) within Processes. Think of preconditions and postconditions of each process with respect to the other processes, activity sequences, etc.
23. Discuss each Entity and summarize the related information.
24. Find out which Entities can/should be merged and which Entities can/should be split.
25. Find out the hierarchies in the structure of the Entities.
26. Find out the associations (relationship, inheritance, etc.) within Entities. Think of ownerships, collaborations, etc.
27. Find out the associations between the Processes and Events. Think of activities triggered by events, events caused by activities, etc.
28. Find out the associations between the Events and Entities. Think of changes in states as a result of events, events as a result of changes in states, etc.
29. Find out the associations between the Entities and Processes. Think of actors participating in activities, activities initiated by actors, etc.
30. If any new concept is recognized, go to 13.
31. Find out the attributes of each Event. Think of characteristics, associations, etc.
32. Find out the operations of each Event. Think of manipulation of attributes, activities, etc.
33. Find out the attributes of each Process. Think of characteristics, associations, etc.
34. Find out the operations of each Process. Think of manipulation of attributes, activities, etc.
35. Find out the attributes of each Entity. Think of characteristics, associations, etc.
36. Find out the operations of each Entity. Think of manipulation of attributes, activities, etc.

In addition, one can use a hypermethod, namely a combined top-down and bottom-up method. That is, the analysis methods can be put at the top and the definitions of the three categories at the bottom. Then a compromise between these two can be sought from both ends.

Furthermore, to devise Business Objects, one can take advantage of other (non Object-Oriented) methods for analysing the business in question. The applied method should make it possible to extract the characteristics and requirements of the business concerned, and present the results in suitable models. The resulting models can then be applied as input for the devising process.

An appropriate method for this purpose is the Dynamic Essential Modelling of Organizations (DEMO) [Dietz 1999] and [DEMO]. DEMO is a method for modelling business processes (in its general sense) that considers organizations as systems consisting of people with specific roles who act according to specific responsibilities and coordinate their actions by means of communication. This method can help one to identify the essential activities performed in the business concerned and describe these activities in a structured manner.

4.3 BUILDING BUSINESS OBJECTS

Business Objects should be the "right" means for realizing (business) software systems, according to the concept of the Multi-Tier Architecture. That is, it should be possible to use them with different data sources and user interfaces. They should also ease interoperation between different systems and platforms in distributed and heterogeneous environments. In other words, the built Business Objects should offer the "right" means for providing the Business Logic layer in the Multi-Tier Architecture.

There are a number of approaches for building Business Objects in the literature, such as those presented by [Ehnebuske et al. 1997] and [Eeles and Sims 1998]. Again, these approaches are based on different definitions of Business Objects. For example, [Ehnebuske et al. 1997] suggests building "(fine-grained) objects" based on the underlying databases, "Business Objects" based on objects and "business components" based on Business Objects and their user interfaces.

In general, building (the devised) Business Objects, as Business Objects or Business Object Components, is not different than building any other kind of object or component. Therefore, building Business Objects and Business Object Components can be based on the existing approaches for building objects and components as software constituents, such as those presented by [Dellarocas 1995], [Nierstrasz and Tsichritzis 1995] and [Heineman and Councill 2001].

Furthermore, there are certain differences between Business Objects and Business Object Components. Business Objects, in contrast to Business Object Components, do not have to support the notion of software plug and play. Besides, Business Objects should be transparent, but this is not a prerequisite for Business Object Components. Consequently, while building Business Objects provides a more visible and flexible means for realizing business software systems, building Business Object Components results in more reusable and interoperable software building blocks.

The interfaces of Business Objects should contain attributes, operations, relationships and descriptions, and the interfaces of Business Object Components should expose inputs and outputs, as well as introspection/self-describing and customisation/adjustment capabilities.

For implementing the interfaces of both Business Objects and Business Object Components, one can begin from scratch or reuse the existing software resources.

Implementing Business Objects from scratch can (potentially) result in a higher level of efficiency, although it can require more work. In this case, the devised Business Objects are directly implemented from the beginning. It is obvious that in this way one is free to take any factor into account to increase the efficiency.

Implementing Business Objects based on reuse can result in a lower level of efficiency, although it can (potentially) require less work. In this case, the devised Business Objects are implemented through the use of existing software constituents, including objects, components, modules and applications, in different forms, related to the possibilities and requirements. As such, searching for suitable software constituents is an indispensable part of reuse.

Although finding and integrating (the right) software constituents can be very difficult, and can restrict performance to the constituents used, taking advantage of existing software as much as possible is the preferred option for contemporary enterprises. [Skörd 1998] refers to the advantages of reuse for companies as follows:

> *One of the motives for companies to reuse classes is to save money in the long run. Another is that reuse gives the system a higher quality since the classes have been tested and verified in earlier systems. Reuse also makes it possible to achieve a higher pace of production since the classes can be used as building blocks when the system is being created. Another reason for reusing classes is to lower the maintenance costs. This is possible since the developers are more familiar with the classes used in the systems, plus there will be fewer classes to maintain.*

Then it names the most important characteristics of a reusable class as follows:

> *- The class should be flexible in the context and it should be easy to modify and adapt to one's needs.*
>
> *- A class interface should be easy to understand and it should also follow the company's coding guidelines.*
>
> *- A class should be independent and have minimal relations to other classes.*
>
> *- A class is quality assured since it has either been used in earlier systems or since it has been tested and verified according to the company's standards.*
>
> ...
>
> *- A reusable class is more expensive to develop than a specific class. This is due to the fact that the surrounding environment has more requirements on a reusable class than it has on a specific class, since the reusable class is to be used in many different contexts.*

Wrapping the legacy systems by objects is a special form of implementing Business Objects based on reuse. However, applying this form can decrease the efficiency, among others, due to the following reasons; there may be:

- entities that are repeated in different systems,
- superfluous functionality in different systems or
- complex relationships between different elements of different systems.

In addition, as [Semaphore 1997] notes, using existing systems implies a kind of compromise in modelling efforts and choosing a middle-out approach. Accordingly, this may lead to models that deviate from the desired model.

Moreover, the mentioned options can be applied together. In this way, Business Objects can be partly implemented from scratch and partly implemented based on reuse. Besides, Business Objects implemented based on reuse can later be gradually replaced with new ones that are implemented from scratch and/or are more efficient.

Finally, Business Object Components can be combined with other Business Object Components related to other (business) concepts. In this case, the resulting component, which is concerned with two or more (business) concepts, can be called a Business Object Block.

The functionality of Business Object Components can also be extended with other (macro) functionalities. With respect to the Five-Layer Architecture, the functionality of Business Object Components can be extended with the (macro) functionalities related to Presentation Logic and Information Logic. In this case, the resulting component, which is concerned with Core Logic, can be called a Business Core Component.

Accordingly, with respect to the Five-Layer Architecture, the functionality of Business Object Blocks can be extended in order to offer the whole functionality related to Core Logic for a number of interrelated (business) concepts. In this case the resulting component can be called a Business Core Block.

4.4 STANDARDIZATION OF BUSINESS OBJECTS[2]

Standard is *something established by authority, custom or general consent as a model or example* [Merriam-Webster]. The International Organization for Standardization (ISO) [ISO] has defined standards as follows:

[2]The background of this section can be found in [Abolhassani 2000-2].

Standards are documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines or definitions of characteristics, to ensure that materials, products, processes and services are fit for their purpose. ... International standards thus contribute to making life simpler, and to increasing the reliability and effectiveness of the goods and services we use. ... International standardization is well-established for many technologies in such diverse fields as information processing and communications, textiles, packaging, distribution of goods, energy production and utilization, shipbuilding, banking and financial services. It will continue to grow in importance for all sectors of industrial activity for the foreseeable future.

In the field of information and communication technology, different standards regarding different aspects, such as character sets, programming languages and operating systems, are devised and utilized. For instance, the Open Systems Interconnection (OSI), a subcommittee of ISO, has developed a reference model for communications architecture. This model, known as the OSI model, has provided the field of information and communication technology with important advantages.

Concerning the close relationship between Business Objects and components (and Business Object Components) as discussed in {3.6}, the following paragraph can clarify the necessity and importance of standards in this area [Herzum and Sims 2000]:

Traditionally, the main objective of good design has been to cost-effectively build a given system satisfying its functional and extra-functional requirements. Two additional objectives for component-based development are to make it simple to develop components themselves ... and to make life simpler for the developer who uses components. This last objective requires the provision of easy-to-understand and well-specified interfaces corresponding to a clean and scalable architectural model. Achieving this, in turn, requires consistency across interfaces which requires standards.

Furthermore, according to the definition of Business Objects, given in {2.3.2}, standardization is one of the main distinguishing characteristics as well as advantages of this concept, compared to the other (Object-Oriented) approaches in realizing software systems. The standardization feature of Business Objects makes them suitable for serving as a means of communication for the cooperating parties, for interoperation within or between different organizations and for utilization of software in (different) organizations.

In the area of information technology, the existing standards are mostly concerned with the syntax of data. This leads to (the need for) many different standards. For instance, the Reference Model for Open Distributed Processing (RM-ODP)[3] aims to provide a "big picture" that organizes the pieces of an ODP system into a coherent whole, but it does not try to standardize the components of such systems. Business Objects, on the other hand, are concerned with the semantics of data and processes, as they focus on the Business Logic and reflect the reality of the business domains. Since Business Objects represent semantical standards, they can provide a basis that leads to uniform standards for different business domains.

Moreover, Business Objects can also be intended for other "organizational units" than business domains, such as sections, departments and enterprises. Besides, countries and regions can also be considered as organizational units. The relationship between the organizational units can then be one of "subordination", "intersection" or "distinction".

In order to be able to take advantage of the standardization feature of Business Objects, one must deal with issues that stem from the very nature of business; the varieties of business methods, disagreements about concepts, as well as differences in cultures, circumstances, possibilities and requirements. Besides, the Business Logic may change over time due to reasons such as invalidation, obsolescence or changes of (parts of) semantics of the real-world concepts over time. Accordingly, some conflicts are known in advance and some are not. Therefore the differences should be handled differently. Keeping standards at the lowest possible level [Casanave 1995] is the easiest way

[3]The Reference Model for Open Distributed Processing (RM-ODP), a joint effort of International Organization for Standardization (ISO) and International Telecommunication Union (ITU), presents standards for distributed and heterogeneous systems through the use of a common interaction model. The goal of RM-ODP is to achieve portability of applications across heterogeneous platforms, interoperability through exchanging information, as well as offering functionality between distributed systems and distribution transparency through hiding the consequences of distribution from both the application programmers and users.

to deal with these difficulties.

The following subsections discuss how standards for Business Objects can/-should be devised and complied with.

4.4.1 DEVISING STANDARDS

Devising standards for Business Objects is closely related to devising and building Business Objects.

4.4.1.1 ABSTRACTION LEVELS

The abstraction levels involve three, orthogonal, and albeit interrelated, aspects, namely organizational unit, semantics and granularity.

ORGANIZATIONAL UNIT

Standard Business Objects should make "interoperation" and "reuse" possible. Therefore, as interoperation and reuse can take place at different organizational units, standards should also be concerned with different organizational units. Consequently, the intended organizational unit(s) can affect the level of abstraction that is suitable.

SEMANTICS GRADE

Standard Business Objects should present the semantics of the real world in such a way that they can accommodate different requirements and changes. Therefore, the grade of the semantics reflected by standards is essential. In fact, there is a trade-off between the grade of semantics on the one hand, and the level of applicability and flexibility of the devised standards, as well as the level of difficulty involved in devising standards on the other hand. These trade-offs must be weighed carefully.

GRANULARITY LEVEL

Standard Business Objects should form the Business Logic. Therefore, they should offer the right granularity levels. The granularity levels do not only involve modelling and realizing the Business Logic, they also influence the system performance, especially in distributed environments. Fine-grained Business Objects offer more flexibility, as they can be composed and re-composed into different shapes. Coarse-grained Business Objects offer more efficiency, as they hide details and take care of their inherent relationships.

In the literature, both fine-grained and coarse-grained Business Objects are advocated. For instance, according to [Sutherland et al. 1997]:

> *Technically, Business Objects encapsulate traditional lower-level objects that implement a business process (i.e., they are a collection of lower-level objects that behave as single, reusable units). User interfaces can be thought of as views of large-grained Business Objects.*

However, standards should be devised with respect to different granularity levels, as no single level would be sufficient.

4.4.1.2 CONCEPTS - ORGANIZATIONAL UNITS

When the same concepts apply to more than one organizational unit, one should take into account on the one hand the relationship between the organizational units concerned, and on the other hand the abstraction level of the concepts concerned.

Assuming that the concept C is applicable to organizational units OU1 and OU2, and that only OU1 and OU2 and no other organizational units are involved, six major situations, as shown in figure 4.1, can occur.

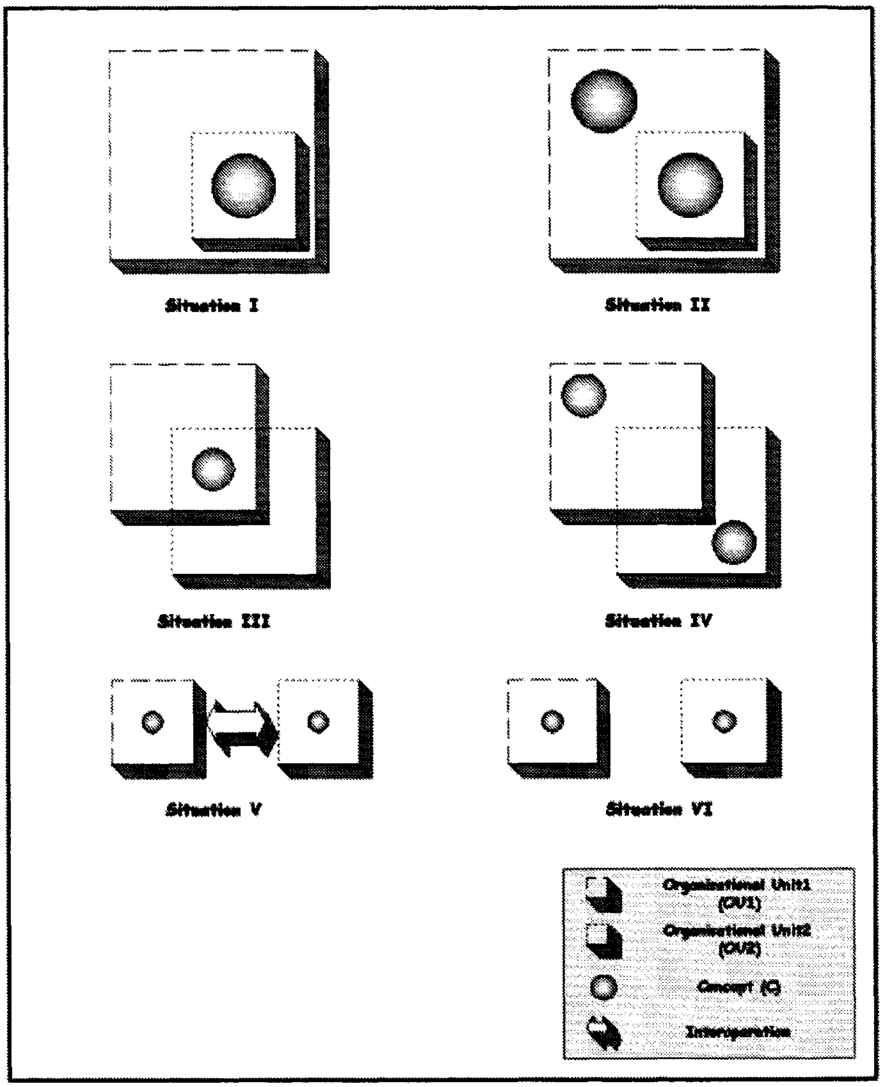

Figure 4.1: Different situations with respect to the relationship between concepts and organizational units.

In the following these situations are explained:

- I) OU2 is subordinate to OU1, and C is only inside OU2: The - single - C should be standardized for both OUs.
- II) OU2 is subordinate to OU1, and C is both inside and outside OU2: The abstraction levels of the Cs can be different, and most probably the

C inside OU2 is a more specific and detailed version of the C outside OU2. Therefore, at least, two Cs should be standardized for each OU, probably with the one outside OU2 as the basis for the one inside OU2.

- III) OU1 and OU2 intersect, and C is inside the intersection: The - single - C should be standardized for both OUs.

- IV) OU1 and OU2 intersect, and C is outside the intersection: The Cs can be of the same or of different abstraction levels. In the former case, the C in common should be standardized for both OUs. In the latter case, the two Cs should be standardized for each OU. However, it is better to provide a general standardized C, which can serve as the basis for the two specific Cs.

- V) OU1 and OU2 are distinctive, but interoperate: The Cs can be of the same or of different abstraction levels. In the former case, the C in common should be standardized for both OUs. In the latter case, the two Cs should be standardized for each OU. However, it is better to provide a general standardized C, which can serve as the basis for the two specific Cs.

- VI) OU1 and OU2 are distinctive, and do not interoperate: Two - different - Cs should be standardized for each OU.

4.4.1.3 ASSOCIATIONS

As mentioned in {4.1}, the associations between Business Objects should be kept at the lowest possible level. As not all of the real-world concepts are supposed to be reflected by (the standard) Business Objects, there are also associations between Business Objects and the other constituents of the Business Logic. Furthermore, not all of the standards, not even those concerned with a specific organizational unit, may be devised together. Therefore, in addition to specifying what the standard Business Objects can offer, one should also specify what these can expect from the other Business Objects or other constituents of the Business Logic.

4.4.2 COMPLYING WITH STANDARDS

The semantics of the concepts, on the basis of which the standard Business Objects are devised, may be different for different situations or may change over time. As change lies at the heart of modern business, it is the main reason why flexible business software is required.

Consequently, adopting the devised and built (standard) Business Objects for modelling business systems and realizing business software systems may not be straightforward, and some adaptations may be required. These adaptations may be of a conceptual or a technical nature.

In order to be able to take advantage of these adaptations, one needs to devise the standards such that they can accommodate the changes required. Besides, in order to utilize these adaptations, one needs to consider the available resources, application environment and desirable performance.

4.4.2.1 CONCEPTUAL ADAPTATIONS

The conceptual adaptations are mainly based on the definition and the interior traits of Business Objects.

ABSTRACTION LEVELS

The different abstraction levels of Business Objects, as discussed in {4.4.1.1}, can be taken into consideration. Business Objects that offer a lower grade of semantics and/or a higher level of granularity can be used as a basis, and Business Objects that offer a higher grade of semantics and/or a lower level of granularity can be partly or completely ignored.

CATEGORIES

The distinction between Entity Business Objects and Process Business Objects can be taken into consideration[4]. Entity Business Objects are more data oriented. Each Entity Business Object provides the functionality that is dedicated to itself, with no or minor involvement of other Entity Business Objects (of the same abstraction level). Process Business Objects are more function oriented. Each Process Business Object provides the functionality that is dedicated to handling specific business tasks, according to specific strategies, using the data and functionality of Entity Business Objects.

[4]These categories of Business Objects are defined in {4.5.1}.

Therefore, Process Business Objects reflect those concepts that are more subject to change in different situations and over time, and Entity Business Objects reflect those concepts that are, by nature, more stable and generally acceptable. Accordingly, (the standard) Entity Business Objects can be used as a basis, and Process Business Objects can be partly or completely ignored.

4.4.2.2 TECHNICAL ADAPTATIONS

The technical adaptations are mainly based on the location and the exterior traits of Business Objects.

DIRECT

In case the built Business Objects are objects consisting of an interface and implementation, as described in {4.3}, conventional Object-Oriented methods can be applied:

- Implementation: The implementation of Business Objects can be partly or completely changed, while the interface remains the same.
- Specialization: Business Objects can be specialized. This can take place both at the interface and at the implementation.
- Delegation: Business Objects can be delegated. This means that a Business Object can be included in a new object (class) as an instance variable. In this way, interaction with the Business Object can be controlled through the new object (class).

INDIRECT

As the connection between Business Objects and the User Interface layer takes place through Presentation Logic, defined in {3.3}, the desired changes can also be imposed thereby. That is, the constituents of the Presentation Logic layer, namely Presentation Objects, defined in {3.5}, can take care of desired modifications. However, this implies a deviation from the nature of Business Objects.

4.5 CATEGORIZATION OF BUSINESS OBJECTS[5]

"Categorization" (or "classification") means *systematic arrangement in groups or categories according to established criteria* [Merriam-Webster], and is closely related to "taxonomy", *the study of the general principles of scientific classification, ... orderly classification of plants and animals according to their presumed natural relationships* [Merriam-Webster].

Accordingly, categorization can be defined as the act of arranging things in, or assigning things to, groups, based on specific characteristics. It is a helpful practice, used in many fields of science and technology. Categorization is in fact a method for maintaining information in a logical manner. It is beneficial for organizing the conceptual requirements at different levels in an efficient way. The categories at each level can be mutually exclusive or not, and taken together, may include all possibilities or not. The taxonomies can then be used in two ways:

- Looking for the things with specific characteristics, beginning from the general ones.
- Determining the position of things according to their specific characteristics, with respect to the other ones.

In the field of computer science and technology, categorization has been applied in many different areas, like computer architecture [Skillicorn 1998], software development [Chance and Melhart 1999], information visualization [Shneiderman 1996], multimedia [Heller and Martin 1999] and security [Landwehr et al. 1993]. Moreover, as is well known, the Object-Oriented paradigm is closely involved in categorization. The most important feature of this paradigm is that concepts are divided into "classes[6].

As a concept that is closely associated with Object-Oriented paradigm, Business Objects should also take advantage of categorization. However, despite

[5]The background of this section can be found in [Abolhassani 2001-2].

[6]The terms "category" and "class" can often be used interchangeably. However, in the Object-Oriented paradigm, "class" has got a more specific meaning than "category".

some efforts, this concept has not yet adequately benefited from categorization. Besides, as Business Objects are at a higher semantical level than objects, categorization of Business Objects should go further than that applied in the Object-Oriented paradigm. In this way, categorization can be helpful for the creation and utilization of Business Objects.

Furthermore, categorization of Business Objects is closely related to standardization of Business Objects, discussed in {4.4}. Categorization can provide standardization with a more disciplined basis, and standardization can confirm categorization.

In general, categorization can help us to:

- Recognize real-world business concepts and concentrate on their main features.
- Recognize different abstraction levels.
- Represent different abstraction levels.
- Comply with the main features of the concept of Business Objects.
- Provide comprehensive and disciplined information about each concept and its related Business Object.
- Provide a general view on the existing (and missing) Business Objects.
- Use the existing Business Objects.
- Recognize the facilities and services required for Business Objects.
- Assign the required facilities and services to Business Objects.
- Adapt Business Objects.

In the following, the basic categories of Business Objects are described in more detail and some criteria for categorizing Business Objects (further) are suggested.

4.5.1 THE BASIC CATEGORIES

The definition of Business Objects given in {2.3.2} is based on the categorization of objects according to their roles in information systems. Accordingly, objects are divided into five main categories, namely Reflection Objects, Interaction Objects, Completion Objects, Connection Objects and Construction Objects. Business Objects are classed under the Reflection Objects.

In this way, Business Objects can also be divided into different categories. A well-known categorization of Business Objects is the following [Shelton 1995]:

> *Business entity objects: Are what usually come to mind in a discussion of Business Objects. They represent people, places and things, in much the same manner as a data-modelling entity...*
>
> *Business event objects: Represent business events, such as business time boundaries (end of quarter or fiscal year, elections, etc), changes in the business environment, product life cycles, etc. Many business event objects represent boundaries in time, while others recognize that some significant action has taken place...*
>
> *Business process objects: Represent business verbs. They represent business processes ..., where a process is characterized by the interaction of a set of Business Objects...*

This categorization is also in agreement with the view of BODTF on different Business Object categories [OMG 1997-2]. In fact, the idea is not new. A similar categorization was already assigned to objects in general [Shlaer and Mellor 1988]:

> *An object is an abstraction of a set of real-world things... Most of things are likely to fall into the following five categories: tangible things, roles, incidents, interactions and specifications.*

Assuming that here the term "object" refers to the Reflection Objects, as in many other cases, and in view of the definitions of the categories that are given, we can note that:

- *Tangible things* and *specifications* are comparable with *entities*,

- *Incidents* are comparable with *events* and
- *Interactions* are comparable with *processes.*

(Dedication of a special category to *roles* is a matter of taste.)

Furthermore, this approach is also supported by the systems analysis and design Methods [Tudor and Tudor 1995]:

> *Early structured methods (De Marco, Yourdon) concentrated on modelling the system in terms of processes and the data flowing between them. It soon appeared that, in all but small, real-time systems, the structure of data was important. Techniques to handle data structure were added (entity modelling being the principal one). It then emerged that it was not just data and processes which were important, but the timing and effects of the interaction between them. Thus, now that structured methods have matured, they have adopted a three-sided approach to specification of the system:*
>
> *- Data*
>
> *- Processes (functions)*
>
> *- Events (and the effect of these on processing and data)*
>
> *What the modelling techniques need to model is the following situation:*
>
> *An event in the outside world will trigger a process. This will cause an effect on data in a given state and (if this is an updating event) will transform data into another recognized state.*

Accordingly, the Business Object categories can be defined as follows:

Entity Business Objects (EBOs) are those Business Objects that represent tangible and intangible business facets, such as "person", "bank account" and "order", and encapsulate enterprise data and their manipulations according to business rules and constraints. Attributes play the most important role in the structure of this category.

Process Business Objects (PBOs) are those Business Objects that represent business activities, such as "purchase", "money transfer" and "delivery", and encapsulate units of enterprise work and their data according to business rules and constraints. Operations play the most important role in the structure of this category.

Event Business Objects (VBOs) are those Business Objects that represent persistent records of, and notification means for, business happenings, such as "reception", "in debt" and "end of year", and are concerned with state changes, action occurrences and time boundaries.

In fact, Entity Business Objects are the means that are necessary to carry out the activities described by Process Business Objects, and Event Business Objects mainly result from interactions between Entity Business Objects with respect to Process Business Objects and can trigger other Process Business Objects.

The importance of categorization can be underlined irrespective of the definition of Business Objects and the categories concerned. For instance, regarding Entity and Process Business Objects, based on a different view on these categories, [Schmid et al. 1998] declares:

> *A business entity provides services embodying business rules, which are common to and used in different applications. ... A business process, or, as a smaller part, a business procedure or a business activity, is a processing sequence that requests services from business entities; it is specific for an application.*

Consequently, it mentions the three following differences between Entities and Processes:

- Processes request services from Entities and not vice versa.
- Entities have a permanent state and Processes do not.
- Sharing of Entities involves transaction mechanisms and sharing of Processes does not.

4.5.2 THE OTHER CATEGORIES

The most obvious and straightforward basis for categorizing Business Objects (further) is to follow the basic categorization, and to proceed with the Entity, Process and Event categories. For instance, [Schmid and Simonazzi 1998] represents a kind of taxonomy for Process Business Objects that includes "Procedures", "Activities" and "Sub-Activities". The same can be done for Entity Business Objects and Event Business Objects. For example, Entity Business Objects can be divided into "Active" and "Passive" categories, and Event Business Objects can be divided into "Process-Dependent", "Entity-Dependent" and "Time-Dependent" categories.

Obviously the existing taxonomies of various fields of life, including business (domains), can also be used and extended to categorize Business Objects in general, and Entity Business Objects in particular.

However, there are many other criteria that can (and must) be used as a basis for categorization of Business Objects. These criteria are mainly related to semantical and structural aspects.

4.5.2.1 SEMANTICAL CRITERIA

The semantical criteria are mainly based on the definition and the interior traits of Business Objects, and are mainly related to the devising of Business Objects.

APPLICATION DOMAIN

Business Objects can be applied to different business domains. Therefore, the application domains of Business Objects can be divided into two major categories:

- Horizontal applications, which can be used for a wide range of business activities.
- Vertical applications, which are related to particular business activities.

Accordingly, Business Objects can be divided into two categories with respect to Horizontal and Vertical applications. The Vertical category can then be further extended for each specific domain, sub-domain and so on.

[Bonar 1997] presents classifications with respect to applications of Business Objects.

APPLICATION TYPE

Business Objects can be used for different application types [Abolhassani 2000-1]. The application types can be divided into two major categories:

- OnLine Transaction Processing (OLTP) applications, which are involved in the day-to-day activities of an enterprise.
- OnLine Analytical Processing (OLAP) applications, which are involved in the decision-making activities of an enterprise.

Accordingly, Business Objects can be divided into two categories with respect to OLTP and OLAP applications. Each of these categories can possess specific features needed for the related type of application.

ABSTRACTION LEVEL

Business Objects can be related to different abstraction levels, as discussed in {4.4.1.1}. The abstraction levels are based on three aspects, namely:

- Organizational unit
- Semantics grade
- Granularity level

Accordingly, Business Objects can be divided into categories with respect to target organizational units, grades of semantics and levels of granularity.

4.5.2.2 STRUCTURAL CRITERIA

The structural criteria are mainly based on the location and the exterior traits of Business Objects, and are mainly related to the building of Business Objects.

STATE

Business Objects can outlive the session in which they are instantiated or not. In the former case they are called "stateful" and in the latter case they are called "stateless".

Therefore, Business Objects can be categorized according to their state.

ACCESS RIGHT

Business Objects can be used according to their access rights. (In the general sense, "access right" determines who has the "right" to "access" what and in which manner.) For instance, Business Objects can be "read-write" or "read-only".

Therefore, Business Objects can be categorized according to their access rights.

UTILIZATION

Business Objects may (need to) utilize different facilities and services, such as "persistence", "transaction" and "security".

Therefore, Business Objects can be categorized according to the facilities and services they utilize.

DEPENDENCY

To be able to start, continue, accomplish or finish their task, Business Objects may, or may not, depend on:

- Other Business Objects
- Other objects
- Other system constituents
- Other systems
- Users

Therefore, Business Objects can be categorized based on their dependencies.

Chapter 5

USING BUSINESS OBJECTS

The real problem is not whether machines think but whether men do.

[B. F. Skinner]

The use of Business Objects is related to their location and creation. This chapter discusses the issues concerned on the basis of the Five-Layer Architecture.

The first section outlines the general issues. The second section concentrates on the layers of the Five-Layer Architecture and the relationship between these layers. The third section elaborates on a number of options regarding the relationship between Business Objects and data sources, and different perspectives on the basis of which Business Objects can be created[1].

5.1 GENERAL ISSUES

In order to use Business Objects in an organizational unit one should take some general issues into consideration. These issues are related to a number of conceptual and technical aspects.

The conceptual aspects include:

[1]The background of this chapter can be found in [Abolhassani and van Groen 2000].

- The relationship between the organizational unit concerned and other organizational units.
- The viewpoint of the organizational unit concerned on the concept of Business Objects.
- The specification of the universe of discourse concerned.

With respect to the universe of discourse, one should decide within which borders the real-world concepts should be searched for, and which concepts should be reflected (by Business Objects). Besides, there are concepts that can only exist within software systems, and hence not in the real world.

The technical aspects include resources, possibilities and requirements, such as:

- The development time.
- The development approach.
- The development resources, including people, platforms and communication means.
- The (legacy) databases.
- The usage characteristics, including frequency of use and (the desired) run-time speed.

5.2 SYSTEM LAYERS

There are two (contradictory) trends in information systems, namely distribution and centralization. On the one hand, by virtue of the new possibilities, especially with respect to communications, information systems are spread over different locations. On the other hand, because of the importance attached to information, each enterprise wants to have a centralized control over it. The Five-Layer Architecture and Business Objects can be used to accommodate both of these trends. As mentioned in {3.5}, objects, including Business Objects, can be spread over different locations in order to take advantage of different software and hardware resources for storage, processing, communications, etc., and to deal with issues such as performance, authorization and security. Besides, the Information Logic layer can deal with the distributed data, and provide the Business Logic layer with the needed information. The Business Logic layer, in turn, can provide a centralized information model, accessible for the Presentation Logic layer.

However, specifying the location of each layer according to its nature, possibilities and requirements is an important issue, and asks for careful engineering and management of distributed resources. For instance, placing two (or more) layers on the same location can simplify communication and increase speed, while distributing the layers can increase flexibility and optimise utilization of resources.

In any case, reducing the volume of exchanged data and information on the wire to the lowest possible amount can increase both speed and flexibility. However, this costs more time and effort in the system development.

A convenient distribution of the layers of the Five-Layer Architecture, which resembles the Three-Layer Architecture, is as follows:

- User Interface and Presentation Logic at the client side.
- Business Logic and Information logic at a central location accessible for all clients.
- Data Source at different locations accessible for Information Logic.

However, as the Five-Layer Architecture divides a system into five logical layers, it offers more flexibility for the distribution of system according to each specific situation.

In the following subsections, each layer is discussed further.

5.2.1 USER INTERFACE

The User Interface layer is an essential and indispensable layer, and is the only layer of the system that deals directly with the user. This layer can be realized based on Interaction Objects or other software constituents. Besides, the IOs (and other constituents of UI) that are used very often can be componentized. This is the idea on which components like "Controls"[2] are based.

Furthermore, if one supplies UI constituents as Internet-enabled components and places them in a central location, a Web browser at the client side is sufficient for a user interface. In this way, the UI components can be installed, updated and upgraded only in one location.

5.2.2 PRESENTATION LOGIC

The Presentation Logic layer is not essential for the system, although it can improve application development. It bridges the gap between the Business Logic layer and the User Interface layer, and can be realized based on Presentation Objects. Besides, the POs (and other constituents of PL) that are used very often can be componentized.

In case the constituents of the User Interface layer are based on a specific model, the Presentation Logic can also be considered as a "mapper" between that model and the model on which the Business Logic layer is based[3].

5.2.3 BUSINESS LOGIC

The Business Logic layer is an essential layer, and forms the central point of the system. This layer should be (preferably) realized based on Business Objects and/or Business Object Components, although other software constituents can be used as well.

However, despite the important role of BL in the system, it can be omitted in the following two situations:

- When the Business Logic is trivial.
- When the Data Source is sophisticated.

[2]Controls include items such as text boxes, list boxes, check boxes, combo boxes, option groups, option buttons, command buttons and toggle buttons.

[3]For an example of this case see [Abolhassani and Szentivanyi 1999].

In these situations the concepts can be directly presented through the Data Source (and Information Logic), and the rules and constraints concerning these concepts can be maintained through the Data Source (and Presentation Logic). In this way, by providing a direct connection between the Presentation Logic or User Interface on the one hand, and the Information Logic or Data Source on the other hand, one can increase the performance.

5.2.4 INFORMATION LOGIC

The Information Logic layer is an essential and indispensable layer for the system in case there are two or more data sources. It bridges the gap between the Business Logic layer and the Data Source layer, and can be realized based on Information Objects. Besides, the FOs (and other constituents of FL) that are used very often can be componentized.

The Information Logic layer hides the details of data access from Business Objects, and hence insulates them from migration of data sources (databases). In this way, the Information Logic layer acts as an intermediate between the existing data models on the one hand, and the Business Object model on the other hand. In this way when the constituents of the Data Source layer are based on a model (for instance the "relational" model) that differs from the model on which the Business Logic layer is based (most likely the "Object-Oriented" model), the Information Logic can also be considered as a "mapper" between those models, and the Information Logic layer should take care of conversions between the constituents of the two models (for instance tables and objects).

Information Logic can be as simple as a table consisting of data entries and their location in a data source, along with the required characteristics of the respective data source, or as complex as procedures dealing with contradictions, duplications, etc. of data in different data sources.

Accordingly, the Information Logic layer can get involved in issues such as querying, locking, transaction management and communication. Therefore, the Information Logic layer should take advantage of the Control Logic, and can directly affect the performance of the whole system.

5.2.5 DATA SOURCE

The Data Source layer is an essential and indispensable layer of the system. Although each layer of the system may store and retrieve (temporary) data through the environments embracing them, DS is the only layer of the system that takes care of (persistent) data of the whole system. As mentioned in {3.5}, in case Object-Oriented database systems are used, this layer can be realized based on Data Objects.

Furthermore, the applied database system, its characteristics and in particular its category influences the design, implementation and performance of the whole system.

5.3 OPTIONS AND PERSPECTIVES

As regards the effect of the applied data sources (databases) on the whole system, the relationship between the Business Objects that form the Business Logic layer and the data sources that form the Data Source layer is of crucial importance. Although this relationship is taken care of by the Information Logic layer, there are a number of interrelated "options" that should be taken into consideration. Decisions about the (preferred) relationship between Business Objects and data sources should be based on an evaluation of these options. In addition, the creation of Business Objects can be based on different "perspectives".

The following subsections discuss these options and perspectives.

5.3.1 OPTIONS

5.3.1.1 LEGACY/NEW

Business Objects may use legacy data sources (databases) or new customized ones. In the former case, the data sources and their characteristics dominate the relationship with Business Objects. In the latter case, the data sources are realized based on the requirements of Business Objects. Therefore, choosing one of these options also involves the perspectives discussed in {5.3.2}.

The use of legacy data sources is in agreement with the "reuse" principle, and can result in optimal utilization of resources, while the use of new data sources can increase flexibility and efficiency. For instance, realizing new data sources can lead to simpler Information Logic, and even eliminate the need of an Information Logic layer.

Furthermore, it is also possible that Business Objects use both legacy and new data sources at the same time. That is, Business Objects can retrieve data from the legacy data sources and save their state in the new data sources. Therefore, these options also involve the Persistent/Temporal options discussed in {5.3.1.2}.

5.3.1.2 PERSISTENT/TEMPORAL

Business Objects may or may not pass on the changes made to them to data sources. In the former case, the changes can outlive "sessions" and become persistent. In the latter case, the changes cannot outlive "sessions" and are temporal. Therefore, these options also involve the One/More options discussed in {5.3.1.3}.

The use of Business Objects as temporal software constituents only makes sense if there is no "crucial" data involved, or if the existing data can/should not be changed during the session. Process Business Objects that do not posses any "crucial" data can be examples of the first case, and Read-Only Business Objects can be examples of the second case.

The use of Business Objects as persistent software constituents is more likely, and, in itself, involves the "state" category of Business Objects. In case of Stateless Business Objects, data should be written back to its origin in data sources. In case of Stateful Business Objects, there are three possibilities:

- The changes are inherently saved as "states" of Business Objects in their allocated data source.
- The "states" of Business Objects are inherently saved as changes in their original data sources.
- The "states" of Business Objects are saved in their allocated data source, and the changes are saved in their original data sources.

Furthermore, in some cases it is also possible that Business Objects are stateful in relation to one specific user, and stateless in relation to the other users. That is, the state of Business Objects can be saved for a specific user during a specific application.

5.3.1.3 ONE/MORE

Business Objects may be instantiated separately for each individual (real-world) concept or for each (real-world) concept as a whole. In the former case, data can be extracted in due course, and in the latter case all data can be extracted in one go from the data sources concerned.

The choice for one of these options strongly depends on the specific cases for which the Business Objects are used. Both options can provide efficiency or result in inefficiency.

Although the first option is more consistent with the nature of Business Objects, in many cases where relational database systems are used as data sources, there is a one-to-one relationship between Business Objects (and in particular Entity Business Objects) and "tables". Besides, in some cases, Business Objects can be associated with a number of the fields of one table (called "projection"), or a number of the fields of two or more tables (called "view"). Furthermore, different classes of Business Objects may also be associated with a common table. In this way, multiple classes of an inheritance tree can be associated with a single table.

5.3.2 PERSPECTIVES

5.3.2.1 DATA

Business Objects can be created in agreement with data existing in (legacy) data sources. This perspective suggests a one-to-one relationship between data source (database) entities, such as tables, and Business Objects, while data source entities are the starting point[4]. Accordingly, the entities concerned should be chosen based on factors such as the characteristics of the respective data sources and the One/More options, discussed in {5.3.1.3}.

[4]This perspective is described in [Jerke et al. 1999].

The main advantage of this perspective is that it makes the relationship between Business Objects and the Data Source layer straightforward. Consequently, Information Logic can be simpler, or it might even be possible to leave out the Information Logic layer altogether. However, the Business Objects created based on this perspective would not be suitable for (new) data sources that are added to the system later on.

The main disadvantage of this perspective is that it may result in Business Objects that are not consistent with the definition of Business Objects, and (even) deviate from the nature of this concept.

5.3.2.2 APPLICATION

Business Objects can be created in agreement with the intended applications. This perspective suggests an association between application requirements and Business Objects.

The main advantage of this perspective is that it makes the relationship between Business Objects and the User Interface layer straightforward. Consequently, Presentation Logic can be simpler, or it might even be possible to leave out the Presentation Logic layer altogether. However, the Business Objects created based on this perspective would not be suitable for new applications of the system.

The main disadvantage of this perspective is that it may result in Business Objects that are not consistent with the definition of Business Objects, and (even) deviate from the nature of this concept.

5.3.2.3 REAL WORLD

Business Objects can be created in agreement with real-world concepts. This perspective suggests a one-to-one relationship between real-world concepts and Business Objects.

The main advantage of this perspective is that it can result in Business Objects that are consistent with the definition of Business Objects, and the nature of this concept.

The main disadvantages of this perspective are that it may demand more effort, and may decrease the efficiency with respect to the relationship between Business Objects and both user interfaces and data sources.

Chapter 6

STATE-OF-THE-ART

Protocol is everything.

[Françoise Giuliani]

The preceding chapters discussed different aspects of Business Objects. Using the discussed subjects as a background, we can survey the environments in which software systems based on the concept of Business Objects can be realized, and technologies with which these environments can be enabled. Accordingly, this chapter surveys the representative enabling technologies and system environments with respect to the discussed subjects concerned with architectural and construction aspects[1].

The first section addresses the general issues. The second section specifies the architectural and constructional criteria for the assessment of the technologies and environments for Business Objects. The third and fourth sections survey a number of enabling technologies and system environments according to the specified criteria.

6.1 GENERAL ISSUES

The emergence of high-bandwidth communication channels and the extreme growth of computer networks, Internet and Intranets on the one hand, and

[1]For a survey and comparison of different enabling technologies for Business Objects from another viewpoint see [Emmerich et al. 1998].

the emergence of several information and communication standards on the other hand have promoted the distribution and heterogeneity of software systems. Business software systems should take advantage of the new possibilities, and at the same time deal with continuous change in business.

Obviously, Monolithic software systems cannot benefit from the new possibilities, nor comply adequately with the changes in business. Although software systems based on the Client/Server architecture can be more appropriate, inherently they cannot fully benefit from the new possibilities and are not flexible enough to comply with the changes in business. Separating the whole functionality of a software system into loosely coupled layers (tiers) is a better way to comply with the business requirements, in view of the technical possibilities.

Furthermore, as mentioned in {3.5}, although Business Objects can be used for systems based on Monolithic and Client/Server architectures, software systems based on a Multi-Tier Architecture are most suitable for applying the concept of Business Objects. Therefore, our survey is confined to those enabling technologies and system environments that can be used for realizing software systems based on a Multi-Tier Architecture.

Enabling technologies should provide basic facilities and services, and in particular communication means for realizing distributed and heterogeneous software systems. System environments should use these enabling technologies to make realizing such software systems possible. Besides, system environments should let users construct Business Objects, and should support separation of the Business Logic from other parts of the system.

Moreover, systems should be able to interoperate and cooperate with other systems. A prerequisite for this is that the system environment is open and extensible. Besides, ease of use for the users and developers can be considered as an extra advantage.

6.2 ASSESSMENT CRITERIA

In the previous chapters we discussed what the use of Business Objects involves: locating, devising, building, standardizing and categorizing Business Objects. In the following subsections, we specify the criteria for assessing the existing resources and tools for Business Objects according to these architectural and construction aspects.

6.2.1 ARCHITECTURAL ASPECTS

Architectural aspects are related to the constituents that form software systems in which Business Objects are located, and the means that take care of communication between these constituents.

6.2.1.1 SYSTEM CONSTITUENTS

Each system constituent should handle a specific (macro) functionality and isolate the other constituents from its (implementation) details. These details should be encapsulated behind an appropriate abstraction or interface that presents only the essential constructs to the other constituents. In this way, an entire system constituent can be replaced without impacting the other constituents. Parallelism in development is another advantage of this approach. An appropriate architecture should at least include the following system constituents:

- A User Interface layer
- A Business Logic layer
- A Data Source layer
- A Control Logic

The existence of these layers should be supported, and assistance in realizing these layers may be given at different levels.

The Control Logic should provide different facilities and services, including:

- Naming: allows naming of Business Objects so that they can be accessed through the use of a unique name.

- Life Cycle: supports the creation (providing a default state), copying, moving, activation (acquiring the state from persistent storage), deactivation (dumping the state into persistent storage) and deletion (removing the state from persistent storage) of Business Objects.
- Persistence: helps in storing and retrieving (the states) of Business Objects, and in accessing data in (legacy) data sources.
- Relationship: helps in creating dynamic associations between Business Objects and helps to provide mechanisms for traversing the links that group the related Business Objects.
- Transaction: coordinates the changes of (the states of) Business Objects shared by different clients that may perform actions at any time.
- Event: makes the clients aware of changes in (the states of) Business Objects.
- Security: helps in making Business Objects secure.

6.2.1.2 COMMUNICATION MEANS

Communication means should provide (seamless) access to Business Objects and other constituents of the system. Moreover, the system constituents should be spread over computer networks in order to take advantage of resources efficiently. However, the decisions and assumptions involved in performance issues may change over time due to changes in the technical infrastructure and other areas. Besides, load balancing and fail-over handling may also be desired. Accordingly, it should be possible to migrate system constituents with no or minor changes in the involved codes or specifications. Therefore, communication means should also provide transparent migration of system constituents.

Furthermore, in addition to the synchronous communication, communication means may also offer asynchronous communication. If this is the case, the client can simply be notified when the requested results are ready. If this is not the case, the client must either wait or repeatedly poll until the requested results are ready. As such, in large systems consisting of many constituents this can cause problems and reduce the performance[2]. While the synchronous "call and wait" communication can be sufficient for communications among coarse-grained constituents, communications among fine-grained constituents can be maintained better by the asynchronous "send and continue" communication.

As a familiar communication means, the Object Request Brokers (ORBs) are communication buses that allow objects to interoperate across address spaces, programming languages, operating systems and networks. They can, in principle, offer the desired means, albeit with different levels of flexibility.

6.2.2 CONSTRUCTION ASPECTS

Construction aspects are related to technical and conceptual factors concerned with the creation of Business Objects.

6.2.2.1 TECHNICAL

The construction of Business Objects can be technically supported in different ways, such as by providing modelling tools, taking care of source or binary code production and offering ready-made constituents. This support may be bound to specific programming languages and platforms.

The programming languages used for constructing Business Objects may or may not support Object-Oriented concepts, such as inheritance and polymorphism. Furthermore, when system constituents are tightly integrated with a specific platform, low-level system services can be used. This may reduce the complexity of the code and can result in a higher performance.

The ready-made constituents come in different forms and at different abstraction levels. For instance, an "abstract class" is a design model for a single object, and a "framework" is a design model for a set of objects that collaborate to carry out a defined set of responsibilities. Thus frameworks have a higher granularity level than abstract classes, and abstract classes can have a higher semantical grade than frameworks.

[2]There are two methods to work around this shortcoming, namely 1) using threads and 2) simulating asynchronous call backs through calls from server to client. However, the first option makes code more complex and the second option is not pragmatic when there is more than one client.

6.2.2.2 CONCEPTUAL

The construction of Business Objects involves the following conceptual issues:

- The reuse of legacy software constituents.
- The deployment of standard Business Objects.
- The deployment of Business Object categories.
- The specification of the relationship between Business Objects and data sources.

In this respect the user can be provided with different options and can be supported with relevant means.

It may be necessary to use the existing software constituents for the construction of Business Objects to save on resources.

Standardization is the main characteristic of the concept of Business Objects. As such, in addition to support the construction of Business Objects, enabling technologies and system environments may support the use of off-the-shelf Business Objects and building-block software constituents, and may assist in the customisation of standard Business Objects.

Categorization of Business Objects based on different criteria can be supported in order to enhance the design and implementation of Business Objects.

Business Objects may use legacy or new data sources, be persistent or temporal and represent one or more instances of a real-world concept.

6.3 ENABLING TECHNOLOGIES

6.3.1 CORBA[3]

The Common Object Request Broker Architecture (CORBA) [CORBA], presented by Object Management Group (OMG), is a specification that

[3]This assessment is based on the CORBA specifications. Of course each implementation of the specifications may have its own peculiarities.

forms the major part of a technological infrastructure intended for reusable, portable and interoperable components in distributed and heterogeneous environments, through a set of interfaces and protocol specifications, named Object Management Architecture (OMA).

Interface Definition Language (IDL) is a descriptive programming language that forms the basis of CORBA concepts, interfaces and protocol specifications.

6.3.1.1 ARCHITECTURAL ASPECTS

The OMA architecture is based on a cooperative paradigm that should allow applications to cooperate dynamically across the network.

SYSTEM CONSTITUENTS

CORBA is appropriate for realizing software systems based on a Multi-Tier Architecture. However, the realization of the Business Logic layer is the only area that is directly supported.

Furthermore, CORBA presents the Control Logic in the form of OMG object services, which are collections of system-level services packaged as objects, belonging to the Completion Objects category, with an IDL interface. The purpose of these services is to provide a set of standard interfaces in order to implement generic object services. These include:

- Naming (CORBA Naming Service)
- Life Cycle (CORBA LifeCycle Service)
- Persistence (CORBA Persistence Service)
- Relationship (CORBA Relationship Service)
- Transaction (CORBA Transaction Service)
- Event (CORBA Event Service)
- Security (CORBA Security Service)

COMMUNICATION MEANS

The basic part of CORBA is an Object Request Broker. It enables sending messages to objects, and receiving the corresponding responses transparently, in a distributed environment, irrespective of the locations. The requests to objects may be issued by processes related to a procedure or an object. The requested objects may be present in the same process that initiates the request or be distributed over different processes. The ORB can link objects both statically, that is, interfaces are known at compilation time, and dynamically, that is, interfaces are discovered at run-time.

The first versions of CORBA did not support asynchronous communication, but the later versions did.

6.3.1.2 CONSTRUCTION ASPECTS

TECHNICAL

The CORBA object model strongly separates the "interface" from the "implementation". The interfaces should be defined through IDL, which supports the Object-Oriented mechanisms of inheritance and polymorphism. However, IDL can only be used for defining interfaces, and only in a rigorous, programming-language-independent manner. IDL definitions can then be compiled for different programming languages. IDL mapping exists towards the main Object-Oriented languages such as C++, Java and Smalltalk. Therefore, Business Objects do not have to be implemented in a specific Object-Oriented language, and on/for a specific platform. In this way, CORBA provides an architecture that can encompass various programming languages and platforms. Besides, clients and servers can be implemented in different languages and still interact.

However, being programming-language-independent, CORBA must convert any object to flat records or opaque byte streams before sending them over the wire, and back to objects again at the receiving end. Besides, CORBA objects cannot directly exploit the specific features of Object-Oriented languages. Being platform independent, CORBA objects cannot take advantage of low-level system services.

Furthermore, directly converting interfaces into implementations may cause significant performance problems. For instance, performance may decrease when objects have to serve too many purposes. Moreover, IDL cannot capture the semantics of the real-world (business) concepts adequately.

Component Description Language (CDL), promoted by the OMG Business Objects Domain Task Force (BODTF), can be considered as a superset of IDL, and is another language for specifying Business Objects. CDL is meant to capture the semantics of business concepts, including business rules. CDL can then be compiled to IDL interfaces. However, in mapping CDL to IDL the business rules (may) get lost, and hence additional code must be generated in an Object-Oriented programming language in order to embody the lost semantics.

OMG provides standard IDL interfaces in specific domains such as telecommunications, management, health care and finance. However, working at IDL level is not easy. Frameworks and other tools can assist the developer in this respect. For instance, high-level interfaces can hide the highly technical IDL interfaces from application developers, and frameworks can provide interfaces appropriate for directly developing Business Objects. Besides, CORBA offers vertical application frameworks that can be used directly by Business Objects.

CONCEPTUAL

CORBA is dedicated to objects of a relatively high granularity level. Therefore, objects of low granularity levels are not efficiently supported by CORBA. Furthermore, CORBA is based on the notion that not all the software constituents in an enterprise are Object Oriented. That is the reason why CORBA IDL should provide descriptive features that are not tied to any language or object model. This makes CORBA appropriate for the "reuse" of legacy software constituents. Using CORBA, one can represent a (server) component as an object, even if it is a wrapped chunk of legacy code.

The CORBA Component Model (CCM), included in the newer versions of CORBA, extends the CORBA object model by defining features that enable application developers and users to develop and deploy components that integrate CORBA services in a standard environment. This can, in particular, ease the development of CORBA applications. In addition, CCM can promote "reuse", as well as the concept of software plug and play.

With respect to standardization of Business Objects, CORBA is involved in the standardization efforts of OMG. These standards, which are based on different real-world (business) concepts related to different business domains, do not address different abstraction levels and do not categorize adequately.

With respect to the relationship between Business Objects and data sources, CORBA does not impose any restriction. Besides, it recognizes a specific kind of objects called "active", which represents those objects that provide services for different objects in the form of a single object. These objects share data among different objects and are not instantiated by each requesting object separately.

6.3.2 ACTIVEX

ActiveX [ActiveX], presented by Microsoft, embraces a set of Object-Oriented technologies and tools, including Component Object Model (COM), Distributed COM (DCOM) and COM+. It provides an environment for the creation and deployment of components named ActiveX controls. COM is a framework for the development and deployment of ActiveX components, DCOM extends the capabilities of COM to support interoperation of components in a distributed environment and COM+ extends COM with a set of new services for application development.

This concept originates from Object Linking and Embedding (OLE), a framework for assembling and managing compound documents, and an application environment containing objects originating from different sources under Microsoft Windows. In turn, OLE had replaced Dynamic Data Exchange (DDE), which was based on string interchange through shared memory. OLE improved DDE by supporting data interchange through a clipboard. OLE controls, named OCX, were software constituents with specific functionality like Graphical User Interface (GUI) entities.

6.3.2.1 ARCHITECTURAL ASPECTS

SYSTEM CONSTITUENTS

The User Interface layer, Presentation Logic layer and Business Logic layer can be realized using any programming language that supports ActiveX.

The realization of the Information Logic layer depends on the applied data sources (databases). For instance, using MS SQL Server, a database management system, as data source, the Information Logic layer can use Stored

Procedures[4] to collect data. Compared to Queries, which can be transmitted to databases in the form of strings inside the "calling code", Stored Procedures have two important advantages:

- Stored Procedures are precompiled and are therefore faster.
- Stored Procedures can be reused in different places.

Furthermore, the Information Logic layer can use Open Database Connectivity (ODBC)[5].

For the Control Logic, ActiveX offers:

- Naming (Monikers/Repository)
- Life Cycle (IClassFactory interfaces)
- Persistence (IPersistStorage, IpersistStream)
- Transaction (Microsoft Transaction Server (MTS))
- Event (Connections interface)

COMMUNICATION MEANS

ActiveX controls can be deployed by different applications within a single computer, as well as by different applications in computers distributed over a network.

ActiveX does not support asynchronous communications.

[4]Stored Procedures, introduced by Sybase, are groups of Structured Query Language (SQL) statements that provide flow control facilities and accept parameters, except "table" names, for execution of operations and exchange of data. They are more efficient than sending SQL statements over the network.

[5]Open Database Connectivity (ODBC) is an open standard Application Programming Interface (API) for accessing different databases, including Access, dBase and DB2, without having to know the proprietary interfaces to them.

6.3.2.2 CONSTRUCTIONAL ASPECTS

TECHNICAL

ActiveX technology is language independent. That is, ActiveX controls can be created using different languages, such as: C, C++, Visual Basic and Visual Java. ActiveX controls are implemented as Dynamic Link Library (DLL) modules.

ActiveX technology depends on the Microsoft platform, and is tightly integrated with Microsoft products. It can use low-level system services of the Microsoft platform, and hence can reduce the complexity of code and result in a higher performance.

ActiveX components can be deployed as ready-made building blocks. They are delivered in binary code and expose their interfaces in a well-defined binary form.

Furthermore, Microsoft Foundation Classes offer a set of customisable services that can be used in some general domains.

CONCEPTUAL

ActiveX is not dedicated to any granularity level. However, it is more appropriate for lower granularity levels. Furthermore, ActiveX does not address the notion of "reuse" explicitly.

With respect to standardization of Business Objects, ActiveX is not involved in any standardization effort. Besides, it does not address categorization (of Business Objects).

ActiveX is, in principle, "stateless"[6]. Therefore, it has certain problems in realizing business software systems that require that the state is maintained.

6.3.3 JAVA

Java [JAVA], presented by Sun Microsystems, provides component technology in the form of Java Beans [JavaBeans] and Enterprise Java Beans [EJB]. In addition, Java Applets can be used as a special kind of component.

[6]It is possible to reconnect to an instantiation of a particular object through Persistent Intelligent Names.

6.3.3.1 ARCHITECTURAL ASPECTS

SYSTEM CONSTITUENTS

Java Beans can be used as client-side constituents. GUI components are well-known examples of Java Beans. As such, the User Interface layer and Presentation Logic layer can be realized using Java Beans.

Enterprise Java Beans (EJB) can be used as server-side constituents. As such, the Business Logic layer can be realized using EJB.

For the realization of the Information Logic layer, one can use technologies such as the Visual Business Sight Framework (VBSF). VBSF is an object-relational Java framework that makes it easy to store and retrieve Java objects in and from relational databases.

Furthermore, the Information Logic layer can use Java Database Connectivity (JDBC)[7].

For the Control Logic, EJB provides services including Transaction and Security.

COMMUNICATION MEANS

Interoperation among the Java constituents is based on Java Remote Method Invocation (RMI). RMI can make distribution transparent.

RMI does not support asynchronous communications.

6.3.3.2 CONSTRUCTION ASPECTS

TECHNICAL

Java is platform independent, and can be deployed on almost all of the existing platforms. However, due to this fact, the performance of this technology is not high.

Java Beans technology is based on an Object-Oriented Application Program Interface (API).

Enterprise Java Beans (EJB) is a component-container architecture specification for the development and deployment of distributed applications. It consists of:

[7] Java Database Connectivity (JDBC) is an Application Program Interface (API) specification for connecting programs written in Java to different databases.

- A server that provides run-time services.
- A container that provides a host environment for components and insulates them from the servers by acting as an intermediary layer, and helps servers in managing the services.
- Enterprise Beans, which construct components that embody (business) concepts and use the services of the servers through containers.

Java Applets are (small) application modules that can be used as Web page elements, and can be transformed from the remote Web server to the Web client.

CONCEPTUAL

Java is not dedicated to any granularity level. However, it is more appropriate for lower granularity levels. Furthermore, Java does not address the notion of "reuse" explicitly.

With respect to standardization of Business Objects, Java is not involved in any standardization effort.

With respect to categorization of Business Objects, in the terminology of Enterprise Java Beans, two categories of components (Business Objects) are addressed. These categories are Entity (Entity Business Objects) and Session (Process Business Objects). However, the only main difference between these two categories is their persistence capabilities. In principle, being persistent and belonging to either the Entity or the Process category of Business Objects are two orthogonal issues. That is, both the Entity Business Objects and the Process Business Objects can be persistent or not, and therefore combining these two criteria is not helpful.

With respect to the relationship between Business Objects and data sources, Java does not impose any restriction.

6.3.4 EXTENDED C++

Extended C++, presented by PowerBroker, is an Object-Oriented programming language that extends C++ in order to provide an integrated environment for the development and management of large-scale distributed systems.

6.3.4.1 ARCHITECTURAL ASPECTS

SYSTEM CONSTITUENTS

The User Interface layer, Presentation Logic layer, Business Logic layer and Information Logic layer can be realized using any conventional C++ programming language method.

For the Control Logic, Extended C++ provides services such as Naming, Life Cycle, Persistence and Event, through a class library and explicit service objects.

COMMUNICATIONS MEANS

Extended C++ allows direct manipulation of objects through Remote Method Call (RMC) arguments and call backs. Therefore, it offers a flexible communication means that provides scaling and event-driven real-time notification capabilities.

Since Extended C++ works at object level, objects can be moved around in the networks. This can provide the clients with location-independent services and makes it possible to provide load balancing and fail-over handling that are transparent to the client.

6.3.4.2 CONSTRUCTION ASPECTS

TECHNICAL

Extended C++ services are unified within an object model and are provided through a class library with uniform and simple Application Programming Interfaces. Services can be applied transparently, or can be called explicitly by instantiating the classes involved.

Additional distributed object services can be specified as C++ classes. The pre-processor parses the header file concerned and generates standard C++ code including a local interface to the remote object service.

Extended C++, in keeping with the spirit of C++, offers all of the known advantages of the Object-Oriented paradigm. For instance, it supports both multiple inheritance and polymorphism, and places no unnecessary restrictions on arguments and return values. Furthermore, built-in types and types inherited from the Extended C++ class library can be sent as arguments or return values. Arguments and return values may even encapsulate complex data structures, and can be passed by value.

CONCEPTUAL

Extended C++ is not dedicated to any granularity level. However, it is more appropriate for lower granularity levels. Furthermore, Extended C++ does not explicitly address the notion of "reuse".

With respect to standardization of Business Objects, Extended C++ is not involved in any standardization effort. Besides, it does not address categorization (of Business Objects).

6.4 SYSTEM ENVIRONMENTS

6.4.1 SANFRANCISCO

SanFrancisco (SF) [SF], presented by IBM, is a framework for creating business management applications, and is based on different layers, including Base, Common Business Objects (CBOs), Core Business Processes (CBPs) and Commercial applications.

6.4.1.1 ARCHITECTURAL ASPECTS

SYSTEM CONSTITUENTS

SanFrancisco supplies a Graphical User Interface (GUI) framework, including base classes for view and control of remote server business components, along with a full set of client area controls needed in different business solutions, and ActiveX components that can be used at the client side. As such, the User Interface layer and Presentation Logic layer can be realized using this GUI framework.

SanFrancisco supplies Common Business Objects (CBOs), including several domain-independent objects that can be used for most business applications, and Core Business Processes (CBPs), including basic structure and behaviour according to traditional business procedures. As such, the Business Logic layer can be realized using CBOs and CBPs.

Furthermore, SanFrancisco presents the Control Logic through the Foundation layer, which offers a comprehensive set of kernel services, including Naming, Persistence, Transaction and Security.

COMMUNICATIONS MEANS

SanFrancisco uses Java Remote Method Invocation (RMI) as the basis of its communication infrastructure, and has extended it to include support for areas such as server process management.

6.4.1.2 CONSTRUCTION ASPECTS

TECHNICAL

SanFrancisco is based on the Java technology. The server runs on IBM OS/400 and AIX, as well as on Windows NT and Unix platforms. It provides Java-based development tools, as well as specific tools, including an automated code generator.

Applications can be implemented using the basic classes and utilities offered by the Foundation layer, as well as using the components offered by the CBO and CBP layers.

CONCEPTUAL

SanFrancisco applications can be integrated with legacy applications using "calls", and with data sources through an "extended schema mapper" utility. Therefore, SanFrancisco complies with the notion of "reuse".

SanFrancisco deals with standardization of Business Objects through Common Business Objects (CBOs) and Core Business Processes (CBPs). CBO components are fine-grained objects that are abstract enough for providing the functionality needed for most business applications. CBP components are coarse-grained objects that are related to traditional business procedures, and maintain the functionality needed for specific business domains. The difference between the granularity levels of CBO and CBP components is large.

SanFrancisco deals with the changes through a specific concept called "command", which can embrace different (unchangeable) Business Objects and can be changed to deal with each specific situation. Another way in which SanFrancisco deals with the changes is based on a notion called "active company", which allows different applications to have different views on the same data.

With respect to categorization of Business Objects, SanFrancisco only distinguishes CBOs and CBPs.

SanFrancisco delivers skeletal applications covering areas such as ledgers, accounts payable and accounts receivable. Use of these generic frameworks can quickly yield business solutions with reduced development effort.

6.4.2 R/3

R/3 is a comprehensive set of integrated business applications, presented by Systems, Application and Products in data processing (SAP) [SAP], using the Client/Server architecture model for providing the ability to store, retrieve, analyse and process data in different ways for different business domains. R/3 replaced R/2, which is still in use.

6.4.2.1 ARCHITECTURAL ASPECTS

SYSTEM CONSTITUENTS

The User Interface layer, Presentation Logic layer, Business Logic layer and Information Logic layer can be realized by means of the variants of the Advanced Business Application Programming (ABAP) programming languages - such as ABAP/4 and ABAP Objects - using R/3 Business Objects.

One of the main ideas behind R/3 is to make it possible to use a common database for a comprehensive range of applications.

Furthermore, R/3 presents the Control Logic internally, and not externally. That is, the Control Logic is maintained for the R/3 Business Objects that exist within the system.

COMMUNICATION MEANS

R/3 uses CORBA, COM+ and Java Remote Method Invocation (RMI) as the basis for its communications infrastructure for accessing Business Objects. For communications between Business Objects, R/3 uses a message broker, based on Application Link Enabling (ALE).

The R/3 (persistent) Business Objects can be transparently accessed in distributed environments through a unique "identifier" assigned to them.

The R/3 Business Objects can also be accessed dynamically, that is, their interfaces can be discovered at run-time through a Dynamic Invocation Interface.

6.4.2.2 CONSTRUCTION ASPECTS

TECHNICAL

R/3 does not support the construction of Business Objects but only supports the use of them through their exposed interfaces. The R/3 Business Objects are presented through a Business Object Repository (BOR), and their operations are presented through Business Application Programming Interfaces (BAPIs).

For developing R/3 applications, BASIS offers a development environment. It consists of a set of middleware programs and tools that provide the underlying base, which enables applications that are seamlessly interoperable and portable across operating systems and databases. BASIS is based on the Client/Server architecture, and offers a relational database management system and a Graphical User Interface (GUI).

ABAP Objects, an extension to ABAP/4, is an Object-Oriented language that supports Object-Oriented concepts, such as inheritance and polymorphism.

R/3 runs on a number of platforms including Windows.

CONCEPTUAL

R/3 is principally based on the notion of "reuse" and aims to provide a background for the interoperation of constituents based on different standards and technologies, such as CORBA, COM+ and Java.

R/3 deals with standardization of Business Objects through Business Application Programming Interfaces (BAPIs), which define a business standard for communication between business applications.

R/3 considers Business Objects coarse-grained constituents that cannot be altered directly. New Business Objects can be created based on those offered by R/3 through inheritance. Moreover, R/3 does not address or apply categorization (of Business Objects).

In general, the R/3 Business Objects can be considered as ready-made solutions in the form of Business Object interfaces.

Chapter 7

CASE STUDY

To find a fault is easy; to do better may be difficult.

[Plutarch 46 AD - 120 AD]

Applying Business Objects for a real-world case can help us to elaborate on the subjects discussed in the previous chapters from a practical viewpoint.

This chapter presents an example case of using Business Objects in an enterprise as the organizational unit concerned. It discusses technical and theoretical issues of applying Business Objects in practice. The practical result of this case study is the addition of new functionality to an information system.

First the background of the case study is introduced. Then the system within which the case study is carried out is introduced, by explaining the structure, production, usage and theoretical aspects of the system. Subsequently the main subject of this case study is discussed. This includes a description of, and the requirements for a new part of the system, as well as the theoretical aspects and the realization of it. Finally, the conclusions are presented[1].

[1]The case study is presented in [van Groen 2000].

7.1 BACKGROUND

The case study was carried out in the company Professionals In Dienstverlening[2] (PID) in Amersfoort, the Netherlands[3], within the framework of the realization of an information system called Geïntegreerde Registratie- en Informatie-systeem PID[4] (GRIP). GRIP is an information system for the registration and administration of data about the employees of PID. It consists of several modules[5] and each module deals with a specific kind of data.

The case study concentrated on the construction of one of the GRIP modules, which deals with the registration and administration of lending data related to lending articles such as book, software and mobile phone. The module is called the "Lending Administration". During this case study the module was designed, implemented, tested and used.

The case study began with a study and analysis of the existing system, GRIP, including an investigation of the structure, production and usage of the system. For this purpose, we studied the existing documents about GRIP and tested the GRIP applications.

Then we interviewed the (intended) users of the Lending Administration module. Since the users were the only people who could provide us with information about the area where the module was going to be applied, the aim of these interviews were twofold. The interviews helped us to recognize the related real-world concepts, and at the same time it helped us to get acquainted with the user demands. In this way, we found out which articles could/should be lent, how they should be lent, who should have which permissions and how the users want to enter, change and read data.

Subsequently, based on the interviews, we designed the input panels. Then, using the feedback from users about the panels, we specified the data and

[2]The name "Professionals In Dienstverlening" means "professionals in provision of services".

[3]The company Professionals In Dienstverlening (PID) is a temporary employment agency that is dedicated to Web technology and embedded software systems. It offers services to companies in different domains such as telecommunication, banking and insurance.

[4]The name "Geïntegreerde Registratie- en Informatie-systeem PID" means "integrated registration and information system of PID".

[5]Here the word "module" does not (exactly) represent its specific meaning in Software Engineering, but stands for a certain part of a system with a specific functionality.

procedures concerned.

After realizing a prototype for a few lending articles and a number of functions, and getting feedback from users about it, we designed the detailed models. This led to the implementation of the Lending Administration module. Business Objects were our focal point during these steps.

Finally, we tested the module and wrote a user manual for it.

7.2 THE GRIP SYSTEM

7.2.1 STRUCTURE

GRIP is an integrated system consisting of several modules that uses a central database. It substitutes a few smaller systems in order to, among others, avoid information redundancy. In this system the concept "employee" plays a major role and data are mainly related to this concept. Some of the main modules of GRIP are:

- System Administration
- Employee Administration
- Project Administration
- Insurance Administration
- Connections Administration

Each module administrates a specific area of the internal activities of the company. They offer different applications for entering, changing and reading data. In some applications it is also possible to provide an overview of the whole data. The system administrator controls the access that users have through these applications to the company data.

The modules of GRIP consist of three (distinct) layers. These layers are:

- User applications

- Business data and procedures
- System data

The first layer, as the front end of the system, takes care of interaction with users. It consists of forms and other user interface constituents. For example, the applications of the Employee Administration module contain forms for entering and changing data about employees. The constituents of this layer were written in Visual Basic 6.0 under Windows 95/98.

The second layer, as the middle layer of the system, provides an abstraction means for accessing the company data in the database, so that the applications of the first layer do not have to get involved in the details of the used database(s) in the third layer. The constituents of this layer are objects, which are created in Visual Basic 6.0 under Windows 95/98, in the form of ActiveX components, and are grouped in Dynamic Link Library (DLL) modules. For example, in the group "System", there are "Application" and "Employee" Business Objects.

The third layer, as the back end of the system, takes care of storage and retrieval of data. It is a MS SQL server 6.5 under Windows NT, and it is accessed through Open Database Connectivity (ODBC). This layer also contains Stored Procedures (SPs).

This architecture resembles the Three-Layer Architecture, discussed in {3.2}. Besides, the middle layer consists of objects that not only resemble Business Objects but are also referred to by this term.

The layers make it possible to distribute the system over different locations. Accordingly, the applications of the first layer are located on the users' workstations, the Business Objects of the second layer are installed on a remote server that can be accessed by (the applications of) the first layer, and the database of the third layer is located in a central place.

Each module can use other modules in order to fulfil its tasks. This cooperation takes place through Business Objects. That is, Business Objects of each module can be (re-)used by other modules through their interfaces.

However, the existing Business Objects do not always provide the whole functionality required for new modules. In case an existing Business Object should be adapted in order to meet the (new) needs of the (new) modules, it should be designed, implemented and installed again.

Furthermore, the Business Objects use other kinds of objects such as the so-called Server objects and Read-Only (RO) objects to accomplish their tasks.

The Server objects load the Business objects and take care of their connection with the database through ODBC. For example, a Server object in the Project Administration module, Server_Project, contains the operation Load_Project, which returns an instance of the Project Business Object. In general, one Server object can load more than one Business Object.

The Read-Only (RO) objects are comparable with the records of a table of the database. These objects are instances of the same class and each of their attributes are related to one field in the corresponding table. In this way, each Business Object receives data from a table of the database; sending data back to the database takes place through the Business Object itself. For example, the Project Business Object receives data in the form of a collection of RO_Project objects. The RO objects are created by the same Business Object that uses them.

Figure 7.1 depicts a simplified object model showing the mutual relationships between an example GRIP Business Object and its related Server and RO objects.

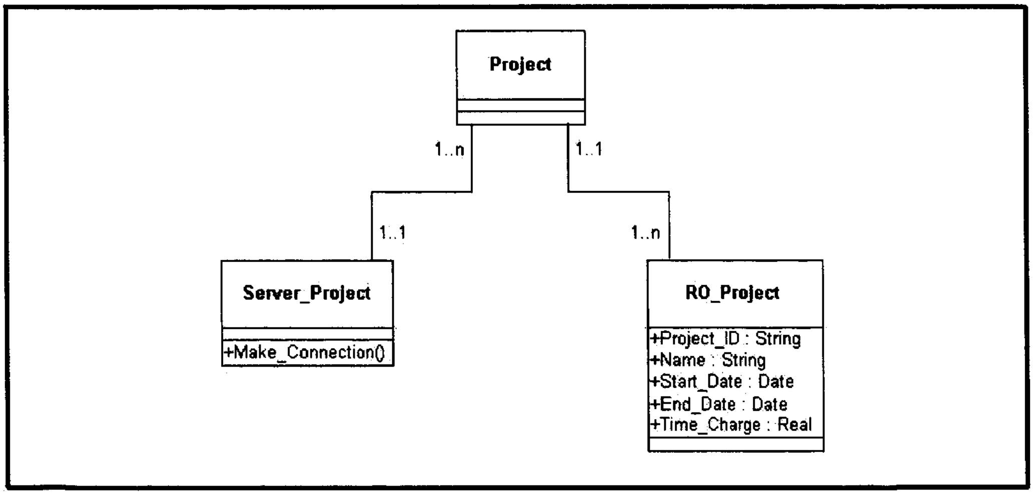

Figure 7.1: An example GRIP Business Object and its related Server and RO objects.

7.2.2 PRODUCTION

The production of GRIP takes place in phases. At the end of each phase a new release is delivered. Each new release extends the system so that users are provided with more options and can work with more information. This is realized either through the improvement of the existing modules or by adding new modules. The new modules take advantage of the functionality of the existing modules, such as those provided by the Business Objects and database entities. In this way, the existing modules are re-used and the system retains its uniform character.

Each release consists of, among others, installation programs and scripts. Using the installation programs, one can set up all of the implemented applications of the modules on a client and create Business Objects on the Business Object server. Using the scripts, one can create the needed tables and Stored Procedures in the database.

7.2.3 USAGE

Through the System Administration module, one can specify which employees can have access to which applications, by assigning combinations of user names and passwords and the name of applications.

To be able to use the system users should first run a standard client application called GRIPOffice. This application asks the user to enter his user name and password, and checks them with the access data existing in the System Administration module. If they are correct, the icons of the applications for which the user is qualified are shown. The user can then start an application by clicking on its icon. Starting an application leads to the running of an executable. There are different executables with respect to different access rights for each module. For example, if there are two groups that use (the applications of) the Project Administration module, there should be two (groups of) applications, such as Read_Projects and Administrate_Projects.

The same combinations of user names and passwords are also used by the ODBC connection and the SQL database for the evaluation of read and write rights.

7.2.4 THEORETICAL ASPECTS

Based on our studies about the GRIP system we derived the theoretical aspects related to the creation and usage of Business Objects within GRIP.

With respect to the possible "options" and "perspectives" for the creation and usage of Business Objects, discussed in {5.3}, these aspects can be itemized as follows:

- Legacy/New: the Business Objects are not inherited from legacy databases; their corresponding data is added as new tables to the database according to the needs of modules.
- Persistent/Temporal: the Business Objects are temporal, thus not persistent. They are instantiated and filled with data at the beginning of each session. Changes in their attributes do not directly result in changes in the database; one can only make these changes by calling a specific operation explicitly. This makes coordination easier.
- One/More: the Business Objects correspond to each concept as a whole, and not to each individual concept separately.
- Perspective: the Business Objects are created and used based on the Data perspective. That is, they are derived directly from the data entities. Therefore, the Business Objects are in fact Information Objects that contain Business Logic.

Furthermore, there is no inheritance among the Business Objects, and their relationships are shifted to the database. Therefore, there is no association among the Business Objects, although they have some similar attributes and operations, and they share some specific construction characteristics.

7.3 THE LENDING ADMINISTRATION MODULE

7.3.1 DESCRIPTION

In the area of lending administration, like the other areas of the internal activities of the company, the concept "employee" is the central point; the main goal is to obtain an overview of what an employee has borrowed. This should include an overview of what an employee has borrowed at a specific moment (lending overview), and what the employee has borrowed in the past (lending history).

There are two groups of users, namely administrators such as secretaries and system administrators, and managers such as office managers, sales managers, field managers and accountants. There should be a clear distinction between different (groups of) users, as the users can perform different functions.

Administrators should be able to administrate a specific kind of article and its lending, and should be able to acquire information and get overviews about the article and its lending. However, one person can perform the role of more than one administrator. For example, a secretary administrates books and mobile phones, and a system administrator administrates software and hardware.

Managers should be able to acquire information and get overviews of all of the lending articles and the employees who have borrowed those articles. They should not be able to change data.

Some of the articles are not, and do not need to be, registered. While one should be able to administrate lending of this kind of articles, there is no need to be able to get overviews of them.

For articles such as books and software there may be a number (of copies) of the same item, so that the same article can be lent more times. Besides, it should be possible to lend an article such as a book more than once to the same employee on the same day, although this may not happen very often.

Most of the time, when a mobile phone is lent, the device and the SIM card are handed over together. However, it should be possible to lend a SIM card without a device.

In brief, it should be possible for the administrators to enter, change and read information about one or more articles and their lending, and it should be possible for the managers to read information about all of the articles and their lending. Accordingly, the Lending Administration module should allow users to:

- Maintain lists of articles

- Inspect overviews of lists of articles
- Administrate lending of articles
- Monitor lending of articles

7.3.2 REQUIREMENTS

Reuse of Business Objects was a major requirement. On the one hand we had to use the existing Business Objects wherever possible for creating new Business Objects, and on the other hand we had to create new Business Objects so that they could be used by the other modules of the system (in the future).

Furthermore, the user interfaces of the Lending Administration module had to look like the user interfaces of the existing modules, so that users could learn to work with the new module quickly and easily. Besides, it was important that GRIP could retain its uniform look and feel.

Moreover, we had to use the existing hardware and software resources. With respect to the platform, programming language and database system, we had to adapt to the existing modules; we had to use MS Windows, Visual Basic and MS SQL.

7.3.3 THEORETICAL ASPECTS

For the creation and usage of Business Objects we tried to follow our own view, while complying with the requirements as far as possible. It should be noted that our starting point was a Three-Layer Architecture. Accordingly, we considered the possibilities with respect to the "options" and "perspectives", discussed in {5.3} as follows.

7.3.3.1 OPTIONS

LEGACY/NEW

We had to use the existing data in the database through the existing Business Objects. For the new concepts we could create our own Business Objects and add their corresponding data structures (tables) to the database.

However, as mentioned in {7.3.2}, we had to use the existing database. Therefore, our data had to be placed in a relational database.

PERSISTENT/TEMPORAL

The Business Objects did not need to preserve their state and outlive the sessions in which they are instantiated. Therefore, we did not have to deal with the following issues:

- Type and location of the storage place of (the states of) the Business Objects.
- Relationship between (the states of) the Business Objects and their underlying data in the database.
- Timing and access control of the updates of (the states of) the Business Objects and their underlying data in the database.

ONE/MORE

The Business Objects had to correspond to each concept as a whole. Therefore, the whole data could be extracted in one go from the database.

7.3.3.2 PERSPECTIVES

Having decided about the options, we had to choose our perspective. In the following we go through the perspectives and discuss them by using an example. The example is related to a representative part of our case, that is, lending books to employees.

DATA

The Data perspective suggests a one-to-one relationship between database entities and Business Objects, while database entities are the starting point. As mentioned in {7.2.4}, the existing modules of GRIP were all based on this perspective.

Following this perspective for our example leads us to the model shown in figure 7.2.

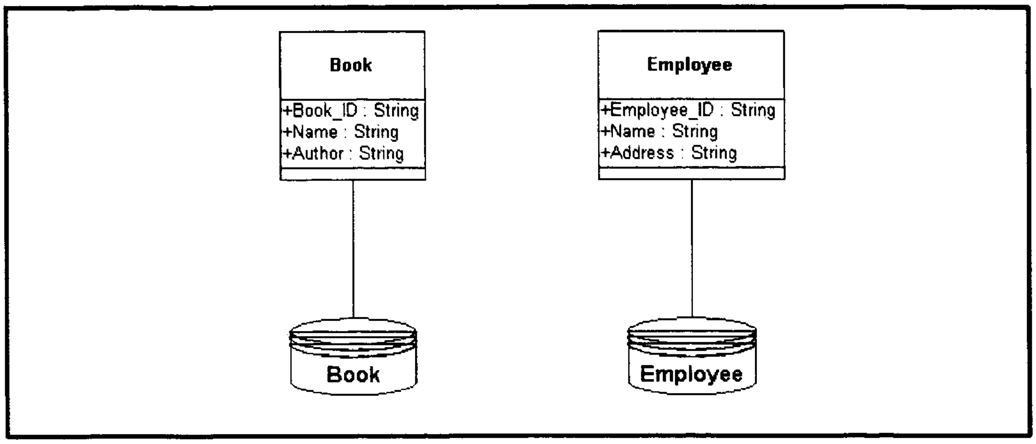

Figure 7.2: Business Objects of the example according to the Data perspective and their relationship with database entities.

In this model there are two Business Objects, or so-called Data Business Objects[6], corresponding to two database entities, namely Book and Employee. Each Business Object handles one database table and contains Business Logic concerned with the corresponding concept. Therefore, Business Logic concerned with two or more concepts cannot be directly supported.

Accordingly, the Business Objects can be reused for other applications (of other modules), but there is no direct support for the administration of lending books to employees in the Business Logic layer. Passing the required support to the front-end layer could result in inefficiency.

APPLICATION

The Application perspective suggests an association between application requirements and Business Objects.

Following this perspective for our example leads us to the model shown in figure 7.3.

[6]Do not confuse with Data Objects defined in {3.5}.

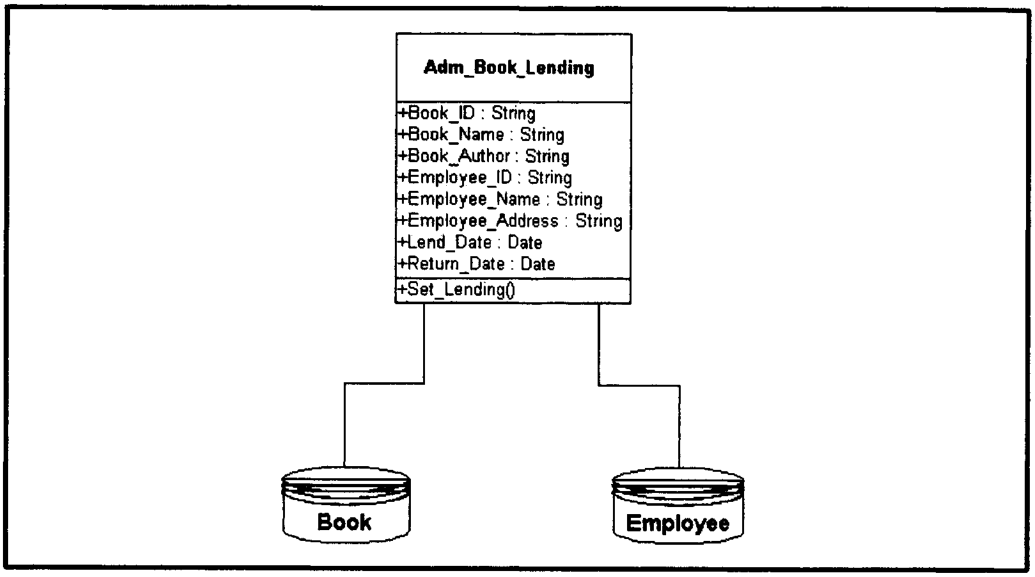

Figure 7.3: Business Object of the example according to the Application perspective and its relationship with database entities.

In this model there is one Business Object, or so-called Application Business Object, corresponding to the lending of books, namely Adm_Book_Lending. This Business Object exchanges data with two database tables and contains Business Logic concerned with both the concepts represented by the tables at the same time.

Accordingly, there is direct support for the administration of lending books to employees in the Business Logic layer. However, applications may need different functionalities, even when they use the same data. Therefore the usage of this Business Object would be restricted to the lending of books.

REAL WORLD

The Real-World perspective suggests a one-to-one relationship between real-world concepts and Business Objects.

Following this perspective for our example leads us to the model shown in figure 7.4.

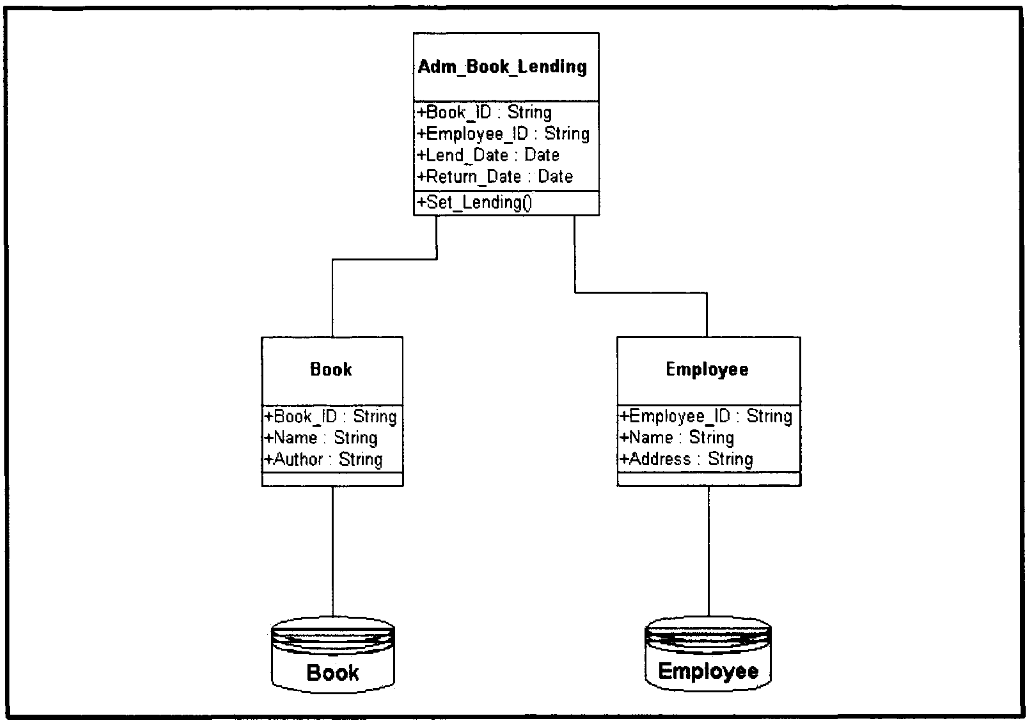

Figure 7.4: Business Objects of the example according to the Real-World perspective and their relationship with database entities.

In this model there are three Business Objects, corresponding to three real-world concepts; two Entity Business Objects, namely Book and Employee, and one Process Business Object, namely Adm_Book_Lending. The Entity Business Objects handle two database tables and contain Business Logic related to the corresponding concepts. The Process Business Object is associated with the two Entity Business Objects and contains Business Logic related to both of them at the same time.

Accordingly, there is support for the administration of lending books to employees through the Process Business Object in the Business Logic layer, and the Entity Business Objects can be reused for other applications (of other modules).

We can also consider the resulting model as a hybrid model - a combination of Data Business Objects resulting from the Data perspective and an Application Business Object resulting from the Application perspective - with two layers.

7.3.4 REALIZATION

In conformance with the description, requirements and theoretical aspects of the Lending Administration module, discussed in the preceding subsections, we realized the module.

7.3.4.1 DESIGN

The Business Object model consists of two types of Business Objects, namely Data and Application Business Objects, and is shown in figure 7.5.

Data corresponding to the Data Business Objects are maintained in the database. That is, we added two groups of tables to the database, namely Article and Lend_Article tables.

The Article tables represent articles themselves. Their fields correspond to the characteristics of articles and their ID.

The Lend_Article tables represent articles that are lent. Their fields correspond to the lending of articles such as the lending date and the return date. These tables have relationships with both the Article tables and the Employee table, in order to specify the lent item and the employee who has borrowed the item. This relationship is maintained through two Foreign Keys.

Therefore, these tables can keep data about the available and lent items of articles. For most of the articles which need to be registered, the whole data is there; one can get overviews and select items for lending. For the other articles the corresponding data do not turn up in the database until they are lent.

For the articles such as books and software of which there may be a number (of copies) of the same item, there is an additional attribute, namely Number. This attribute gives the number of the existing copies of the same item. In this way, the same item can be lent several times concurrently. Attribute "Available" shows the number of available copies. Besides, a unique key has been assigned to each record of the Lend_Article tables so that an article such as a book can be lent on the same day to the same employee more than once.

Since mobile phones and SIM cards can be lent separately, they are represented by separate tables.

The Overview Application Business Object uses the Adm_Article_Lending and Adm_Article Application Business Objects in order to present information about the articles and the employees who have borrowed those articles. The Data Business Objects have no mutual relationships.

All of the Application Business Objects use the Server_Lending Completion Object in order to connect to a Data Business Object or another Application Business Object. The application forms in the User Interface layer also use this Completion Object in order to connect to the Application Business Objects. It makes instances of the requested Business Objects and passes them on to the requester objects.

The Data Business Objects use Stored Procedures for reading and writing data. The Stored Procedures in turn exchange data with one or two database tables; two for most of the articles and one for those articles that do not need to be registered.

Furthermore, the Data Business Objects do not communicate with the table Employee, as shown in the Real-World perspective model (figure 7.4), but instead with the Employee Business Object, which already existed in the GRIP system.

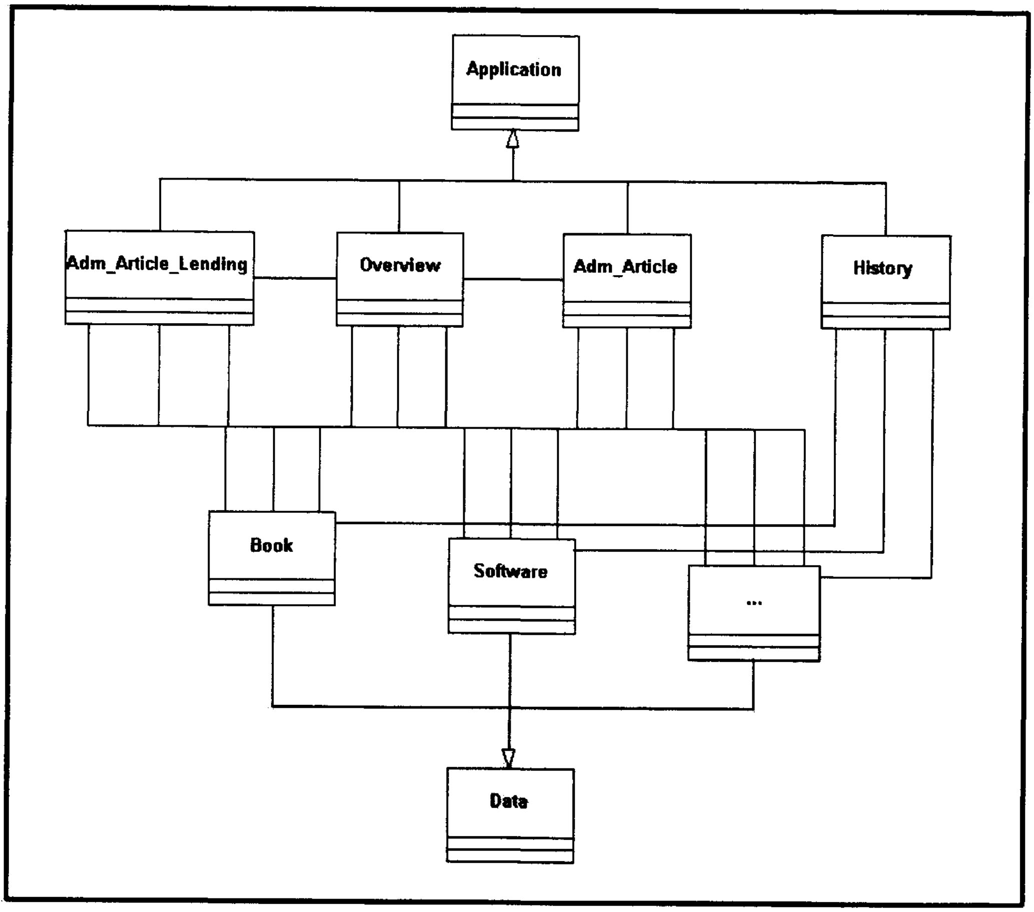

Figure 7.5: The Business Object model.

7.3.4.2 IMPLEMENTATION

The front-end layer presents a number of applications consisting of Graphical User Interface (GUI) constituents such as menus, forms and tables. These applications use the Application Business Objects of the middle layer and provide the functionality required by the administrators and managers for the administration of and getting overviews of articles and their lending. Each application has a main panel where a user can enter, change and read data. The applications are:

- Administrate_Articles: to register and maintain lists of articles. For example, to add a new purchased item to the corresponding list.

- Overview_Articles: to provide overviews of the lists of articles. For example, to see if an article is lent and which employee has borrowed the article.
- Administrate_Overview_Lending: to register and maintain lending of articles to employees, and to provide overviews of the lists of borrowed articles.
- Overview_History: to provide overviews of lists of all articles that an employee has ever borrowed.

The constituents of this layer are implemented using Visual Basic 6.0 under Windows 95/98, and are installed on the users' workstations. They provide users with a uniform interface for all applications, and are similar to those already offered by other modules.

The middle layer (mainly) consists of Application and Data Business Objects.

The constituents of this layer are implemented using Visual Basic 6.0 under Windows 95/98 in the form of ActiveX components, and are installed on a remote server.

At the beginning of each session, the Business Objects are instantiated and filled in, and at the end of the session they are deleted. Besides, the changes in the Business Objects do not directly result in changes in the related data; these only take place after calling of the specific operations.

The back-end layer consists of data in the form of tables and Stored Procedures set up in the system database, which is a MS SQL-server 6.5 under Windows NT, and is installed on a central server.

7.4 CONCLUSIONS

As discussed in {2.3.1}, there is no globally accepted definition of the term Business Objects. Therefore, different people who are familiar with this term often have different definitions of it. People in PID were not an exception. However, their familiarity with Business Objects and the fact that they had a background in using them was very helpful. Otherwise, it would have been extremely difficult (or even impossible) to perform this case study.

During this case study, we tried to comply with the demands imposed by the company as far as possible. For example, with respect to the programming language, database system and enabling technology, other choices could have led to better solutions. Anyhow, the company was satisfied with the result.

The Three-Layer Architecture of the existing system made it easier for us to study the system and its parts, while applying the same architecture for the new module made its development more convenient.

Furthermore, in the applied architecture, Data Business Objects are dependent on the Data Source layer, and Application Business Objects are dependent on the User Interface layer. We could use Business Objects that - in conformance with their definition, given in {2.3.2} - only contain Business Logic and shift the task of exchanging data with the Data Source layer to Information Objects and the task of exchanging data with the User Interface layer to Presentation Objects. This was one of the ideas that later resulted in the concept of Five-Layer Architecture, discussed in {3.3}.

Chapter 8

ILLUSTRATIVE EXAMPLES (I)

One machine can do the work of fifty ordinary men. No machine can do the work of one extraordinary man.

[Elbert Hubbard]

Applying Business Objects is closely related to the underlying architecture. This chapter illustrates the application of Business Objects within the framework of a Five-Layer Architecture, for two domains, namely the health-care domain and the banking domain.

The first section concentrates on the role of Information Objects and Event Business Objects, and elaborates on those requirements of the health-care domain that can benefit from these concepts.

The second section concentrates on the role of Presentation Objects, and addresses one of the requirements of the banking domain that can benefit from this concept.

8.1 THE HEALTH-CARE DOMAIN

Automation in the health-care domain, like in many other domains, underwent different stages. At present, automation designers face the challenge of having to find a way to enable and maintain the interoperation between different information systems, prevent redundancy of data and of procedures that are present in different systems, support reuse and deal with the dynamic work environment. This section discusses the use of Business Objects for complying with the general requirements of this domain.

The section begins by introducing automation in the health-care domain, through reviewing its history and general requirements. Then it describes how one can use Business Objects to comply with these requirements. In particular it describes how to deal with different data sources and to react to changes in the work environment by concentrating on two specific and representative example cases[1].

8.1.1 BACKGROUND

8.1.1.1 HISTORY

Automation in the health-care domain began with mainframe-based systems such as Hospital Information Systems (HISs). These systems were meant for supporting the administrative functions of Health-Care Institutes[2] (HCIs), such as billing, inventory and scheduling. However, this support remained at the (higher) enterprise levels. The special requirements of different departments of HCIs were a barrier to extending the mainframe-based information systems in order to provide support at the (lower) departmental levels.

Later, the minicomputer-based departmental information systems supported the (administrative) functions, similar to those of HISs, for different departments of HCIs, such as laboratory, radiology and pharmacy. In addition to information systems, departments also use special equipment, such as patient monitoring and imaging devices.

Unfortunately, (most of) the automation units[3] are stand-alone. They do not interoperate with the other automation units. This makes, mainly, the interoperation between the information systems of different departments within HCIs very difficult. Accordingly, the department-level information systems form distinct islands. This situation obstructs an enterprise level view not

[1]This section is based on the author's work in the Academical Medical Centre (AMC), Amsterdam, the Netherlands and the Reinier de Graaf Hospital, Delft, the Netherlands.

[2]This term is used to refer to all of the different organizations involved in health care, including different kinds of hospitals and clinics.

[3]Automation units refer to all automation systems and equipment in HCI.

only on data but also on data exchange routes and procedures. Obviously, in this way, interoperation among HCIs, as well as interoperation between HCIs and other enterprises that are active in other domains is not easy, if possible at all.

In order to make interoperation between systems possible, different standards have been devised and used. However, some of these standards are, in fact, imposed on HCIs as a side effect of the purchased products. Anyhow, the traditional standardization approaches have not adequately dealt with providing interoperability.

Many of the standards are related to "data exchange". These standards are merely conventions for exchanging data between systems with respect to a specific or general area. Digital Imaging and Communications in Medicine (DICOM) [DICOM], Medical Information Bus (MIB) [MIB] and Health Level 7 (HL7) [HL7] are examples of this category of standards. DICOM, devised by the medical imaging business, supports interfacing of various pieces of imaging equipment; MIB provides cross-vendor connectivity for clinical monitoring equipment; HL7 is based on a set of messages for data definition and transmission.

Following another approach, the Clinical Context Object Workgroup (CCOW) [CCOW 1998] aims to enable interoperability in the form of application control, based on the concept of "clinical contexts". The clinical contexts represent common state information including parameters that should uniformly affect the behaviour or operation of multiple health-care applications. The user should be able to manage the clinical contexts by establishing and modifying them, and applications should be able to coordinate their behaviour as the clinical contexts change.

Accordingly, it is difficult to take advantage of (a group of) existing systems, in order to realize a new functionality. In fact, in order to maintain the right interoperation between systems, one has to deal with many different platforms, languages, models, standards, etc.

Furthermore, one of the main difficulties of using the (ready-made) software systems in the health-care domain is "adjustment" of the system. It demands a lot of time and effort to adjust a system to the specific requirements of a HCI. This is due to the fact that each HCI not only has its own peculiarities with respect to resources and requirements, but also its own methods. Therefore, software systems should be so general that different HCIs can use them. As such, even when a software system is flexible enough, adjusting it may require a lot of time and effort; when a software system is not flexible enough, adjusting it may be more difficult than developing it from scratch.

Moreover, supporting of the administration procedures has been the main goal of many of the information systems in the health-care domain. Therefore, these information systems have been developed based on concepts that are not appropriate for purposes other than administration support. Consequently, the administration-oriented information systems cannot be extended to comprehend support for clinical procedures, such as treatment of patients as well.

An important issue in this respect is dealing with what is called "Patient Logistics". This term represents a large number of (interrelated) activities related to examination, treatment, hospitalisation, etc. of patients.

At present there are a number of systems that can only help in consulting dispersed information with respect to personnel work schedules, (laboratory) equipment, etc. Still the main part of the work takes place through personal contact (oral, telephonic, email, etc.).

In order to support Patient Logistics the information system concerned should get involved in the whole procedure from the time that a patient arrives at the hospital until the time that the patient leaves the hospital. For instance, it should also include issues like "waiting list" and "release policy".

What makes support for Patient Logistics more difficult is the high dynamism of the procedures and the continuous change in the work environment. For example, patients can always reject an appointment and ask for a new one. This makes planning extra difficult.

Together, the above-mentioned problems with information systems in the health-care domain have resulted in a situation with the following characteristics:

- inability or difficulty of interoperation between different systems,
- redundancy of data in different systems,
- redundancy of procedures in different systems,
- lack of reuse and

- inflexibility of procedures in reacting to changes in the work environment.

Besides, the health-care domain has a number of distinctive characteristics that should be taken into consideration, including:

- high sensitivity of data,
- high complexity of procedures,
- huge diversity of systems,
- low operating budgets for automation and
- little or *hopefully* no intention of making profit.

8.1.1.2 GENERAL REQUIREMENTS

To improve automation in the health-care domain, one should take the described situation into consideration, and should deal with the mentioned characteristics.

All in all, HCIs require affordable, flexible, distributed and heterogeneous information systems that are able to interoperate seamlessly and securely across a variety of organizations, including hospitals and (outpatient) clinics, and their different departments. The systems should, among other things, support HCIs in managing, organizing and scheduling different activities involved in the treatment and hospitalisation of patients, and hence serve different kinds of users, including physicians, nurses and employees, while maintaining authorization and guaranteeing privacy. Furthermore, the systems should be based on the existing resources; it should use and improve the hospital automation infrastructure efficiently and reliably.

Accordingly, the main general requirements for information systems in the health-care domain can be summarized as follows:

- maintain (secure) interoperation between different departments of HCIs, among different HCIs[4] (and between HCIs and other enterprises active in other domains),

[4]Each HCI should preserve the ownership of its own data. This makes centralization of data undesirable, if possible at all.

- maintain flexibility for information systems, in order to be able to apply them under different circumstances and changing work environments,
- support administrative, as well as clinical procedures,
- maintain authorization of data access and manipulation, without any restrictions with respect to time, location, platform, etc. and
- guarantee privacy of patients.

8.1.2 APPLYING BUSINESS OBJECTS

Business Objects can be used to build an information system for the health-care domain that meets all the requirements, in particular those related to dealing with different data sources and reacting to changes in the work environment. In the following, the benefits of applying Business Objects for these purposes are discussed and shown through two representative example cases.

8.1.2.1 RATIONALE

The main characteristics of Business Objects, as defined in {2.3.2}, and the Multi-Tier Architecture, as defined in {3.4}, make them appropriate for the outlined requirements for an information system of the health-care domain.

STANDARDS

The existing standards in the health-care domain are mainly based on data formats and control information. Business Objects are based on the real-world concepts of the health-care domain, and reflect the changes appropriately in an organized way, with respect to organizational units concerned. Therefore, Business Objects provide a better basis for interoperation than those based on data or applications.

Applying the notion of Business Objects results in a common model that comprises data and procedures of the health-care domain. In this way, those concepts that are basically common in the whole domain do not have to be repeated for each separate organizational unit such as "department" and hence redundancy can be eliminated. In addition, this makes interoperation between applications based on Business Objects easier, as they either use the same Business Objects or use different Business Objects whose (direct or indirect) relationships are always maintained to reflect the real world.

Furthermore, we can "implement" the same "interfaces" in different ways, and have the Business Objects related to the more specific concepts "inherit" from those related to the more general concepts. In this way, we can deal with the specific requirements and at the same time preserve interoperability. In addition, the "encapsulation" of data through Business Objects can help us to protect data, maintain data ownership and provide controlled access to the (critical) data.

However, to realize the new enterprise-level systems, we should deal with and take advantage of the existing departmental-levels systems as much as possible. In fact, the legacy systems present and support many fundamental procedures. Consequently, the existing standards for data exchange have also become a *real-world concept*, and one should also take them into consideration when one wants to realize systems based on the concept of Business Objects.

In principle, using (standard) Business Objects in accordance with the Multi-Tier Architecture is orthogonal to the use of other standards. That means that one can use Business Objects and still apply different standards for different purposes. For instance, in the context of the Multi-Tier Architecture, the Information Logic layer can exchange data between the Business Logic constituents, the Business Objects, according to different standards. In this way, the existing standards can co-exist with the standard Business Objects.

PATIENT LOGISTICS

Support for Patient Logistics mainly involves planning and plan adjustment, and in broad outlines should take the following factors into consideration:

- Human resources such as physicians and nurses,
- Material resources such as workplaces and equipment and
- Data such as those registered on the medical records of patients.

While planning (mainly) copes with the (relatively) static aspects of these factors, plan adjustment should deal with their dynamic aspects.

In fact, here we have to deal with the "supply and demand" problem between the existing resources on the one hand and the patients (and their treatment requirements in its general sense) on the other hand. The responsible personnel should always be able to get an overview of the existing resources, order (new) resources and adapt planning of resources (to the current situation). It should also be possible to make ad hoc modifications. The result of changes should then be propagated to everyone involved.

An important issue in plan adjustment is the capability of reacting to the "real-time" data such as presence/absence and availability/unavailability of resources, as well as patients. For example, in case of an equipment defect, patients should be informed and new appointments should be made.

Accordingly, in order to be able to support Patient Logistics we need to have an integrated and flexible system which can deal with both clinical and administration procedures at the same time. Systems based on Business Objects can fulfil these demands.

SOFTWARE DEVELOPMENT

As Business Objects are not tightly linked to the data they represent, one can develop (new) applications, or change (existing) applications, without having to deal with the underlying databases. In addition, the validity of (common) data, which is presented through Business Objects and is used by different applications, is also always guaranteed. This is of crucial importance in HCI environments, where many different databases are used.

Moreover, one can reuse the legacy systems through wrapping them by Business Objects or their underlying objects. This is also very important in HCI environments, where many different systems are in use.

Furthermore, for realizing Business Objects and the related applications, one can use the existing enabling technologies and system environments that are (specifically) intended for this purpose. Besides, the ready-made Business Objects presented by third parties can also be evaluated, adjusted and used.

As Business Objects are organized and arranged according to the categories they belong to, they can easily be searched and located. In addition, changing Business Objects in accordance with changes in their real-world counterparts (with respect to the organizational unit concerned) can be accomplished at one place and be made available for all applications that use them.

REASONS TO APPLY BUSINESS OBJECTS

In general, the main reasons for applying Business Objects in health-care-domain information systems can be summarized as follows. Business Objects can help us to:

- integrate data in different systems,
- integrate procedures in different systems,
- provide a general view on data and procedures,
- prevent the redundancy of data,
- prevent the redundancy of procedures,
- make the extension of systems and the addition of new applications easier; this can help us to support different kinds of users, including physicians, nurses and employees that are active in different areas, such as diagnosis, hospitalisation and accounting,
- reuse existing systems,
- provide conceptual and technical flexibility for creation, distribution and usage,
- take advantage of enabling technologies and system environments,
- use off-the-shelf components,
- provide a better basis for cooperation with other HCIs and
- provide the possibility for interoperation with systems of enterprises active in other domains, for instance in order to manage purchases of a HCI through interaction between the information systems of the HCI and vendors of the health-care materials.

CORBAmed [CORBAmed 1998], the Domain Task Force of Object Management Group (OMG) in the health-care domain, aims to offer the mentioned advantages based on the notion of Business Objects.

CORBAmed intends to provide a model of the health-care domain, including concepts such as "Person", "Institution", "Ordering", "Tracking", "Scheduling" and "Financial Services". CORBAmed has already been active in different areas, including:

- Electronic Patient Record (EPR)
- Clinical Image Access Service (CIAS)
- Clinical Observation Access Service (COAS)
- Patient Identification Services (PIDS)
- Health care Resource Access Control (HRAC)

8.1.2.2 EXAMPLE CASES

In broad outlines, the treatment of patients in the health-care domain is carried out by general practitioners, hospitals and complementary care organizations. Figure 8.1 shows a simplified view of this process in the Netherlands. Each of the following examples discusses specific aspects of this process.

The first example concentrates on dealing with different data sources, and the second example concentrates on reacting to changes in the work environment.

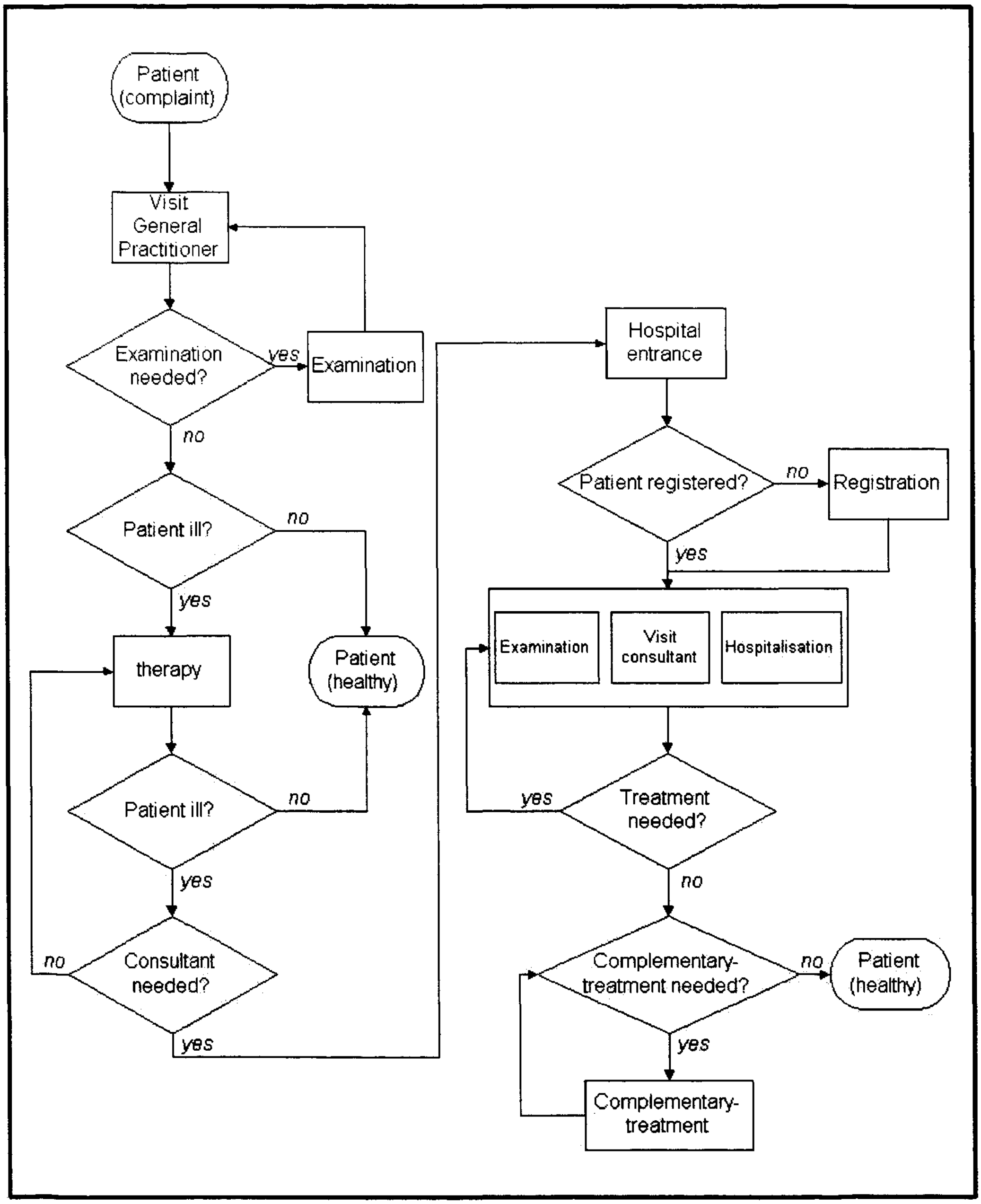

Figure 8.1: A simplified view of the health-care process in the Netherlands.

FIRST EXAMPLE

The "general internal" department of the outpatient clinic deals with a wide range of diseases and patients, and hence is involved in different kinds of data and procedures.

This department accepts the patients who are sent by general practitioners or consultants from other departments. The secretary of the department arranges the appointments for the consulting hours of the consultants of the department.

For the consulting hours, it must be checked whether the appointments have not been cancelled or changed, and if all the patient's documents that are needed, like laboratory results and x-rays, are ready. Before each consulting session, all of the existing documents related to the patient concerned should be available to the consultant.

During the consulting sessions, different types of information about the patient are acquired through asking questions, examination, etc. and subsequently a diagnosis is made. The diagnosis may result in medication, hospitalisation, etc. In some cases, the consultant may need complementary information before making the diagnosis.

After each consulting session the consultant provides a report about the patient, including (new) information. All the documents about the patient, including the mentioned report, should be collected and added to the patient's file.

The consultants continuously consult with other consultants, general practitioners, assistants, nurses, etc. through different means, including special forms. Besides, the consultants supervise the activities of their assistants.

The outpatient clinic in general and the "general internal" department in particular use different systems for different purposes. These systems are not able to interoperate; they include overlapping data and do not cover all desired data and functions.

The "general internal" department uses two legacy relational database systems and intends to use a new relational database system for its documents.

Business Object Models

In order to provide a basis for complying with the requirements of the example case, we should recognize the real-world concepts of this area, as the organizational unit concerned, and should realize Business Objects for these concepts.

Figure 8.2 shows the Entity Business Object model of the example case. These Business Objects mainly represent data related to this area.

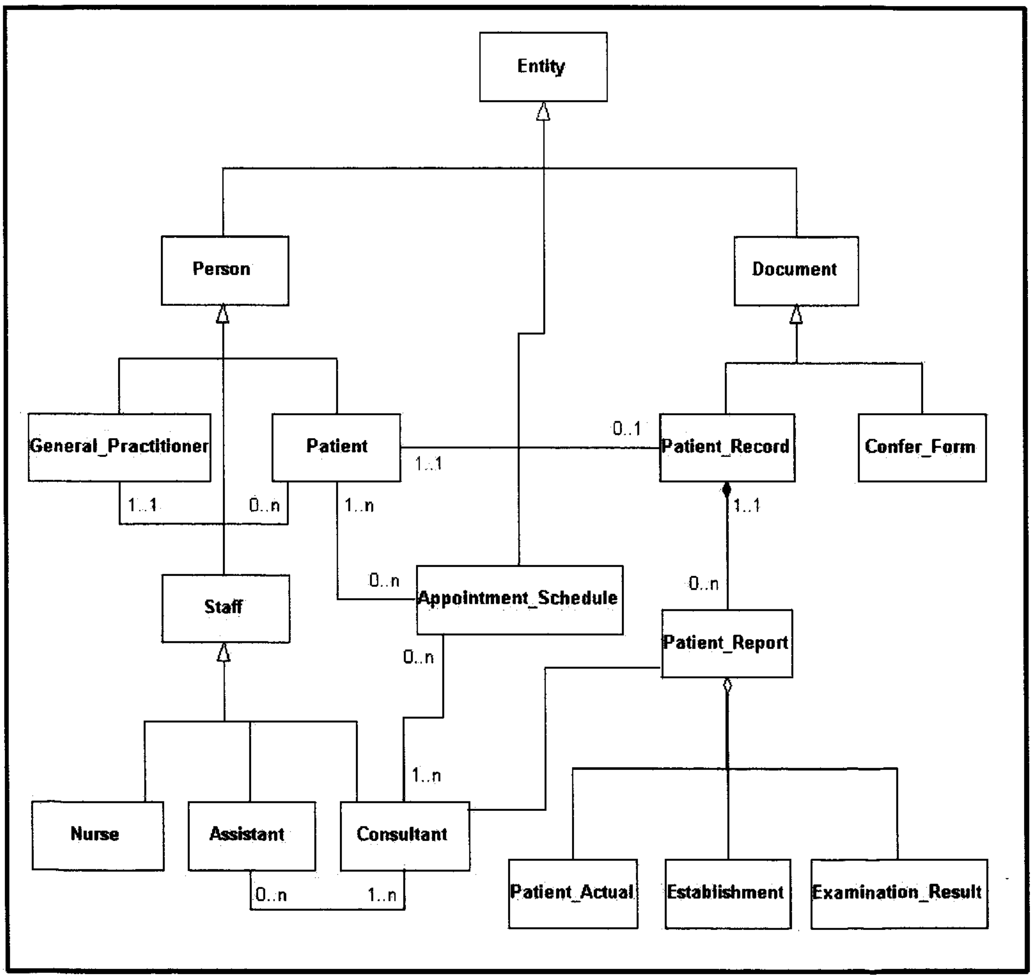

Figure 8.2: The Entity Business Object model of the first example case.

Figure 8.3 shows the Process Business Object model of the example case.

These Business Objects mainly represent procedures related to this area.

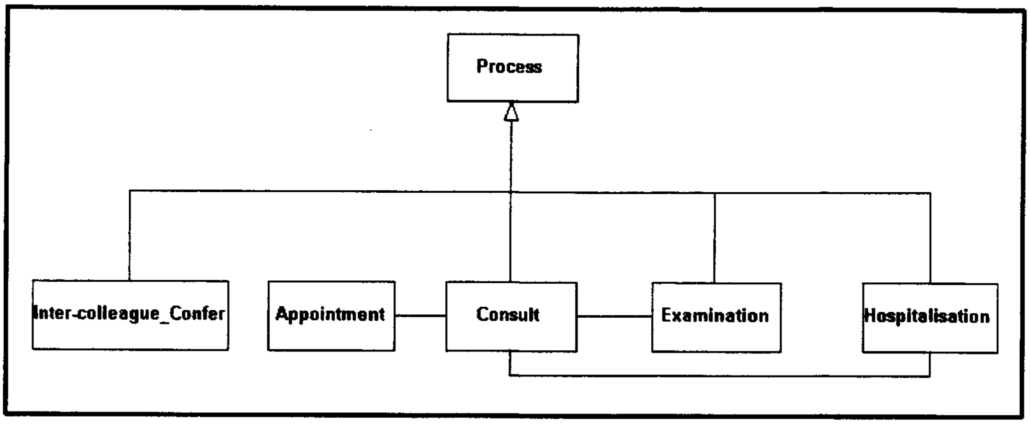

Figure 8.3: The Process Business Object model of the first example case.

Figure 8.4 shows the Event Business Object model of the example case. These Business Objects represent the main events that initiate procedures related to this area.

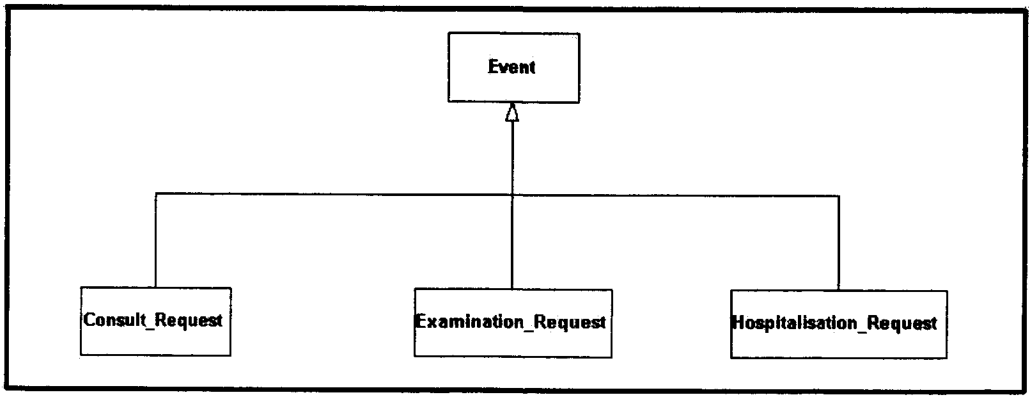

Figure 8.4: The Event Business Object model of the first example case.

The shown Business Objects can be applied for both administration and clinical procedures, and can be used and reused for different applications. Furthermore, they can be modified and extended to deal with new requirements.

Dealing With Different Data Sources

The main purpose of applying Business Objects to this area is to deal with different data sources in an effective and efficient way. This can be demonstrated by concentrating on a specific part of the Business Object model that focuses on "consult".

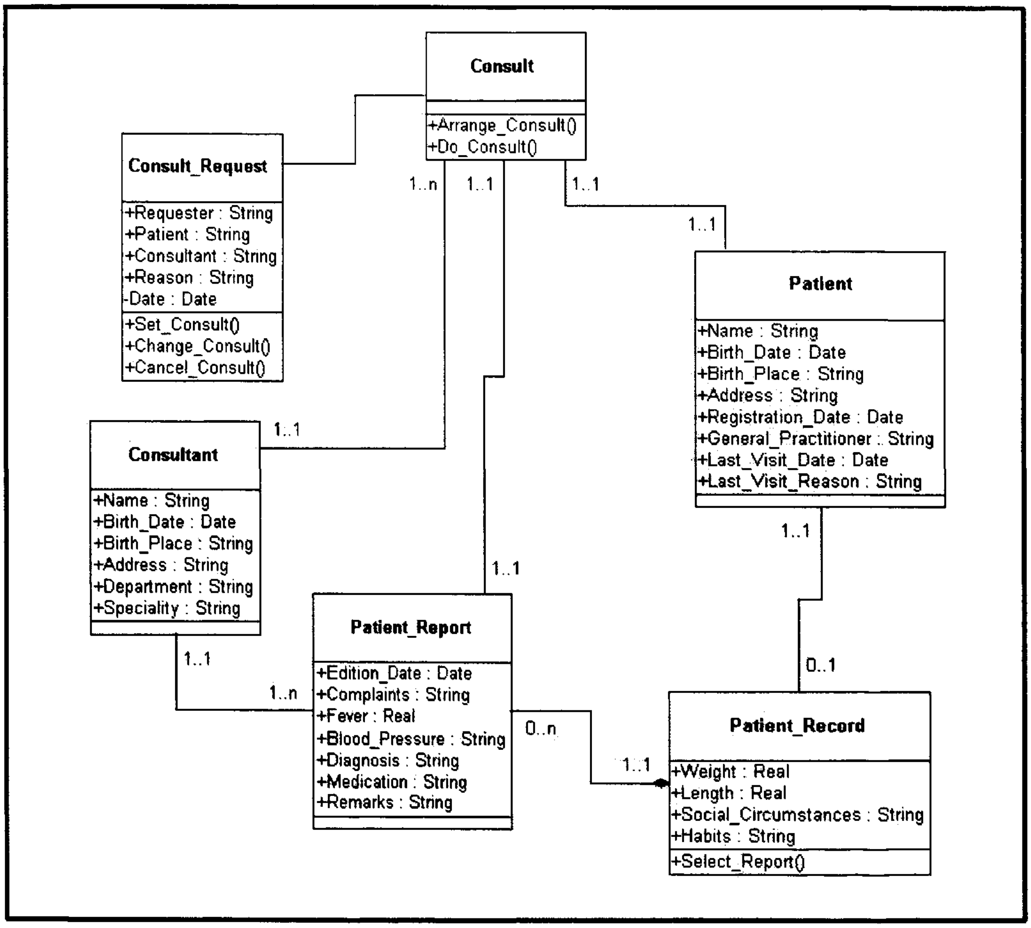

Figure 8.5: A part of the Business Object model of the first example case.

Figure 8.5 shows a part of the Business Object model of the example case that includes a number of Entity Business Objects. These Business Objects represent data distributed over different data sources. Therefore, the (Information Objects of the) Information Logic layer should store and retrieve data related to these Business Objects with no concern about the underlying data sources.

In general, for each Business Object there is an Information Object that stores and retrieves its data. This Information Object knows the exact location of data, and accomplishes the required conversions and transformations. Furthermore, the relationship between Business Objects and data sources also depends on a number of aspects discussed in {5.3}.

In the following, the relationship between the "Patient" Entity Business Object and the data sources concerned is explained.

"Patient" has a number of attributes, including "Name", "Birth_Date", "Birth_Place", "Address", "Registration_date", "General_Practitioner", "Last_Visit_Date" and "Last_Visit_Reason". The first five attributes are stored in a legacy relational database that is intended for administrative purposes. One can only change data in and add data to this database through native applications. Other systems can only read data stored in this database. The last three attributes are stored in a legacy relational database that is intended for clinical purposes.

During each session, when a specific "Patient" is called, the system searches for the instance concerned. If it does not find the instance, it creates a new instance. During the creation, the (only) instance of the related Information Object, "F_Patient", is called. This instance, in turn, searches the data sources concerned. It searches for two different tables in two different relational database systems, in order to find the related records. If such records exist, the "F_Patient" instance collects data and passes it to the "Patient" instance, otherwise the corresponding exception is thrown.

The explained procedure is shown in figure 8.6.

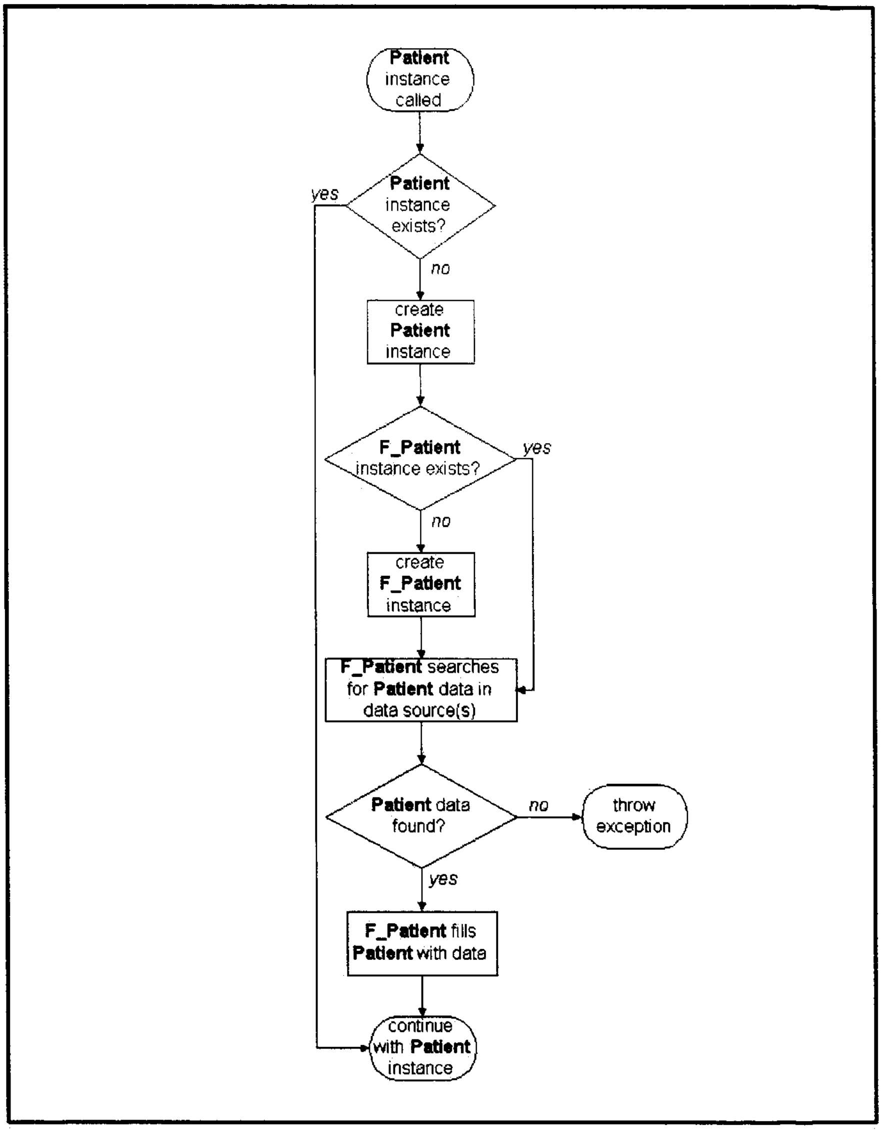
Patient instance called
Patient instance exists?
yes
no
create Patient instance
F_Patient instance exists?
yes
no
create F_Patient instance
F_Patient searches for Patient data in data source(s)
Patient data found?
no
throw exception
yes
F_Patient fills Patient with data
continue with Patient instance

Figure 8.6: Flow of actions for using a specific instance of the "Patient" Business Object (data) through the corresponding Information Object.

Furthermore, "Patient_Record" has a number of attributes, including "Weight", "Length", "Social_Circumstances" and "Habits". These attributes are stored in a legacy relational database that is intended for clinical purposes.

One can only add data to this database through native applications. Other systems can read and change data stored in this database.

During each session, when a specific "Patient_Report" is called, the system searches for the instance concerned. If it doesn't find the instance, it creates a new instance. During the creation, the (only) instance of the related Information Object, "F_Patient_Record", is called. This instance, in turn, searches the data source concerned. It searches for a table in a relational database system, in order to find the related record. If such record exists, the "F_Patient_Record" instance collects data and passes it to the "Patient_Record" instance, otherwise the corresponding exception is thrown. At the end of the session, existing instances of "Patient_Record" are written back to the database through the "F_Patient_Record" instance. However, during the session, at specified points in time the instances are stored in the database as well.

"Patient_Report" has a number of attributes, including "Edition_Date", "Complaints", "Fever", "Blood_Pressure", "Diagnosis", "Medication" and "Remarks". These attributes are stored in a new relational database that is intended for clinical purposes. One can only change data in and add data to this database through the related Business Objects. Other systems can only read data stored in this database.

During each session, when a specific "Patient_Report" is called, the system searches for the instance concerned. If it does not find the instance, it creates a new instance. During the creation, the (only) instance of the related Information Object "F_Patient_Report" is called. This instance, in turn, searches the data source concerned. It searches for a table in a relational database system, in order to find the related record. If such record exists, the "F_Patient_Report" instance collects data and passes it to the "Patient_Report" instance, otherwise the corresponding exception is thrown. When a new "Patient_Report" must be created, the system creates a new instance. At the end of the session, existing instances of "Patient_Report" are written back to the database through the "F_Patient_Report" instance. For new instances, a new entry in the database is created. However, during the session, at specified points in time the instances are stored in the database as well.

"Consultant" has almost the same story as "Patient". "Consult" and "Consult_Request" are of temporary nature and thus not persistent.

SECOND EXAMPLE

The "opthalmology" department of the outpatient clinic manages appointments telephonically or directly, and registers them in a file.

At the beginning of each workday the presence of consultants is checked. In case a consultant is not available, his appointments must be cancelled and new appointments should be set. In some cases it is possible to hand the appointments to other consultants. It can also happen that a consultant leaves the clinic during the working hours for a specific or unknown period of time. This can also have consequences for the appointments.

If a patient wants to change or cancel his appointment, the freed time slot should be made available for setting new appointments. The time slot should then be used as optimally as possible. In general, when everything runs according to the agenda, patients should check in with the clinic at least a quarter of an hour before their appointment. If a patient does not show up on time, his appointment may be changed. In that case, his time slot will be given to another patient who is present.

Business Object Models

The target environment of this example is very similar to the environment of the previous example. Therefore, most of the real-world concepts of this example are the same as those presented in the previous example. Accordingly, the Business Objects of the previous example can be used for this example as well.

The additional concepts of this example are those related to the main events with respect to the availability of consultant and patient. They can be presented by two Event Business Objects, namely Consultant_Availability and Patient_Availability. Figure 8.7 shows the Business Object model of this example case.

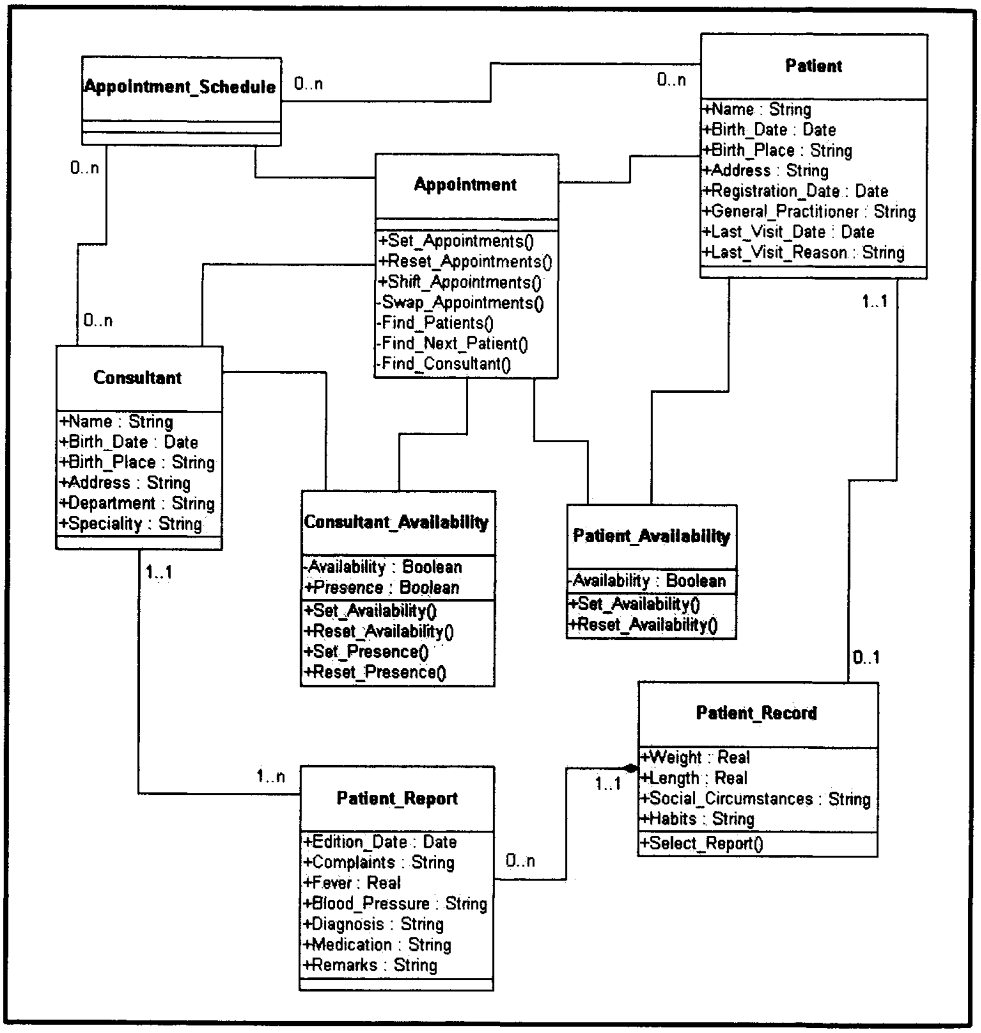

Figure 8.7: The Business Object model of the second example case.

Reacting To Changes In The Work Environment

A main purpose of applying Business Objects to this area is to react to changes in the work environment. This can be demonstrated by going over a number of scenarios with respect to the availability of consultant and patient.

- Consultant not available the whole day: Generally, at the beginning of the workday the Presence and Availability attributes of the Consultant_Availability Event Business Object of the consultant concerned are set. If it becomes known that the consultant will not be present, his Presence Attribute is reset. This change triggers the Reset_Presence operation of the Consultant_Availability Event Business Object. This operation in turn triggers the Reset_Appointments operation of the Appointment Process Business Object, in order to cancel the appointments of the consultant. This operation then finds all patients that have an appointment with the consultant concerned on that day. It then tries to find free time slots of the other appropriate consultants on the same day for the patients. The appropriateness of a consultant for a patient is derived from the corresponding patient record. In any case the result is registered in the involved Entity Business Objects and can be used for further steps, for example to notify the patients.

- Consultant not on time: If the consultant does not come before a specific amount of time, the Availability attribute of his corresponding Consultant_Availability Event Business Object is reset. This change triggers the Reset_Availability operation of the Consultant_Availability Event Business Object. This operation in turn triggers the Shift_-Appointments operation of the Appointment Process Business Object, in order to shift the appointments of the consultant for a specific amount of time. This action continues for a certain number of times for specified time intervals. If during this period the consultant arrives, his Availability attribute is set. This change triggers the Set_Availability operation of the Consultant_Availability Event Business Object, and stops the Reset_Availability operation from triggering the Appointment Process Business Object. Otherwise, the Presence Attribute of the Consultant_Availability Event Business Object is reset. This change triggers the Reset_Presence operation of the Consultant_Availability Event Business Object, and it acts the same as in the first scenario.

- Consultant not available for a while: If during the workday the consultant leaves the clinic for any reason, the approach is similar to that described in the second scenario. The Availability attribute of his corresponding Consultant_Availability Event Business Object is reset, and so on.

- Patient not available: If the patient checks in on time, the Availability attribute of his corresponding Patient_Availability Event Business Object is set. This attribute is checked after a specific amount of time. If it is set, the Set_Availability, and otherwise the Reset_Availability operation of his Patient_Availability Event Business Object is triggered. The Reset_Availability operation in turn triggers the Shift_Appointments operation of the Appointment Process Business Object, in order to shift the appointment of the patient to a later time on the same day. This operation then finds the next patient who has an appointment with the same consultant on that day, and has already checked in. The operation then swaps the appointment time slots of the two patients. In case the patient shows up for his (new) appointment, his Availability attribute is set. This change triggers the Set_Availability operation of the Patient_Availability Event Business Object and stops the Reset_Availability operation from triggering the Process Business Object. Otherwise the same procedure will be repeated for a certain number of times for specified time intervals, and subsequently the Reset_Availability operation of the Patient_Availability Event Business Object triggers the Reset_Appointments operation of the Appointment Process Business Object, in order to cancel the patient's appointment and assign his time slot to another patient. This is registered in the involved Entity Business Objects and can be used for further steps.

In this way, Event Business Objects can play an important role in reacting to changes in the work environment.

THE MAIN ADVANTAGES

Figure 8.8 shows an example architecture of a health-care information system in a hospital. Each hospital unit is connected to the system through a workstation which presents the User Interface layer. The clinical and administration applications are installed on these workstations. The Presentation Logic can exist on the same workstations or on other locations accessible for two or more workstations. The Business Logic layer is placed in a central location. It can exchange data with different data sources through the Information Logic layer. This layer can exist in the same location as the Business

Logic layer. The legacy information systems such as Hospital Information System (HIS) can be dealt with as legacy databases.

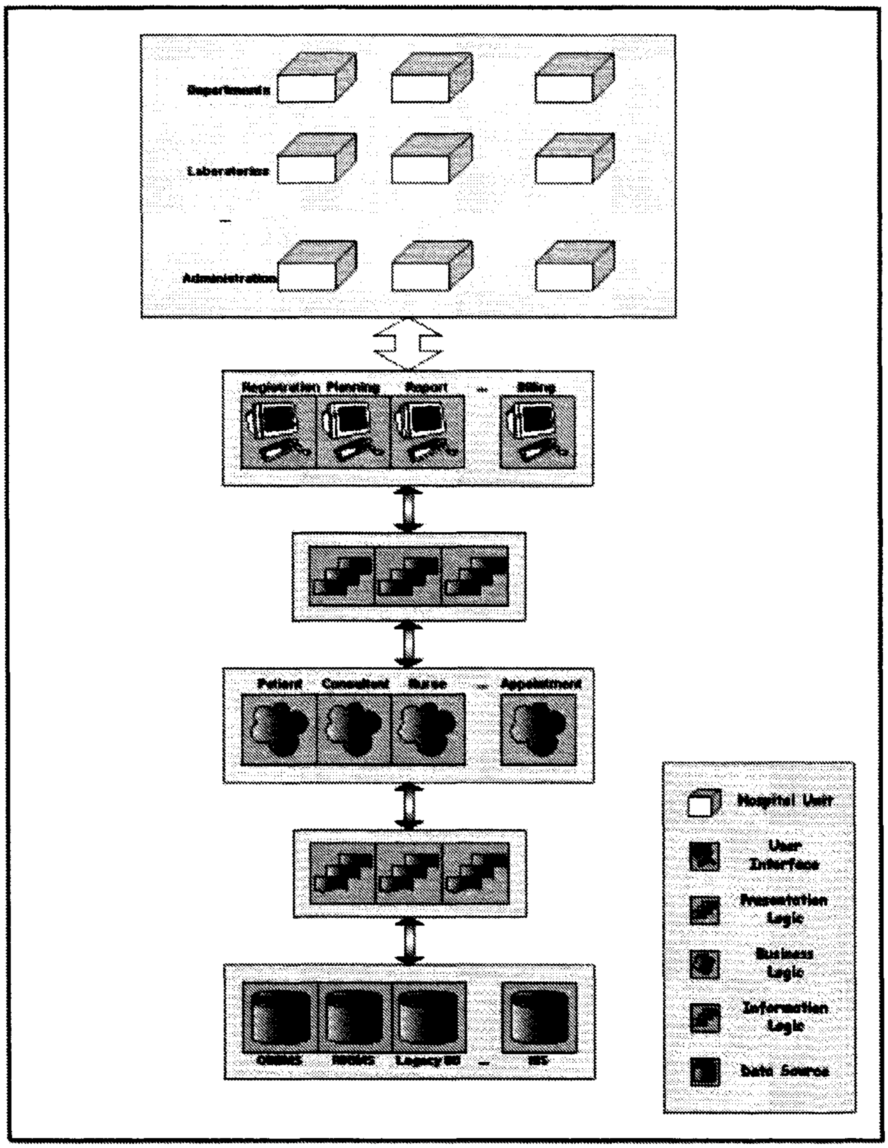

Figure 8.8: An example architecture of a health-care information system in a hospital.

The presented examples can be observed in the context of this architecture. Since the basic administration functions of different departments of the out-patient clinic are almost the same, issues and solutions discussed for one department are valid for and can be applied to other departments.

As the examples show, applying Business Objects and the Multi-Tier Architecture can mainly help us to:

- provide a general view on data,
- prevent redundancy of data,
- recognize the missing data,
- reuse the existing data sources,
- provide the missing data and
- react to the changes in the working environment.

The (Entity) Business Objects represent the data distributed over the databases. Accordingly, the "general internal" department can have a general view on its data through these Business Objects. For example, the patient data, which exists in two relational databases, is represented through "Patient".

It would be possible for some of the attributes, e.g. "Address" of "Patient" (patient's address) to exist in both databases. This would be detected during the development of the Information Objects concerned, e.g. "F_Patient", and the desired policy could be enforced through them. Anyhow, the Business Objects concerned would not have to get involved in this problem.

Moreover, as Business Objects are based on the real-world concepts (their real-world counterparts), the missing data can be spotted and added to system in an organized manner that makes them available and reusable for the whole system. For example, in order to be able to comply with (the changes in) the real world, due to administration reasons, the "general internal" department would have to know about the patient's occupation. This would affect the creation of the "Patient" Entity Business Object, and consequently the related attribute would be added to (one of) the underlying databases.

In addition, there will be no (new) redundant data in future, as the (new) data will be added based on Business Objects.

All applications - the existing applications and the new ones - have a well-defined place in the process flow and use the Business Objects of the Business Logic layer as a communication means. This maintains the integrity of the system.

Business Objects encompass the Business Logic with respect to the ordinary, as well as (predictable) occasional situations. Data, and in each specific situation possible operations on data are based on the valid rules with respect to the situation concerned. Changes in the work environment are associated with the physical and medical state of patients and physical state of resources. Event Business Objects help us to manage events that can lead to, or result from, these changes. In this way, the system can be adapted to the changing situations in the work environment.

The extendibility of the system makes it possible to cover the whole area of Patient Logistics. The new internal activities, as well as external activities, can be supported through the realization of new Business Objects, and interoperation through the Business Logic layer or the Data Source layer. For example, in this way the system can exchange data with information systems of general practitioners, recreation centres and pharmacies.

Furthermore, Business Objects can be reused in order to form new systems. Therefore, new systems/applications can be realized in a relatively short time and at low cost, as one does not have to begin from scratch and can take advantage of already applied and tested software constituents. Furthermore, to develop applications, one can use the Business Objects without having to deal with (or even without having to know about) the underlying data sources. For example, any application can use "Patient" without dealing with the two databases that maintain its data. This not only promotes reuse of the existing resources, but also supports integration of different applications. Besides, one can use the existing software constituents for creating Business Objects.

Reuse of existing software constituents cannot only reduce the time and effort required for software development, but also maintenance efforts.

Analysis and design can directly affect maintenance efforts. Standardization and categorization of Business Objects leads to clear view on functionality of system constituents and the relationship among them.

For realizing Business Objects and the related applications, one can take advantage of the existing resources and tools for Business Objects, or even use the ready-made Business Objects in case they are available.

8.2 THE BANKING DOMAIN

Automation in the banking domain, like in many other domains, involves various dimensions. In broad outlines, automation designers should deal with the core procedures of banking on the one hand, and the different channels for communicating with the customers on the other hand. This leads to a high level of complexity, and requires a lot of work to establish and maintain information systems. This section discusses the use of the Multi-Tier Architecture based on the concept of Business Objects for dealing with different channels in this domain[5].

The section begins by introducing automation in the banking domain. Then it describes how one can use the Multi-Tier Architecture based on the concept of Business Objects for this domain in order to deal with different communication channels, by concentrating on a specific and representative example case.

8.2.1 BACKGROUND

Automation in the banking domain should support a wide range of activities. These activities can be as simple as transferring money from one account to another, or as complex as providing full budgeting and financial planning advisory services. Furthermore, banks persistently aim at reducing their costs and paperwork, broadening their activity fields and offering new services, in order to attract new customers and to preserve the old customers. Therefore, information systems in the banking domain should support these efforts as well. Accordingly, the architecture of these information systems should make it possible to comply with the requirements of these activities and efforts. However, the existing information systems are mainly based on an old banking model, which has resulted in an enormously complex structure.

[5]The advantage of using Business Objects for another financial domain, E-Commerce, is addressed in [Abolhassani 2000-1].

For a long time, physical branches and the (snail) mail were the only channels through which banks offered their services to their customers. In the last years, banks have started to provide different channels for communicating with their customers. Accordingly, new channels, mainly based on telephone (lines) and computer (devices), have gradually been added to the existing ones. Therefore, the system architecture in the banking domain should also deal with the (new) channels as they come into view.

Moreover, each channel should be able to reach the core of the banking system according to its characteristics. Therefore, the existing systems should continuously be altered and extended with proprietary protocols, standards and software packages. This all can result in a complicated situation, which is not easily manageable and asks for an architecture which can embrace reusable, flexible and scalable constituents in a heterogeneous and distributed environment.

8.2.2 APPLYING BUSINESS OBJECTS

Applying the Multi-Tier Architecture together with the concept of Business Objects can be beneficial for the banking domain, in particular for dealing with different communication channels. In the following, the use of the Multi-Tier Architecture and Business Objects for this purpose is discussed and shown through a representative example case.

8.2.2.1 RATIONALE[6]

The Multi-Tier Architecture, in accordance with the concept of Business Objects, is an appropriate architecture for information systems in the banking domain. It can result in manageable systems, as it separates the core concepts from their usage issues, such as communicating with customers through different channels. This architecture makes it possible to change, enhance and improve the core procedures, while the (new) channels for communicating with the customers can use those procedures in the same way. Moreover, the channels should be balanced and consistent with each other, and transactions should be reflected across all channels. This can also be maintained by applying the Multi-Tier Architecture.

[6]The items discussed in {8.1.2.1} can also be valid here.

Furthermore, although all channels should in principle be able to access the core of the system, each specific channel should provide the customers with possibilities according its characteristics. This means that the services offered by each channel should be in agreement with the nature of that channel and nothing else. The Presentation Logic layer of the Multi-Tier Architecture can take care of such a relationship between the Business Logic layer as the core of the system on the one hand, and the User Interface layer of each channel on the other hand.

Moreover, maintaining security is an important issue in the banking domain, and except for the common parts of the system, each channel may have its own security principles and requirements[7].

8.2.2.2 EXAMPLE CASE

A bank offers different services to its customers such as accounts, personal loans, mortgages and insurance. The bank accounts are of different types, intended for different purposes. Two main types of the accounts are "Current Account" and "Saving Account". While the former can be used for ordinary transactions, the latter is intended for yielding interest.

Each customer can have several accounts at the same bank, possibly at different physical branches. In general, the bank accounts offer services including account transactions - such as account-to-account transfer and bill payment - and balance inquiry. The customer can receive money through his account. This can happen casually or regularly such as monthly salary. In the same way, the customer can pay money through his account, casually or regularly such as monthly rent. The customer should also be informed about his account balance, both on demand and regularly. That is, in addition to the periodically issued statements, the customer may want to be informed about his transactions and balance of an account at any arbitrary point in time.

The bank intends to offer the above-mentioned services through different channels, including physical branch (applying to the desk), telephone (dialling the special number of the call centre and following the instructions), and computer, using standard Web browsers (from the bank's home page, selecting the "online banking" section and then choosing the desired option), or installing and using specific software packages.

[7]This issue is beyond the scope of this work.

However, the channels may not offer the services in the same way, and some channels may not be appropriate for some services at all. For instance, for some services it might be necessary that the customer goes to the physical branch.

Furthermore, it is of crucial importance for the bank, and also for the customers, to prevent unauthorized people from attempting to use (some of) the bank services through different channels. For instance, the bank needs to prevent unauthorized people from logging into its "online banking" section of its "home page".

A typical scenario of the relationship between the bank and the customers is as follows:

> The customer contacts the bank, and then, on request, identifies himself to the bank. In case the bank recognizes him as a legal customer, he can ask for the desired service, for example checking an account balance. Consequently, the bank asks the customer to choose the desired account, in case he as more than one account. Subsequently, the bank verifies the chosen account and lets the customer know the account balance.

The described scenario can be realized through different channels. For instance, realizing this scenario through the Web can result in the following session:

> The customer goes to the bank's "home page" through his Web browser, and selects the "online banking" section. Then, he enters his "user name" and "password" in the form displayed on his monitor. The system verifies if there is an existing customer with the specified "user name" and if the "password" is valid. If the system cannot identify and/or validate the customer, it notifies the customer and requests him to re-enter the information. In case the customer is registered, and the system recognizes him as a legal customer, a menu with a number of options is shown. The customer chooses the "Account Services" option, and then "Show Balance" from the following submenu. Then, the system lists all of the accounts owned by the customer with the type

of each account. The customer can then choose the desired account from that list. Subsequently, the system verifies the chosen account and shows the balance of the account.

BUSINESS OBJECT MODELS

To provide the basis for complying with the requirements of the example case, we should recognize the real-world concepts of this area, as the organizational unit concerned, and should realize Business Objects for these concepts. Applying Business Objects in this area can mainly help us to:

- provide a structured system and
- support different channels.

Figure 8.9 shows the Entity Business Object model of the example case. These Business Objects mainly represent data relevant to this area.

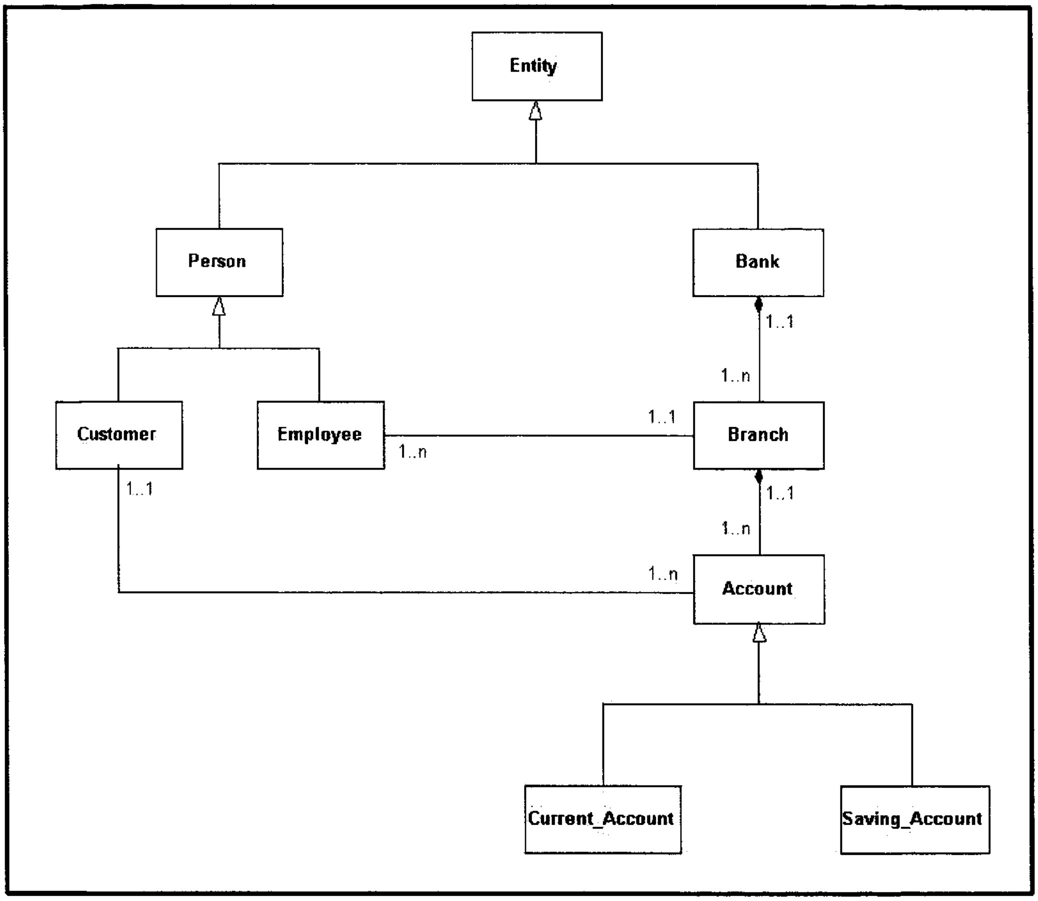

Figure 8.9: The Entity Business Object model of the example case.

Figure 8.10 shows the Process Business Object model of the example case. These Business Objects mainly represent procedures relevant to this area.

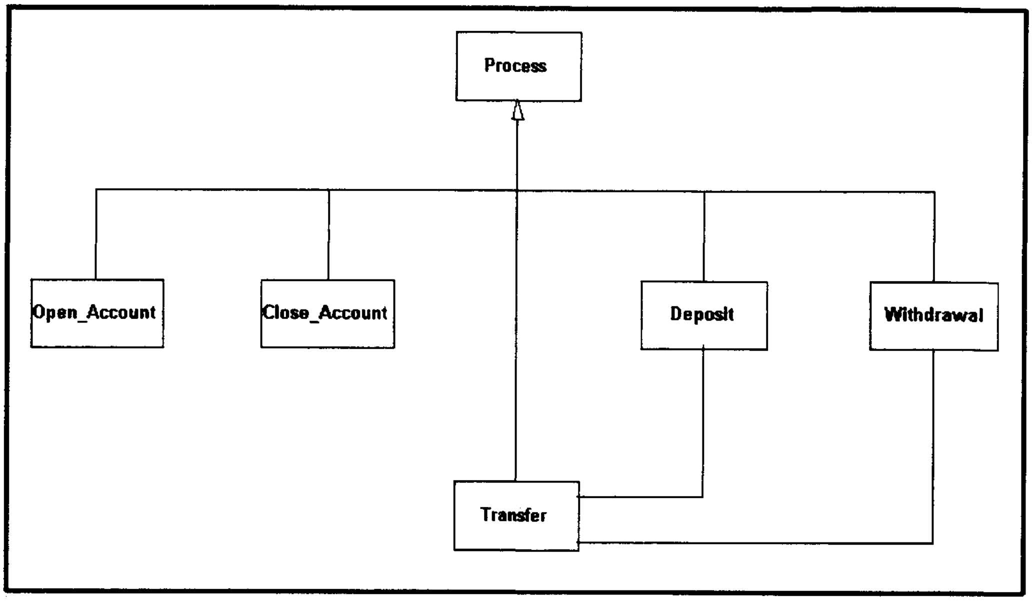

Figure 8.10: The Process Business Object model of the example case.

Figure 8.11 shows the Event Business Object model of the example case. These Business Objects represent the main events that initiate procedures relevant to this area.

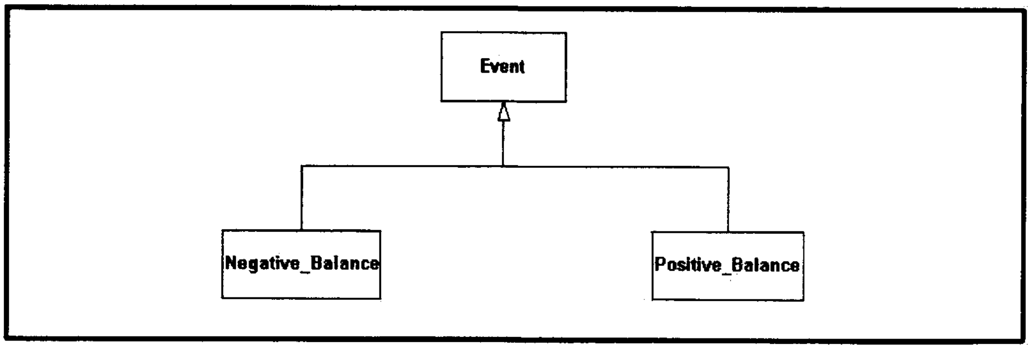

Figure 8.11: The Event Business Object model of the example case.

The shown Business Objects can be used and reused for different applications. Furthermore, they can be modified and extended to deal with new requirements.

DEALING WITH DIFFERENT COMMUNICATION CHANNELS

The main purpose of applying the Multi-Tier Architecture based on Business Objects for this area is to deal with different communication channels. This can be demonstrated by concentrating on a specific part of the Business Object model.

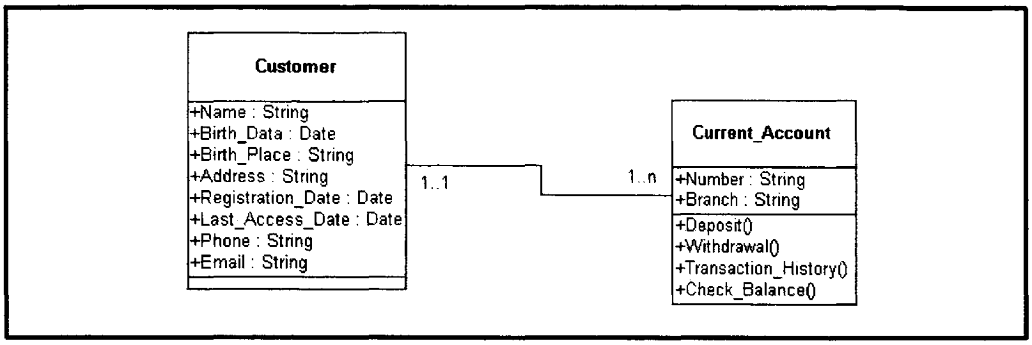

Figure 8.12: A part of the Business Object model of the example case.

Figure 8.12 shows a part of the Business Object model of the example case that includes two Entity Business Objects. These Business Objects should communicate with the customer through different channels. Therefore, the (Presentation Objects of the) Presentation Logic layer should present these Business Objects to the User Interface according to the channel concerned.

Therefore, as the Presentation Logic layer deals with different communication channels, the Business Logic layer, and hence the Business Objects, do not have anything to do with the peculiarities of the channels. Accordingly, the system can support different channels, only by maintaining the appropriate Presentation Logic, while the same Business Logic can be used across different channels.

In general, for each Business Object there can be a Presentation Object that presents it to the User Interface. This Presentation Object knows the characteristics of the channel concerned, and accomplishes the required conversions and transformations. Besides, the Presentation Logic layer blocks the services that are not appropriate for each specific channel. As such, the relationship between Business Objects and Presentation Objects does not have to be a one-to-one relationship. In addition, in case there is a one-to-one relationship between Business Objects and Presentation Objects, the relationship between their attributes and operations does not have to be a one-to-one relationship. Furthermore, it should be noted that the presentation of the Business Logic layer is not the only task of the Presentation Logic layer; for instance, authentication is another one. (In the general sense, "authentication" is checking the legitimacy and access right(s) of the user.)

In the following, the Presentation Objects related to the "Customer" and "Current_Account" Entity Business Objects are explained. The Presentation Objects, namely "P_Customer" and "P_Current_Account", are used for the Web, which is the communication channel here. These Presentation Objects are shown in figure 8.13.

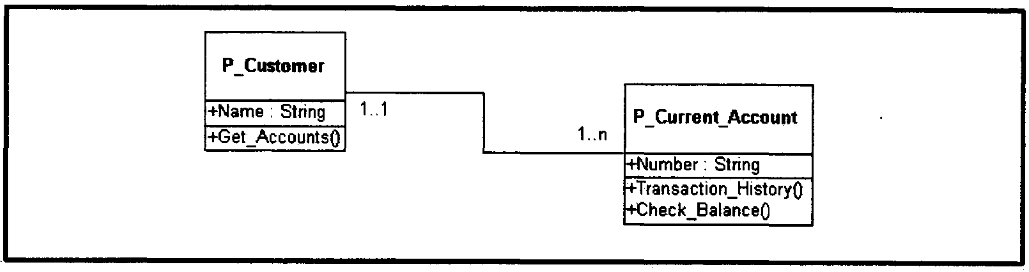

Figure 8.13: The Presentation Object model of a part of the Business Object model of the example case.

"Customer" has a number of attributes, including "Name", "Birth_Date", "Birth_Place", "Address", "Registration_date" and "Last_Access_Date", as well as "Phone" and "Email". The "Name" attribute is required and appropriate to be presented through the channel concerned. Moreover, an operation, such as "Get_Accounts", is required to provide the User Interface layer with all the accounts of the customer.

"Current_Account" has a number of attributes, including "Number" and "Branch", and a number of operations, including "Deposit", "Withdrawal", "Transaction_History" and "Check_Balance". The "Name" attribute and the "Check_Balance" operation are required and appropriate to be presented through the channel concerned.

THE MAIN ADVANTAGES[8]

Business Objects represent the core of the system. Accordingly, it is possible to handle different channels based on the same concepts and software constituents that provide a common basis. For example, the concept "Account" and its related Business Object can be used by user interfaces at physical branches, as well as Web browsers.

As the core of the system is represented by Business Objects, it can easily adapt to changes in the real world. In addition, as channels are not tightly coupled to Business Objects, the changes do not directly affect them. For example, if the rules concerning the current account change, the "Current_Account" Business Object can be changed while the other Business Objects, as well as the channels can (principally) remain unchanged. Furthermore, as a new kind of account, say "new account", is introduced, its Business Object counterpart, "New_Account", can be added to the system (Business Logic), and its Presentation Object counterpart, "P_New_Account", can be added to the Presentation Logic. Accordingly, the user interface of different channels can use "P_New_Account" for their (new) services.

Moreover, one can use the Business Objects, such as "Current_Account", to add (new) channels. This not only promotes reuse of the existing resources, but also supports integration of different channels.

[8]The items discussed in {8.1.2.2} can also be valid here.

Chapter 9

ILLUSTRATIVE EXAMPLES (II)

In a few minutes a computer can make a mistake so great that it would have taken many men many months to equal it.

[Anonymous]

The previous chapter illustrated the use of the concept of Business Objects and its related (architectural) concepts, for specific domains and requirements. This chapter illustrates the use of Business Objects and the Multi-Tier Architecture, in the general sense, for simulation and system integration.

The first section addresses the (potential) advantage of Business Objects for realizing simulation.

The second section discusses the use of the Multi-Tier Architecture, in the form of the Three-Layer Architecture, and the benefits of Business Objects for architectural system integration.

9.1 SIMULATION

Simulation can be applied in many different areas, including business systems, for different purposes, such as analysis, visualisation of processes and training. Important aspects of realizing simulation are recognition of the essential elements of the system concerned, providing an appropriate representation of these elements and maintaining a convenient relationship between this representation and the simulation models. This section discusses the use of Business Objects for realizing simulation, which can be considered as a side effect of using Business Objects in general.

The section begins with the background of the use of Business Objects for realizing simulation. It shows the lack of, and the need for, such a concept in simulation. Then it discusses how one can use Business Object models to provide Petri-Net and (other) simulation models, by concentrating on a representative example case[1].

9.1.1 BACKGROUND

Simulation can be applied in many different areas, such as manufacturing, transportation and education, and for different purposes, including:

- Analysis,
- Capturing the dynamic behaviour,
- Design,
- Visualization of processes,
- Testing,
- Optimisation and
- Training.

The realization of simulation in any area encompasses different phases. The three main, interrelated, phases of simulation are model design, model execution and model analysis [Fishwick 1996]. In addition, the ability to represent the system under study correctly and adequately is a key issue in realizing simulation. Accordingly, recognition of the essential elements of the system

[1]The background of this section can be found in [Abolhassani and Barjis 2000] and [Abolhassani and Barjis 2001].

concerned is of crucial importance and should be accomplished before the mentioned phases of simulation. As such, this effort can be considered as a "pre-simulation" phase.

Furthermore, it is also important to provide an appropriate representation of the achieved results of the "pre-simulation" phase and maintain a convenient relationship between these results on the one hand, and the main simulation phases on the other hand. In other words, a convenient translation from the models of the recognized elements of the system under study into the simulation models should be possible.

9.1.1.1 SIMULATION ELEMENTS

In the area of business systems, similar to the other areas, simulation asks for a thorough insight of the system under study, without any unnecessary details.

The notion of Business Objects imposes a strict distinction between the "pure" business concepts and anything else. Therefore, this notion can be used for the recognition of the "right" elements of the business system concerned that should be part of a simulation.

Business Object models provide a comprehensive and organized view on the system under study and they can flexibly react to the changes.

9.1.1.2 REALIZING SIMULATION

To realize simulation, one should construct a simulation model. The notion of Petri-Nets, [Peterson 1981] and [Petri-Nets], is one of the well-known means for constructing simulation models, as discussed in [Oren and Birta 1995] and [Agostini and Michelis 1998].

One of the main advantages of applying the notion of Petri-Nets for this purpose is its concise form. Petri-Nets represent information in a compact way, and can be used for realizing simulation directly, through specific Petri-Net tools, or indirectly, through general simulation tools. In the former case Petri-Net tools should offer simulation facilities, in addition to those intended for analysing and editing the Petri-Net models. In the latter case Petri-Net models can serve as input for the construction of simulation models, as they contain all necessary information.

In broad outlines, business processes, like many other processes, consist of a number of activities, which can take place sequentially or in parallel. Furthermore, for each activity to follow another activity or activities, there might be one or more conditions that must be fulfilled, or there might be timings that must be complied with.

Petri-Net models contain information about the logical sequence of processes, as well as causal, conditional and optional links, choice elements and synchronization; simulation tools support and apply this information for realizing simulation.

Models based on the notion of Business Objects provide a (reasonable) straightforward way for the construction of Petri-Net models. Therefore, applying Business Objects with Petri-Nets offers a comprehensive way for realizing simulation for business systems.

Furthermore, the standardization and categorization features of Business Objects can be helpful in providing simulation models, and for communication and cooperation among the people involved in realizing simulation, as well as among these people and those involved in the original system (the system being simulated).

In addition, applying Business Objects for realizing simulation can result in the production of standard simulation software constituents.

9.1.2 EXAMPLE CASE

A travel agency wants to investigate its methods of dealing with its clients, especially with respect to the arrangement of accommodation and transportation. The routine procedure of the travel agency is as follows.

When a client applies to the travel agency to book a trip, the desk clerk must check if he is already registered. Then the client lets the desk clerk know about his preferences for his intended trip. The preferences encompass destination, time, cost, accommodation and transportation. The desk clerk then surveys the options that match these preferences.

For the desired destination, and according to the specified time and cost (range), the desk clerk looks for the accommodation possibilities that match the client's desired accommodation. Then the desk clerk checks the outcomes with the client to see if the result is satisfactory. If the result is not accepted, the desk clerk asks the client to make some changes in his preferences for accommodation, and then follows the same procedure with the new preferences. If after a certain number of trials the result is still not satisfactory, the desk clerk asks the client to change his preferences for other items, namely cost and time. In some cases it may even happen that the client is asked to change his destination, if possible. After each change the desk clerk begins the procedure with the new preferences.

When accommodation that matches all of the (possibly changed) preferences is found, a similar procedure begins for the transportation. In this step, the desk clerk looks for those transportation means that match all of the preferences (fixed at the previous step) and the client's desired transportation. If no satisfactory result can be found, the other preferences for time and cost should be changed. In this case, if the changes in the preferences affect the other items, the desk clerk must repeat the procedure for those items. For instance, if the departure and/or arrival time change, the accommodation must be checked again. If the result is satisfactory, the initiator step is succeeded, otherwise it must be repeated with new preferences.

In case all of the (possibly changed) preferences are matched, and the desired accommodation and transportation are found, the desk clerk informs the client and presents the achieved results to the counter clerk for the payment. After the payment is made by the client, the counter clerk informs the desk clerk and the desk clerk makes the reservations.

When the travel agency receives the confirmations from the parties concerned (hotels, airlines, etc.), the counter clerk informs the client about it. If no confirmation is received (on time), the reservations are considered as failed. Accordingly, the client is invited to either change his preferences, which initiates the whole procedure from the very beginning, or to get his money back.

Finally, in case the whole process succeeds, the travel agency sends the accommodation and transportation documents to the client.

The travel agency is looking for answers to the following questions:

- Is it better to begin the procedure with the transportation and not the accommodation?
- Is it better to search for transportation and accommodation in parallel?

- Is it better to merge the tasks of the desk clerk and the counter clerk?
- Up to which point is it reasonable to change the preferences?
- Which order is more efficient for changing the preferences?
- What other preferences can be taken into account?
- Is it better to let (ask) the client (to) give a range of preferences for each item in advance?

9.1.2.1 BUSINESS OBJECT MODELS

In order to provide the basis for realizing simulation for the example case, we should recognize the main elements of the organizational unit concerned and present them in Business Object models.

Figure 9.1 shows the Entity Business Object model of the example case.

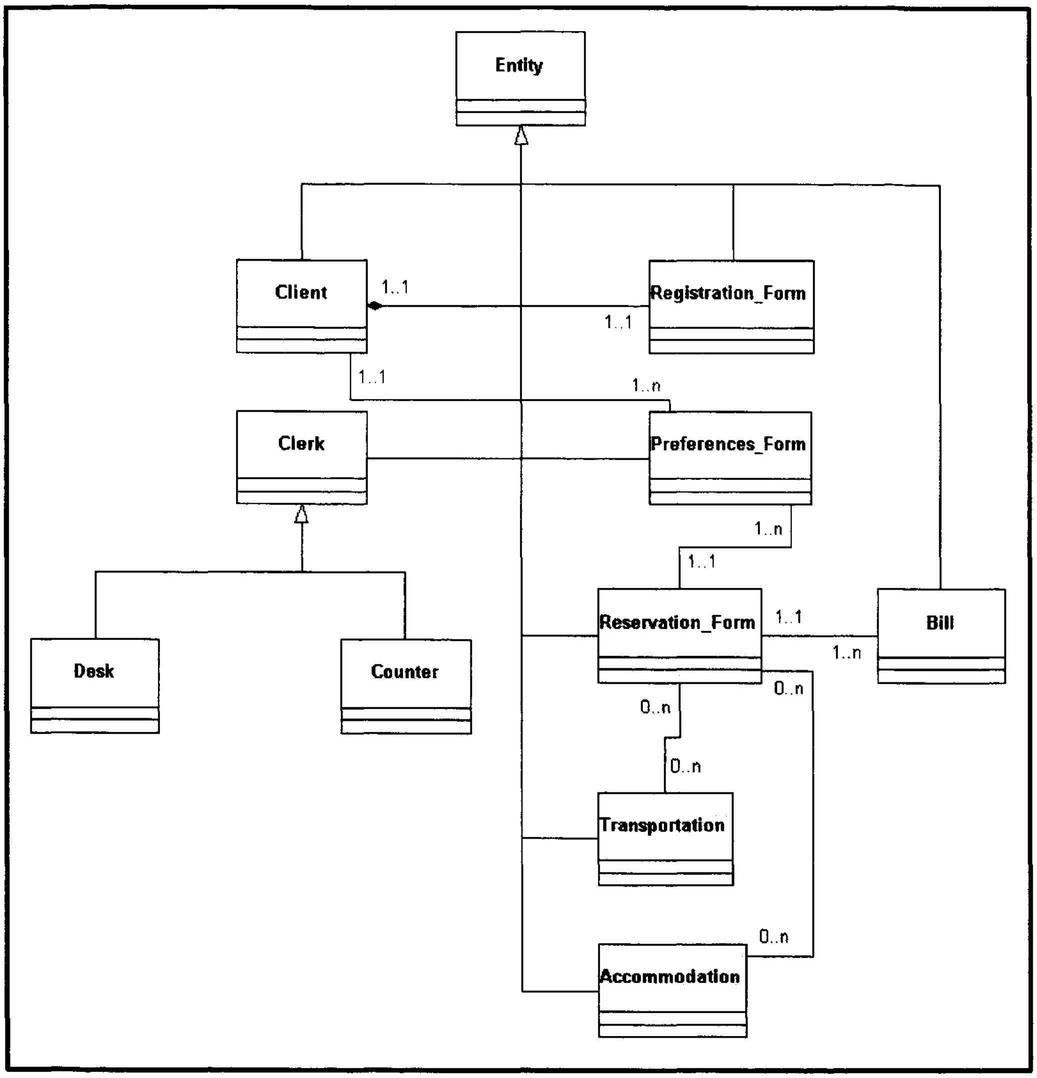

Figure 9.1: The Entity Business Object model of the example case.

Figure 9.2 shows the Process Business Object model of the example case.

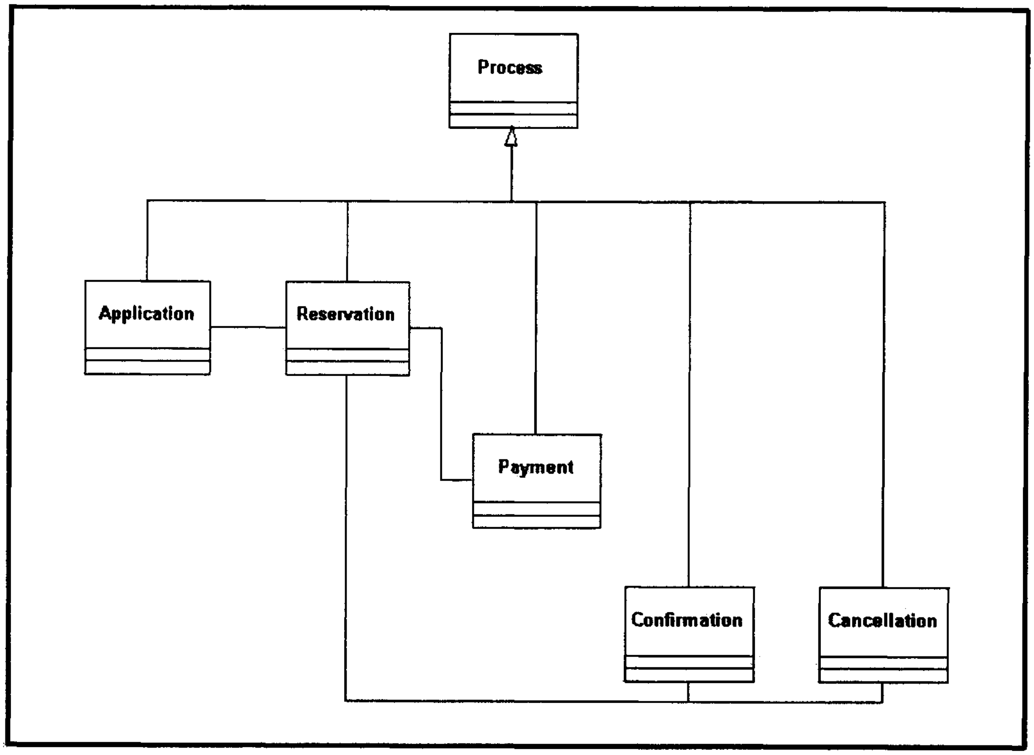

Figure 9.2: The Process Business Object model of the example case.

Figure 9.3 shows the Event Business Object model of the example case.

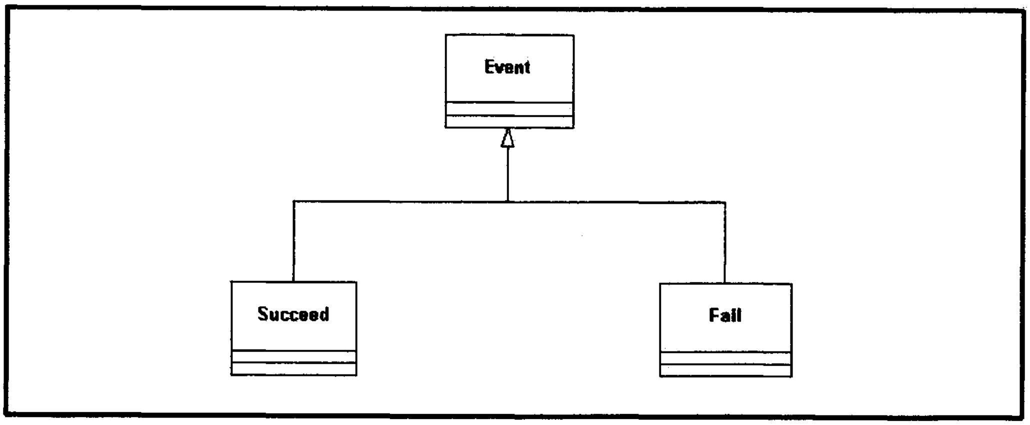

Figure 9.3: The Event Business Object model of the example case.

In addition, figure 9.4 shows the sequence diagrams of the example case.

These diagrams depict three processes related to Reservation, Succeed and Fail.

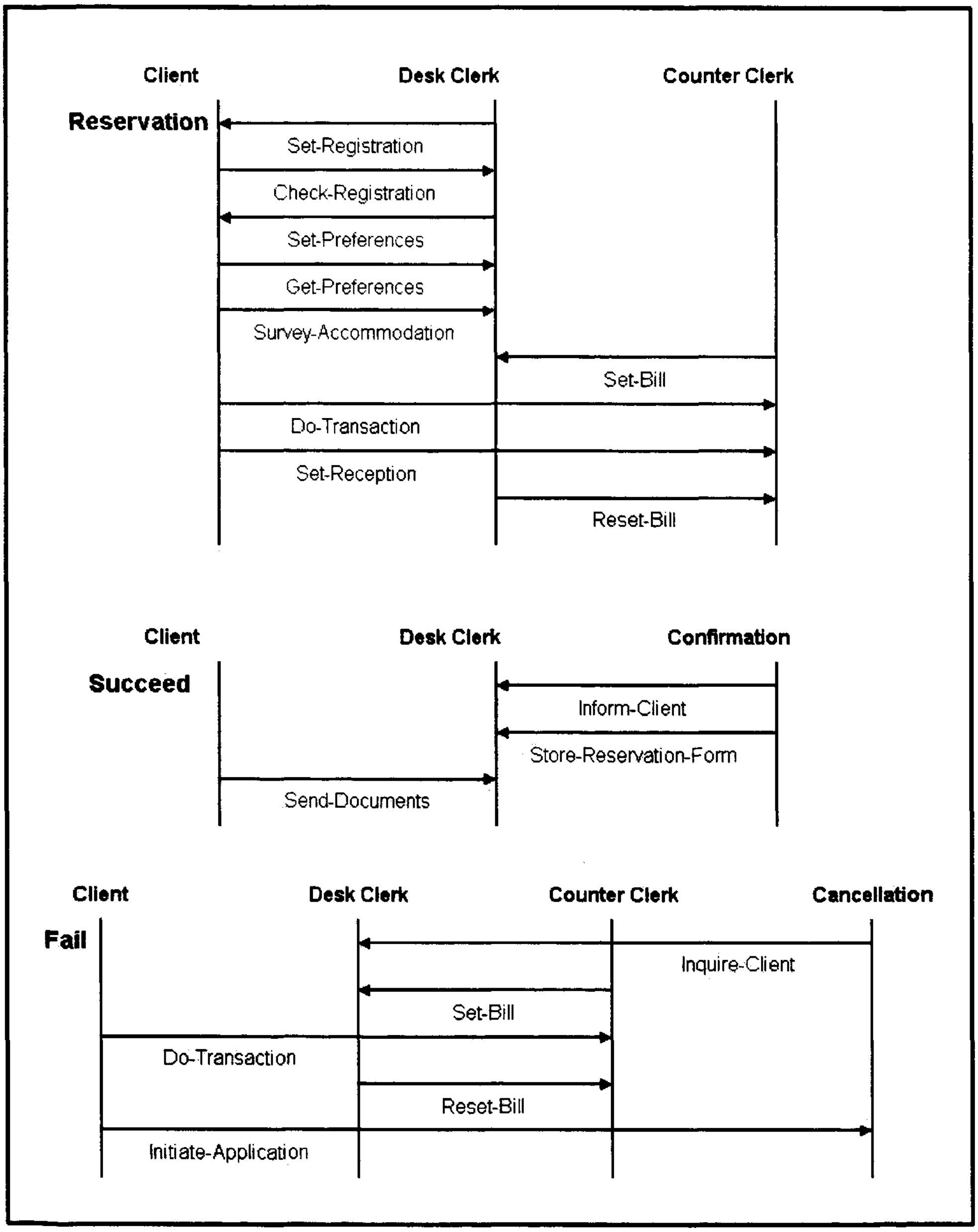

Figure 9.4: The sequence diagrams of the example case.

9.1.2.2 SIMULATION MODELS

The shown Business Object models, and in particular the Process Business Object model, can be used to provide Petri-Net models; the business processes can be identified and the causal and conditional links between these processes can be established.

Figure 9.5 shows the Petri-Net model of the example case.

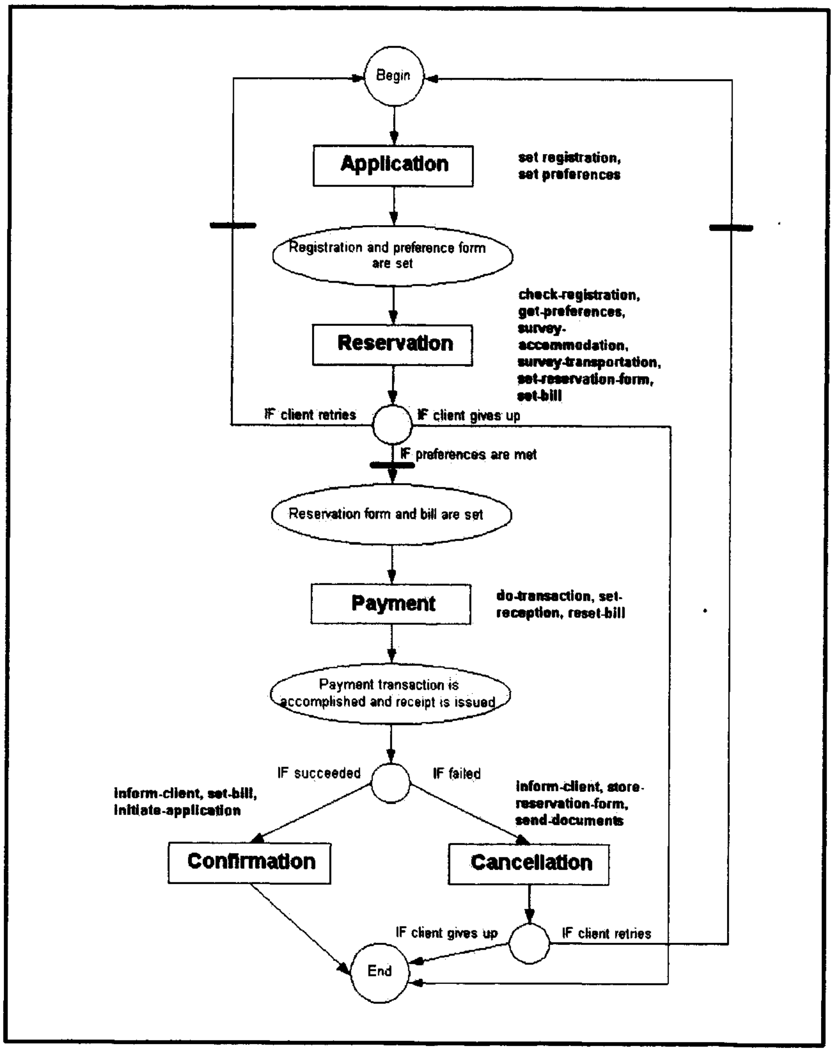

Figure 9.5: The Petri-Net model of the example case.

The model shown in figure 9.5 represents all of the five identified processes, namely Application, Reservation, Payment, Confirmation and Cancellation, as five transitions in the form of five rectangles. The Facts that are created after each process follow the rectangles in the form of ovals. For example, after the Application process, two facts are created. These two facts represent the setting of the Registration form and the Preference form. All of the supplementary activities, related to each process, are listed next to the corresponding rectangle. For example, Application process contains two activities: set registration and set preferences.

As mentioned in {9.1.1.2}, the resulted Petri-Net model can be used for realizing simulation directly, through specific Petri-Net tools, or indirectly, through general simulation tools.

For instance, one may convert the Petri-Net model to a simulation model using the Arena simulation tool [Kelton et al. 2002]. In this simulation tool, each activity, action or atomic process can be represented by three interrelated elements, namely "Arrive", "Server" and "Depart" or "Enter", "Process" and "Leave". In addition, Arena presents logical models that represent typical features of process modelling, such as sequence, parallelism ("Duplicate"), causal and conditional interrelations ("Choose", "If", "Else", "Always"), as well as "synchronization ("Batch").

Accordingly, The Arena simulation model of the example case can be provided (almost) directly from the shown Petri-Net model. In brief:

- "Begin" converts to "Arrive".
- Each "transition" converts to a "Server".
- In case there is more than one "input" for a "Server", a "Batch" should be used.
- Each "condition" converts to "Choose".
- "End" converts to "Depart".

Figure 9.6 shows the Arena simulation model of the example case.

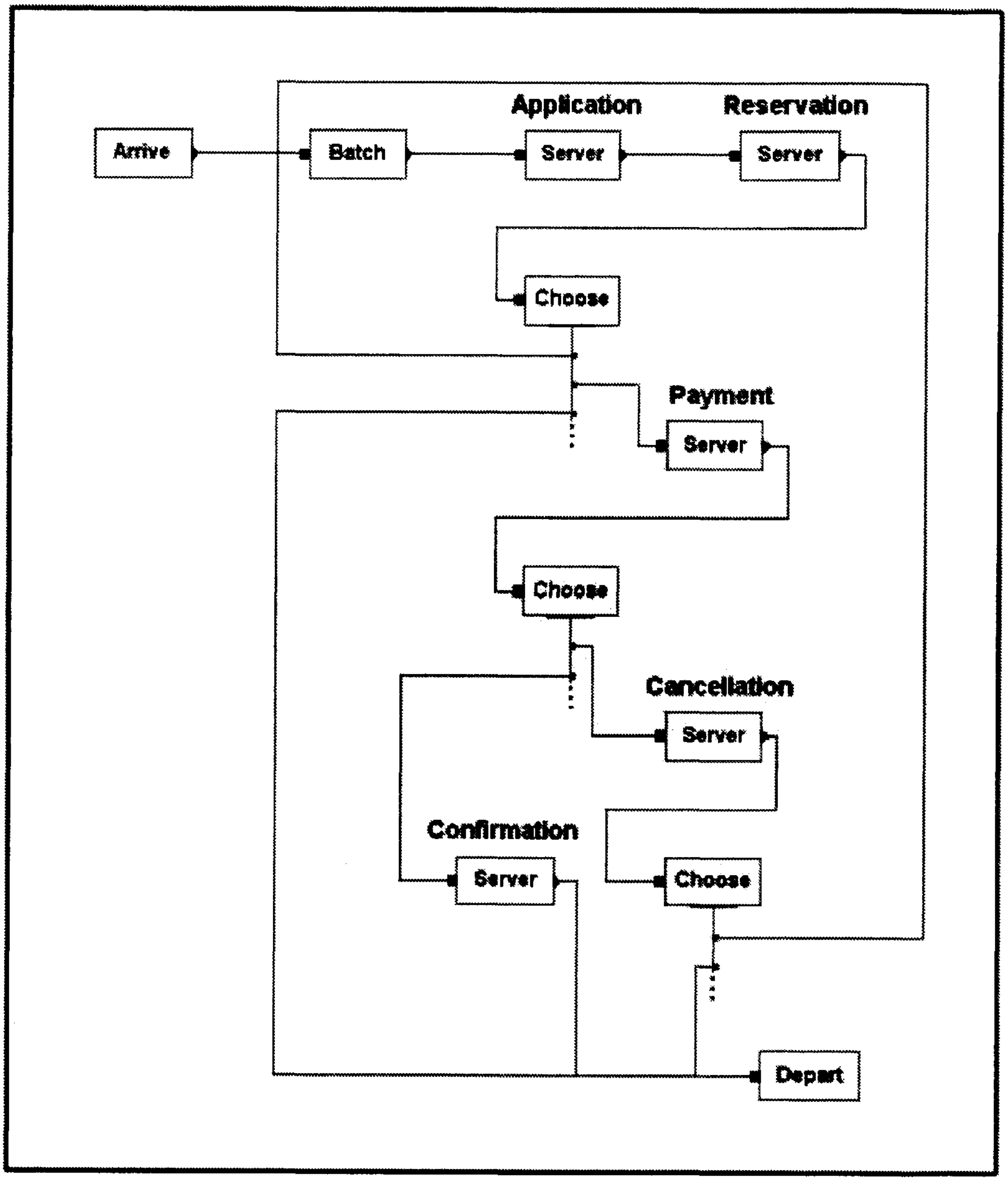

Figure 9.6: The Arena simulation model of the example case.

9.2 SYSTEM INTEGRATION

Over the years, the information systems of enterprises have been realized based on different models and technologies, and they have been spread over different computer networks. Accordingly, enterprises have been locked to separate systems that exist on various locations. To use their systems optimally, enterprises aim to integrate these systems. This practice is called "system integration". This section discusses system integration from an architectural point of view based on the notion of the Multi-Tier Architecture, and points to the role of Business Objects in this respect.

The section begins with a background about system integration. Then, it discusses architectural system integration by going over different integration possibilities. The possibilities are based on four architectural categories and three general methods. Finally, the benefits of Business Objects for architectural system integration are mentioned[2].

9.2.1 BACKGROUND

System Integration is the act of making two or more systems work together in order to achieve new functionality, or to improve the functionality of one or more of the participating systems for a defined period of time. The new or improved functionality should be the result of interoperation and cooperation between the participating systems. The interoperation and cooperation can be realized in different forms related to the desired functionality, the characteristics of the participating systems and the available and applicable means.

The domain of the participating systems can be limited to an individual enterprise, a so-called *inside firewall*, or encompass two or more enterprises, a so-called *outside firewall.* Connecting the Front-Office systems[3] and Back-Office systems[4] is an example of the inside firewall System Integration. The concept of Virtual Enterprise[5] is an example of the outside firewall System Integration.

[2]The background of this section can be found in [Abolhassani 2001-3].

[3]Front-Office systems include Help Desks, Customer Service Desks and Customer Call Centres.

[4]Back-Office systems include General Ledger, Account Payable and Payroll.

[5]This concept refers to a group of Small and Medium Enterprises (SMEs) distributed in

In general, to realize System Integration we can:

- Found the desired functionality upon the existing systems, as is.
- Provide a model for the desired functionality, and then adopt and adapt existing systems based on that model.

9.2.2 ARCHITECTURAL SYSTEM INTEGRATION

System Integration can be considered from an architectural point of view. This viewpoint makes it possible to deal with system integration at an abstract level with respect to the peculiarities of the participating systems. As such, architectural system integration can be of general applicability.

The possibilities for architectural system integration can be explained by dividing systems into four main categories, and using three major methods that can be applied for implementing interoperation and cooperation among systems.

9.2.2.1 CATEGORIES

The notion of the Multi-Tier Architecture, as defined in {3.4}, is an appropriate basis for the architectural categorization of systems. According to this notion, systems consist of the following (macro) functional parts:

- User Interface
- Presentation Logic
- Business Logic
- Information Logic
- Data Source

a given geographical region or even located in different countries and continents that form an individual *virtual* enterprise, for a specific period of time, in order to supply a specific product or service, based on interoperation and cooperation via computer networks by sharing their data and resources.

Consequently, by imposing clear borders between different parts of a system related to different (macro) functionalities, the Multi-Tier Architecture provides us with a robust architectural view on the integration of systems.

By ignoring Presentation Logic and Information Logic for the sake of simplicity, and looking at each part as a pure logical unit, we can bring each system under one of the following categories, defined in {3}:

- Monolithic: User Interface, Business Logic and Data Source are interwoven.
- Fat Client/Server: User Interface and Business Logic are interwoven. Data Source is distinguishable.
- Client/Fat Server: User Interface is distinguishable. Business Logic and Data Source are interwoven.
- Three-layer: User Interface, Business Logic and Data Source are distinguishable.

Accordingly, System Integration can be considered as a matter of:

- Data Source integration
- Business Logic integration
- User Interface integration

or a combination thereof.

9.2.2.2 METHODS

MESSAGING

The "messaging" method is based on exchanging text messages according to the syntax and semantics recognizable for the parties involved. This implies that each party is provided with the facilities and services required for sending and receiving the messages. Extensible Markup Language (XML), [XML] and [XML-W3], is an example of enabling technology for this method.

The advantages of this method are:

- It is easy to implement.
- It is easy to maintain.
- It does not impose much overhead.
- It can be fast at run-time.

The disadvantages of this method are:

- It is flat.
- It has security problem.
- It cannot maintain "state".
- It is not transparent.

This method can be used for the User Interface, Business Logic and Data Source integration. The (sub) system that is integrated with the main system through this method cannot take part in the overall system model as a first-class constituent.

WRAPPING

The "wrapping" method is based on providing an interface for (part of) a system. Accordingly, the desired functionality of the system (part) is exposed through the interface, and requests to the exposed functionality are directed through this interface. The interface should take care of the manipulations required. CORBA is an example of enabling technology for this method.

The advantages of this method are:

- It can be easy to implement.
- It can be easy to maintain.
- It can be secure.
- It can maintain "state".
- It is almost transparent.

The disadvantages of this method are:

- It is flat.
- It imposes overhead.
- It can be slow at run-time.

This method can be used for the User Interface, Business Logic and Data Source integration. The (sub) system that is integrated with the main system through this method can take part in the overall system model as a coarse-grained component.

ASSOCIATION

The "association" method is based on treating part of a system (S1) as part of another system (S2). As such, the same technology used for the communication between the constituents of S2 should be applied for the related part of S1.

The advantages of this method are:

- It is robust.
- It can be secure.
- It is transparent.

The disadvantages of this method are:

- It can be difficult to implement.
- It can be difficult to maintain.
- It can imply duplication of work.
- It can impose overhead.

This method can be used for the Business Logic and Data Source integration. The (sub) system that is integrated with the main system through this method can be considered as a part of the main system.

9.2.2.3 POSSIBILITIES

The starting point for the integration is a Three-Layer system. It is supposed that a system consisting of three distinct layers (main system) should be integrated with another system (subsystem) which belongs to one of the four mentioned categories through one of the three mentioned methods.

MONOLITHIC SYSTEM

The subsystem can be considered as a black box. It can be integrated with the User Interface and Business Logic of the main system through messaging and wrapping.

This case is shown in figure 9.7.

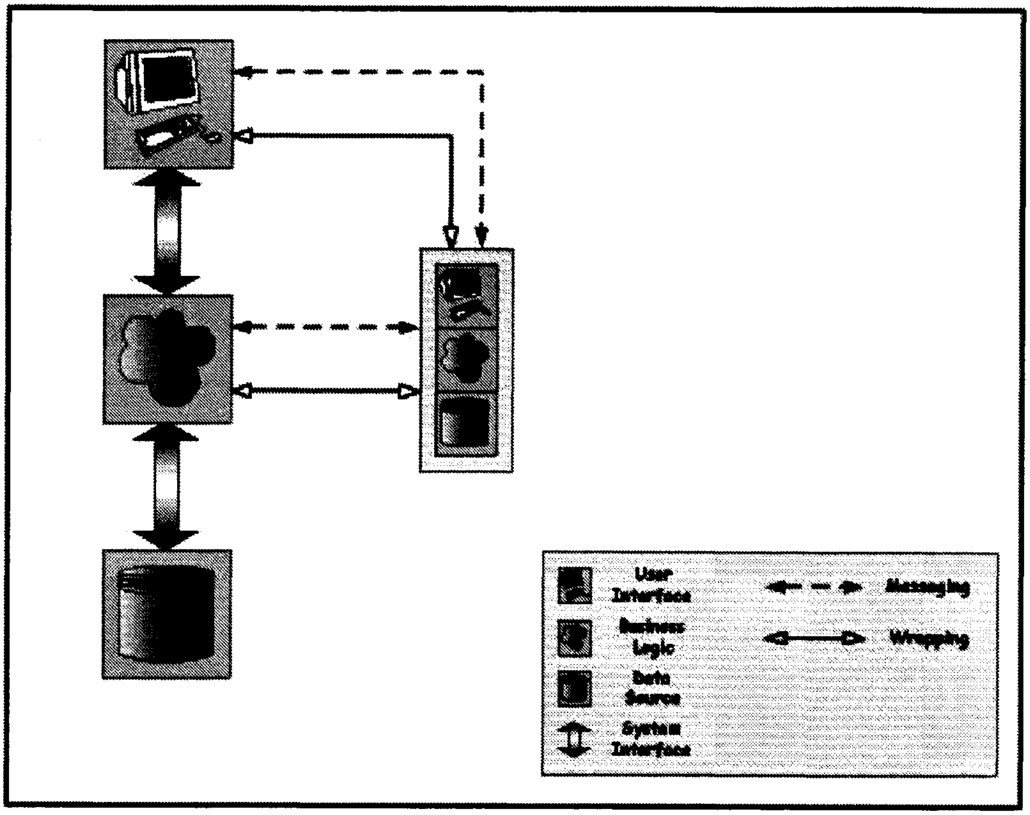

Figure 9.7: Integrating a monolithic system.

FAT-CLIENT/SERVER SYSTEM

The fat client (User Interface and Business Logic) of the subsystem can be integrated with the User Interface and Business Logic of the main system through messaging and wrapping.

The server (Data Source) of the subsystem can be integrated with the Business Logic of the main system through association.

This case is shown in figure 9.8.

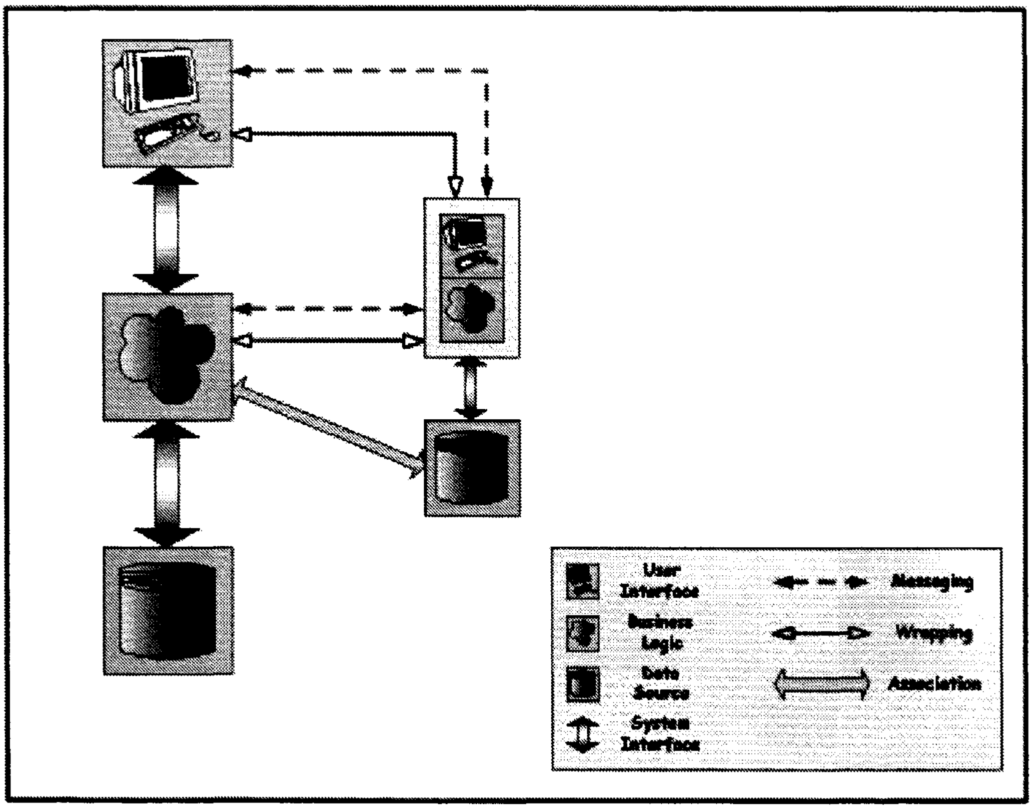

Figure 9.8: Integrating a fat-client/server system.

CLIENT/FAT-SERVER SYSTEM

The client (User Interface) of the subsystem can be integrated with the User Interface of the main system through messaging and wrapping.

The fat server (Business Logic and Data Source) of the subsystem can be integrated with the User Interface and Business Logic of the main system through messaging and wrapping. Besides, the fat server of the subsystem can be integrated with the Business Logic of the main system through association. However, this can result in duplication of work with respect to the Business Logic.

This case is shown in figure 9.9.

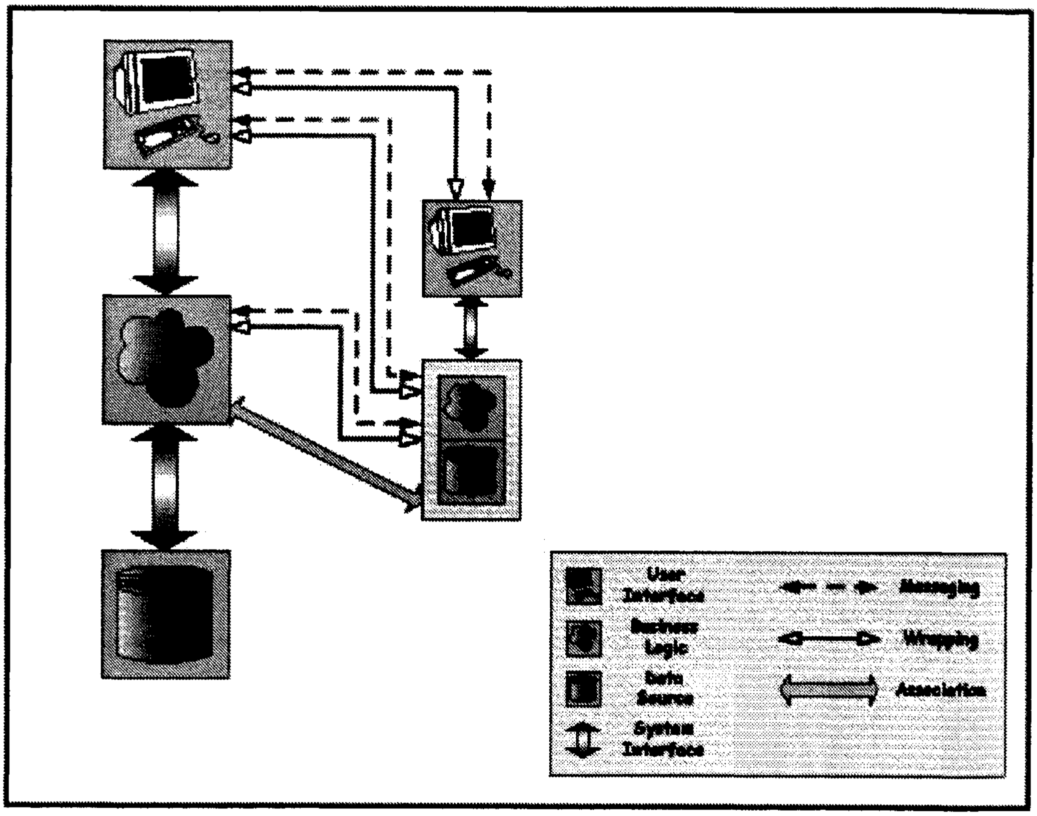

Figure 9.9: Integrating a client/fat-server system.

THREE-LAYER SYSTEM

The User Interface of the subsystem can be integrated with the User Interface of the main system through messaging and wrapping.

The Business Logic of the subsystem can be integrated with the User Interface and Business Logic of the main system through messaging and wrapping. Besides, the Business Logic of the subsystem can be integrated with the Business Logic of the main system through association.

The Data Source of the subsystem can be integrated with the Business Logic of the main system through association.

This case is shown in figure 9.10.

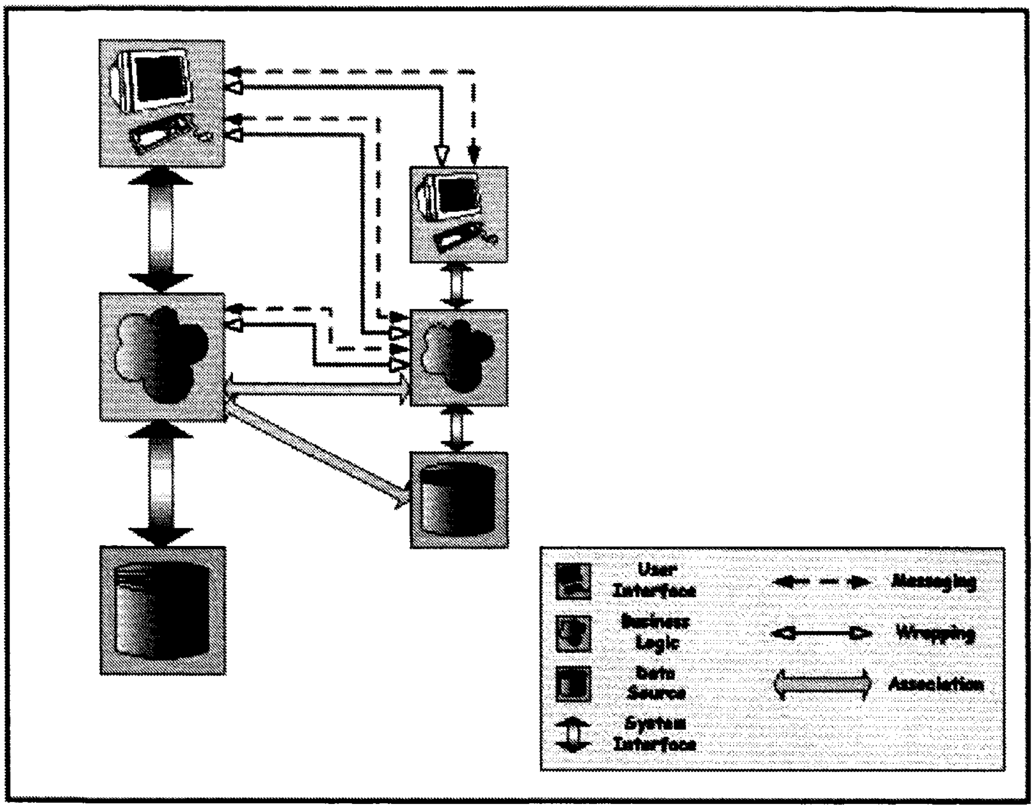

Figure 9.10: Integrating a three-layer system.

TAILORING THE SYSTEM

To realize system integration as explained above, and in particular to choose the right methods, one should take into account many theoretical and practical aspects which are related to the possibilities and requirements of the specific situation in which system integration should be realized. The main theoretical and practical considerations are:

- Integration goal: improvement of the existing functionality/providing new functionality

- Characteristics of the participating systems
- Level of distinguishability between the layers of the participating systems
- Available code: binary/source
- Development time
- Development resources
- Development approach
- Frequency of (data) usage
- Run-time speed
- Security

9.2.3 BENEFITS OF BUSINESS OBJECTS

Although the integration of systems based on the Business Logic layer is, in principle, the most logical and robust way, it is not always easily achievable. The ease of the Business Logic integration is especially related to the constituents that form this layer.

The use of Business Objects for realizing the Business Logic layer makes system integration based on this layer more feasible, in particular because Business Objects:

- are based on the real-world concepts,
- provide the desired level of abstraction and
- are reusable.

Furthermore, the use of Business Objects can result in uniformity and standardization of the constituents of the Business Logic layer. This makes system integration easier.

Chapter 10

CONCLUSIONS

The longest part of the journey is said to be the passing of the gate.

[Marcus Terentius Varro]

In conformance with the research goal, the preceding chapters provide a comprehensive and integrated view on Business Objects. They identify the core concept of Business Objects and its value for software engineering in realizing information systems through answering to the formulated research main and sub-questions.

Furthermore, the preceding chapters show that the hypotheses formulated at the beginning of this research are acceptable.

This chapter reviews and concludes the subjects discussed in the preceding chapters.

10.1 DEFINITION

Chapter two presents a clear and unambiguous definition of the concept of Business Objects. This definition can form an appropriate basis for discussing Business Objects and the accompanying issues. Furthermore, Business Objects created in accordance with this definition can lead to openness and semantical interoperability among information systems.

Using Business Objects, as a specific kind of objects, can result in reusability, flexibility and modifiability. Therefore, using Business Objects can improve system development and reduce system maintenance efforts. Furthermore, Business Objects reflect the real-world concepts they represent and provide a structured view on the business in question. Therefore, in addition to their benefit for software systems, Business Objects can help enterprises to find the inefficiencies and deficiencies of their (business) systems. Accordingly, enterprises can use Business Object models as leading models for (re-)engineering and (re-)constructing their information systems. Consequently, using Business Objects can have considerable financial advantages for companies. At the same time, Business Objects dictate their required infrastructure to the information systems concerned, and hence may give rise to changes in the existing infrastructure.

For realizing business software systems [Taylor 1995] suggests using "model-based" development, as opposed to application development; the model-based software systems, like the business systems they support, are adaptive and able to respond to evolving business requirements. It describes the goal of this approach as providing software models that are not tied to any particular problem and represent the structure and operations of a business as simply and directly as possible. Furthermore, the use of "active components" for dealing with the requirements of (enterprise) information systems is suggested in [Dietz 1994] and [Dietz and Mulder 1998]. The concept of Business Objects, as defined in this thesis, is in accordance with these approaches.

The thesis emphasizes that the concept of Business Objects involves more than merely applying the Object-Oriented technology for business systems. The concept of Business Objects represents a specific approach for modelling business systems and realizing business software systems, and, in turn, leads to new concepts different from those related to applying objects - or even Reflection Objects - to isolated software systems. These concepts are in particular related to the architectural aspects of applying Business Objects.

10.2 LOCATION

Chapter three presents the Multi-Tier Architecture as the appropriate architecture for (applying) Business Objects, and describes the role of Business Objects as constituents of the Business Logic.

In addition to its value with respect to Business Objects, the Multi-Tier Architecture enables and promotes parallelism and reuse in software development. As in this architecture the main constituents of the system are not tightly coupled, they can be designed and implemented in parallel and can be reused for new systems, more effectively and efficiently than is possible in other architectures.

[Taylor 1995] also refers to the software architecture that supports the model-based development approach as a layered architecture which consists of horizontal layers, rather than vertical applications; the models occupy the middle layer and form the integrating structure of the system, the top layer contains the screens and the bottom layer consists of the legacy systems. The architectural context of Business Objects defined in this thesis is in accordance with this viewpoint.

10.3 CREATION

Chapter four discusses two forms of the presentation of Business Objects, namely "abstract" and "object". Presenting Business Objects as "abstract", the OMG approach, can result in semantical standardization and is in particular advantageous for interoperation among systems. Presenting Business Objects as "object" can result in usage standardization and is in particular advantageous for reusability.

However, the use of off-the-shelf Business Objects as components for realizing business software systems is not straightforward. The componentization of Business Logic can lead to new challenges in the design of applications. Choosing the "right" components with the "right" abstraction levels and using the existing software constituents are examples of these challenges.

Furthermore, chapter four deals with standardization of Business Objects as the major characteristic of this concept. It does not limit the standardization to the globally acceptable standards. This is especially meaningful with respect to the difficulty of devising such standards. Accordingly, the standardization feature of Business Objects does not have to be restricted to widely acceptable standards. Standardization can be realized and used at different organizational units.

Whether Business Objects can comply with the standards closely depends on how they have been devised. For instance, the deployment flexibility of the standard Business Objects is directly related to their granularity, as well as the semantics they maintain. Therefore, to devise the standards we should not only deal with the general standardization issues and the specific characteristics of the domain concerned, but we should also take their deployment into consideration.

Moreover, chapter four also points to categorization of Business Objects as an important aspect to which insufficient attention has been paid. It shows that the categorization criteria of Business Objects are (to a large extent) orthogonal. That is, each Business Object can belong to many different categories. In other words, each Business Object can be considered as a point in an n-dimensional space, where each dimension represents a specific criterion. Categorization of Business Objects can be helpful in providing a better view on different aspects of Business Objects, in devising standard Business Objects and in using them.

For instance, in the modelling phase we do not have to deal with issues such as persistence, security and transactions. Moreover, in addition to the core semantical issues that are specific for each business domain there are many other issues that are repeated in different business domains, and there are many issues that are repeated when one realizes software systems based on Business Objects. Capturing these issues in a disciplined manner based on categorization of Business Objects can be advantageous.

Literature shows some similar efforts in the relevant and close areas. For instance, [Vaessen 2001] presents a taxonomy with respect to Business Logic, where each Business Logic rule belongs to "structural assertation", "action assertation" or "derivation assertation".

As the use of Business Objects for modelling business systems and realizing business software systems increases, categorization of Business Objects, and Business Object taxonomies, will grow in importance as well.

10.4 USAGE

Chapter five discusses issues that are involved in the use of Business Objects. These issues are related to the location and creation of Business Objects. The chapter shows the close relationship between the layers of a Five-Layer Architecture. In addition, it points to the different interrelated options with respect to the relationship between Business Objects and data sources, and to the perspectives with respect to the creation of Business Objects. Accordingly, the use of Business Objects (within the framework of a Multi-Tier architecture) closely depends on each specific case.

10.5 STATE-OF-THE-ART

Chapter six points to a number of existing enabling technologies and system environments that can be used to develop, deploy and manage distributed systems, and discusses their relationship with the concept of Business Objects. Not all of these technologies and environments use the term Business Objects in their terminology, nor do they have the same understanding of this term. However, using different aspects of Business Objects discussed in the preceding chapters as background, this chapter surveys a number of these technologies and environments.

10.6 APPLICATION

Chapter seven presents a case study carried out in a company, within the framework of the realization of an information system. The information system is based on the Three-Layer Architecture and uses Business Objects for its middle layer.

Chapter eight discusses the use and advantages of Business Objects and the Multi-Tier Architecture for two domains, namely health care and banking. It illustrates how we can apply Business Objects within the framework of a Five-Layer Architecture in order to deal with specific requirements, namely dealing with different data sources and user interfaces, as well as changes in the work environment.

However, the term "Business Objects" is not familiar to many people in the studied domains, as in many other domains. Besides, among those to whom this term sounds familiar, there is a confusion about its meaning and benefit. In addition, the term "Business" has a negative effect on some people in some domains, especially on those in the health-care domain.

Chapter nine illustrates the use and advantages of Business Objects and the Multi-Tier Architecture, in the general sense, for simulation and system integration.

10.7 OUTLOOK

The trend towards the integration of information systems and the providing of integrated business services, the growing interoperation and cooperation between enterprises and the new technological possibilities motivate the use of Business Objects. Delivering the "right" information (up to date - back end) to the "right" people (customers, employees, partners, etc.) at the "right" time (fast enough) in the "right" form (platform, device and media type - front end) is the ultimate goal of any enterprise information system. Business Objects can help designers of enterprise information systems to achieve this goal.

There is another trend towards the use of the Internet services instead of software packages. These services can deliver the last version of an application through the Internet. These applications require neither installation nor upgrades, and have no multiple versions the user has to deal with. The Internet services can enable and result in multi-party business systems. These systems require Business Objects that are different from those required by traditional business systems.

Bibliography

[Reference]	Description[1]
[Abolhassani 1999]	M. Abolhassani, Defining The Business Objects, International Workshop on Information Integration and Web-based Applications & Services (IIWAS'99), November 1999, Yogyakarta, Indonesia
[Abolhassani and Szentivanyi 1999]	M. Abolhassani and G. Szentivanyi, A Component-Oriented Approach For

[1]References are listed alphabetically and belong to one of the following categories:

- Books
- Articles of conferences, workshops, etc.
- Technical reports
- Web pages

They identify items belonging to each category with the same details, as far as possible, while they aim to identify each item in sufficient detail, as follows:

- For a Book the details, in order, are: first initial(s) and surname(s) of the author(s), title, publisher, date (year) of publish.
- For an article the details, in order, are: first initial(s) and surname(s) of the author(s), title, name, date (month, year) and place (city, country) of the conference, workshop, etc.
- For a technical report the details are: first initial(s) and surname(s) of the author(s), organization name, title, date of publish.
- For a Web page the details, in order, are: title, Internet address.

	Enterprise-Devoted E-Commerce, International Workshop on Advance Issues of E-Commerce and Web-based Information Systems (WECWIS'99), April 1999, Santa Clara, USA
[Abolhassani 2000-1]	M. Abolhassani, Reviewing The Requirements Of Traditional E-Commerce, The Fourth CollECTeR Conference on Electronic Commerce (CollECTeR (USA) 2000), April 2000, Breckenridge, USA
[Abolhassani 2000-2]	M. Abolhassani, Business Objects And Standards, International Conference on System Research, Informatics and Cybernetics (InterSymp 2000), August 2000, Baden-Baden, Germany
[Abolhassani and Barjis 2000]	M. Abolhassani and J. Barjis, Applying Business Objects For Simulation, Summer Computer Simulation Conference (SCSC 2000), July 2000, Vancouver, Canada
[Abolhassani and van Groen 2000]	M. Abolhassani and A. van Groen, Business Objects In Practice: Options And Perspectives, International Workshop on Practical Information Mediation, Brokering and Commerce on the Internet (IMEDIAT 2000), October 2000, Tokyo, Japan
[Abolhassani 2001-1]	M. Abolhassani, Business Objects And Enterprise Applications, Information Resources Management Association International Conference (IRMA 2001), May 2001, Toronto, Canada

[Abolhassani 2001-2]	M. Abolhassani, Categorization Of Business Objects, International Conference on System Research, Informatics and Cybernetics (InterSymp 2001), August 2001, Baden-Baden, Germany
[Abolhassani 2001-3]	M. Abolhassani, System Integration, Architectural Approach And Business Objects, Scuola Superiore Guglielmo Reiss Romoli Conference (SSGRR2001), August 2001, L'Aquila, Italy
[Abolhassani and Barjis 2001]	M. Abolhassani and J. Barjis, Realizing Simulation For Business Management, Summer Computer Simulation Conference (SCSC'01), July 2001, Orlando, USA
[ActiveX]	ActiveX, `http://www.activex.org`
[Agostini and Michelis 1998]	A. Agostini and G. de Michelis, Simple Workflow Models, Workshop on Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98), June 1998, Lisbon, Portugal
[Bonar 1997]	J. Bonar, Business Objects for Front-Office Applications: Making Domain Experts Full Partners, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97), October 1997, Atlanta, USA
[Booch 1994]	G. Booch, Object-Oriented Analysis and Design, The Benjamin/Cummings Publishing Company, Inc., 1994
[Casanave 1995]	C. Casanave, Business-Object Architecture and Standards, Object-Oriented

Programming Systems, Languages and Applications (OOPSLA'95), October 1995, Austin, USA

[Cattel and Barry 1997] R. Cattel and D. Barry, The Object Database Standard: ODMG 2.0, Morgan Kaufmann Publishers, Inc., 1997

[CCOW 1998] Common Clinical Context Architecture Specification 1.1, The Clinical Context Object Workgroup (CCOW), 1998

[Chance and Melhart 1999] B. Chance and B. Melhart, A Taxonomy for Scenario Use in Requirements Elicitation and Analysis of Software Systems, IEEE Conference and Workshop on Engineering of Computer-Based Systems, March 1999, Nashville, USA

[Cook and Daniels 1994] S. Cook, J. Daniels, Designing Object Systems, Prentice-Hall, Inc., 1994

[CORBA] Common Object Request Broker Architecture (CORBA), `http://www.omg.org/corba`

[CORBAmed 1998] The CORBAmed Roadmap, CORBAmed: The OMG Health-Care Domain Task Force, February 3, 1998

[Cummins 1996] F. Cummins, Business Objects Issues, Electronic Data Systems, 1996

[Dellarocas 1995] C. Dellarocas, Toward a Design Handbook for Integrating Software Components, the Fifth Symposium on Assessment of Software Tools and Technologies, June 1997, Pittsburgh, USA

[DEMO] Dynamic Essential Modelling of Organizations (DEMO), `http://www.demo.nl`

[DICOM]	Digital Imaging and Communications in Medicine (DICOM), http://medical.nema.org
[Dietz 1994]	J. Dietz, Modelling Business Processes For the Purpose of Redesign, Business Process Re-Engineering: Information Systems Opportunities and Challenges, May 1994, Queensland Gold Coast, Australia
[Dietz 1999]	J. Dietz, Understanding and Modelling Business Processes with DEMO, International Conference on Conceptual Modelling (ER'99), November 1999, Paris, France
[Dietz and Mulder 1998]	J. Dietz and H. Mulder, Organizational Transformation Requires Constructional Knowledge of Business Systems, Thirty-First Annual Hawaii International Conference on System Sciences, January 1998, Kohala Coast, USA
[Edwards 1997]	J. Edwards, 3-tier Client/Server At Work, John Wiley & Sons, Inc., 1997
[Eeles and Sims 1998]	P. Eeles, O. Sims, Building Business Objects, John Wiley & Sons, Inc., 1998
[Ehnebuske et al. 1997]	D. Ehnebuske, B. Mc Kee, I. Rouvellou and I. Simmonds, Business Objects and Business Rules, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97), October 1997, Atlanta, USA
[EJB]	Sun's Enterprise Java Beans (EJB), http://java.sun.com/products/ejb

[Emmerich et al. 1998]	W. Emmerich, E. Ellmer, B. Osterholt and R. Zicari, Business Object Facilities - A Comparative Analysis, Workshop Integration heterogener Softwaresysteme, September 1998, Magdeburg, Germany
[Fishwick 1996]	P. Fishwick, What is Simulation?, IEEE Potentials, February 1996
[Fowler 1997]	M. Fowler, Analysis Patterns, Addison-Wesley, 1997
[van Groen 2000]	A. van Groen, Business Objects in GRIP, master thesis, Delft University of Technology, August 2000
[Heineman and Councill 2001]	G. Heineman and W. Councill, Component-Based Software: putting the pieces together, Addison-Wesley, 2001
[Heller and Martin 1999]	R. Heller and C. Martin, Using a Taxonomy to Rationalize Multimedia Development, IEEE International Conference on Multimedia Computing and Systems, June 1999, Florence, Italy
[Hertha et al. 1995]	W. Hertha et al., An Architecture Framework: From Business Strategies to Implementation, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'95), October 1995, Austin, USA
[Herzum and Sims 1998]	P. Herzum and O. Sims, The Business Component Approach, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98), October 1998, Vancouver, Canada

[Herzum and Sims 2000]	P. Herzum and O. Sims, Business Component Factory, John Wiley & Sons, Inc., 2000
[HL7]	Health Level 7 (HL7), `http://www.hl7.org`
[Hung and Patel 1997]	K. Hung and D. Patel, Modelling Domain Specific Application Frameworks with a Dynamic Business Object Architecture: An Approach and Implementation, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97), October 1997, Atlanta, USA
[Hung et al. 1998]	K. Hung, T. Simons and T. Rose, "The Truth Is Out There?" A Survey of Business Objects, The International Conference on Object-Oriented Information Systems (OOIS98), September 1998, Paris, France
[ISO]	International Organization for Standardization (ISO), `http://www.iso.ch`
[Jacobson et al. 1992]	I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, Object-Oriented Software Engineering, Addison-Wesley, 1992
[JAVA]	Sun's Java, `http://java.sun.com`
[JavaBeans]	Sun's JavaBeans, `http://java.sun.com/beans`
[Jerke et al. 1999]	N. Jerke, G. Szabo, D. Jung and D. Kiely, Visual Basic 6 Client/Server How-To, Sams Publishing, 1999

[Kelton et al. 2002]	W. Kelton, R. Sadowski and D. Sadowski, Simulation with Arena, McGraw-Hill, 2002
[Landwehr et al. 1993]	C. Landwehr, A. Bull, J. McDermott and W. Choi, A Taxonomy of Computer Program Security Flaws with Examples, Naval Research Laboratory, November 1993
[Larman 1998]	C. Larman, Applying UML and Patterns: an introduction to Object-Oriented analysis and design, Prentice-Hall, Inc., 1998
[Marshall 1997]	C. Marshall, Business Object Management Architecture, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'97), October 1997, Atlanta, USA
[Merriam-Webster]	Merriam-Webster OnLine, `http://www.m-w.com`
[MIB]	Medical Information Bus (MIB), `http://www.mib.com`
[Nierstrasz and Tsichritzis 1995]	O. Nierstrasz and D. Tsichritzis, Object-Oriented Software Composition, Prentice Hall, 1995
[Oestereich 1999]	B. Oestereich, Developing Software with UML, Object-Oriented Analysis and Design in Practice, Addison-Wesley, 1999
[OMG]	Object Management Group (OMG), `http://www.omg.org`
[ODMG]	Object Database Management Group (ODMG), `http://www.odmg.org`

[OMG 1995] Object Management Group (OMG), Glossary of Terms, Business Object Management Special Interest Group, OMG Document, 95-09-12

[OMG 1997-1] Object Management Group (OMG), A Discussion of the Object Management Architecture, Object Management Group, Inc., 1997

[OMG 1997-2] Object Management Group (OMG), Common Business Objects, Business Object Domain Task Force, OMG Document, 97-12-04

[OMG 1999] Object Management Group (OMG), Business Object Concepts, Business Object Domain Task Force, OMG Document, 99-01-01

[Oren and Birta 1995] T. Oren and L. Birta, Petri-Nets and Simulation: A Tutorial, Summer Computer Simulation Conference (SCSC 1995), July 1995, Ottawa, Canada

[Orfali and Harkey 1997] R. Orfali and D. Harkey, Client/Server Programming with JAVA and CORBA, John Wiley & Sons, Inc., 1997

[Orfali et al. 1996] R. Orfali, D. Harkey and J. Edwards, The Essential Distributed Objects Survival Guide, John Wiley & Sons, Inc., 1996

[Orfali et al. 1999] R. Orfali, D. Harkey and J. Edwards, Client/Server Survival Guide, John Wiley & Sons, Inc., 1999

[Peterson 1981] J. Peterson, Petri-Net Theory and the Modelling of Systems, Prentice-Hall, Inc., 1981

[Petri-Nets] Petri-Nets, http://www.petrinets.org

[Ramackers and Clegg 1995] G. Ramackers and D. Clegg, Object Business Modelling, requirements and approach, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'95), October 1995, Austin, USA

[Rumbaugh 1991] J. Rumbaugh, Object-Oriented Modelling and Design, Prentice-Hall, Inc., 1991

[SAP] Systems, Application and Products in data processing (SAP), http://www.sap.com

[Schmid and Simonazzi 1998] H. Schmid and F. Simonazzi, Business Procedures are not Represented Adequately in Business Application Frameworks!, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98), October 1998, Vancouver, Canada

[Schmid et al. 1998] H. Schmid, M. Riebisch, T. Heverhagen and H. Liessmann, A Business Object Framework Architecture, Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98), October 1998, Vancouver, Canada

[Semaphore 1997] Semaphore, Building Business Objects with Distributed Object Computing, 1997

[SF] IBM SanFrancisco (SF), http://www.ibm.com/Java/Sanfrancisco

[Shelton 1995] R. Shelton, Business Objects, Open Engineering Inc., 1995

[Shlaer and Mellor 1988] S. Shlaer and S. Mellor, Object-Oriented Systems Analysis: Modelling the World in Data, Prentice-Hall, Inc., 1988

[Shneiderman 1996] B. Shneiderman, The eyes have it: A task by data type taxonomy of information visualizations, IEEE Symposium on Visual Languages, September 1996, Los Alamos, USA

[Sim 1996] E. Simon, Distributed Information Systems: From client/server to distributed multimedia, McGraw-Hill International Ltd., 1996

[Skillicorn 1998] D. Skillicorn, A Taxonomy for Computer Architectures, Computer, IEEE, November 1988

[Skörd 1998] P. Skörd, Reuse and characteristics of reusable classes, master thesis, Växjö University, June 1998

[Stacey 2001] M. Stacey, Component Design, Architectural Press, 2001

[Sullo 1994] G. Sullo, Object Engineering: Designing Large-Scale Object-Oriented Systems, John Wiley & Sons, Inc., 1994

[Sutherland 1995] J. Sutherland, The Object Technology Architecture: Business Objects for Corporate Information Systems, Symposium for VMARK Users, November 1995, Albuquerque, USA

[Sutherland et al. 1997]	J. Sutherland, D. Patel, C. Casanave, J. Miller and G. Hollowell, Business Object Design and Implementation, Springer, 1997
[Szyperski 1998]	C. Szyperski, Component Software: beyond Object-Oriented programming, ACM, 1998
[Taylor 1995]	D. Taylor, Business Engineering with Object Technology, John Wiley & Sons, Inc., 1995
[Tudor and Tudor 1995]	D. Tudor and I. Tudor, Systems Analysis and Design: A Comparison of Structured Methods, Blackwell Publishers Inc., 1995
[Vaessen 2001]	R. Vaessen, Business Logic Extraction, master thesis, Eindhoven University of Technology, June 2001
[XML]	Extensible Markup Language (XML), `http://www.xml.com`
[XML-W3]	Extensible Markup Language - the World Wide Web consortium (XML-W3), `http://www.w3.org/XML`
[Yin 1994]	R. Yin, Case Study Research: Design and Methods, Sage Publications, 1994

SAMENVATTING

HET ONDERZOEK

Midden jaren zeventig begon men met het toepassen van objectgeoriënteerde technologie voor het modelleren van systemen en het realiseren van softwaresystemen, en sinds het begin van de jaren negentig is deze technologie algemeen geaccepteerd. Meer recentelijk begon de term "Business Objects (BOs)" veelvuldig te verschijnen in de literatuur over informatiesystemen, software engineering en de corresponderende gebieden. Deze term wordt gebruikt om verschillende ideeën te representeren en wordt geassocieerd met verschillende onderwerpen. Vandaar dat basiskennis over het kernconcept dat door deze term is (of moet) gerepresenteerd (worden) praktisch is of zelfs noodzakelijk. Tot deze kennis behoort de rol die het concept van Business Objects speelt in informatiesystemen en haar waarde voor software engineering. Dit onderzoek beoogt deze kennis te verschaffen.

Om zulke kennis te kunnen verschaffen, moet men alle verschillende aspecten betreffende het concept Business Objects bestuderen en deze aspecten met hun onderlinge samenhang behandelen. Alleen op deze manier kan de (vermeende) toegevoegde waarde van Business Objects voor software engineering en het realiseren van informatiesystemen beseft en gerealiseerd worden. Dientengevolge is het hoofddoel van dit onderzoek: het verschaffen van een veelomvattend en geïntegreerd beeld van Business Objects. Om die reden streeft dit onderzoek ernaar het kernconcept van Business Objects te identificeren door middel van een grondige discussie over de verschillende aspecten ervan en naar het leggen van logische verbanden tussen die aspecten.

Om het genoemde doel te kunnen bereiken, moet men echter alle betreffende aspecten op een abstract niveau behandelen. Dit is noodzakelijk om de verbanden tussen die aspecten toe te kunnen lichten terwijl de geldigheid van de besproken kwesties gewaarborgd kan blijven. Bovendien is de verworven kennis bruikbaar voor het bestuderen van de literatuur over specifieke kwesties.

DE VRAAGSTELLING

Om een veelomvattend en geïntegreerd beeld van Business Objects te kunnen verschaffen, moet men de betreffende aspecten in een georganiseerde manier met betrekking tot de (vermeende) toegevoegde waarde van Business Objects voor software engineering en het realiseren van informatiesystemen bespreken en behandelen. Derhalve formuleren we de fundamentele vraagstelling van dit onderzoek als volgt:

Hoe kan het concept Business Objects bijdragen aan de vooruitgang op het gebied van "informatiesystemen engineering"?

Vervolgens moet het onderzoek zich richten op het concept Business Objects met betrekking tot haar relatie met de ontwikkeling en het onderhoud van softwaresystemen. Het moet bestudeerd worden of de toepassing van Business Objects de systeemontwikkeling kan verbeteren en het benodigde systeemonderhoud kan terugdringen.

De realisering van het onderzoeksdoel kan verder worden bewerkstelligd door het formuleren van een aantal kernvraagstukken, die van de fundamentele vraagstelling afgeleid kunnen worden. Deze subvraagstellingen moeten zich enerzijds richten op de hoofdkwesties behandeld in de literatuur, en anderzijds richten op de kwesties die niet (voldoende) behandeld zijn.

Vandaar dat dit onderzoek zich richt op de volgende vragen:

- **Wat zijn Business Objects?**
- **Wat is de plaats van Business Objects?**
- **Op welke manier kunnen Business Objects gecreëerd worden?**
- **Op welke manier kunnen Business Objects gebruikt worden?**
- **Wat zijn de hulpmiddelen en gereedschappen voor Business Objects?**
- **Op welke manier kunnen Business Objects toegepast worden?**

Als uitgangspunt voor de beantwoording van de genoemde vraagstellingen gebruiken we de volgende hypotheses:
- Een universeel aanvaardbare definitie van Business Objects kan geformuleerd worden.
- Business Objects kunnen voordelen hebben voor het realiseren van informatiesystemen.
- Het concept Business Objects kan een centraal punt vormen voor een bepaalde discipline van het gebied van software engineering. Deze discipline kan rond een aantal architectonische en bouwkundige aspecten met betrekking tot informatie systemen gevormd worden.

Vanwege de aard van dit onderzoek zijn literatuurstudie en casestudie de meest geschikte onderzoeksmethoden voor het beantwoorden van de vragen.

DE UITVOERING

In de literatuur wordt de term "Business Objects" hoofdzakelijk gebruikt om de toepassing van objectgeoriënteerde technologie voor het modelleren van "business" systemen en het realiseren van business softwaresystemen (in gedistribueerde omgevingen) te representeren. Er zijn echter veel verschillende, uiteenlopende en (zelfs) elkaar tegensprekende definities van deze term, en er ontbreekt een heldere en overal geldende definitie. Dit vormt een probleem voor het onderzoek over (en de toepassing van) Business Objects. Na een discussie over het concept "object" en het categoriseren van objecten op basis van hun rol in informatiesystemen en het doornemen van representatieve definities voor Business Objects in de literatuur, formuleren we een definitie van dit concept en geven we de belangrijkste voordelen daarvan aan.

Om de rol van Business Objects in informatiesystemen te specificeren, bepalen we de plaats van Business Objects met betrekking tot de bestaande architecturen. Derhalve bestuderen we eerst de basistypen van architecturen, namelijk "Monolithic" en "Client/Server" architecturen. Dan bestuderen we de "Three-Layer Architecture", een specifiek soort Client/Server architectuur. Vervolgens definiëren we de "Five-Layer Architecture". Deze architectuur breidt de Three-Layer Architecture uit en pakt haar tekortkomingen aan. Daarna is de "Multi-Tier Architecture" gedefinieerd. Dan beoordelen we de rol en plaats van Business Objects binnen de genoemde architecturen, en in het bijzonder in de Five-Layer Architecture. Uiteindelijk wordt het concept "component" behandeld, de relatie tussen component en Business Objects besproken en het concept "Business Object Components" gedefinieerd.

Business Objects moeten gecreëerd worden in overeenstemming met hun beoogde kenmerken en voordelen en hun locatie met betrekking tot informatiesystemen; alleen op deze manier kunnen ze ons doelmatig en doeltreffend helpen in het modelleren van (business) systemen en het realiseren van (business) softwaresystemen. Om die reden behandelen we de hoofdkwesties voor het creëren van Business Objects. Dan bespreken we het "beschrijven" van Business Objects en presenteren we een algoritme hiervoor. Daarna bespreken we het opbouwen van Business Objects en definiëren we de concepten "Business Object Blocks", "Business Core Components" en "Business Core Blocks". Vervolgens bespreken we standaardisatie en categoriseren van Business Objects. We beschouwen standaardisatie als een belangrijk kenmerk van Business Objects, en een voordeel van dit concept ten opzichte van andere (objectgeoriënteerde) methoden voor het realiseren van softwaresystemen. Daarnaast beschouwen we categoriseren als een handig hulpmiddel voor het gebruik van Business Objects bij het realiseren van softwaresystemen.

Verder wordt het gebruik van Business Objects in het kader van de Five-Layer Architecture behandeld. In dit verband bespreken we de hoofdkwesties, de lagen van deze architectuur en de relatie tussen deze lagen. Daarnaast bespreken we een aantal "opties" met betrekking tot de relatie tussen Business Objects en "data sources", en verschillende "perspectieven" op basis waarvan Business Objects kunnen worden gecreëerd.

Gebruik makend van de besproken onderwerpen over Business Objects kijken we naar een aantal representatieve omgevingen voor het realiseren van softwaresystemen op basis van Business Objects, en technologieën die deze omgevingen hiertoe in staat stellen.

Vervolgens tonen we de toepassing van Business Objects aan de hand van een casestudie en illustratieve voorbeelden. De casestudie is uitgevoerd in een bedrijf, in het kader van de realisering van een informatiesysteem dat gebaseerd is op de Three-Layer Architecture en gebruik maakt van Business Objects voor het "middel layer". De eerste twee voorbeelden laten zien hoe Business Objects en de Five-Layer Architecture kunnen worden toegepast in twee gebieden, namelijk de gezondheidszorg en het bankwezen, om aan hun specifieke eisen tegemoet te komen. De volgende twee voorbeelden tonen aan hoe Business Objects en de Multi-Tier Architecture in het algemeen kunnen worden gebruikt voor systeemintegratie en simulatie.

Acknowledgements

I would like to thank Prof. Jan Dietz for his productive supervision and support.

Thanks are due to Prof. Waltraud Gerhardt, Prof. Henk Koppelaar, Prof. Maarten Looijen, Prof. Vaclav Repa, Prof. Roberto Zicari and Dr. Eugen Kerckhoffs for all their constructive comments and valuable remarks.

I want to express my gratitude to all of the members of the promotion committee for their cooperation and participation.

I thank Mrs. Mirjam Nieman for her friendly assistance and effort in checking the text.

Mohammad Abolhassani
January 2003, Dortmund, Germany

About the author

Mohammad Abolhassani was born on the 10th of January, 1962 in Teheran, Iran. In 1979 he finished his secondary school and began his higher education studies. After an interruption from 1980 to 1983 he continued his studies and in 1987 he got his diploma in Computer Hardware from Shahid Beheshti University (former National University of Iran). From 1986 to 1992 he worked in different positions such as programmer, teacher and system analyst.

In 1993 he began his studies at Delft University of Technology, the Netherlands, and in 1997 he graduated in Technical Informatics, in the Knowledge-Based Systems group. From December 1997 to November 2001 he was doing his PhD research at Delft University of Technology, in the Information Systems group.

In February 2002 he joined the Information Retrieval group at the University of Dortmund, Germany. In January 2003, together with the group, he moved to Duisburg-Essen University.