Object-Oriented Analysis and Top-Down Software Development

Dennis de Champeaux
HP-Labs
1501 Page Mill Rd, 1U
Palo Alto, CA 94304-1181
USA

Abstract

In this paper, we address the issue of how to provide an analyst that uses the object-oriented paradigm with a top-down approach. An analyst gets this approach for free when working within the structured paradigm. Ensembles are introduced that differ from objects in that they connote entities with internal parallelism. Preliminary experimentation suggests that ensembles allow for information hiding.

electronic address: champeaux@hplabs.hp.com
telephone #: (415) 857 6674
key words: OO-Analysis, top-down, ensemble
1 Introduction

In this paper, we outline a top-down object-oriented analysis (OOA) method. Top-down OOA allows an analyst to employ well-established strategies like divide-and-conquer.

We start by clarifying some of our terminology:

Analysis is the activity that yields a description of what a target system is supposed to do; detailing functional, performance and resource requirements. This description could be the basis for a contract between the client and the developer and aims to be the unambiguous input to the designer.

Design is the activity which yields an artifact description of how a target system will work. The design satisfies the requirements, while it is still implementation language independent. The artifact description aims to be the unambiguous input to the implementor.

Object-oriented analysis describes a target system with a characterization of the entities in the domain, their inherent interrelationships, and their intended behavior in isolation as well as their interactions. The order in which these aspects are addressed varies, but usually the entity characterization precedes the behavior description. This contrasts with the order in which structured analysis deals with these aspects; behavior first and entity characterization (data dictionaries) second.

Our work is grounded on the assumption that neither structured analysis nor structured design provide a natural characterization for subsequent implementation in an object-oriented language, as supported by experience in Hewlett-Packard. At the same time, we do not suggest that object-oriented analysis and design make sense only when a subsequent implementation employs an object-oriented programming language.

The necessity of analyzing a system in a top-down fashion arises specially in the characterization of large systems. While the analysis of a toy example like the popular car cruise control system yields only a "flat" set of objects, the analysis of a corporation like Hewlett-Packard, an airline reservation system or a bank will yield "objects" at different abstraction levels.

The problem that we encounter is caused - we conjecture - by an uncritical adoption of the notion of object from the realm of the object-oriented programming languages. We suspect that this is the core reason why identifying objects is a hard task. We can wonder for instance whether the following notions are proper objects:

In the realm of Hewlett-Packard:
- a division, a department, an employee, a project, a production unit, a product, an order,
- a floor in a building, a location code, etc.

In the realm of an airline system:
- a flight, an airplane, a flight attendant, a client, a flight schedule, a special meal order,
- a service schedule, a luggage door, a payment scale, etc.

In the realm of a bank:
- an interest rate, a branch office, a teller machine, a corporate account, a loan officer, the overseas department, a monthly statement, etc.