Relating Evolving Business Rules to Software Design

Wan M.N. Wan-Kadir and Pericles Loucopoulos
Department of Computation,
University of Manchester Institute of Science & Technology
P.O. Box 88, Manchester, M60 1QD, U. K.
e-mail: w.wan-kadir@acm.org, pl@co.umist.ac.uk

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 position paper, we start our discussion with the recent approaches in evolvable software system that consider business rule as an integral aspect of their model. Next, we describe our approach that consists of a model that links business rules and software design. The link model is intended to improve requirements traceability in software design, as well as minimizing the efforts of software changes due to the changes of business rules. XML DTD is used to specify the metamodel of our solution for automation and portability purposes. Finally, we briefly discuss the future developments of the current research.

Keywords: Software Evolution, Business Rules, Software Architecture / Design, User Requirements, Metamodel.

1. Introduction

Nowadays, nearly all of the commercial and government organizations are highly dependent on software systems. Due to the inherent dynamic nature of their business environment, software evolution is inevitable. The changes generated by business policies and operations are propagated onto software system. A large portion of total lifecycle cost is devoted to introduce new requirements, and remove or change existing requirements. However, the evolution of a successful software system is inevitable for the software to remain useful in its environment.

Many research projects attempt to find a more applicable way for building a software system that is adaptive to changes [1]. They strive to propose a software model, or architecture, that has the ability to minimize the effect of changes as well as providing requirements traceability in their model. Most of them utilize the benefit of object-oriented, distributed system, software architecture and component-based technologies. Our research attempts to address these issues in a slightly different way namely by considering business rules as volatile part of a software system. In this position paper, we introduce a software architecture which includes a link model that aims to reduce the effects of software changes by improving requirements traceability.

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 [2], and introducing good abstractions of the components for composition improves software evolvability [3]. In distributed systems, ‘mediator’ is used as a middle component between changed server and older client [4], change absorbers in DRASTIC architecture [5], ports [6], and actor ‘liaison’ [7].

In object-oriented system, the formalization of object behaviour specification [8], and the generalization of reuse contract formalism and its integration into Unified Modeling Language (UML) metamodel [9] proved useful in supporting software evolution. Ling increases adaptability of object-oriented design against requirement changes using adaptive schema style rules [10]. The rules are used to transform any object-oriented design into a more adaptable design. Diaz et al. provide method to explicitly identify, design and implement business policies in object-oriented software system [11]. 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 [12], the decomposition into a set of components based on business consideration [13], and the use of precise specifications to determine and maintain the semantic dependencies between components [14] are 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 strengthen the developed technological solution. Among the important sources of changes are business rules that represent business policy which of course when it changes, the underlying software system should support that change. It is important to explicitly consider business rules in software modelling for assisting future evolution [15].

Thus, the premise of this research that the explicit consideration of business rules in software modelling may improve the evolvability of a software system is based on the fact that business rules are the most volatile component in information systems [16]. Moreover, business rule changes bring the highest impact on both software and business process compared to other changes such as altering code for elegance and readability, changing data naming convention, speeding execution, or reducing internal storage usage [17, 18].

### 3. Related Works

The approach presented here is related to other initiatives in two different areas i.e. *business rules modeling* and *evolvable software architecture*. The former is mainly concerned with the representation of business rules as structured requirements whilst the latter is concerned with the role of business rules in software design.

#### 3.1 Business Rules Modelling

**Business Rule-Oriented Conceptual Modeling (BROCOM)** provides a metamodel that formalizes business rules in conceptual modelling [19, 20]. In BROCOM, a business rule is composed of three components namely event (E) that triggers business rules, condition (C) that should be satisfied before an action, and action (A) that describes the task to be done. Thus, the business rules are specified in ECA structure. Apart from that, BROCOM also provides a metamodel that links business rules with organizational facts such as origin, processor, and organizational unit as its components.

The **Business Rules Group (BRG)** classified business rules into three main types i.e. structural assertions, action assertions, and derivations [21]. Structural assertion is a statement about concept or relationship of something of importance to the business. There are two kinds of structural assertions i.e. terms and facts. A term is a word or phrase, which has a specific meaning to business. A fact asserts an association between two or more terms. Action assertion is concerned with the dynamic aspect of the business. It includes a conditional action, integrity constraints, and optional actions. Finally, a derivation is a derived fact that is created by an inference or a mathematical calculation from terms, facts, other derivations, or action assertions. A mathematical calculation derives facts using a specified mathematical algorithm whilst an inference produces a derived fact using logical induction or deduction.

*von Halle* classified business rules into term, fact, mandatory constraint, guideline, action enabler, computation, and inference [22]. This classification can be considered as a simpler version of BRG classification scheme.

*Morgan* provides formalization in terms of the pattern of business rule statements which is capable to be translated into formal logic [23]. The basic form of the patterns is `<subject> must <constraint>`. This basic form is extended to a more complex form based on the five patterns namely basic constraint, list constraint, classification, computation, and enumeration. Morgan also suggests to link business rules to a fact model which contain business objects, their relationships, and their attributes. For a greater flexibility, he suggests to keep the classes and relationships in the fact model as simple as possible and represent their constraints as business rules.

#### 3.2 Evolvable Software Architecture

Evolvable software architecture considers business rule as a key component. Among the leading approaches is **Adaptive Object Model (AOM)**, which is defined as “a system that represents classes, attributes, and relationships as metadata” [24, 25]. It is a meta-architecture that allows users to manipulate the concrete architectural components of the model such as business objects and business rules. These components are stored in a database instead of code. Thus, a user only needs to change the metadata instead of changing the code to reflect domain changes. Strategies [26] and RuleObjects [27] are used to model complex rules which are ‘functional or procedural in nature’ [28] such as legal types of values for certain attribute, certain entity-relationships are only legal if certain values and other constraints that involves conditions. Simple rules such as defined types of entities, legal subtypes, relationships, and cardinality, are normally controlled by object-oriented modeling semantics.

**Coordination Contract** aims to separate core business entities which are relatively stable and volatile business products which keep changing for the business to remain competitive [29]. It is desirable to have a separate implementation of these two entities so that the changes are localized only to the volatile parts, with minimum impact on the core services already implemented in the system. Volatile business products are implemented as
contracts. Contract aims to externalize the interactions between objects (core entities) by explicitly define them in the conceptual model. It also provides a coordination role similar to other components in architecture-based approaches such as architectural connectors [2], glue [12], actor [7] or change absorbers [5]. By separating computation from coordination [18], the coupling between components is minimize. If there is any change to business product or its business rules, the relevant contract is 'superposed' on the existing implementation of core entities, which are considered as black boxes.

**Business Rule Beans (BRBeans),** formerly known as Accessible Business Rules [16, 30], is a framework that provides guidelines and infrastructures for the externalization of business rules in a distributed business application. Business rules are externally developed, implemented and managed to minimize the impact of their changes on other components such as core business, application, and user interface objects. They are implemented as server objects, which are fired by embedded trigger points in application objects. The rule management facility is provided to help users to understand the existing rules and to locate the rules when changes are required. BRBeans is implemented as a part of WebSphere Application Server by IBM [31]. In a textual specification, keywords such as 'except when', 'unless', and 'depend on' can be used to indicate the points of rule externalization. During implementation, each point of variability is translated into a trigger point, that is a piece of code in a rule driven object that executes business rules.

**4. The Link Model**

In developing an architecture for evolvable software systems, we found that it is important to satisfy the following two prerequisites: (i) structured business rule expressions and (ii) minimum extension to design elements. The former aims to strike a balance between natural (to reflect a user language) and formal (to reflect a machine language). The latter attempts to minimize the extension to the existing standard design metamodel elements.

In developing a set of templates for structured business rule expressions, we first identify the appropriate structural typology of business rules as well as developing a metamodel to formally specify them. The metamodel is important for the automated software development tools. Table 1 summarizes the typology which is adopted for our current purpose.

In terms of software design, we use the widely accepted method namely Unified Modeling Language (UML) [32]. We strive to minimize the extension to the standard UML modelling elements for many reasons. For example, the introduction of rule components may decrease the system performance because it would increase the number of objects and function calls during implementation. Moreover, the introduced modelling components would need new semantic definitions, which makes understanding of the design model more difficult.

In order to fill a gap between business rules and software design, we propose a link model which comprises additional information associated to each business rule such as *rule owners* and *rule set*. This information is important for business rule management since the authorized user of the intended system is allowed to add, delete or modify any rule in the rule set. In addition, the link model also maps business rules to UML model elements such as classes, relationship, and constraints. We consider only the effect of changes of business rules against software design and not vice-versa, thus one-to-many mapping is appropriate for the current purpose. XML DTD is used to specify the metamodel of UML design, link model and business rule for the purposes of automation and portability. The overall view of our proposed software architecture is shown in Figure 1.

<table>
<thead>
<tr>
<th>Types</th>
<th>Definition and Formalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>A word or phrase that relevant to the business.</td>
</tr>
<tr>
<td>Fact</td>
<td>A statement that assert a relationship (attribute / generalization / aggregation) between two terms.</td>
</tr>
<tr>
<td>Constraint</td>
<td>A statement that specifies a mandatory feature (business behaviour or characteristics) that must be satisfied by a business entity.</td>
</tr>
<tr>
<td>Guideline</td>
<td>A statement that specifies an optional feature that should be satisfied by a business entity.</td>
</tr>
<tr>
<td>Action</td>
<td>A statement that specifies a condition that will start a business action.</td>
</tr>
<tr>
<td>Inference</td>
<td>A statement that derives a fact using logical deduction or induction.</td>
</tr>
</tbody>
</table>

![Table 1: Structural typology of business rules](image-url)
managing CASE tool capability is required for software design in achieving evolvable software system. Needed to evaluate the quality of business rules and its typology. The study of the effect of design changes to refinement, exploring in order to provide enough formalization of requirements traceability using the link model for linking frequently changing business rules to software design. We improve the well as a link model for linking frequently changing business rules to software design. We improve the requirements traceability using the link model and provide enough formalization for future automated implementation.

However, there are still further issues that need exporing in order to achieve a complete solution to evolvable software systems. They include: (a) metamodel refinement, (b) change impact evaluation, and (c) CASE tools development. For metamodel refinement, more case studies are needed to achieve a complete business rule typology. The study of the effect of design changes to business rules will also has a significant contribution to metamodel refinement. Change impact evaluation is needed to evaluate the quality of business rules and software design in achieving evolvable software system. CASE tool capability is required for automation in managing the design, link model, and business.

5. Conclusions and Future Works

In developing a metamodel for evolvable software systems, we moved one step closer to considering the most important source of requirement changes i.e. business rules. We provide a set of structured templates for specification and management of business rules, as well as a link model for linking frequently changing business rules to software design. We improve the requirements traceability using the link model and provide enough formalization for future automated implementation.

References


