Chapter II

Tropos: A Requirements-Driven Methodology for Agent-Oriented Software

Paolo Giorgini
University of Trento, Italy

Manuel Kolp
University of Louvain, Belgium

John Mylopoulos
University of Toronto, Canada

Jaelson Castro
Federal University of Pernambuco, Brazil

Abstract

Software systems of the future will have to perform well within ever-changing organizational environments. Unfortunately, existing software development methodologies have traditionally been inspired by programming concepts rather than organizational ones, leading to a semantic gap between the system and its operational environment. To reduce this gap, we propose a software development methodology named Tropos that is founded...
on the i* organizational modeling framework. i* offers the notions of actor, goal, and (actor) dependency. Tropos uses these concepts as a basis to model early and late requirements, architectural design, and detailed design for a software system. The chapter outlines Tropos phases through an e-business example. The methodology is a good complement to proposals for agent-oriented programming platforms.

Introduction

Organizational software systems have traditionally suffered from an impedance mismatch between their outer and inner environments: their operational environment is understood in terms of actors, responsibilities, objectives, tasks, and resources, while their inner environment is conceived as a collection of (software) modules, entities (e.g., objects, components), data structures, and interfaces. This mismatch contributes greatly to poor software quality and to frequent failures of software system development projects.

One reason for this mismatch is that development methodologies have traditionally been inspired and driven by the programming paradigm of the day. So, during the era of structured programming, structured analysis and design techniques were proposed (DeMarco, 1978; Yourdon & Constantine, 1979), while object-oriented programming has given rise more recently to object-oriented design and analysis (Booch, Rumbaugh, & Jacobson, 1999; Wirfs-Brock, Wilkerson, & Wiener, 1990). For structured development techniques, this meant that, throughout software development, the developer could conceptualize the system in terms of functions and processes, inputs and outputs. For object-oriented development, on the other hand, conceptualizations consist of objects, classes, methods, inheritance, and the like, throughout.

Using the same concepts to align requirements analysis with system design and implementation makes perfect sense. For one thing, such an alignment reduces impedance mismatches between different development phases. Moreover, such an alignment can lead to coherent toolsets and techniques for developing software. In addition, it can streamline the development process itself.

However, why base such an alignment on implementation concepts? Requirements analysis is arguably the most important stage of software development. This is the phase where technical considerations have to be balanced against social and organizational ones. This is also the phase where the operational environment of the system is modeled and analyzed. Not surprisingly, this is also the phase where the most and costliest errors are introduced compared to other development phases. Even if (or rather, when) the importance of design and