Chapter 6
Enterprise Applications: From Requirements to Design

Christine Choppy
LIPN, University Paris 13, France

Denis Hatebur
University Duisburg-Essen, Germany

Maritta Heisel
University Duisburg-Essen, Germany

Gianna Reggio
Universita di Genova, Italy

ABSTRACT

The authors provide a method to systematically develop enterprise application architectures from problem descriptions. From these descriptions, they derive two kinds of specifications: a behavioral specification describes how the automated business process is carried out. It can be expressed using activity or sequence diagrams. A structural specification describes the classes to be implemented and the operations they provide. The structural specification is created in three steps. All the diagrams are expressed in UML.

1. INTRODUCTION

We provide a method to systematically develop enterprise application architectures from problem descriptions. The problem descriptions are based on Jackson's problem frame approach (Jackson, 2001). For enterprise applications, we have developed a specialized problem frame that takes the specifics of such applications into account (Choppy & Reggio, 2006). In particular, new domain types – such as *business worker* and *business object* – are introduced. We describe the requirements through problem diagrams that are instances of the enterprise application frame and of other problem frames.

In addition to Jackson's problem frame approach, we represent the business model underlying the business application by a domain

DOI: 10.4018/978-1-4666-2199-2.ch006

knowledge diagram. Such a diagram identifies the domains relevant for the business process to be automated and states how they are related. It serves to analyze the business process to be automated and helps to construct the appropriate instances of the enterprise application frame, thus obtaining the problem diagrams.

From the problem diagrams, we derive two kinds of specifications: a behavioral specification describes how the automated business process is carried out. It can be expressed using activity or sequence diagrams. A structural specification describes the classes to be implemented and the operations they provide. This specification is expressed as a class diagram, where all operations are specified in OCL.

With these different models, we have described the software development problem in a detailed way, taking into account the specifics of business applications. This makes it possible to develop a suitable software architecture in a systematic way. We proceed similarly as described in earlier work (Choppy, Hatebur, & Heisel, 2011), where we derive software architectures from problem descriptions for arbitrary software. In this work, we make use of the fact that the software development problem is decomposed in subproblems that fit to the enterprise application frame. Taking into account the identified business objects, we focus on how these objects can best be stored and accessed. In this context, questions of distributing the software to be developed and the information to be stored are addressed, too.

In a first step, we create an initial architecture. To obtain that architecture, we first have to decide on the responsibilities of the software to be developed (which is called *machine* in the problem frames approach). We have to inspect all domains occurring in the problem diagrams and decide if they will be part of the machine or not. Some domains may reside in the environment but still need to have an internal representation in the machine. The rules given in (Choppy & Reggio, 2006) support this task. Moreover, the initial architecture reflects the problem decomposition that was obtained by applying the problem frame approach. Each subproblem machine becomes a component in the initial architecture.

The initial architecture need not be implementable, because the interaction between the different components has not yet been taken into account. Therefore, we transform the initial architecture into an implementable architecture. To create the implementable architecture we have to consider technical requirements, for example that some functionality should be implemented on another computer. In business applications, there is usually a database, which may be located on a different computer than the machine we are building. In that case, we have to split the problem diagrams and create corresponding subproblem diagrams that separate the different machines accordingly. Conversely, in many cases, only one database is used to store different kinds of information, such that the components representing the different domains have to be merged into the database component. Moreover, we introduce coordinator components, considering the formal descriptions of the business model, and facade components. Finally, we allocate all machines that solve the different problems or subproblems and the considered domains to physical components of the machine to be built.

The implementable architecture that is obtained in this way does not follow a particular architectural style. If, for example, a layered architecture is wished for, the implementable architecture can further be transformed into a layered one, as is described in (Choppy et al.., 2011).

All the diagrams that we present in this chapter are expressed in UML instead of the original notation used by Jackson (Jackson, 2001). This makes it possible to exploit the additional expressive power provided by UML class diagrams and also to represent software architectures and problem descriptions in the same notational framework. Carrying over the problem-frame approach to UML is achieved by defining a UML profile for