Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 50 (2004) 367–382
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Relating evolving business rules to software design

W.M.N. Wan-Kadir [1], Pericles Loucopoulos *

Department of Computation, University of Manchester Institute of Science and Technology (UMIST), P.O. Box 88, Manchester M60 1QD, UK

Available online 27 November 2003

Abstract

In order to remain useful, it is important for software to evolve according to the changes in its business environment. Business rules, which can be used to represent both user requirements and conditions to which the system should conform, are considered as the most volatile part in today's software applications. Their changes bring high impact on both the business processes and the software itself. In this paper, we present an approach that considers business rules as an integral part of a software system and its evolution. The approach transcends the areas of requirements specification and software design. We develop the Business Rule Model to capture and specify business rules, and the Link Model to relate business rules to the metamodel level of software design elements. The aim is to improve requirements traceability in software design, as well as minimizing the efforts of software changes due to the changes of business rules. The approach is demonstrated using examples from an industrial application.

Keywords: Software evolution; Business rules; Software architecture/design; User requirements

1. Introduction

In today's business environment of relentless change, software evolution is inevitable since changes generated by business policies and operations need to be propagated onto the support software system. A software system is directly related to the business system within which it operates and is thus a manifestation of some business requirements for operational control and support of decision making. A large portion of total software lifecycle cost is devoted to introducing new requirements and removing or changing existing requirements. It is therefore important to consider development paradigms that attempt to improve effectiveness and efficiency of software in situations that legacy systems need to evolve.

The need for dealing with software evolution has led researchers and practitioners to develop techniques that strive to build software systems that can be adaptive to changes [9]. Such techniques propose a software model, or architecture, that has the ability to minimize the effect of

* Corresponding author. Tel.: +44-161-200-3332; fax: +44-161-200-3378.
E-mail address: peri.loucopoulos@co.umist.ac.uk (P. Loucopoulos).
[1] W.M.N. Wan-Kadir is currently on research leave at UMIST, Manchester. His permanent address is Software Engineering Department, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia.

1383-7621/$ - see front matter
doi:10.1016/j.sysarc.2003.09.006

changes as well as providing requirements traceability in their model. A number of different technologies are utilized such as object-oriented, distributed system, software architecture and component-based technologies.

In this paper we explore a complimentary technique that focuses on enterprise knowledge and its relationship to a software system [22,25, 32,43].

To address the problem of software evolution successfully, we propose that developers must be provided with a process which identifies areas of potential change and offers explicit representation of these *volatile concepts*. Such a paradigm would provide the means of maintaining specifications at a high level, minimizing the effect of changes on software maintenance. We define these volatile concepts as those aspects of the business environment which explicitly refer to policy statements about the operations of the business namely the *business rules*.

Our research attempts to address software evolution by considering business rules as a volatile part of a software system. Our work focuses on a user-driven software evolution paradigm and in particular on the way that changes in business rules affect changes to software systems. The approach known as MBRM (Manchester Business Rules Management) comprises of a number of activities, techniques and support systems. This paper focuses on one aspect namely that of *linking conceptual specifications of business rules to software designs*.

The paper is organized as follows. Section 2 introduces the problem of software evolution and its relationship to business change. It briefly overviews key developments in Software Engineering in relation to software evolvability and develops the main arguments for the business rules paradigm. Section 3 is concerned with research initiatives in business rules modelling and evolvable software architecture. Section 4 introduces the MBRM approach in terms of its main processes and the underlying metamodels. These metamodels constitute the 'theory' on which the linking between business rules and software designs in the MBRM approach is based and details on these are given in Section 5. Section 6 demonstrates the principles using a case study. Section 7 concludes the paper with a summary of the work to date and a discussion on future directions.

2. Problem background

There are many approaches which aim to enhance the evolvability of software artifacts such as the study of software architectures, distributed, object-oriented and component-based software systems. For example, refining the role of connectors makes run-time evolution of software architectures feasible [30], and introducing good abstractions of the components for composition improves software evolvability [11]. In distributed systems, 'mediator' is used as a middle component between changed server and older client [39], change absorbers in DRASTIC architecture [8], ports [26], and actor 'liaison' [5].

In object-oriented systems, the formalization of object behaviour specification [17], and the generalization of reuse contract formalism and its integration into Unified Modeling Language (UML) metamodel [27] has proved useful in supporting software evolution. Liu increases adaptability of object-oriented design against requirement changes using adaptive schema style rules [21]. The rules are used to transform any object-oriented design into a more adaptable design. Diaz et al. [7] provided a method to explicitly identify, design and implement business policies in object-oriented software systems. The explicit description of business policy that is capable to separate volatile parts from the stable ones localizes change and supports evolution. In component-based software, the separation of components, connectors and configuration [38], the decomposition into a set of components based on business consideration [18], and the use of precise specifications to determine and maintain the semantic dependencies between components [31] have proved to facilitate software evolution.

However, most of these approaches only consider the software technology aspect. There is another important aspect i.e. the *sources of changes* that should be seriously considered in order to reach at the root of the evolution problem, and to