A UML reuse framework and tool for Requirements Engineering

Vitor A. Batista, Daniela C. C. Peixoto, Thiago R. V. Anjos, Wilson Pádua

Synergia – Computer Science Department
UFMG – Federal University of Minas Gerais
Belo Horizonte, Brazil

{vitor, cascini, trva, wilson}@dcc.ufmg.br

Abstract. Requirement Engineering (RE) activities are manual and critical by nature. Providing some automated support for the RE tasks helps analysts to reduce manual labor, and in consequence, reduce defects rates and increase reuse and motivation. In this paper, we introduce a UML framework and tool support which automates part of the RE process. Using UML stereotypes concepts as the core of this solution, we created a set of integrated tools composed by: (1) a reusable framework that models some common RE behavior patterns that are typically present in information system projects; (2) a function that allows the reuse of information provided by entity modeling; (3) a tool that automates the generation of application prototypes; and (4) a tool that analyzes specific types of defects. Our preliminary findings indicate that the framework and the automated support are effective at RE modeling and review. In addition, they increase motivation and promote team engagement, through elimination of repetitive activities.

Keywords: Requirements Engineering, UML framework

1 Introduction

Requirement Engineering (RE) is a very labor-intensive and critical activity. It is manual by nature, since requirements are elicited by analysts from users through workshops and interviews, and recorded in informal or semi-formal specifications[1]. It is critical because its defects inevitably lead to later problems in design and implementation, whose repair is, usually, expensive and difficult. Many studies have shown that requirements with poor quality are a major cause of project failures [2].

One way to help analysts is to provide some integrated, automated support for the RE tasks. This helps to reduce manual labor, and eases the early detection of errors. Since RE is as a naturally labor-intensive process, only a few tasks may be significantly automated. However, their contribution to decrease rework and increase productivity can be substantial. Another way to improve RE support is to provide reusable model elements, with appropriate guidance.

In this work, we present the adoption and improvement of an RE sub-process in a software development organization, Synergia. It is embedded in a process, called Praxis-Synergia; this is a professional version of Praxis, an educational software development process[3]. Here, we discuss some challenges faced and solutions proposed during this effort towards an automated and integrated approach to RE.

Our main contribution lies in providing a UML reuse framework and a set of automated tools to support RE activities. This framework has the goal to standardize the modeling of the most common software transactions in information systems, yet remaining flexible enough to let analysts to extend it when needed.

The expected benefits of this approach consist of improving model comprehension and reuse, reducing errors and increasing productivity throughout the software development cycle.

The article is structured as followed. Section 2 introduces basic concepts of Praxis and the problems faced by requirement analysts. Section 3 presents our UML framework and the proposed automation tools. Section 4 provides a discussion of the results and open issues. Section 5 presents the related work. Section 6 concludes and presents future work.

2 Background

Praxis is a model-based, use-case-driven process. In its prescribed development sequence, a project is divided in iterations, where each iteration implements one or more functional requirements, modeled as use cases.

The Praxis Problem model captures and analyses requirements. Despite some similarity with RUP's Analysis Model, its structure closely follows the IEEE-830 standard for Requirements Specification. The Problem model is divided into two main views: Requirement view and Analysis view.

The Requirement view describes the desired product from the user viewpoint, representing desired functions as UML use cases. Each use case behavior is described by one or more event flows (scenarios). Event flows may be modeled as UML activities, or described with text. The former is more adequate for complex interactions, the latter for simpler ones.

The Analysis view describes the desired product from the developer viewpoint, but still in the problem-domain; it models concepts, procedures and interfaces as classes, and their interactions by UML collaborations.

Synergia, the organization where this study was carried out, uses a professional version of Praxis; a detailed description of requirements activities and artifacts can be found in [4].

The adoption of Praxis at Synergia faced some problems during execution of real-life projects:

- User interfaces are prototyped in the Requirements view and also modeled as UML Classes in the Analysis view. Keeping both artifacts consistent during project is hard and error prone (25% of the defects found in our requirements reviews).

- Since Praxis does not offer guidelines to model requirements, each use case is modeled from scratch, according to the experience of each analyst. Common transactions, like CRUDs, are often modeled quite differently by different analysts, and this is a serious impediment to reuse.
- Praxis uses a large subset of UML elements, and a lot of information needs to be recorded in stereotype properties (tagged values). This results in large checklists, reviews effort and subsequent rework.

Given such problems faced at Synergia, we proposed a set of improvements in Praxis-Synergia, in order to overcome them. In next section we present our proposals.

3 The UML framework and automated tools

The solution we proposed includes a set of UML tools which automate common RE activities, and ease the specification of common user interactions in a typical information system.

3.1 The UML Profile

The core of our automated solution is a set of stereotypes (with tagged values and constraints), used to enhance and increase the formality level of requirements modeling. To model user interfaces (UI), a hierarchy of stereotypes for UI widgets, fields, commands, navigation was created. A partial view of these stereotypes is in **Fig. 1**. UIs are modeled as a set of stereotyped classes; composition associations represent how a widget is embedded inside another. The root of these composition trees are classes with «screen», «modalScreen» or «report» stereotypes. They have state machines that model their behavior.

UI fields are represented by class attributes with a concrete stereotype of «baseUIAttribute». Commands are modeled as class operations with «command». Navigation between screens is a directed association with «navigate» stereotype, which holds the operation(s) that triggers that navigation. This association also holds the target state in the target's state machine.

Appearance changes, such as visibility and enablement, are modeled for each of the UI widgets, fields or commands, by tagged values which may hold "Yes", "No" and "Depends on State". In case of depends, the engineer models in which States of the UI the values Yes/No applies.

Application menus are also modeled as stereotyped classes («menu», «subMenu») and their «menuItem» operations.

All stereotypes are encapsulated in a UML profile, deployed as a plug-in in our UML modeling tool, IBM Rational Software Architect (RSA). This plug-in also provides task automation resources, as discussed next.

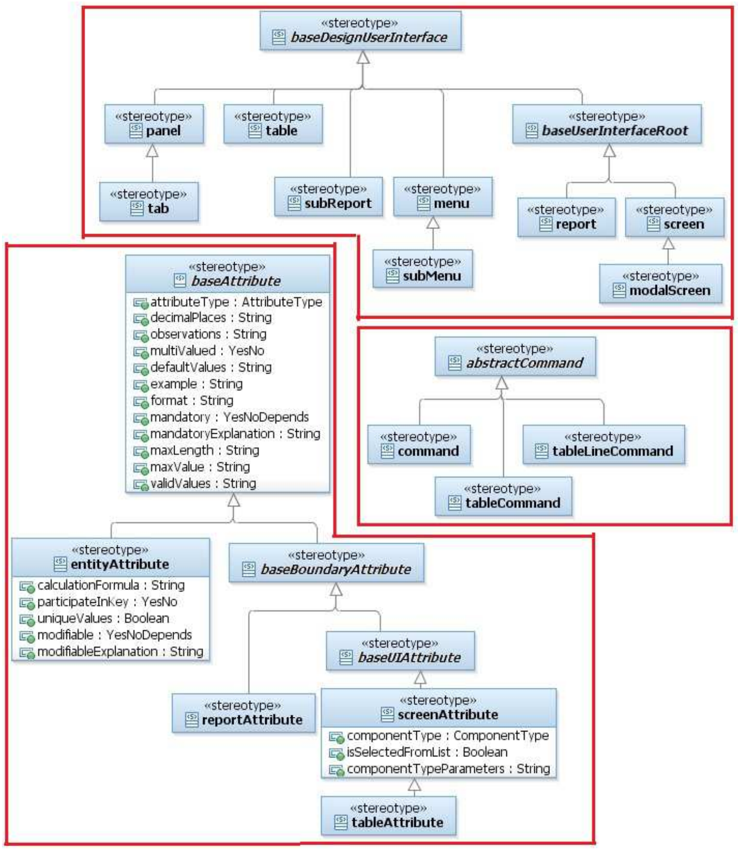

«stereotype» baseDesignUserInterface
«stereotype» panel
«stereotype» table
«stereotype» baseUserInterfaceRoot
«stereotype» tab
«stereotype» subReport
«stereotype» menu
«stereotype» report
«stereotype» screen
«stereotype» subMenu
«stereotype» modalScreen
«stereotype» baseAttribute
attributeType : AttributeType
decimalPlaces : String
observations : String
multiValued : YesNo
defaultValues : String
example : String
format : String
mandatory : YesNoDepends
mandatoryExplanation : String
maxLength : String
maxValue : String
validValues : String
«stereotype» abstractCommand
«stereotype» command
«stereotype» tableLineCommand
«stereotype» tableCommand
«stereotype» entityAttribute
calculationFormula : String
participateInKey : YesNo
uniqueValues : Boolean
modifiable : YesNoDepends
modifiableExplanation : String
«stereotype» baseBoundaryAttribute
«stereotype» baseUIAttribute
«stereotype» reportAttribute
«stereotype» screenAttribute
componentType : ComponentType
isSelectedFromList : Boolean
componentTypeParameters : String
«stereotype» tableAttribute

Fig. 1. Stereotypes to model UI elements.

3.2 UML Framework and interaction copy wizard

Different information system projects usually have common behavior patterns, such as CRUD transactions. Instead of developing them from scratch every time, a reusable UML framework should provide enough reuse to make the engineer's task more productive and less error prone. We gathered information from past projects and modeled the most common interactions as a set of abstract UML collaborations. Their participants are abstract UI classes and their widgets and commands. Navigation between participants and their behavior are modeled in such abstract interactions.

The task of the requirements engineer is to create his own concrete collaboration, related by an aggregation association to framework collaborations. A wizard in the RSA plug-in helps to replace participating classes by concrete counterparts, while preserving useful base structures and behavior. **Fig. 2** shows the result of a wizard execution to copy a CRUD interaction from framework. The engineer just has to inform names for the concrete instances.

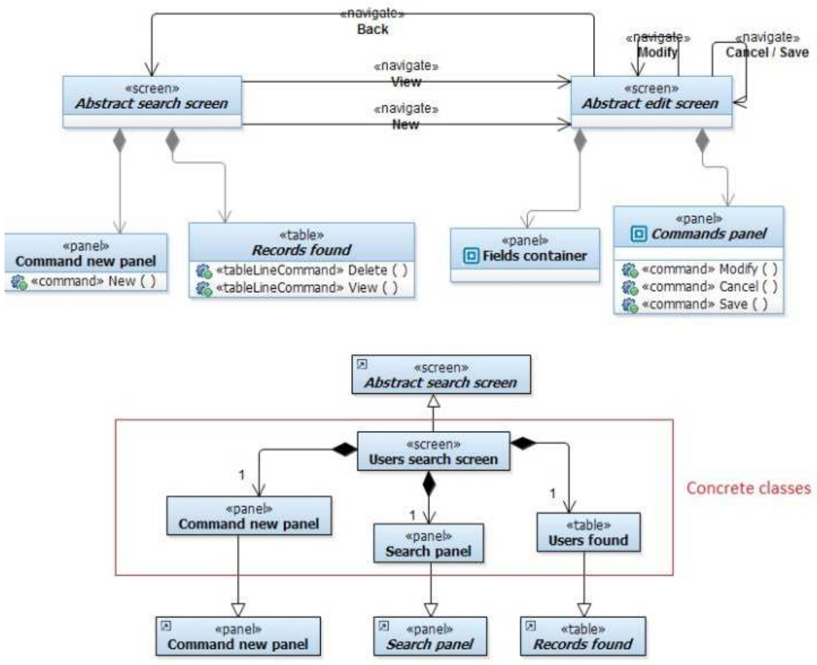


Fig. 2. Concrete participants generation

During the course of a project, engineers are able to create other abstract interactions, thus extending framework capabilities. The interaction copy wizard will be able to work on the new base interactions.

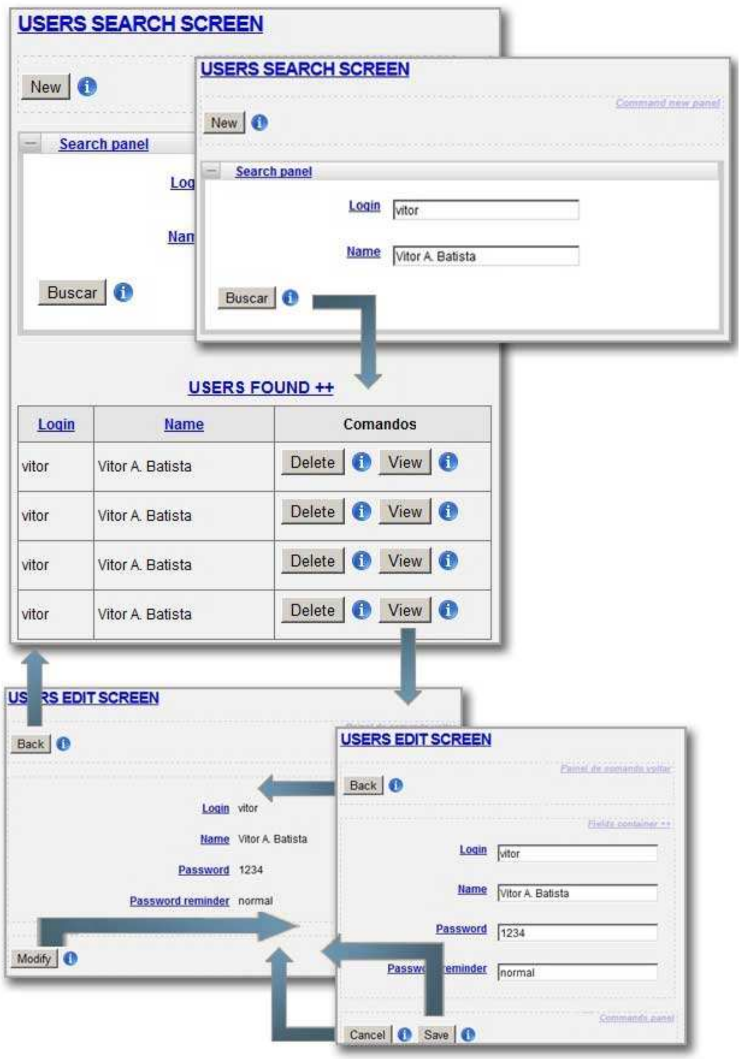
3.3 Referring to persistent entities attributes

Fields in UIs usually correspond to entity class attributes. With this in mind, our stereotype for UI fields holds an association to the entity attribute from which the data come from or where they are stored. This way, after the engineer generates his concrete instance of the framework collaboration, he creates attributes in the UI widgets, applying on them a stereotype whose properties refer to an entity attribute. In Praxis-Synergia, this means that entity attribute documentation is “inherited” by the corresponding UI field. Figure 1 shows a group of stereotypes and their tagged values for attributes.

The RSA plug-in helps to create UI fields from entity attributes. The engineer just select attributes from entities, and new attributes are created in UI widgets referring to the selected entity attributes.

3.4 Prototype generation

After all application UIs, and their attributes, commands and associated behavior are modeled, the RSA plug-in can generate the application prototypes. This is made possible by the level of formality imposed by Praxis-Synergia. The RSA plug-in converts each class, attribute, operation and stereotype's tagged values into navigable HTML.



Login: vitor
Name: Vitor A. Batista

Login	Name	Comandos
vitor	Vitor A. Batista	Delete View
vitor	Vitor A. Batista	Delete View
vitor	Vitor A. Batista	Delete View
vitor	Vitor A. Batista	Delete View

Login vitor
Name Vitor A. Batista
Password 1234
Password reminder normal

Login: vitor
Name: Vitor A. Batista
Password: 1234
Password reminder: normal

Fig. 3. - Generated prototype of a CRUD interaction

The generated prototype also provides specification pop-ups for each element, making easier to validate requirements with the users, and to have them understood by the developers that will provide the real code for application.

Fig. 3 shows an example of the generated prototype for the interaction presented in **Fig. 2**.

3.5 Profile Constraints

Finally, but not less important, we analyzed all our reviews checklists and noticed that most of their items could be automatically verified. So, we introduced OCL constraints into our stereotypes to check the completeness and correctness of UML models.

After they finish the modeling of their concrete interactions, the engineers can execute the RSA validation procedure, and this will check all OCL constraints. Violations are marked as model errors, which the process requires to be fixed before reviews.

3.6 Other RSA plug-in automation

The RSA plug-in provides more automation functions than described in this article. A short list of other functions includes:

- Automatic generation of documentation requirements specification, often demanded by clients, from the UML model into a PDF or MS Word format, including all diagrams, elements and their stereotype tagged values;
- An Eclipse View to help filling stereotypes tagged values in RSA;
- Support to count Function Points directly from UML elements[4];
- Enhanced refactoring support to change references from one UML element to another;
- A Listener to model changes to helps to prevent inconsistencies in model.

4 Preliminary Results

In this section we present some benefits using the RSA plug-in, at Synergia.

The automated prototype generation feature of the RSA plug-in was first used in a small requirements specification project (813 Function Points - FP). We compared the effort needed to write the specifications of two current and similar projects, using the same process version and the same set of tools. Table 1 presents the results. The total effort measures the work in activities that directly produce the requirements specification. Activities like management and training were excluded, but reviews and rework efforts were included.

Although there is a difference between these two projects in size, the data seem to indicate that automated prototype generation indeed increases productivity and reduces rework. Requirements engineers that participated in both projects confirmed this impression.

Table 1 - PRODUCTIVITY COMPARISON BETWEEN MANUAL AND AUTOMATED PROTOTYPES

Project	*# use cases*	*# FP*	*Total Effort (h)*	*Productivity (PF/h)*
Manual Prototypes	128	2586	6727.32	0.38
Automated Prototypes	37	826	1441.83	0.57

All current projects at Synergia use stereotype constraints, but we were not able to compare them with older projects, since they also differ in many other aspects. Nevertheless, we interviewed our requirement engineers, and all of them confirmed that automated model validation, provided by OCL Constraints, significantly helps them to keep model integrity and consistency.

So far, we have not validated the UML reuse framework and interaction wizard in production, but we presented them to their future users, and its reception was very positive. We also made some informal experiments, asking engineers to model a CRUD interaction using the framework and the wizard tool, after they listened to their presentation. All of them finished their models in less than one hour.

5 Related Work

A number of researchers discussed the use of specifics tools and frameworks to support the software development process. X and Y [5] present a framework for formalizing a subset of UML diagrams, enabling their analysis. Examples of tools that check requirements consistency using semi-formal specifications are BVUML [6] and CDET [7] and VIEWINTEGRA [8]. Genius[9] and Janus[10] research projects allows the generation of user interfaces from specification of the application domain. Another approaches [11] [12] use scenarios as an input for the user interface prototype generation.

In the context of Requirement Engineering, our framework covers different types of integrated functionalities that support the RE activities. Regarding prototype generation, our approach focused on the UML class models. It does not require any model transformation or inclusion of additional descriptions (using other formalisms) in the models. The profile constraints bound to the stereotypes provide a quick way to enforce a set of rules mandated by process standards.

6 Conclusions and Future Work

In this paper we presented some problems faced at Synergia, regarding requirement engineering activities. We proposed a set of tools to automate those activities, and used UML extension mechanisms to improve their execution.

Although, so far, we do not have research-level data to support the claims of increased productivity and quality of the UML requirements models, we collected feedback from the requirements engineers; these were very satisfied with the improvements brought by the RSA plug-in. They identified other positive points, like significant reduction of manual work and increase in detected defects.

This development was carried out for one specific proprietary tool, IBM RSA. However, since we used strict UML 2 concepts, without tool-specific notation extensions, the environment should be portable to other tools that support full UML 2 in the Eclipse platform.

As future work, we plan to invest in model transformations to generate complete Java code for framework interactions, as well as its automated tests and tests specifications.

7 References

1. S. S. Rachida, R. Dssouli, and J. Vaucher, "Toward an Automation of Requirements Engineering using Scenarios," vol. 2. Journal of Computing and Information, pp. 1110-1132, 1996.
2. T. Hall, S. Beecham, and A. Rainer, "Requirements problems in twelve software companies: an empirical analysis," IEE Proceedings - Software, vol. 149, no. 5, p. 153, 2002.
3. B. Pimentel, W. P. P. Filho, C. Pádua, and F. T. Machado, "Synergia: a software engineering laboratory to bridge the gap between university and industry," International Conference on Software Engineering, 2006.
4. V. A. Batista, D. C. C. Peixoto, E. P. Borges, W. Pádua, R. F. Resende, and C. I. P. S. Pádua, "ReMoFP: A Tool for Counting Function Points from UML Requirement Models," Advances in Software Engineering, vol. 2011, pp. 1-7, Jan. 2011.
5. B. H. C. Cheng and L. A. Campbell, "Integrating informal and formal approaches to requirements modeling and analysis," in Proceedings Fifth IEEE International Symposium on Requirements Engineering, pp. 294-295.
6. B. Litvak, S. Tyszberowicz, and A. Yehudai, "Behavioral consistency validation of UML diagrams," in First International Conference onSoftware Engineering and Formal Methods, 2003.Proceedings., pp. 118-125.
7. J. Scheffczyk, U. M. Borghoff, A. Birk, and J. Siedersleben, "Pragmatic consistency management in industrial requirements specifications," in Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05), 2005, pp. 272-281.
8. A. Egyed, "Scalable consistency checking between diagrams - the VIEWINTEGRA approach," in Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), pp. 387-390.
9. C. Janssen, A. Weisbecker, and J. Ziegler, "Generating user interfaces from data models and dialogue net specifications," in Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93, 1993, pp. 418-423.
10. H. Balzert, "From OOA to GUIs: The janus system," JOOP, vol. 8, pp. 43-47, 1996.
11. J. Shirogane and Y. Fukazawa, "GUI prototype generation by merging use cases," in Proceedings of the 7th international conference on Intelligent user interfaces - IUI '02, 2002, p. 222.
12. M. Elkoutbi, I. Khriss, and R. K. Keller, "Automated Prototyping of User Interfaces Based on UML Scenarios," Automated Software Engineering, vol. 13, no. 1, pp. 5-40, Jan. 2006.