Towards Model-Driven Approach for Rapid ERP Development

Igor Miletić[1], Marko Vujasinović[2], and Zoran Marjanović[3]

[1]Breza Software Engineering, Kraljice Natalije 23a,
11000 Belgrade, Serbia
igor.miletic@brezasoftware.com
[2] Research consultant, B. Knezova 10,
22300 Knin, Croatia
marko.vujasinovic@acm.org
[3] Faculty of Organizational Sciences, Jove Ilica 154,
11000 Belgrade, Serbia
zoran.marjanovic@fon.rs

Abstract. In this paper we describe an MDA (model-driven architecture) approach that we successfully apply in our ERP development process. We have developed platform-independent models of certain ERP aspects and tools that use these models to generate database scripts and source code for a target platform, ERP documentation, and help system. The paper outlines these models using patterns, which are considered common solutions for recurring challenges in ERP development.

Keywords: enterprise resource planning, model-driven development, application generators

1 Introduction

An enterprise resource planning (ERP) system integrates business data to coordinate, consolidate, and support execution of business processes inside a company. Our experience from the ERP systems development (BrezaERP [1]) and several successfully completed ERP implementations exposed that ERP systems typically need to provide various applications and reports that support a wide range of business processes, quick and reliable responses to business processes, and simple and low-cost extensions to such business processes. To address these issues, we propose certain best-practice solutions. We have formalized these solutions through a set of patterns expressed as platform and application independent MDA (www.omg.org/mda) models using UML (Unified Modeling Language; www.uml.org). These model-based patterns have been incorporated into a several ERP development projects and have significantly helped us to speed-up the implementation activities. Importantly, the patterns apply in any ERP development project. In the following sections, we start with the short overview of the related work. Then, we introduce the key requirements that any ERP should satisfy, and discuss the proposed model-based patterns that meet these requirements. We conclude the paper with some future work directions.

2 Related Work

Several approaches to speed-up ERP development have been proposed. Govedarica et. al. [2] propose development of a system specification using XML (Extensible Markup Language; www.w3c.org/xml), which is used by the XSLT (XSL Transformation; www.w3c.org/TR/xslt) tools to generate executable information systems. Govedarica and his colleagues propose structures that describe specification of transactional programs as well as user forms. However, this approach is too general - it is not strictly focused on the ERP development issues, it does not provide concepts and models that can be used to express commonality of user forms as well as behavior of applications, nor does it address generation of a help system of the application from a system specification. Lazarevic and Misic [3] introduced an approach that uses a formal specification of the ERP systems to generate the database procedures that would control the integrity-constraint rules of the entity-relationship models. Dugerdil and Gairals [4] focus on the development of the MDA models that can capture an ERP configuration, but they start from an assumption that the ERP system is already developed and needs only to be customized. Deursen and Kuipers [5] focus on the problems related to the synchronization of ERP documentation with the changes in the source code; they introduce so-called island grammars, which are syntactic structures that generate documentation from the source code and merge it with a hand-written documentation.

Our work differs from others in that we provide precisely defined, formalized, and in-practice proven MDA models that speed-up ERP development through (1) automated generation of the user forms, documentation, and help system, (2) simple introduction of new ERP modules and new documents, and (3) simplified customization of system actions and behavior.

3 Design Patterns (Generic Models) for ERP Development

An ERP system depends on a number of requirements imposed from the different perspectives, e.g., usability from the end-users' perspective, or simplicity of development, extension, and maintenance from the developers' perspective. Table 1 lists the key requirements that have been revealed during BrezaERP development. However, these requirements are not BrezaERP-specific; they apply to any ERP system.

Table 1. The key requirements for the ERP development

Requirement No.	Requirement	Addressed by pattern
1.	Uniform user interface	Interface configuration
2.	User system support (Help system)	Interface configuration
3.	Updated user specification of the system	Interface configuration
4.	Simple integration of new modules	System configuration
5.	Support for large number of documents	Document configuration
6.	Execution of actions over the documents	Behavior configuration
7.	Reliable communication between subsystems	Behavior configuration

We have identified four generic models (or *patterns*) that can be used to meet these requirements and reduce manual effort and development time.

3.1 Interface Configuration Pattern (Model)

Separate design and coding of the hundreds of different user forms that may exist in an ERP is a hard, error-prone, and time-consuming work. Even a small change in specifications may require significant effort. For instance, consider tens of user forms embedded within an ERP financial module, all of them containing a date-picker component. Assume that we want to replace the component with a new improved one. It may take up to one hour (in our experience) to implement and test the change in just one form. If, say, twenty user forms are being changed and all are maintained separately, such a small change will have taken few days to complete. Moreover, additional time is needed to change the help and the system specification documents. To address such issues and to reduce development time, we propose the *interface configuration* pattern (shown in Fig. 1).

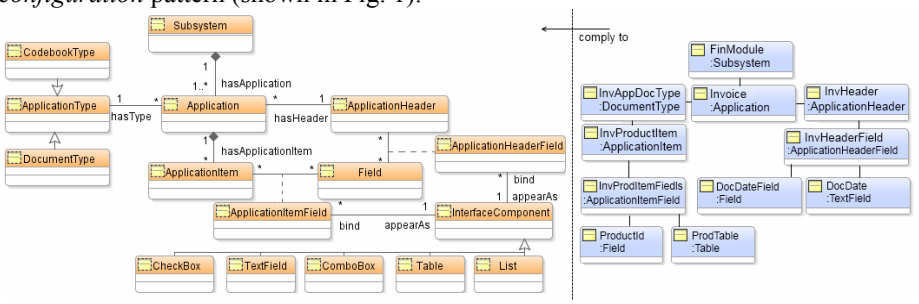

Fig. 1. The interface configuration pattern. (The left part shows the interface configuration concepts; the right part shows their instances, given as an example)

The interface configuration pattern proposes development of models that contain descriptions of the user forms in a generic way. `Subsystem` concept represents an ERP module (e.g. financial module). `Subsystem` contains `Applications` (e.g. Invoice application). Each `Application` belongs to the `ApplicationType` that represents either a business document or a codebook. `Application` is composed of `ApplicationHeader` and `ApplicationItems`, which are sets of `Fields` (e.g. document date, item quantity). The `Application` elements on the user forms are specified by `InterfaceComponents` such as check-boxes, tables, text-fields, etc.

We model all the user forms to comply to the interface configuration pattern. (This pattern can be considered as a metamodel for the models of specific user forms.) Changes on the user forms are performed over the interface configuration that defines these forms. Our *user form definition tool* allows us to simply define new or to change existing user form definitions. The generators operate on the interface configuration definitions, and guarantee the *uniform* user interface, and *harmonized* help system and system specification documents.

3.2 System Configuration Pattern (Model)

Traditionally, when developing a new ERP module, developers engage in several activities, such as creation of new database objects manually or by a tool that generates executable database scripts for new objects, execution of the database

scripts, and development environment preparation (e.g., setting up new project in a development tool, placing the project under a version control, and so on). All these activities take significant time. If the number of new modules increases, the effort and duration of activities increase as well. To address this issue, we propose the *system configuration* pattern (shown in Fig. 2).

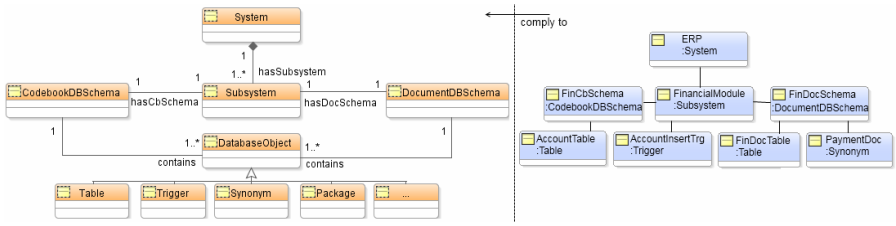

Fig. 2. The system configuration pattern. (The left part of figure shows the system configuration concepts; the right part shows their instances given as an example)

The system configuration pattern proposes the model that is supported by a tool that clones a (previously prepared) template of a database structure of a standard module (the one defined as a standard-template). `System` concept represents any ERP system. `Subsystem` represents an ERP module. Database structure of the ERP module may contain codebook schema and document schema. These are captured by `CodebookDBSchema` and `DocumentDBSchema` concepts. `CodebookDBSchema` or `DocumentDBSchema` can contain a number of database objects - `tables`, `triggers`, `packages`, etc. When we develop new modules we use system configuration pattern and our tool, which provides us a faster ramp-up time. In particular, we automatically get initial table structures, initially prepared projects in a development environment, as well as application objects that map to database objects.

3.3 Document Configuration Pattern (Model)

The ERP system automates the flow of the (electronic) business documents within a company. New document types, however, appear rapidly as the business grows. Appearance of a new document type is reflected with the ERP system as a creation of new electronic document type. Assume a case in which a government administrator introduced a new type of invoice document that is obligated for all the companies in the country. The new invoice type introduces new type of document items that contain important tax information. Traditionally, the new document type, such is that invoice, requires a new database structure. It will cost time to design such a new database structure, and to implement business logic that supports the document type. If the document has relationships with other document(s), this leads to an even more complicated situation. The number of database table objects will grow rapidly with more and more new document types. To avoid such issues we propose the *document configuration* pattern (shown in Fig. 3).

The document configuration pattern proposes the database structure that provides flexibility in storing documents, document items and document connections of various document types in the same database tables of a particular subsystem (module). The pattern enables addition of new document types in the ERP without

any change of the data structure or change in the application code. In fact, the pattern is a metamodel for document types (models). Each document complies with the document configuration. `DocumentTable`, `DocumentItemTable` and `DocumentConnectionTable` concepts represent physical structures (e.g. database tables) that store documents, document items and document connections, respectively.

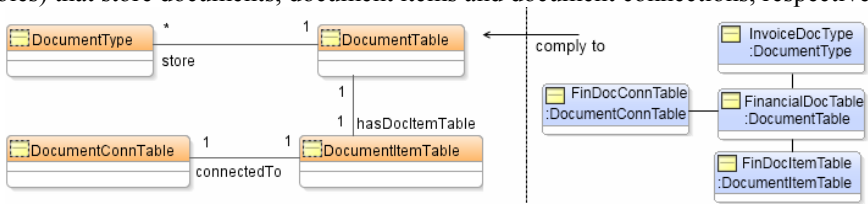

Fig. 3. The document configuration pattern. (Left part of the figure shows the document configuration concepts; the right part shows their instances given as an example)

We have developed a generic ERP component that is based on the document configuration pattern and that implements the standard functions for the document manipulation (e.g., create document, delete document, add document item). When new document type appears, we just register new document type. The ERP system distinguishes the types of documents by the provided field value that classifies the documents.

3.4 Behavior Configuration Pattern (Model)

Behavior and business logic of every ERP system is woven into the number of actions over the documents. Actions may be standard (i.e., supported by document configuration pattern and above mentioned component), or document-specific such as the specific processing, accounting, and document content auditing. The specific actions executed on a document in one module can cause creation of one or more new documents in other modules. (Such creation of new documents is followed by strict mapping rules. The mapping rules define how to map fields from one document to another when transformation among documents occurs; mapping can be simple one-to-one mapping or more complex if additional operations are included.) Adding a new action to the ERP system usually brings additional costs because business logic related to all document types that new action will apply to has to be re-implemented. To address such issues we propose the behavior configuration pattern (Fig. 4).

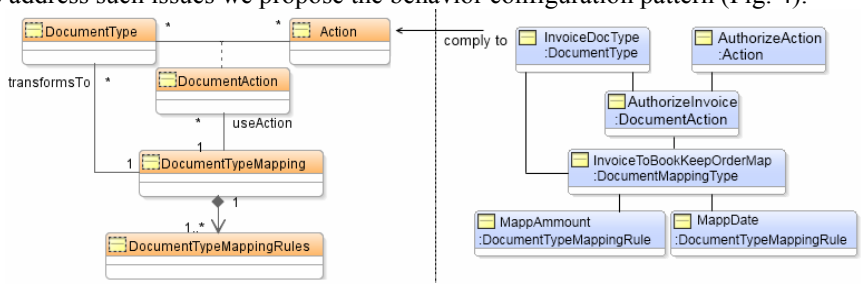

Fig. 4. The behavior configuration pattern (Left part of the figure shows the behavior configuration concepts; the right part shows their instances given as an example)

The behavior configuration pattern proposes a metamodel for modeling and configuring document types' related actions as well as the mapping rules among two or more document types. The proposed model is expressed as an aggregation of the `DocumentType` concept and the actions related to the document type and mapping rules (concept `DocumentActions`). The mapping rules are used by the corresponding action when transformation among documents is needed. `DocumentAction` is in fact some executable component (e.g. Java class) that is general enough to be executed over the documents. To support the behavior configuration pattern, we developed mechanism (component) of the action execution. The component executes the allowed actions over the certain type of document. In order to transform one document to another, the component reads mapping rules (represented by `DocumentTypeMappingRules` concept) and executes defined transformation.

The behavior configuration pattern allows changes in behavior of the ERP system without code modification, as we have developed component which is able to load defined actions from the model and execute them over the documents. It is sufficient to register new action into the underlying structure.

4 Conclusion and Future Work

The proposed patterns are in fact generalized MDA metamodels of certain ERP aspects. Their constructs can be also considered as a domain-specific language constructs for ERP modeling. We describe each particular ERP module and application by using these constructs, and use the tools to generate database scripts, source code, help system and system specification documents. Our target platform was Oracle database and Java environment; however, the generators could be extended to support other similar platforms as well. We now plan to reveal more patterns for ERP development and formalize them as metamodels. That will lead towards ERP model-based, rather than hard-code, development. Additionally, we plan to develop a tool that provides a visual environment for ERP modeling.

5 References

1. BrezaERP. On-line, last accessed August 2010 at www.brezasoftware.com.
2. Govedarica, M., et. al. (2004) Generating XML Based Specifications of Information Systems, Computer Science and Information Systems, Vol. 1, Issue 1, pp. 117-140.
3. Lazarevic, B. and Misic, V. (1991). Extending the entity-relationship model to capture dynamic behavior. European Journal of Information Systems, Vol. 2, pp. 95-106.
4. Dugerdil, P., and Gaillard, G. (2006), Model-Driven ERP Implementation, Proc. of the 2nd Int. Workshop on Model-Driven Enterprise Information Systems, Paphos, Cyprus.
5. Deurse, A., and Kuipers, T. (1999) Building documentation generators, Proc. of the IEEE Int. Conf. on Software Maintenance, pp. 40-49.