Towards More Flexible Enterprise Information Systems

Rogerio Atem de Carvalho[1,2] and Björn Johansson[3]

[1] Federal Fluminense Institute, NSI, R. Dr. Siqueira 273, Sala F104, Campos, Brazil
[2] Federal Fluminense University, Latec, R. Passo da Patria 156, Bloco E, Niteroi, Brazil
ratem@iff.edu.br
[3] Department of Informatics, School of Economics and Management,
Lund University, Sweden
bjorn.johansson@ics.lu.se

Abstract. The aim of this paper is to present the software development techniques used to build the EIS Patterns development framework, which is a testbed for a series of techniques that aim at giving more flexibility to EIS in general. Some of these techniques are customizations or extensions of practices created by the agile software development movement, while others represent new proposals. This paper also aims at helping promoting more discussion around the EIS development questions, since most of research papers in EIS area focus on deployment, IT, or business related issues, leaving the discussion on development techniques ill-treated.

Keywords: Enterprise Information Systems, Domain Specific Languages, Design Patterns, Statechart Diagrams, Natural Language Processing.

1 Introduction

In Information Systems, flexibility can be understood as the quality of a given system to be adaptable in a cost and effort effective and efficient way. Although it is usual to hear from Enterprise Information Systems (EIS) vendors that their systems are highly flexible, the practice has shown that customizing this type of system is still a costly task, mainly because there are still based on relatively old software development practices and tools. In this context, the EIS Patterns framework[1] is a research project which aims at providing a testbed for a series of relatively recent techniques nurtured at the Agile methods communities, and ported to the EIS arena.

The idea of suggesting and testing new ways for developing EIS was born from accumulated research and experience on more traditional methods, such as Model Driven Development (MDD), on top of the open source ERP5 system [1]. ERP5 represents a fully featured and complex EIS core, making it hard to test the ideas here presented in their pure form, thus it was decided to develop a simpler framework to serve as a proof of concept of proposed techniques.

[1] Initially discussed at the EIS Development blog through a series of posts entitled EIS Patterns, starting in December 2010 (http://eis-development.blogspot.com).

G. Poels (Ed.): CONFENIS 2012, LNBIP 139, pp. 157–164, 2013.

This paper is organized as follows: the next topic summarizes the series of papers that forms the timeline of research done on top of ERP5; following this, the proposed techniques are presented, and finally some conclusions and possible directions are listed.

2 Background

In order to understand this proposal, it is necessary to know the basis from where it was developed, which is formed by a series of approaches developed on top of ERP5. Following the dominant tendency of the past decade, which was using MDD, the first approach towards a formalization of a deployment process for ERP5 was to develop a high-level modeling architecture and a set of reference models [2], as well as the core of a development process [3]. This process evolved to the point of providing a complete set of integrated activities, covering the different abstraction levels involved by supplying, according to the Geram [4] framework, workflows for Enterprise, Requirements, Analysis, Design, and Implementation tasks [5].

Since programming is the task that provides the "real" asset in EIS development, which is the source code that reflects the business requirements, programming activities must also be covered. Therefore, in *"ERP5: Designing for Maximum Adaptability"*[6] it is presented how to develop on top of the ERP5's document-centric approach, while in *"Using Design Patterns for Creating Highly Flexible EIS"*[7], the specific design patterns used to derive concepts from the system's core are presented. Complimentary, in *"Development Support Tools for ERP"* [8] two comprehensive sets of ERP5's development support tools are presented: (i) Product-related tools that support code creation, testing, configuration, and change management, and (ii) Process-related tools that support project management and team collaboration activities. Finally, in *"ERP System Implementation from the Ground up: The ERP5 Development Process and Tools"*[9], the whole picture of developing on top of ERP5 is presented, locating usage of the tools in each development workflow, and defining its domain-specific development environment (DSDE).

Although it was possible to develop a comprehensive MDD-based development process for the ERP5 framework, the research and development team responsible for proposing this process developed at the same time an Enterprise Content Management solution [10] and experimented with Agile techniques for both managing the project and constructing the software. Porting this experimentation to the EIS development arena lead to the customization of a series of agile techniques, as presented in *"Agile Software Development for Customizing ERPs"*[11].

The work on top of ERP5 provided a strong background, on both research and practice matters, enough to identify the types of relatively new software development techniques that could be used on other EIS development projects. Even more, this exploration of a real-world, complex system, has shown that some other advances could be obtained by going deeper into some of the techniques used, as well as by applying them in a lighter framework, where experimentations results could be quickly obtained.

3 Enters EIS Patterns

EIS Patterns is a simple framework focused on testing new techniques for developing flexible EIS. It was conceived having the Lego sets in mind: very basic building blocks that can be combined to form different business entities. Therefore, it was built around three very abstract concepts, each one with three subclasses, representing two "opposite" derived concepts and an aggregator of these first two, forming the structure presented in Fig. 1.

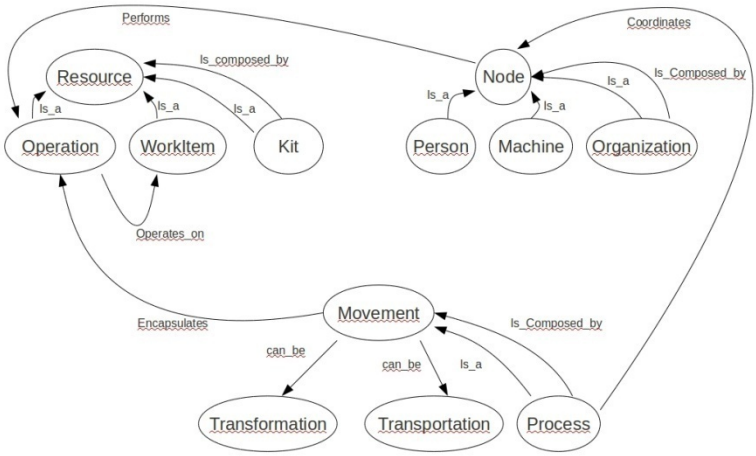

Fig. 1. Ontology representing the EIS Patterns core

Fig 1 is interpreted as follows:

Resource: is anything that is used for production.
-Material: product, component, tool, document, raw material etc.
-Operation: human operation and machine operation, as well as their derivatives.
-Kit: a collective of material and/or immaterial resources. Ex.: bundled services and components for manufacturing.
Node: is an active business entity that transforms resources.
-Person: employee, supplier's contact person, drill operator etc.
-Machine: hardware, software, drill machine, bank account etc.
-Organization: a collective of machines and/or persons, such as manufacturing cell, department, company, government.
Movement: is a movement of a Resource between two Nodes.
-Transformation: is a movement inside a node, in other words, the source and destination are the same node, it represents the transformation by machine or human work of a resource, such as drilling a metal plate or writing a report.

-Transportation: is a movement of resources between two nodes, for example, moving a component from one workstation to another, sending an order from the supplier to the customer.
-Process: a collective of transformations and/or transportations, in other words, a business process.

Besides the obvious "is a" and "is composed by" relationships presented in the ontology in Fig. 1, a chain of relationships denote how business processes are implemented: "a Process *coordinates* Node(s) to *perform* Operation(s) that *operates on* Work Item(s)". The semantic meaning of this chain is that process objects control under which conditions node objects perform operations in order to transform or transport resources. This leads to another special relationship which is "a Movement *encapsulates* an Operation", which means that a movement object will encapsulate the execution of an operation. In practical terms, an operation is the abstract description of a production operation, which is implemented by one or more node objects' methods. When this operation is trigged by a process object, it defers the actual execution to a pre-configured node object's method, and this execution is logged by a movement object, which stores all parameters, date and time, and results of this execution. Therefore, an operation is an abstract concept which can be configured to defer execution to different methods, from different objects, in accordance to the intents of a specific business process instance. In other words, a business process abstraction keeps its logic, while specific results can be obtained by configuration.

Although this execution deference can appear to be complex, it is a powerful mechanism which allows that a given business process model may be implemented in different ways, according to different modeling-time or even runtime contexts. In other words, the same process logic can be implemented in different ways, for different applications, thus leveraging the power of reuse.

It is important to note that in this environment, Processes control the active elements, the Nodes, which in turn operate on top of the passive ones, the Resources. In programming terms, this means that processes are configurable, nodes are extended, and resources are typically "data bag" classes. Therefore, extending the nodes for complying with new business requirements becomes the next point where flexibility must take place.

3.1 Using Decorators to Create a Dynamic System

Usually, class behavior is extended by creating subclasses, however, this basic technique can lead to complex, hard to maintain, and even worse, hard-coded class hierarchies. One of the solutions to avoid this is to use the Decorator design pattern [12], taking into account the following matters:

- While subclassing adds behavior to all instances of the original class, decorating can provide new behavior, at runtime, for individual objects. At runtime means that decoration is a "pay-as-you-go" approach to adding responsibilities.

- Using decorators allows mix-and-matching of responsibilities.
- Decorator classes are free to add operations for specific functionalities.
- Using decorators facilitates system configuration, however, typically, it is necessary to deal with lots of small objects.

Hence, by using decorators it is possible, during a business process realization, to associate and/or dissociate different responsibilities to node objects - in accordance to the process logic, and providing two main benefits: (i) the same object, with the same identifier, is used during the whole business process, there is no need for creating different objects of different classes, and (ii) given (i), auditing is facilitated, since it is not necessary to follow different objects, instead, the decoration of the same object is logged. Moreover, it is possible to follow the same object during all its life-cycle, including through different business processes: after an object is created and validated - meaning that it reflects a real-world business entity - it will keep its identity forever[2].

An important remark is that decorators must keep a set of rules of association, which is responsible for allowing or prohibiting objects to be assigned to new responsibilities. If a given object respects the rules of association of a given decorator, it can be decorated by it. At this point, defining a flexible way of ensuring contracts between decorators and decorated objects is of interest.

Should-dsl: a language for contract checking

Although Should-dsl was originally created as a domain specific language for checking expectations in automated tests [13], in the EIS Patterns framework it is also used to provide highly readable contract verifiers, such as:

```
associated |should| be_decorated_by(EmployeeDecorator)
```

In the case above the rule is auto-explanative: “the associated object should be decorated by the Employee Decorator”, meaning that for someone to get manager’s skills he or she should have the basic employee’s skills first. Besides being human readable, these rules are queryable, for a given decorator it is possible to obtain its rules, as well as the symmetric: for a given node object, it is possible to identify which decorators it can use. Query results, together with the analysis of textual requirements using Natural Language Processing, are used to help configuring applications built on top of the framework.

Using Natural Language Processing to Find Candidate Decorators

It is also possible to parse textual requirements, find the significant terms and use them to query decorators’ documentation, so the framework can suggest possible decorators to be used in accordance to the requirements. Decorators’ methods that represent business operations - the components of business processes - are specially

[2] A more complete discussion on using decorators, with examples, can be found at `http://eis-development.blogspot.com.br/2011/03/enterprise-information-systems-patterns_09.html`

tagged, making it possible to query their documentation as well as obtain their category. Categories are used to classify these operations, for instance, it is possible to have categories such as “financial”, “logistics”, “manufacturing” and so on. In that way, the framework can suggest, from its base of decorators, candidates to the users’ requirements.

3.2 A Domain-Specific and Ubiquitous Language for Modeling Business Process

The ontology presented in Fig. 1, although simple, is abstract enough to represent entities involved in any business process. Moreover, by appropriately using a statechart diagram, it is possible to use a single model to describe a business process, define active entities, as well as to simulate the process.

In order to better describe this proposal, Fig. 2 shows a simple quotation process. Taking into account that a class diagram was used to represent the structural part of the business process[3], by explicitly declaring the objects responsible for the transitions, it is possible to identify the active elements of the process, all of the Person type: sales_rep, verifier, approver, and contractor; as well as how they collaborate to perform the business process, by attaching the appropriate methods calls. Additionally, in some states, a method is declared with the “/do” tag, to indicate that a simulation can be ran when the process enters these states.

To run these state machine models, Yakindu (www.yakindu.org) could be used. By adapting the statechart execution engine, it is possible to run the model while making external calls to automated tests, giving the user the view of the live system running, as proposed by Carvalho *et al*. [14].

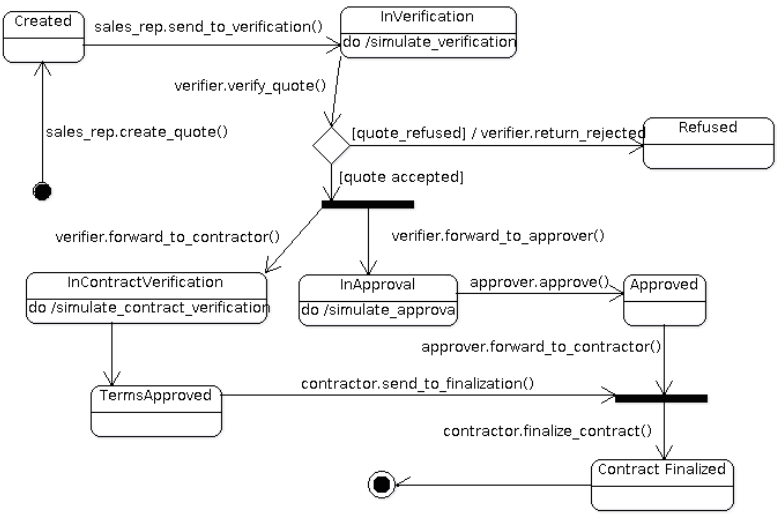

Fig. 2. A simple quotation process using the proposed concepts

[3] Not shown here due to the lack of space.

3.3 An Inoculable Workflow Engine

Workflow engines provide the basis for the computational realization of business processes. Basically, there are two types of workflow engines: (i) associated to application development platforms or (ii) implemented as software libraries.

EIS patterns uses Extreme Fluidity (xFluidity), a variation of the type (ii) workflow engine, developed as part of the framework. xFluidity is an inoculable (and expellable) engine that can be injected into any Python object, turning it workflow-aware. Symmetrically, it can be expelled from the object, turning the object back to its initial structure when necessary. It was developed in this way because type (i) engines forces you to use a given environment to develop your applications, while type (ii) forces you to use specific objects to implement workflows, most of times creating a mix of application specific code and workflow specific statements. With xFluidity it is possible to define a template workflow and insert the code necessary to make it run inside the business objects, while keeping the programming style, standards, naming conventions, and patterns of the development team. In EIS Patterns, xFluidity is used to configure Process objects, making them behave as business processes templates.

Currently xFluidity is a state-based machine, however, it can be implemented using other notations, such as Petri Nets. In that case, no changes are necessary in the inoculated objects, given that these objects do not need to know which notation is in use, they simple follow the template.

4 Conclusions and Further Directions

This paper briefly presents a series of techniques that can be applied to turn EIS more flexible, including the use of dynamic languages[4]. Although the EIS Patterns framework is a work in progress, it is developed on top of research and practical experience obtained on the development of the ERP5 framework.

This experience led to the use of an abstract core to represent all concepts, while providing flexibility through the use of the Decorator pattern. On top of this technique, Natural Language Processing (NLP) and automated contract checking is used to improve reuse even more and, as a side effect, enhance system documentation, given that developers are forced to provide code documentation as well as to define association contracts through should-dsl, which is a formal way of defining the requirements for the use of decorators to expand the functionality of Node objects.

The integrated use of an inoculable workflow engine, a domain-specific and ubiquitous language, and should-dsl to check association contracts, is innovative and provides more expressiveness to the models and the source code, by the use of a single language for all abstraction levels, which reduces the occurrence of translation errors through these levels. This is an important point: more expressive code facilitates change and reuse, thus increasing flexibility.

[4] For a discussion on this see `http://eis-development.blogspot.com.br/2010/09/is-java-better-choice-for-developing.html`

Further improvements include the development of a workflow engine based on BPMN, in order to make the proposal more adherent to current tendencies, and provide advances on the use of NLP algorithms to ease identification and reuse of concepts.

References

1. Smets-Solanes, J.-P., Carvalho, R.A.: ERP5: A Next-Generation, Open-Source ERP Architecture. IEEE IT Professional 5, 38–44 (2003)
2. Campos, R., Carvalho, R.A., Ferreira, A.S.: Modeling Architecture and Reference Models for the ERP5 Project. In: Confenis 2006. Research and Practical Issues of Enterprise Information Systems, pp. 677–682. Springer, New York (2006)
3. Carvalho, R.A., Campos, R.: A Development Process Proposal for the ERP5 System. In: IEEE International Conference on Systems, Man, and Cybernetics. IEEE Press, New York (2006)
4. IFIP – IFAC GERAM: Generalized Enterprise Reference Architecture and Methodology, IFIP – IFAC Task Force on Architectures for Enterprise Integration (1999)
5. Monnerat, R.M., Carvalho, R.A., Campos, R.: Enterprise Systems Modeling: the ERP5 Development Process. In: 23rd Annual ACM Symposium on Applied Computing, vol. II, pp. 1062–1068. ACM, New York (2008)
6. Carvalho, R.A., Monnerat, R.M.: ERP5: Designing for Maximum Adaptability. In: Wilson, G., Oram, A. (orgs.) Beautiful Code: Leading Programmers Explain How They Think, pp. 339–351. O'Reilly Media, Sebastopol (2007)
7. Carvalho, R.A., Monnerat, R.M.: Using Design Patterns for Creating Highly Flexible Enterprise Information Systems. In: The III IFIP International Conference on Research and Practical Issues of Enterprise Information Systems. IFIP Series (2009)
8. Carvalho, R.A., Monnerat, R.M.: Development Support Tools for Enterprise Resource Planning. IEEE IT Professional 10, 39–45 (2008)
9. Carvalho, R.A., Campos, R., Monnerat, R.M.: ERP System Implementation from the Ground up: The ERP5 Development Process and Tools. In: Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization, pp. 423–438. IGI Global (2009)
10. Carvalho, R.A.: An Enterprise Content Management Solution Based on Open Source. In: Research and Practical Issues of Enterprise Information Systems II, vol. 1, pp. 173–184. Springer, New York (2007)
11. Carvalho, R.A., Johansson, B., Manhaes, R.S.: Agile Software Development for Customizing ERPs. In: Enterprise Information Systems and Implementing IT Infrastructures: Challenges and Issues, pp. 20–39. Information Science Reference, IGI Global, Hershey (2010)
12. Gamma, E., et al.: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
13. Tavares, H.L., Rezende, G.G., Mota, V., Manhaes, R.S., Carvalho, R.A.: A tool stack for implementing Behavior-Driven Development in Python Language, arXiv:1007.1722v1(cs.SE)
14. Carvalho, R.A., Carvalho e Silva, F.L., Manhaes, R.S.: Business Language Driven Development: Joining Business Process Models to Automated Tests. In: V IFIP International Conference on Research and Practical Issues of Enterprise Information Systems. LNBIP (2011)