Chapter 2.1
Ontology Based Object-Oriented Domain Modeling:
Representing Behavior

Joerg Evermann
Memorial University of Newfoundland, Canada

Yair Wand
The University of British Columbia, Canada

ABSTRACT

An important step in developing the requirements for an information system is analyzing the application domain. In this step, conceptual models are used for representing an application domain. However, while languages for software design are available and widely used, no generally accepted language exists for conceptual modeling. This work suggests the use of object-oriented software modeling languages also for conceptual modeling. Such use can support a more accurate transition from domain models to software models. As software-modeling languages were not intended for modeling application domains, their constructs lack the required semantics. While previous papers addressed the representation of structural elements of domains using object concepts, this paper addresses behavioral aspects, related to change and interaction. The proposed semantics are based on a mapping between ontological concepts that describe behavior and object-oriented constructs related to dynamics. Based on these mappings, modeling rules are proposed to guide the modeler in creating ontologically well-formed models. The mappings and rules are exemplified using UML and are demonstrated using a case study.

INTRODUCTION

A good understanding of the application domain is necessary to develop the requirements for information systems (IS). Such understanding can be facilitated with the use of conceptual models. Conceptual modeling is the "activity of formally describing some aspects of the physical and social world for the purpose of understanding" (Mylopoulos, 1992).

Despite possible benefits to IS development of using conceptual models, no widely used formal or semi-formal language for conceptual modeling exists. In contrast, formal and semi-formal languages, notably object-oriented languages, are commonly used in software design. As reported in (Dobing & Parsons, 2006, 2008) and also found in our case study (Sec. 8), practitioners, for lack of a language specific to conceptual modeling, have been using software design languages for this purpose. However, this often occurs in an unguided way, possibly leading to confusion and difficulties in understanding. Without guidance, the support of UML for describing domains other than software is poor and this can lead to miscommunication (Smolander & Rossi, 2008).

Adopting widely used and well-accepted object-oriented languages, usually employed for software design, in a guided way and with clearly specified semantics for conceptual modeling, has several potential benefits: (1) It can provide a shared language to support better communication between analysts and software designers. (2) It can help mitigate translation problems between the conceptual and the software models, (also called "impedance mismatch" (Cilia, Haupt, Mezini, & Buchmann, 2003; Kolp, Giorgini, & Mylopoulos, 2002; Roe, 2003; Rozen & Shasha, 1989). More specifically, because the domain model is specified in the same language as software, the domain model can also serve as an initial model of the software system (Coad & Yourdon, 1991), which can subsequently be adapted to particular technologies. Such technology-driven refactoring is beyond the scope of this paper. The discussion in Section 9 will revisit this point in more detail. (3) A clear representation of application aspects can reduce possible confusion of business and implementation aspects in conceptual models (Parsons & Wand, 1997). (4) Assigning semantics to language constructs for domain representation purposes can provide modeling rules (Evermann & Wand, 2005a).

Because object-oriented languages were not developed for conceptual modeling, they lack application domain semantics. For example, while language constructs such as "Method" or "Operation" have clear meaning for software design, it less clear what they represent in the application domain. However, assigning application domain semantics to language constructs, while necessary for their use in application domain modeling, is insufficient. It is also desirable to identify *modeling rules* to ensure that the created models represent only really possible situations in the application domain. Modeling rules can improve the ability to communicate and reason about the domain by restricting the possible interpretations of a model (Hadar & Soffer, 2006), and hence can support convergence of the domain understanding among different stakeholders, a pre-requisite for development and implementation success. Therefore, such rules can improve the effectiveness of the created models as ways to communicate and reason about the domain (Reinhartz-Berger & Sturm, 2008).

Previous research (Evermann & Wand, 2005b) proposed the use of object-oriented design languages for modeling the structural aspects of application domains. That research proposed specific application domain semantics for the static structure constructs found in UML class diagrams, and suggested modeling rules to develop well-formed and meaningful (with respect to perceptions of the real world application domain) models. The present work addresses the behavioral aspects of conceptual modeling, focusing on constructs to describe change and interaction. We exclude use case related constructs as they describe external interactions with a system, whereas the remaining UML constructs describe the system itself.

Our approach is based on the use of ontology, a specification of concepts that exist in a domain. Previously, ontologies have been used mostly to *evaluate* modeling languages (Green & Rosemann, 2000; Opdahl & Henderson-Sellers, 2002; Opdahl, Henderson-Sellers, & Barbier, 1999).