Chapter 4
Derivation of Use Cases from the Topological Computation Independent Business Model

Janis Osis
Riga Technical University, Latvia

Erika Asnina
Riga Technical University, Latvia

ABSTRACT

In Model Driven Architecture (MDA), business requirements for the information system are described in a Computation Independent Model (CIM), which additionally can describe knowledge of the business, and structure and behavior of both business and supporting information system. In object-oriented software development requirements are described by use cases. Use cases and identification of them are informal and application-oriented. Goal-based approaches provide a more systematic way for discovering use cases from informal knowledge about a system. The main and very important difference of the approach suggested in this chapter is that we ground our domain analysis on a mathematical engineering model, Topological Functioning Model. It is a formal holistic computation independent business model, whose characteristics help in avoiding challenges in functional requirements caused by non-systematic approaches and fragmental nature of use cases, namely, completeness, traceability and compliance with the problem domain.

INTRODUCTION

Model-Driven Architecture (MDA) developed by the Object Management Group (OMG) proposes three (or at least two) transformable models for system specification. The first one is a Computation Independent Model or CIM. The CIM is a model that should eliminate the gap between business people and software developers. Two other models are a Platform Independent Model (PIM) and a Platform Specific Model (PSM). The last two models specify the system structure and behavior according to the object-oriented paradigm and definitely are transformable.

This chapter discusses the CIM. This model specifies domain information: business vocabulary, business rules, business knowledge, system requirements (in a broad sense), etc. This means

DOI: 10.4018/978-1-61692-874-2.ch004

it may be as transformable as non-transformable. Moreover, this model may have different users and thus may describe different domains – a problem as well as a solution.

Summarizing authors' viewpoints in (Hendryx, 2003a; Hendryx, 2003b; Grangel, Chalmeta, & Campos, 2007; Che, Wang, Wen, & Ren, 2009; Jeary, Fouad, & Phalp, 2008), the CIM can contain three parts: CIM-Knowledge Model, CIM-Business Model, and CIM-Business Requirements for the System.

The *CIM-Knowledge Model* reflects an enterprise from the holistic point of view, thus providing the general vision of the enterprise with focus on enterprise knowledge. It reflects the problem domain.

The *CIM-Business Model* focuses on the business scope and goals as well as terminology, resources, facts, roles, policies, rules, organizations, locations and events of concern to the business. It does not reflect considerations of the computerized information system. However, the scope of the Business Model in the CIM must include, at a minimum, those business functions which are planned to be served by the computerized system. Thus, this model reflects both problem domain and solution domain.

The *CIM–Business Requirements for the System* contains the contract between the business and IT about what the business people expect the computerized information system to do. It reflects functional and non-functional requirements for the computerized system. Thus it reflects the solution domain.

Thus, models used for business analysis and requirements gathering/specification may be used as the CIM within MDA. However, if we speak about the object-oriented paradigm in software development, these models usually are expressed by natural language in an informal way as somehow structured text. This means that correspondence between the solution and the problem domain is intuitive and weak. If we want to incorporate object-oriented software development (OOSD) with MDA and all its models, greater attention to problem domain analysis should be devoted.

There are two fundamental aspects for system modeling that need to be distinguished. The first one is *analysis*, which defines *what* an application has to do with a problem domain to fit customer's requirements. The second one is *design*, which defines *how* an application will be built. This means that at the analysis stage, a developer breaks a complex system under consideration into smaller parts to gain a better understanding of it, and at the design stage he/she makes a synthesis (combines) these separated and analyzed parts into the whole to reproduce the complex system under consideration as complete as possible. Therefore, at the beginning a developer considers each part independently of others, and only then he/she looks for relationships between those parts. A line between the analysis and design is fuzzy in the OOSD. Besides that, analysis implies that a software developer should analyze client's current situation as precise as possible, gather requirements to the system planned to build and implement them in some suitable modeling language.

Usually, requirement analysis is erroneously considered as determination of what software the client *wants*. Actually, requirements should determine what software the client *needs* (Schach, 1999). And, as Michael Jackson stated in (1999; 2005), not only what software the client needs, but also within what environment this software will work.

The OOSD does not provide such a thing as an "object-oriented requirement". Therefore, requirements are gathered and then specified in the analysis phase in order to determine functionality of the planned system. Object-oriented analysis (OOA), an initial stage of the OOSD, usually is a set of semiformal specification techniques that contains in some order three core steps, which are related to each other, but at the same time they can be performed in parallel (Schach, 1999; Jacobson, Christerson, Jonsson, & Overgaard, 1992):