Open Issues in Requirements Modeling for Reuse [⋆]

Miguel A. Laguna[1], Francisco J. García[2], Oscar López[3], and José M. Marqués[1]

[1] University of Valladolid
{mlaguna,jmmc}@infor.uva.es
[2] University of Salamanca
fgarcia@usal.es
[3] Technological Institute of Costa Rica
olopez@infor.uva.es

Abstract. Due to the many forms that software requirements take, their modeling and relationships with other reusable elements in a repository can signify the success or failure of a whole reuse programme. This article looks at the need to standardize the requirements to make them effectively reusable, and this can initially be achieved by using patterns. In addition, alternatives to obtain consistent sets of requirements related to reusable elements on other levels of abstraction are explored, thus producing an external reuse interface of more complex elements. Finally, guides are proposed for finding patterns common to different elements and for obtaining generic requirements which can be adapted to multiple situations by using parameterization.

Keywords Requirements engineering, requirements reuse, generic requirement, scenario, use case, workflow, mecano model

1 Introduction

The systematic reuse of software is based on the storage of reusable elements in repositories for future use by means of either composition or generation [20]. The approach used by the Research Group in Reuse and Object Orientation (in Spanish, GIRO), at the University of Valladolid, Spain, is based on coarse grain reusable elements, called *mecanos* [12], which can be used with a hybrid outlook consisting of composition plus generation. A mecano is a complex structure made up of fine grain elements (*assets*) that belong to different levels of abstraction (analysis, design and implementation) and which maintain intra- and inter-level relationships. Mecanos are built as static configurations of assets. However, the reuse process offers the possibility of automatically generating mecanos through the composition of the assets that respond to the needs of the reuser.

The generation of mecanos is based on the classification and recuperation of assets. The information concerning the attributes of the assets that make up a mecano can be classified with the help of methods based on natural text and facets [33]. The ontological models [2] and Case-Based Reasoning [13] are alternative solutions to the problem of

⋆ This work is sponsored by "DOLMEN" Project within CICYT, TIC2000-1673-C06-05, Ministry of Technology and Science, Spain.

searching for assets in the repository. However, the classification of mecanos should offer information on the global functionality they have, that is, the set of their functional requirements.

Both the search for reusable elements and the automatic composition of mecanos start from the requirements proposed by a reuser so as to recuperate those elements from the repository that best fit the requirements. In other words, the requirements are the access gate to the elements stored in the repository. The success of software reuse depends, to a great extent, on how the requirements are treated, in both the definition phase for reuse and in the search phase during the development of reuse.

This article explores different possibilities for dealing with the requirements in the reuse context. Section 2 proposes a strategy for dealing with the diversity of requirements and their relationships within the context of reuse. Section 3 deals with the need to standardize the requirements and proposes some solutions, including the transformation of requirements into requirements of a different kind. Section 4 looks at the way of relating and grouping requirements to make up coarse grain assets. Section 5 proposes the definition and inclusion in the repository of generic requirements with the same objectives as the design patterns, but on a higher level of abstraction. In addition, the possibility to instantiate complete mecanos from generic requirements connected to other assets of analysis, design and implementation is explored. The conclusions and future work are included in the last section.

2 Requirements typology, description and structure

Reusable elements show varying degrees of formalization. The code assets, design specifications and formal specifications, for instance, have a greater degree of formalization than those belonging to the requirements level. Requirements assets are frequently based on natural language, which gives rise to problems of ambiguity, uncertainty and a lack of definition. When speaking of requirements, very different elements are being referred to:

- Requirements with different degrees of formalization: natural language, semiformal, formal notation
- Declarative (rules) and procedural (scenarios) requirements
- Aims and means to reach them
- Requirements of varying granularity: from requirements documents, global requirements, functional requirements, atomic requirements
- Functional (including information requirements) and non-functional requirements
- User requirements and developer requirements [4]

Pohl [31] quotes several accepted definitions of requirements and establishes a very detailed classification of the different types. He firstly distinguishes between three major blocks: functional, non-functional and information requirements and he reviews the treatment of the requirements included in 21 documents of standards and methodological guides. On the other hand, he considers three dimensions in requirements engineering: specification (from least to most complete), representation (from least to most formal) and agreement (from different personal views to a common view).

From our perspective of mecano reuse, the requirements which are of interest to us are: user, functional, fine or medium granularity, procedural or declarative format and low level of formalization. The reason for this choice is that the end users see their problems in this way and the search in the repository of one or several mecanos that solve these problems should start from this basis. However, in later phases, the non-functional requirements can be taken into account. In order to deal with these requirements it is necessary to introduce a certain degree of standardization in the assets so they are more easily identifiable, comparable and can be more easily related to each other.

The most frequent approaches, from the point of view of the end user, are the scenarios, in their diverse variations, and the business rules. The most widely used scenarios are the use cases, initially introduced by Jacobson [16], becoming widely known in [17] and updated in UML [3]. However, other variations should be considered, in particular the business processes or *workflows*. The scenarios are usually based on natural language or, in the best of cases, structured language. Thus, from the point of view of reuse, it is convenient that this type of requirements follow some kind of norm which allows them to be compared for their incorporation to new developments later.

Business rules [14, 30], due to their declarative nature, can be described in a fairly formal way, by means of logical expressions, calculation formulas or decision tables. On the other hand, a complex rule can be expressed as a composition of other simpler rules (if *x*, then *y*, where *x* and *y* in turn are rules) and it is thus relatively simple to define a format for its introduction in the repository.

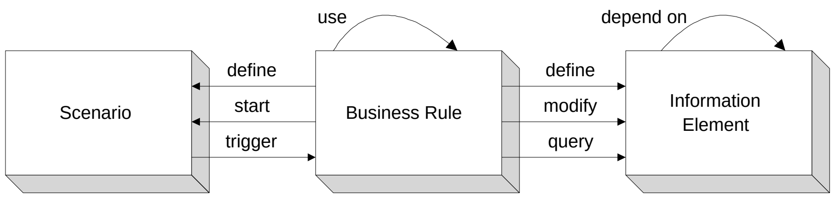


Fig. 1. Relation between business rules and scenarios

Business rules can show different levels of abstraction. From the reuse point of view, what is of most interest are the management rules (the highest level of abstraction). From the formal point of view, Loucopoulos [25] distinguishes three types of rules:

- Static and transition restrictions: Both types of restriction represent rules that must always be respected (the assertion must always be true)
- Rules with a static and transition origin: Are used to define information in terms of other primitive rules
- Rules of action: Invoke transactions if the precondition is met

Static rules can be incorporated in structural diagrams (for instance, using a language of restrictions such as OCL [40] in a class diagram). As for dynamic and action rules, these can be found in the form of triggers in the designs of databases or, also, in the form of restrictions in dynamic models. Although this form of expressing the rules allows a high degree of formalization, the fact that they are embedded in other requirements makes their independent reuse difficult.

In any case, the relationships between rules, scenarios and information requirements can be represented. Figure 1 represents interaction between the system and its users. The scenarios represent a natural way of representing this interaction. The system connects inputs (stimuli) and outputs (replies). The replies to a valid input are constructed with respect to the input, the state of the system and the business rules. The latter relate scenarios with the state of the system and should be taken into account on defining the coarse grain assets and the relationships between the requirements of which it is made.

From the structural point of view, Durán [8] breaks down the scenarios (as use cases), in addition to the information and non-functional requirements, into their elemental parts. This possibility of breaking a requirement down into its atomic parts (for instance, a step in a scenario, an element of information or a business rule condition) is fundamental, not only for storage in the repository in a controlled way, but also in order to exchange, or even automatically generate, requirements in different formats, compatible with different tools.

As for requirements in the reuse process, outstanding authors [10, 32] maintain that their introduction substantially improves the development process. However, since the requirements represent the main means of communication with clients, its format is not suitable for representation in a computer, and much less for reuse in a simple and direct way. Given that the simple classification and search techniques are not too effective, the most promising methods include development environments and CASE tools [32] or knowledge representation techniques, such as those proposed by Lowry [26]. Cybulski [6] has revised the different approaches used. In more recent studies, Sutcliffe & Maiden [39] propose the recognition of patterns by means of analogy as a solution to the problem of scenario comparison.

All these reuse techniques stress the semantics of independently obtained requirements. Alternatively, Domain Engineering proposes the creation of reusable requirements from the start. FODA [19] seems to be the most representative technique within this tendency. The most up to date methods, such as ODM [38] or FeatureRSEB [15] are, directly or indirectly, based on this technique. FORM [23] represents the development of FODA towards the paradigm of object orientation.

3 Standardization of functional requirements

Faced with the diversity of requirements, a standardization of the information concerning requirements included in the assets is necessary. This standardization would hopefully make dealing with the requirements and the individual comparison of the assets present in the repository with the user needs a lot easier. In the GIRO repository a format based on templates has been chosen. The decision to use this format of representation is justified in the following points:

– Natural language is the basis of communication in the requirements elicitation phase, which means that the most natural way of documenting them is to use a kind of mechanism based on natural language.

– The templates are limited to documenting requirements, independently of the development paradigm that has been followed.
– This template format is well suited for presentation in HTML pages, which greatly facilitating the creation of mecano navigation systems through hypermedia.
– Its tabular structure is also useful for finding exchange and storage formats in the repository, in such a way that, not only is its presentation easy, but also its treatment. In this sense XML (*eXtensible Markup Language*) seems to be the ideal means for storing these functional descriptions. It is also extremely simple to define the DTD (*Document Type Definition*) corresponding to the grammar of the labels to describe the template inputs.

Our initial experience is based, particularly, on the templates and linguistic patterns proposed by Durán [8], which can be used both during elicitation meetings with clients and users and to register and manage the requirements stored in the repository. As a result of such experience, for some template fields, standard phrases have been identified that are usual in requirements specifications. These phrases, called by their author *linguistic patterns*, must be completed with the correct information. The proposed templates are as follows:

- Information requirements template
- Functional requirements template
- Non-functional requirements template
- Objectives template
- Actors template
- Conflict template

The structuring of the information in the form of a template and the "standard" phrases proposal facilitate the writing of the requirements, guiding the developers to reuse in feeding the repository. For the purposes of this article, the most interesting templates are those related to functional requirements, information requirements and the objectives. The objectives of the system can be considered as *high level requirements* [34], so the requirements themselves would be the means to achieve the objectives.

The domains dealt with are University management, Disability tools and Digital image treatment. [12] gives details of the experience and the conclusions obtained during the introduction of the corresponding assets in the GIRO Repository. Other groups have used the said repository to introduce their own requirements (specifically, requirements concerning information security, recuperated and used without modification in several industrial projects). Initially, a tool already in existence was used, EUROWARE [35], partially adapted to the mecano model for the implementation of the repository. At present, our own repository is used, developed by a conventional database manager which implements the complete model (http://giro.infor.uva.es).

From this initial experience, the need for some kind of additional standardization that goes beyond the use of templates has become clear. Thus, determining the initial functionality of the system software from the description of the future user's current manner of working is proposed in [21]. One of the lines of research of the GIRO group is aimed at developing this proposal by using a tool to determine the functional requirements of an information system and its storage in the form of analysis assets. The starting point is to use the workflows modelled with Petri nets, thus permitting the automatic generation of use cases and of other reusable elements on the analysis level (*analysis assets*) that can be included in a repository to construct mecanos. In this way,

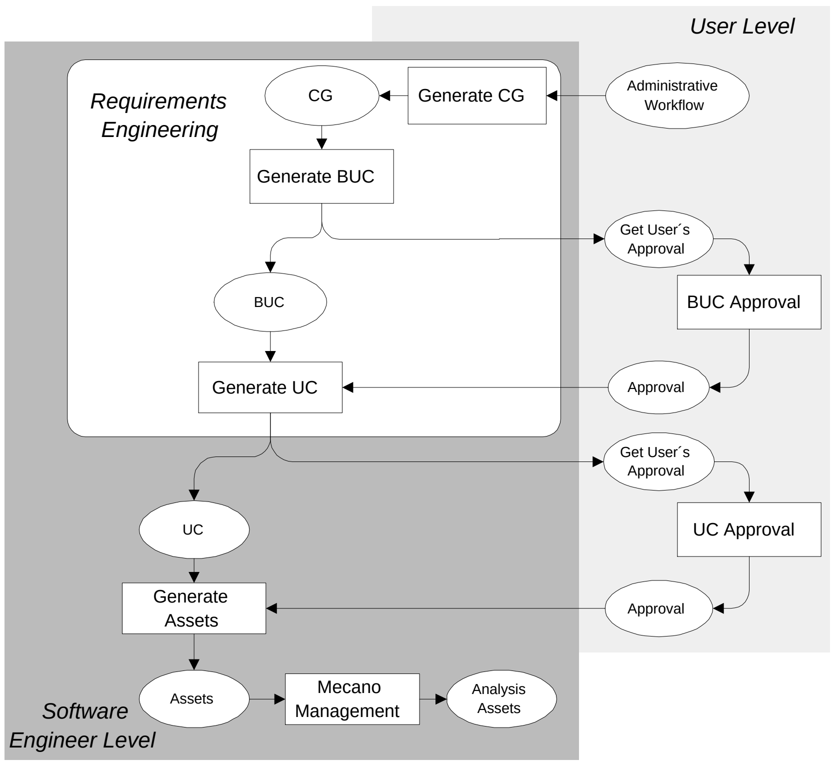


Fig. 2. General framework for the standardization of user requirements capture

the strength of the use cases is reinforced by the use of workflows which allows scalability and traceability. At the same time, the informality of the use cases is corrected through a robust formalism provided by the Petri nets. To sum up, the standardization of the functional requirements capture process is proposed to generate use cases included within mecanos.

Figure 2 shows the reference framework used to standardize the requirements. Two levels are established with relation to its elicitation process: The user level and the software engineer level. The former has an external view of the system as a black box, while the latter corresponds to the interior of the said box. On the software engineer level there is a view of the requirements engineer which acts as an interface between the two previous levels. The user provides the first approach to the system's functional requirements through the definition of *administrative workflows*. A specific type of workflow is used, the Document-Task diagram [21], which, used correctly, complies with the standards defined by the *Workflow Management Coalition (WfMC)* [41]. In this stage, the methodological proposal provides a preliminary definition of the user requirements and the functionality of the system is modeled from this by using a case graph (GC). By analyzing the case graph, business use cases (BUC) and use cases (UC) are obtained. Finally, the assets generated in this way are ready to be used in the development in the context of software reuse. Consequently, the general framework of the proposed solution leads to analysis assets that are suitable for being associated, through the repository management interface, with the corresponding design and implementation assets to form the required mecanos. In [27] this proposal is developed in detail, using a formalism based on Petri nets, and is applied to the real case of an electric company. Obtaining requirements assets automatically guarantees their standardization in the repository.

3.1 Requirements Transformation

Within the requirements standardization actions, the problems of the transformation of requirements from one type to another is dealt with. The aim is to find a common representation for the requirements that implies user interaction (mainly scenarios and workflows). Different views have led to a plethora of techniques that have a lot in common:

- Software engineers handle mainly use cases (and, in general, scenarios) to represent user-system interaction
- Business management technicians use tools for business process reengineering (BPR)
- Starting from use cases, extensions (business use cases or BUC) are proposed for representing business processes and to allow their study and improvement
- Workflow managers take over in the large companies as the basis for co-operative work and for the checking and automation of complex processes
- The *WfMC* [41] proposes a standard framework for defining workflow models and their implantation
- UML derives a specialized statechart diagram (the activity diagram) to represent, exactly, the business processes [3]

The current situation leads us to lend special attention to two extreme levels. One, the global business level, which includes the workflows, BPR, and BUC. The other, the user-computer interaction level, where the scenarios or use cases are applied. In addition, the activities of a workflow can be seen as use cases, so the two techniques are related. The working hypothesis is that all these techniques are essentially identical and can be treated homogeneously so that an activity within a workflow is refined as another workflow, in a recursive manner, until an elemental activity is achieved which represents a simple interaction between an actor and a system.

The search for a common format to represent the different types of functional requirements internally, and to facilitate their comparison, is based on different works. Lee [24] maintains that a use case can be converted into a Petri net and, vice-versa, a workflow can be built internally as a Petri net [1]. How to use a workflow to automatically generate use cases has been illustrated above.

The work being carried out in this line includes the definition of a model that represents the relation between the different types of functional requirements through their common elemental components, defining a unifying language to integrate the different terminologies. To do so, we have defined a requirements metamodel, see figure 3. This metamodel includes the Sequence Specification Template as a common format to internally represent the different types of functional requirements in the repository and to facilitate their comparison.

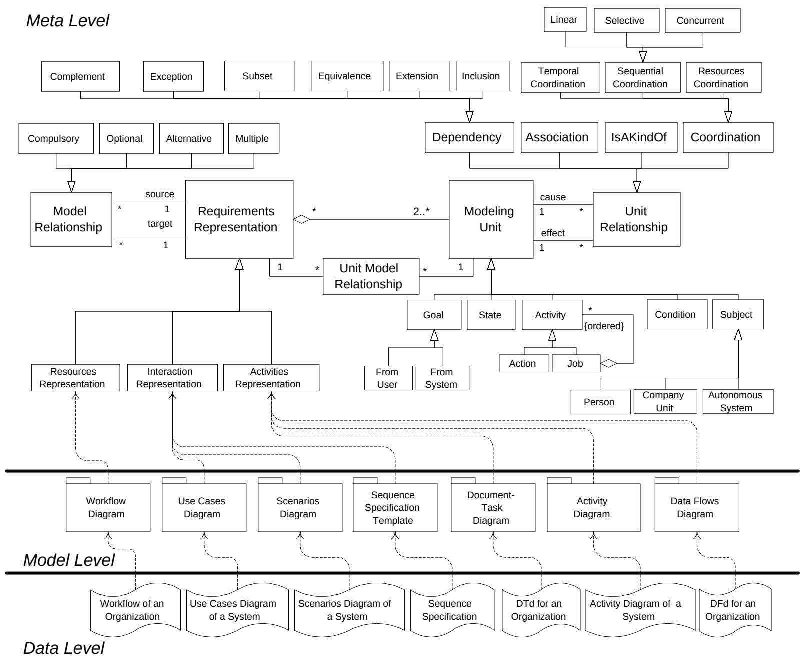


Fig. 3. Metamodel of Software Requirements Representations, expressed in UML. The requirements are represented by Modeling Units related to each other and with Requirements Representations that are also related to each other.

4 Coarse grain requirements: functional descriptors

For the reuse model based on mecanos, not only is an individual requirement important, but also, especially, the set of requirements for which a mecano represents a complete solution from analysis to implementation. This situation poses the problem of how to relate and group the functional requirements.

Some approaches to the problem can be found in the work of other authors, such as the concept representation nets [5], databases linked by means of *plug-ins* to a text processor with which the software requirements document can be modified [36] or the facades that gather the information from a reusable software element considered as relevant [18]. The variability mechanisms related to the facades use the relations of *include* and *extend* between use cases and can be implemented through the relations use and extend of the mecano model. [29] proposes a classification of the requirements in families and considers a system of requirements selection based on a structure in net form. On the other hand, as mentioned in section 2, relations can be established between the business rules and the scenarios: a user initiates a use case that triggers a business rule that uses information elements [7]. This implies that if requirements of various types are included in the repository, the relations between them can be defined in those terms (trigger, initiate, etc., all definable in terms of the intra-level relations of the mecano model).

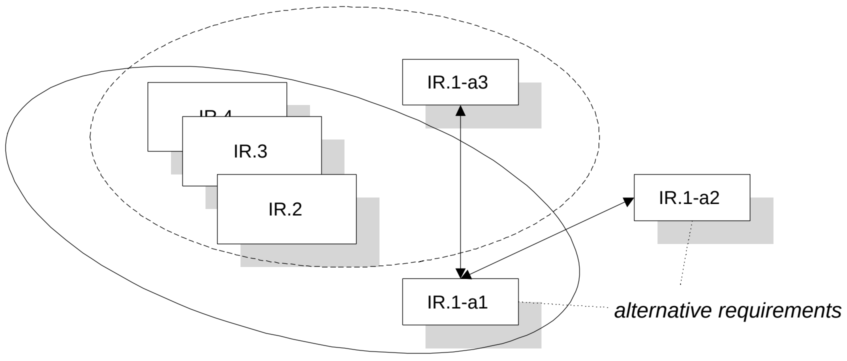


Fig. 4. Coarse grain requirements with alternatives

In the mecano model, the grouping of requirements is dealt with by means of functional descriptors. A functional descriptor is a set of links to those functional requirements that the developer for reuse wanted to stress, so each of these requirements is an asset component of the mecano. The presence of a functional descriptor in every mecano is not obligatory, but a mecano may have several, as functional views of the same mecano. At the same time, several mecanos can share the same functional descriptor [12].

Besides the clear aggregation relationship between the functional descriptor and the assets that form it, other relationships should be explicitly established between these assets which allow alternative but incompatible requirements to be selected or, optionally, to add requirements that are compatible with each other. On the other hand, there are dependency relations between functional requirements and class diagrams or between information requirements and data models, etc. This kind of relations used in the GIRO repository have an antecedent in the domain analysis techniques. The method used in FODA [19], and in FORM [23] which represents the evolution towards the Object Oriented paradigm, proposes three types of characteristics (*optional*, *obligatory* and *alternative*) and three kinds of relations between them (*composed-of*, *generalization* and *implemented-by*). The kind of relation *implemented-by* is really a dependence between requirements so that the implementation of a requirement necessarily implies the implementation of the second.

In the mecano model, the intra-level structural relations that can be used in an equivalent way are aggregation (or composition), extension and use. In addition, also available is the association as a bi-directional type of relation. The assets that describe atomic requirements similar to the graphs with characteristics (*feature graphs*) belonging to FODA or FORM can be linked to these relations, being grouped by aggregation to predefine different functional descriptor. These functional descriptor in turn are linked to other analysis, design assets etc, forming mecanos that suppose variations on the basic model (figure 4).

Mecanos built using these techniques can be found in the GIRO repository. The most immediate application of the mecano model with variations is to be found in the development of product lines. In this sense, once experimentation has been done in academically useful projects, the phase of applying it to real cases is entered. The first of them is being developed in the "Consejería de Agricultura y Ganadería de la Junta de Castilla y León" for controlling subsidies.

5 Generic requirements

Work concerning the reuse of requirements in families of similar applications already exists, for instance the work of Finkelstein [9], which is based more on models than on user requirements. Lam [22] bases his work on analogies and domain analysis to develop domain scenarios as a particular form of abstraction of the problem by means of closed compartments. He suggests that scenario reuse is useful in applications that belong to the same domain. For instance, the scenarios of a library system (book lending, returning, etc) are comparable by means of analogy with those found in a video renting system (video lending, returning, etc). Both applications belong to the resource management domain and share similar scenarios. Scenarios are adapted by means of *walkthroughs* and discussion with the clients. Sutcliffe and others [39] propose generic models of applications and, related to them, libraries of generic use cases that can be used to compare, through scenario generation, the use cases in storage with the new ones requested by the users. They use a strategy based on analogy for requirements reuse. The comparison proposal is based on a *matching* process based on first order and temporal logic.

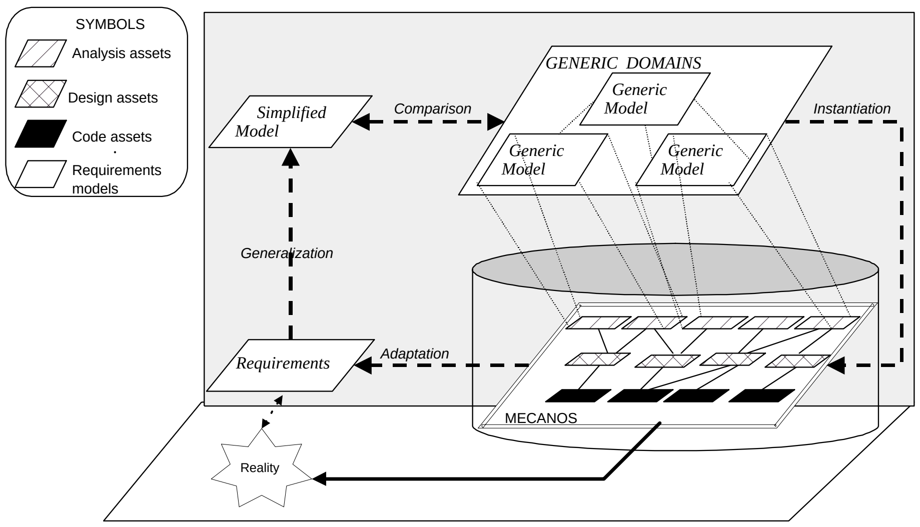


Fig. 5. Generic requirements as the key to mecano reuse

As set out in previous paragraphs, our proposals for reuse from requirements is based on the scenarios as the representation of the functionality. In this section, following the ideas of the above-mentioned authors, although with a different focus, an approach is presented based on generic scenarios as an alternative to the scenarios requested for each particular system. Figure 5 shows the model for reuse from functional requirements (possibly generated automatically from workflows). The scenarios must be generalized, in the sense of achieving generic requirements, in order to gain access to a higher level of abstraction. The generic scenarios obtained pass to the phase of comparison with the scenarios recuperated from the repository. It is after this comparison that the decision must be taken as to whether the recuperated scenarios (generic models) can be adapted to the problem to generate the corresponding mecanos in reuse time. These instantiated mecanos (complex structures formed by analysis, design and implementation elements) will be the basis on which the final application is developed. This adaptation can give rise to new versions that are fed back to the repository. There are four basic tasks for the reuse process of generic requirements:

- Represent the scenarios of the problem
- Generalize the scenarios of the problem
- Compare the generalized scenarios with those in the repository
- Adapt the selected/generated elements so they satisfy the requirements

The problem of representation has been dealt with in section 3. The adaptation of the related reusable elements will be considered in future work. Generalization and comparison make up the central theme of this section, in which we propose a way to deal with them. The model of the problem must be simplified to include the essential functionality and the generic domain models must be extracted from the repository. The simplified model of the problem must be compared with the generic domain model by using some kind of analogy, based on Petri nets.

Generalizing scenarios is based on the analysis by reduction of Petri nets [37]. The reduction techniques allow transitions and/or places based on specific properties, such as vivacity or limitation, to be eliminated or substituted, in such a way that the system is simplified, preserving their behavioral properties. The analysis by reduction is carried out by means of the following steps: eliminate implicit places, carry out the fusion of equivalent places and substitute places with groups of actions. From the conceptual point of view, reduction allows sequences of operations in the system to be grouped together and to substitute them with macro-actions in a reduced model. Figure 6 shows an example taken from [28].

The comparison of scenarios should be carried out on the basis of some significant element, common to the models. The goals of the actors, included explicitly in the models expressed as Petri nets, can be this element of comparison. Thus, a conclusion can be reached concerning the analogy between two models in terms of satisfying the goals. Starting from the generalized problem model, the search for analogies between scenarios can be carried out by selecting one or several generic models from the repository with a basic potential for reuse for the problem. A generic model is potentially reusable in a problem if two conditions are met:

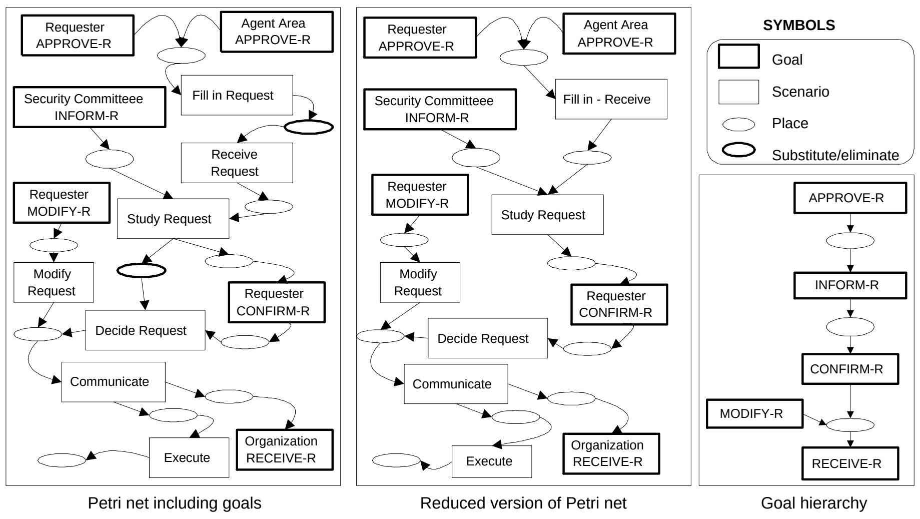


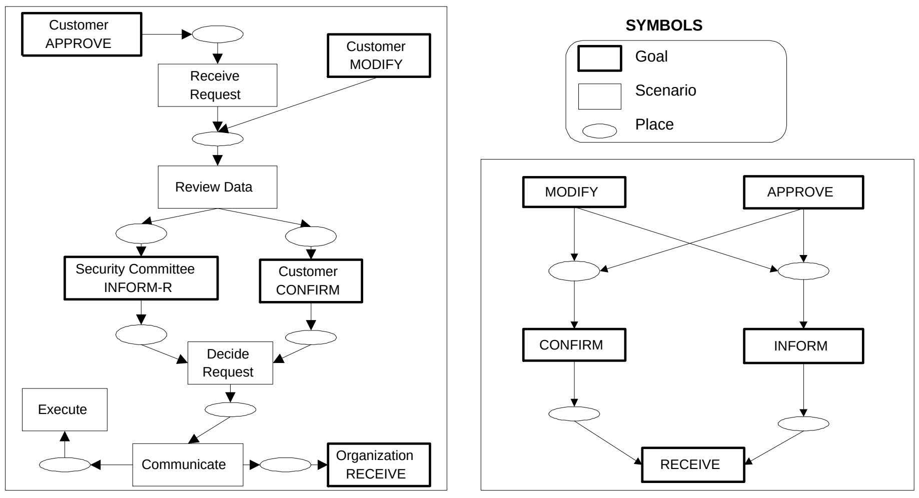
Fig. 6. Reduction of the Petri net of a request processing system

- The root goal and the leaf goals of the goals hierarchy of the problem are included (inclusion of nodes) in the goals hierarchy of the generic model, which requires the use of a common language
- The internal structure of the goals in the generic model (the goals hierarchy without considering the root nodes) satisfies the achievement of the leaf goals of the problem. This condition ensures that the domain has at least one logical path to satisfy the final goals of the problem

Once a generic model with a potential for reuse has been found, the generic scenarios must then be compared. Scenarios of the generic model must be found which can be applied to the problem. A generic scenario is applicable to the problem if:

- It starts from a root node of the generic model and it satisfies a leaf goal of the generic model
- It satisfies a root goal and a leaf goal of the problem

The sequences of generic transitions which satisfy the root and leaf goals of the problem are considered to be analogous scenarios to those corresponding to the problem. In order to prove this analogy, the sequences of transitions corresponding to each root aim, both in the analogous scenarios and the problem model, can be expressed by means of logical disjunctions and logical conjunctions. However, the real applicability can only be seen in terms of the semantics of the problem. This means that the general reuse process starting from the requirements must pass through the scenario adaptation stage. Determining the reuse potential of the generic model for the problem can be carried out by means of automatic theorem proving. The applicability of the generic scenarios in the problem can be expressed in logical terms, even though it can only be decided with respect to the semantics of the problem. The scope of this problem is given in detail in [28] and it is applied to a practical problem. Figure 7 shows one of the generic models used in the comparison.



Petri net for generic model of request processing

Goal hierarchy

Fig. 7. A generic model for the domain of request processing

The comparison process takes as its starting point the aims of the modeled system, which requires a common syntax to express the aims of the problem and the domain. Despite this weakness, our approach has the advantage that it can be made automatic. Unlike other approaches which compare requirements that use complex and interactive pattern recognition processes, including heuristic ones, we propose a process that can be made automatic and can be incorporated in the mecano model. Finally, the way of dealing with the generic requirements in the repository must be different from that of the simple requirements. If the latter were related by concrete design and implementation assets to form a mecano directly usable in reuse time, the generic requirements will form, along with the architectural and design patterns, a special variety of mecanos which cannot be used directly. What is obtained is in fact a generic mecano (or mecano pattern) which must be instantiated in each concrete occasion before being able to use it in a development situation with reuse. The adaptation may be similar to the well-known adaptation methods of the design patterns [11] or it may be based on parameters, so the substitution of a parameter in a generic requirement provokes a parallel substitution in the related design and implementation assets. The design of *wizards* type tools which make adapting the generic mecanos from the requirements to the design and implementation easier is a task which will be dealt with in the near future, once the process to be used and the transformation mechanisms have been defined.

6 Conclusions and future work

Several approaches for the treatment of software requirements, in the context of systematic reuse, have been presented in this paper. Although they have not been measured experimentally (this lies within the bounds of future work) the reuse of standardized individual requirements has been shown to be plausible. The automatic obtaining of requirements assets guarantees their standardization in the repository. The next step is to find a common representation for the greatest possible number of types of scenario so the requirements obtained from different sources can be compared.

When considering complex systems, the reuse of mecanos would seem to be more interesting, as the selection of a functional descriptor makes it easier, through the reification relationships, to obtain the corresponding design and implementation assets. To do so, the explicit definition of the intra-level relationships between the requirements makes the construction of the functional descriptor easier and represents a good basis for the use of mecanos in the development of software product lines.

The last proposal presented implies the use of generic requirements. The comparison of generic scenarios, from which the details have been eliminated, can be carried out through the use of Petri net reduction techniques and the search for analogies between them, based on the goals of the actors. The projected future work includes the design of wizards type tools to make the adaptation phase of both the requirements and the rest of the related assets easier.

References

1. W.M.P. van der Aalst. Three Good reasons for Using a Petri-net-based Workflow Management System. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, 1996.
2. Raquel Anaya. *Desarrollo y gestión de componentes reutilizables en el marco de Oasis*. PhD thesis, Universidad Politécnica de Valencia, 1999.
3. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, 1999.
4. J. W. Brackett. Software Requirements. Curriculum Module SEI–CM–19–1.2, Software Engineering Institute, Carnegie Mellon University, 1990. http://www.sei.cmu.edu.
5. Peter Clark and Bruce Porter. Building Concept Representations from Reusable Components. In *Proceedings of the AAAI'97*, pages 369–376, 1997.
6. Jacob L. Cybulski. Patterns in Software Requirements Reuse. In *Proceedings of 3rd Australian Conference on Requirements Engineering ACRE'98, Deakin University, Geelong, Australia*, pages 135–153, 1998.
7. Oscar Díaz and Juan José Rodríguez. Change case analysis. *Journal of Object-Oriented Programming (JOOP)*, 12(9):9–15,48, February 2000.
8. A. Durán, B. Bernárdez, A. Ruiz, and M. Toro. A Requirements Elicitation Approach Based in Templates and Patterns. In *WER'99 Proceedings*, Buenos Aires, 1999.
9. A. Finkelstein. Re–use of Formatted Requirements Specifications. *Software EngineeringJournal*, 3(5):186–197, May 1998.
10. W. Frakes and S. Isoda. Success factors of systematic reuse. *IEEE Software*, 11(5):15–18, September 1994.
11. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

12. Francisco J. García. *Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*. PhD thesis, Universidad de Salamanca, 2000.
13. Francisco J. García, Juan M. Corchado, and Miguel A. Laguna. CBR Applied to Development with Reuse Based on mecanos. In *Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering*, june 13-15, 2001. Buenos Aires, 2001.
14. C. Goti. A business classification for business rules. Technical Report TR03.563, IBM, Santa Teresa Lab., San José, EEUU, 1994.
15. Martin L. Griss, John Favaro, and Massimo D'Alessandro. Integrating feature modeling with RSEB. In *Proceedings of the Fifth International Conference on Software Reuse (ICSR-5)*, pages 76–85, 2-5 June, 1998. Victoria, B. C., Canada, June 1998. IEEE Computer Society, IEEE Press.
16. I. Jacobson. Object Oriented Development in an Industrial Environment. In *OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications*, pages 183–191, Orlando, FL USA, ACM, 1987.
17. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object–Oriented Software Engineering: A Use Case Driven Approach*. Addison–Wesley, 4^{a} edition, 1993.
18. Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse. Architecture, Process and Organization for Business Success*. ACM Press. Addison-Wesley, 1997.
19. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Feasibility study. Technical Report CMU/SEI-90-TR21 (ESD-90-TR-222), Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, November 1990.
20. Even-André Karlsson, editor. *Software Reuse. A Holistic Approach*. Wiley Series in Software Based Systems. John Wiley and Sons Ltd, 1995.
21. Miguel A. Laguna, José M. Marqués, and Francisco J. García. A user requirements elicitation tool. *ACM SIGSOFT Software Engineering Notes*, 26(2):35–37, March 2001.
22. W. Lam. Achieving Requirements Reuse: A Domain-Specific Approach from Avionics. *The Journal of Systems and Software*, 38(3):197–209, September 1997.
23. K. Lee, KC. Kang, W. Chae, and BW. Choi. Feature-based approach to object-oriented engineering of applications for reuse. *Software-Practice and Experience*, 30(9):1025–1046, July 2000.
24. W.J. Lee, S.D. Cha, and Y.R. Kwon. Integration and Analysis of Use Cases Using Modular Petri Nets in Requirement Engineering. *IEEE Transactions on Software Engineering*, 24(12):1115–1130, December 1998.
25. P. Loucopoulos. Business Rules Modelling: Conceptual Modelling and Object-Oriented Specifications. In *Proceedings of the IFIP Conference, TC8/WG8.1*, 1991.
26. M. Lowry and R. Duran. *The Handbook of Artificial Intelligence*, chapter Knowledge-Based Software Engineering, pages 241–322. Addison-Wesley, 1989.
27. Oscar López, Miguel Angel Laguna, and José Manuel Marqués. Generación Automática de Casos de Uso para Desarrollo de Software Basado en Reutilización. In *Actas de las V Jornadas de Ingeniería del Software y Bases de Datos*, pages 89–101, Valladolid, November 2000.
28. Oscar López, Miguel Angel Laguna, and José Manuel Marqués. Reutilización del Software a partir de Requisitos Funcionales en el Modelo de Mecano: Comparación de Escenarios. In *Actas de IDEAS 2001*, San José, Costa Rica, April 2001.
29. M. Mannion, H. Kaindl, and J. Wheadon. Reusing Single System Requirements from Application Family Requirements. In *Proceedings of the International Conference on Software Engineering*, 1999.
30. T. Moriarty. Business rule analysis. *Database Programming & Design*, 6(4):66–69, 1993.
31. Klaus Pohl. Requirements engineering: An overview. Technical Report TR 96/2, CREWS, 1996.

32. Jeffrey S. Poulin. Integrated support for software reuse in computer-aided software engineering (CASE). *ACM SIGSOFT Software Engineering Notes*, 18(4):75–82, 1993.
33. Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.
34. P. Sawyer and G. Kontoya. Software Requirements, in Guide to the Software Engineering Body of Knowledge, Version 0.95. Technical report, May 2001. http://www.swebok.org.
35. Sema Group. *Euroware User's Manual*. Sema Group, 1996.
36. A. Silva. Across Version/Variant requirement traceability in avionics software development and testing. In *Proceedings of the 2nd European Reuse Workshop (ERW'98)*, pages 179–198, November, 4-6, 1998. Madrid (Spain), November 1998. European Software Institute (ESI).
37. M. Silva. *Las Redes de Petri en la Informática y la Automática*. Editorial AC, Madrid, 1985.
38. Mark Simos, Dick Creps, Carol Klingler, Larry Levine, and Dean Allemang. Organization domain modeling (ODM) guidebook - version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121, June 1996.
39. Alistair G. Sutcliffe, Neil A. M. Maiden, Shailey Minocha, and Darrel Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 24(12):1072–1088, December 1998.
40. J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison–Wesley, 1999.
41. WfMC. Workflow management coalition terminology and glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, http://www.aiim.org/wfmc/standards/docs/glossy3.pdf, 1996.