Available online at www.sciencedirect.com
SCIENCE DIRECT®
Science of Computer Programming 56 (2005) 5–21
Science of Computer Programming
www.elsevier.com/locate/scico

Use-case components for interactive information systems

Eliezer Kantorowitz*, Alexander Lyakas

Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel

Received 28 November 2003; received in revised form 31 August 2004; accepted 6 September 2004

Abstract

Specification-oriented components (SOC's) are designed to facilitate the implementation of a system directly from its specifications. An earlier study has shown cases in which SOC's enabled information systems to be implemented with considerably less code than when implemented with components designed by a typical object-oriented approach. This study goes a further step by considering the essence of an information system to be the flow and processing of data. The components based on this abstraction attempt to hide code that is not implementing data flow or data processing. Based on this approach, an experimental framework called $\mathcal{WebSI}$ has been developed. $\mathcal{WebSI}$ components hide the code for the construction of the user interface (UI), the database access code and the Web-related code. $\mathcal{WebSI}$ was designed to facilitate the manual translation of English language use-case specifications into Java code. $\mathcal{WebSI}$ enabled the construction of information systems with a modest amount of code. The similarity between the $\mathcal{WebSI}$-based Java code and the English language use-case specifications facilitated verifying that the code implements the specifications correctly. The automatically produced UI's were relatively easy to learn and to use. The modification of $\mathcal{WebSI}$-based legacy code was facilitated by the high level of the code and its use-case structure, but remained a labor-intensive task.

Keywords: Component; Specification-oriented; Specification-oriented component; Use case; Information system; User interface; World Wide Web; Web

* Corresponding author.
E-mail addresses: kantor@cs.technion.ac.il (E. Kantorowitz), lyakasal@cs.technion.ac.il (A. Lyakas).
URLs: http://www.cs.technion.ac.il/~kantor (E. Kantorowitz), http://www.cs.technion.ac.il/~lyakasal (A. Lyakas).

0167-6423/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2004.11.002

1. Introduction

A software component can be defined as a piece of software which is designed such that it can be reused in many different systems. The design of components that meet this requirement has proven to be a difficult task. These difficulties may be alleviated by designing components which are limited to a particular application domain and to a particular method of combining the components into a software system. Such a set of components and the corresponding construction method are sometimes called a component *framework*.

One of the measures of quality of a framework is the developer's effort required to design, implement and test a software system with the help of the framework, i.e., the *development costs*. Another quality measure of a framework is the developer's effort required to extend the software, produced with the help of the framework, at a later point in time, i.e., the *extension costs*. This quality may be expressed by the *extension complexity* [10] of the software produced with the framework. A further quality measure of a framework is whether it facilitates manufacturing software systems that have a high *usability* level. Usability expresses both the extent to which the system provides the functionalities that the users need, and whether the system enables the users to accomplish their work with a minimum of human effort and in a pleasant way. Usability is possibly the most important system property, as it may be the determining factor for whether or not the software will sell.

This paper focuses on designing frameworks facilitating the construction of high-usability software systems at low development and extension costs. Other aspects of component design, e.g., the problems of interfacing components from different sources, are not considered in this paper.

One approach in frameworks design is to use a classical object-oriented (OO) design methodology, i.e., to construct a component for each one of the important objects in the application domain. An example of such a framework is the Java's Swing package for graphical user interfaces (GUI) construction. This package contains components for such GUI controls as buttons, menu entries and text fields. A GUI is constructed by combining these components in a special way. The construction involves the specification of the size of the controls, their colors and the layout on the screen. The programmer must also code the operations carried out by the software, when a particular GUI control is operated by the user. Programming at this level of detail can be quite labor intensive.

Another approach in frameworks design is the *specification-oriented* approach, suggested in [12], where the framework's components are designed to enable a direct coding of the system from its specifications. The possibility of deriving an implementation of a system directly from its specifications was demonstrated for reactive systems, in which a state chart specification of such a system can be executed [4]. The developer of such a system has thus only to specify it by its state chart, and need not invest further time in designing and implementing the system. The state chart can be executed, because it explicitly specifies all the computations in the system. Our study considers large complex information systems, for which it is difficult to produce such complete specification of the entire system, as done in the state chart approach. A common industrial method to manage this high complexity is to develop such a system through a software development process