Requirements Processes: An Experience Report

Julio Cesar Sampaio do Prado Leite[*], Soeli T. Fiorini[*], Amador Durán[**], Beatriz Bernárdez[**], Juan Sánchez Díaz[+] and Emilio Insfrán Pelozo[+]

[*]Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brasil.
[**]Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, España.
[+]Departamento de Sistemas Informáticos, Universidad Politécnica de Valencia, España
[*]{julio,soeli}@inf.puc-rio.br. [**]{amador,beat}@lsi.us.es
[+]{jsanchez,einsfran}@dsic.upv.es

Abstract. Processes are certainly a key element in software management. Defining and using processes is believed to be an important factor towards quality. Our paper describes a general process language and the experience on its use by two different research teams with two different requirements processes. The process language is the basic representation for a process reuse repository implemented as a web application. The web application was used in order to describe a process for requirements management and a process for interface generation based on requirements information. We present both processes as well as an evaluation of the prototype. Our results are a confirmation of the importance of process reuse and of the possibility of sharing requirements information by publication, on the web, of requirements processes.

Keywords: requirements engineering, requirements process, process reuse, requirements management, interface

1. Introduction

In the last few years, a large number of new techniques and methods in different areas, have presented the process perspective as a new form of visualizing and improving production, business or support operations in organizations. In terms of organizational theory, there is a shift from the functional perspective to a cross-function one, based on processes. This trend is substantially enforced as globalization of the economy introduced the need for quality standards, and those are fundamentally based on process oriented audits. That is, to meet quality standards, organizations need to document and follow well-defined processes.

The quality movement has reached software engineering. Standards, following the pioneers CMM (Capability Maturity Model) [1] and ISO 9000, have emerged and most of them are centered on processes. Software process is the cornerstone of software quality. No quality can be achieved if the software development organization does not follow established and well-defined processes.

The first ideas about software process were geared towards automation. Defining software processes as programs could lead to greater automation in software construction. Today there are several environments that implement processes in an

automated fashion, however, as researchers found out, automating processes is not a trivial endeavor. As such, as in organizations, software engineering has also started to use the notion of process without relying in automation. The possibility of establishing process description and communicating these descriptions to others has been seen as an important managing tool for software engineers.

The growing interest in Process Engineering was an evolution of studies focused on products which verified that the quality of the developed software has a strong relation with the process which is used to elaborate it. In 1984, at the First International Workshop on Software Processes –(ISPW) a group of researchers learned about the new subject on process technology, which emerged. Afterwards, more than twenty workshops and conferences have been held (ICSP, IPTW, EWSPT, FEAST and PROFES).

Nowadays, attempting to improve processes, many companies try to reach levels of process maturity [2], based on their improvement and definition. However, those improvements are expensive, and they rely on the involvement of managers and teams, process data, people who know process modeling, training, and cultural change. Several factors, imposing difficulties, make companies spend long periods of time to define some processes [2], and some of them give up in the middle of the maturity process. A frequently mentioned process to accelerate this within a company is to replicate one organizational process reusable in other projects. At this point, the process descriptions are very important because they allow the knowledge to be reused.

Our paper reports on results from this standpoint. For us, processes are task descriptions that have a fixed objective and are informative on how humans, using different resources can accomplish an objective. Particularly we are interested in processes for requirements engineering. Working on software processes, Fiorini [3] has identified problems with existing software description languages and on the effective reuse of software process information. In order to meliorate the problems Fiorini has proposed an environment and a process language that enables a more complete description of processes as well as guide and facilitate their reuse.

In this context, we are presenting two different proposals of requirements engineering process. One focused on requirements management and the other on interface generation from user requirements. Both are results of research experience on requirements engineering and were performed by different research groups. As such, we see our article as contributing to the dissemination of processes and evaluating Fiorini´s proposal.

The rest of this paper is organized as follows. In section 2, the process definition language used in the experiences is defined. In section 3, the web tool developed for supporting the process definition tasks is presented. In sections 4 and 5, two experiences of process definition are reported. In section 6 some observations from the experiences are described. Finally, in section 7 some conclusions are presented and future work is pointed out.

2. The Process Description Language (ROpl)

A process modeling language is a formal notation used to express process models. A process model is the description of a process, expressed in a process modeling language [4]. The language we are using, ROpl, is a process description language geared towards reuse [5]. The language is a strong typed description language that focus on the enumeration of atributes for a given process. It is a static language and has no interpreter for its semantics. The grammar is defined in XML[6], a meta language for ROpl, allowing us to specify the structure of ROpl. As such, the information concerning ROpl structure can be sent/received and processed on the Web in an even way. Because the processes naturally have a hierarchical structure, we adopted XML, taking the advantage of its benefits to work with documents structuring.

The XML's DTD (Document Type Definition) contains or points out at the declarative marks, which define a grammar or set of rules to a certain class of documents. Since ROpl has three major types: *usual process*, *pattern process* and *standard process*, its DTD has three different representation for a process.

Following we provide an overall description of the three process types and the operands of each type. See [5] for more details.

2.1 Standard Process

It is a standard (for example, CMM or ISO) in a form of process. It is a base for process definition, according to specifics process improvement and quality assurance standards. It has a normative finality.

A standard process pattern is organized around the idea of *section*, *sub-section* and *item*. The structure of a standard process is described below, where, as in DTD specifications, the comma means sequence. Other features such as attributes or cardinality or have been omitted for the sake of readability.

Process-Standard :=
Name, Keywords, Objective, Applicability, Type, Description, Author, Version, Representation, Where-to-Find, Adaptation, Artifacts, Concepts, Actors, Section

Section :=
Name, Objective, Description, Sub-Section

Sub-Section :=
Name, Reference, Objective, Description, Representation, Training, Verification, Metrics, Tool, Item

Item :=
Name, Reference, Pre-Condition, Input, Description, Constraint, Output, Post-Condition, Pre-Condition, Input,

Recommendation, Constraint, Output, Post-Condition, Use-Pattern

2.2 Process Pattern

It is a process that deals with problems related to processes. According to the definition of patterns, it can not be new or hypothetical [7] [8].

A process pattern is organized around the idea of *community*, *family* and *individual*. That is, a process is composed of *macro activities* which are composed of *micro activities*. The structure of a process pattern is described below, using the same notation that was used for describing the structure of standard processes.

Process-Community :=
Name, Keywords, Origin, Objective, Classification, Problem, Context, Cause, Representation, Artifacts, Family-Process, Individual-Process

Process-Family :=
Id, Context, Cause, Control , Related-Pattern, Representation, Family-Solution

Family-Solution :=
Name, Pre-Condition , Input , Recommendation, Constraint, Output, Pos-Condition, Use-Pattern

Process-Invididual :=
Id, Context, Cause, Control , Related-Pattern, Representation, Individual-Solution

Individual-Solution :=
Name, Pre-Condition , Input , Recommendation, Constraint, Output, Pos-Condition

2.3 Usual Process

It is any existing process that is neither a standard nor a pattern. It is not a standard process because it does not have the normative finality and it is not a process pattern because it has not been tested (applied) a considerable number of times to solve one recurrent problem.

An usual process has 3 abstraction levels: the *process*, the *macro activity* and the *detailed activity*. That is, a process is composed of macro activities which are composed of detailed activities. Its structure is defined as follows:

Process :=

Concept, Actor, Verification, Metrics, Training, Method, Tool, Templates, Police, Artifacts, Name, Author, Classification, Type, Objective, Description, Pre-Condition, Pos-Condition, Macro-Representation, Micro-Representation, Conformance, Characteristics, Macro-Activity

Macro-Activity :=

Name, Description, Pre-Condition, Input, Output, Previous-Activity, Pos-Condition, Constraint, Actor-in-Charge, Method, Tool, Micro-Activity

Micro-Activity :=

Name, Description, Pre-Condition, Input, Output, Previous-Activity, Pos-Condition, Constraint, Actor-in-Charge, Method, Tool

Besides the three basic types, we also have a type called *solution process* which is an instance of either the framework or of any other process (pattern, usual or standard), or of the combination of those.

A process framework models the behavior of the processes in a given domain. A process framework is defined as a reusable process. Its kernel models the common activities among the processes of the domain. Hot -spots model the specific and flexible parts of the processes of the domain and they are specified during the instantiation of the framework. The hotspots are activities or other elements of the process (techniques, tools...), that define characteristics or specific paths of a solution process (process instance). The hotspots are instanced by the process engineer, redefining the description of activities or elements, both on the macro and detailed levels. The framework itself will have to be represented as a process, identifying the parts that are hot spots and common activities. The instantiation of one processes framework generates a solution process. ”[3]. See also figure 1.

However, accomplishing the mapping and documentation of processes is an arduous and painstaking job, particularly in large organizations, because it involves people who are expected to supply information regarding the manner in which they perform their activities.

Given this context, it is important that the infrastructure to support process description and further use be functional and ease to use. Based on a process reuse architecture [3], figure 1, a Web tool was developed, RPS (http://www.re.les.inf.puc-rio.br/soeli), in order to provide this infrastructure.

3. The Web Tool (RPS)

RPS is a Web tool designed to provide support for the edition and retrieval of process information. The tool uses ROpl as its basis. Figure 2 shows the main menu with two types of selection: by the menu on the left or by the chart, where you have

the possibility of reusing a framework or reuse directly patterns, standards or processes.

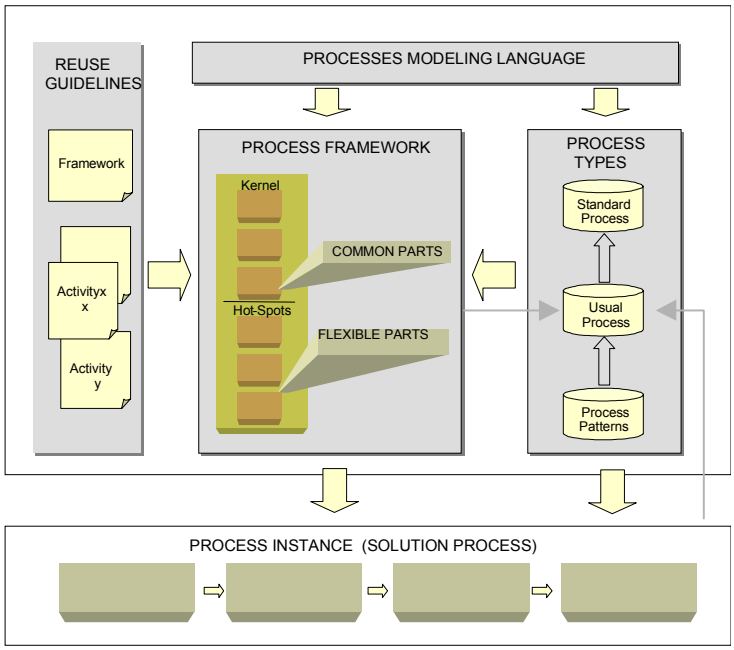

Figure 1. Process Reuse Architecture

The menu on the left give the option of a) data entry (cadastro), for cataloguing new processes, b) view a process (consulta), to visualize in HTML a given chosen process, or c) access control (utilitários), for managing access and passwords.

Several different pages are available according to the function requested. The processes are stored as XML [6] files and are prettyprinted using XSL (Extensible Stylesheet Language) [6], which makes possible different presentation styles for processes. Depending on the type of reuse (figure 1) a special path of the flow below is followed, that implies that there different possibilities of re-using the information stored in the data base.

RPS is a prototype and as such has several limitations, however we believe that it opens an avenue of opportunities regarding the research of how processes are presented, their languages and the reuse possibilities of processes. Our initial results [5] were very positive. On the next two sections we detail two different processes and a respective assessment of the tool for each process.

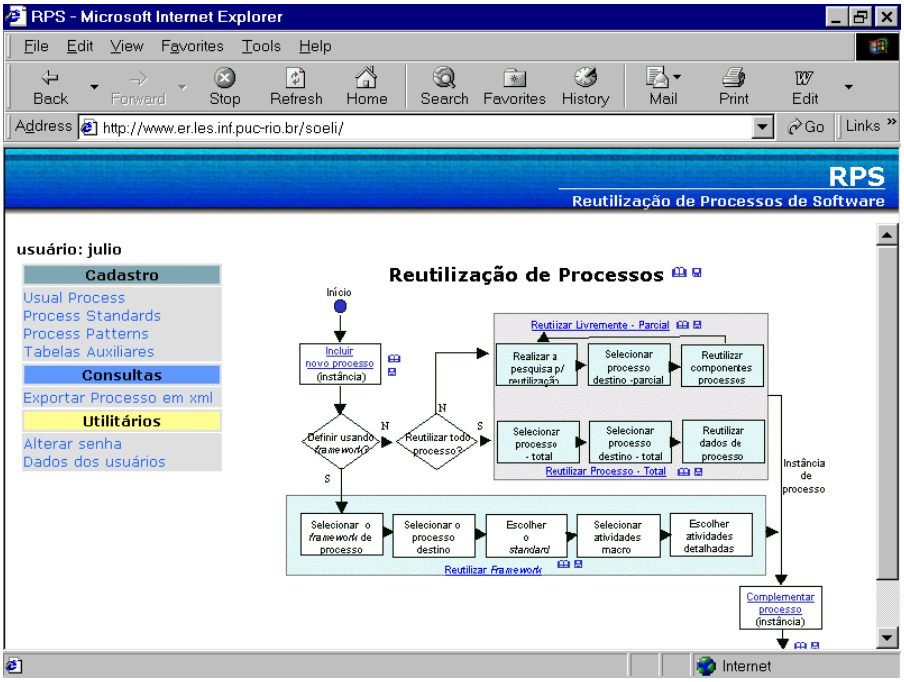

Figure 2. RPS main Page

4. First Experience: Requirements Management Process

The first experience took place at the Departmento de Lenguajes y Sistemas Informáticos at the Universidad de Sevilla and was driven by two members of the local Requirements Engineering Research Group. The goal was to describe a detailed process for Requirements Management (RM), taking the approach described in [9] as a basis and focusing on requirements change management. The RM process was initially modeled using UML activity diagrams [10], as shown in figure 3 and figure 4.

In figure 3 a context diagram for Requirements Engineering (RE) is presented. Following RPS, RE was defined as an usual process, and Requirements Development (RD) and Requirements Management (RM) as macro activities. In figure 4, a detailed description of the RM macro activity is displayed.

The former approach was defining RM as a process itself, but the structure shown in figure 4 was eventually chosen in order to maximize reuse of already defined processes. The reason of this choice was that all RM defined processes in the repository were at the macro activity level, and the current implementation of the web tool do not allow changing the abstraction level (process, macro activity or detailed activity) of an asset

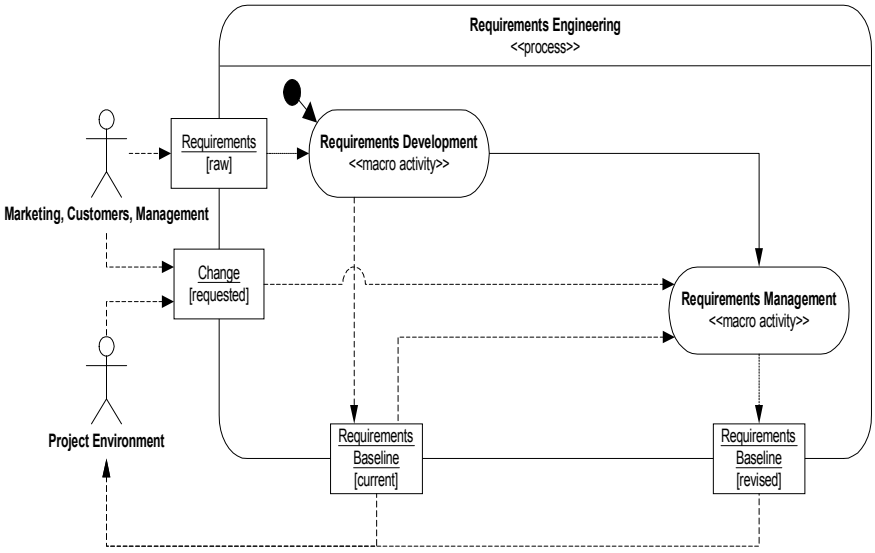

Figure 3. Requirements Engineering Process Overview

The framework reuse strategy was initially discarded after some attempts because the only available framework in the web tool process repository, a framework for RE, was too general and we found an usual process asset that was much closer to our approach. So, the free partial reuse strategy was finally chosen, achieving a high reuse ratio, although some translation work was needed, since the reuse processes was initially described in Portuguese and the target language was English.

5. Second experience: Semi-Automatic User Interface Generation from User Requirements

The second case study used to evaluate the tool is about a semi-automatic user interface generation method. This process is based on user requirements. The case study is organized as follows: first, our user interface generation method, and second each phase and activity is explained in detail. Finally the experience of using the tool for the process definition is reported.

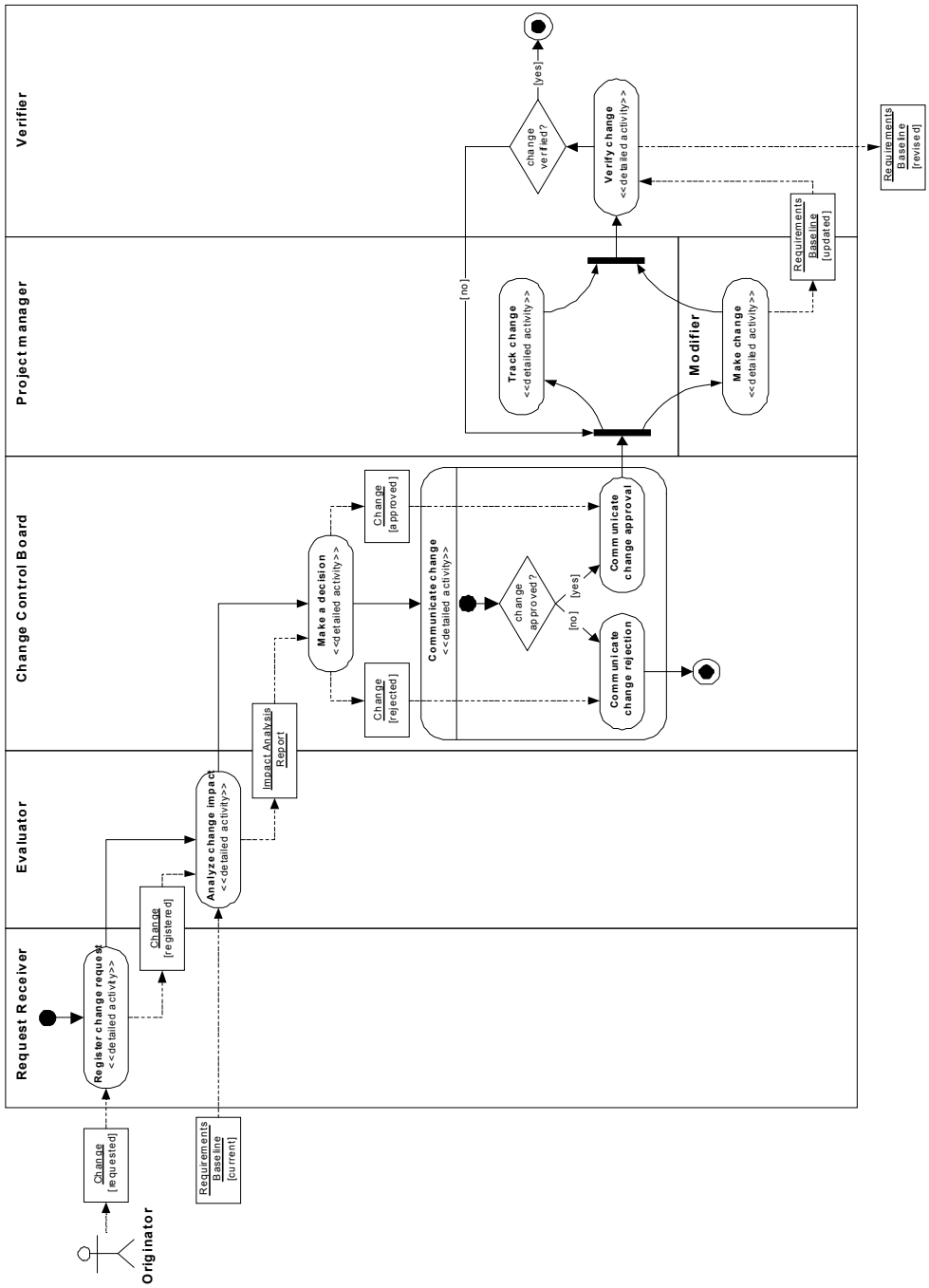

Figure 4. Activity Diagram for Requirements Management

When a software product is designed and implemented, it is very important to assure that the user requirements have been properly represented. To achieve this objective, a guided software production process is needed, starting from the initial requirements engineering activities and through to the resultant software product. In this section, a methodological approach for generating user interfaces corresponding to the user requirements is introduced. As opposed to other proposals, we defend the idea of having a process with a high degree of automation where the generation of user interfaces corresponding to precise user requirements has a methodological guidance. Furthermore, a corresponding visual tool, which allows us to almost completely automate the entire process, has been implemented. A detailed discussion about the tool can be found in [11]. An important contribution of the method is that it automatically generates an inter–form model of navigation which is based on the relationships include and *extend* specified among the use cases. The introduction of this navigation feature makes possible to use the generated interfaces in web environments.

In short, this section presents both a methodological proposal and the associated support tool, which backs it up, within the field of requirements engineering. They are based on the Unified Modeling Language (UML [10]), extended by the introduction of Message Sequence Charts (MSC) [12]. As we view MSCs as extended UML Sequence Diagrams by adding the needed stereotypes, the approach can be considered UML-compliant from the notational point of view.

A clear, precise iterative process allows us to derive user interface prototypes in a semi-automatic way from scenarios. Furthermore, a formal specification of the system is generated and represented through state transition diagrams. These diagrams describe the dynamic behavior of both the interface and control objects associated to each use case or each MSC. The method has four main steps: the first two steps require analyst assistance to some degree, whereas the last two steps make the process of scenario validation fully automated by means of prototyping

5.1 Description of the Proposal

In this section we present precise method or process to guide the generation of user interfaces corresponding to the user requirements, according to the ideas introduced in section 4 and using the UML diagrams class model, state transition diagrams and message sequence charts. Figure 1 shows a schematic representation of the activities contained in the proposed method. As we have commented above, the first two activities, namely scenario representation and synthesis of use cases, are manual activities which the analyst must carry out. The last two, specification generation and generation of prototypes, are totally automatic.

Figure 5 also fixes the order in which these activities should be performed. The process begins at the scenario representation stage where a use case model is created. The next stage consists of describing use cases by means of MSC. During the stage of specification generation, a state transition diagram (STD [13]) for the class User Interface and another STD for the Control Object are automatically obtained from a given MSC. Lastly, the final stage consists of automatically generating the user interface prototypes as well. The method is iterative; in consequence, the prototyping is used to validate and enrich the initial requirements.

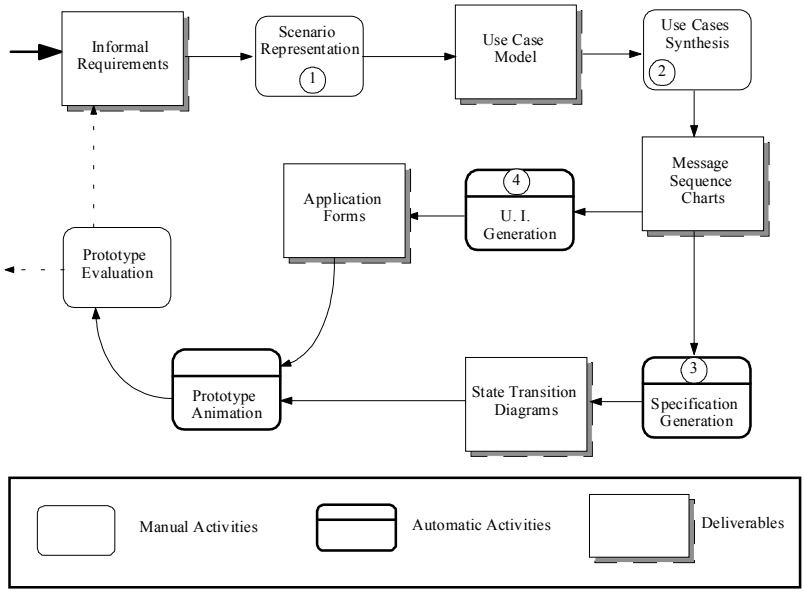

Figure 5. Schematic Representation of the Method

5.2 Tool utilization

We have considered as "usual process" the describe method in the last section. We can define four macro activities: *use case model construction*, *use cases synthesis*, *specification generation* and *user interface generation*. Each macro activity can be decomposed in detailed activities. We shall now proceed to explain the macro activity "use cases synthesis" in detail.

This macro activity can be decomposed in tow detailed activities: *message sequence chart construction* and *message sequence chart labeling*. The former detailed activity can be described as follow. Once we have obtained the Use Case Model, we need to work with the involved use case information to undertake the process of designing a software system. To do this, use cases must be formally described: the formal definition of a use case is achieved by using a graphic, non-ambiguous language, such as MSC. In this phase, which is a manual one, the use case templates are used as help so that the analyst can detect the events sent by the actors and by the classes of problem domain. In each MSC, besides the participating actors (initiator, supporting actors), one class for the user interface and one class that acts as control object are introduced, according to the initial Objectory proposal [14] and according to the UML approaches for a software production process [15].

The last detailed activity also can be described in the sequel manner. An important piece of data that must be introduced in this step, is constituted by the labels that will appear in the user interface to identify relevant pieces of information. When following the flow of events specified in the MSC, a given piece of information enters or exits

the user interface object, the analyst must specify the corresponding label, that will play a basic role in the process of generating the user interface.

Apart from the labels, information about the type of the arguments of each event specified in the diagram must be introduced. The allowed types are the basic data-valued types (number, Boolean, character, strings, enumerated) and the object-valued types corresponding to the system classes. The types of the event arguments together with the class attributes are used in the process of the interface generation.

6 Conclusions

We have briefly presented an infrastructure for process reuse and detailed the ROpl, a process language proposed to cast processes for reuse. The infrastructure is supported by a web application that provides editing, retrieving and visualization mechanisms. In order to gain more understanding on requeriments processes as well as on the effectiveness of the prototype tool, we conducted two experiences with two different research groups. We detail below the final remarks of each research group. We end the section enumerating further work.

The first experience at the Universidad de Sevilla was highly positive. The search facilities were a key aspect, and the facet-based schema has proved to be extremely useful. The reuse guideline for frameworks was also very useful, especially because it offered a broad view of the framework, its activities and some guides for making reuse decisions. The flexible view of the concept of framework was also a positive aspect, since any activity in a framework can be reused or not, not only hot spots, although some of them are strongly recommended but no one is considered as mandatory.

One of the drawbacks was the strictly sequential model of processes and the lack of an standard graphical representation of processes. It would be interesting to have a richer model in which processes could be ordered in a more flexible way. Other found problem was the strict level hierarchy, in which an item initially created at one level (process, macro activity or activity) cannot be changed to an upper or lower level and, in the case of detailed activities, they must be reused together with their parent macro activity, not isolated.

From the point of view of "*Universidad Politécnica de Valencia"*, we have encountered a powerful mechanism of process definition and reutilization, but we have only fully exploded the former characteristic, *process definition*. There is a coherent classification of process types: standard, pattern, usual and solution. The variable level of granularity in process definition and decomposition (macro and detailed activities) let us employ the desire level of abstraction and detail.

In the other hand, there are also negative aspects. There is not a graphical notation to depict process and activities dependency. The precedence process relation is very restrictive. Finally, there is not a view model or users and users groups. Any user can access to all database. We have not found a privacy policy and information protection.

Regarding future work, we understand that different issues vary in terms of effort needed and maturity over time. The short term issues are those related to the tool, like: better user interface, language customization, privacy and security, and of course a more stable platform. Requiring more effort would be things related to the language concept itself, for instance, during the first experience, it was found as interesting the

possibility of having more than 3 levels of abstraction. It would be nice if recursive composition could be applied when defining a process, i.e. if a process could be defined as recursively composed by other process, having as many depth levels as necessary. In terms of long term we would be interested in auditing and populating the repository together with a more robust tool. Collaborations like the one presented in this paper, we believe, will foster the possibility of improving the repository as well as provide a larger user base for testing the tool.

References

[1] Paulk, C.M., Curtis, B., Chrissis, M. B., Weber, V.C.: *Capability Maturity Model for Software*, Ver. 1.1, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-93-TR-24, ESC-TR-93-177, 1993.

[2] SEI, *Process Maturity Profile of the Software Community 1999 Year End Update*, SEMA 3.00, Carnegie Mellon University, March, 2000.

[3] Fiorini, S. T., Leite, J. C. S. P., Lucena, C. P.: Process Reuse Architecture, Klaus R. Dittrich, Andreas Geppert, Moira C. Norrie (Eds.): Advanced Information Systems Engineering, 13th International Conference, CaiSE 2001, Interlaken, Switzerland, June 4-8, 2001, Proceedings. Lecture Notes in Computer Science 2068 Springer 2001, pp. 284-298.

[4] Finkelstein, A., Kamer, J., Nuseibech, B.: *Software Process Modelling and Tecnology*, Research Studies Press Ltda, 1994

[5] Fiorini, S. T., *Arquitetura para Reutilização de Processos de Software* (Software Process Reuse Architecture), PhD Thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, April, 2001.

[6] http://www.w3.org/TR/REC-xml and http://www.w3.org/Style/XSL/

[7] Coplien, J. O.: *Software Patterns*, SIGS Books & Multimedia, USA, 1996.

[8] Gamma, E., Helm, R., Johnson, R., e Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Design*, Addison-Wesley, 1995.

[9] Wiegers, Karl E., *Software Requirements*, Microsoft Press, 1999.

[10] Booch G; Rumbaugh J; Jacobson I, *The unified modeling language,* Addison-Wesley. 1999.

[11] Sánchez J; Pastor O; Fons J; From user requirements to user interfaces: a methodological approach, Klaus R. Dittrich, Andreas Geppert, Moira C. Norrie (Eds.): Advanced Information Systems Engineering, 13th International Conference, CAiSE
2001, Interlaken, Switzerland, June 4-8, 2001, Proceedings. Lecture Notes in Computer Science 2068 Springer 2001, pp. 60-75.

[12] ITU: Recommendation Z. 120: Message Sequence Chart (MSC). ITU, Geneva, 1996.

[13] Harel D.: *State Charts: a visual formalism for complex systems,* Science of Computer Programming, 8(3), 231-274. 1987.

[14] Jacobson I et al. *Object-Oriented Software Engineering: A use case driven approach*. New-York ACM Press, 1992.

[15] Jacobson I; Booch G; Rumbaugh J. *The Unified Software Development Process*. Addison-Wesley, 1999.