Metamodeling for Requirements Reuse

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

[1] Technological Institute of Costa Rica, San Carlos Regional Campus, Costa Rica
olopez@infor.uva.es
[2] Department of Informatics, University of Valladolid, Spain
mlaguna@infor.uva.es
[3] Department of Informatics and Automatics, University of Salamanca, Spain
fgarcia@usal.es

Abstract. Correct requirements determination is a critical factor in software development as it takes resources and it is an error prone activity which can bring tragic consequences to the rest of the software life cycle. Having stored reusable requirements elements, both qualified and classified, in a repository might contribute to taking advantage of software development resources and to reducing the error probability in requirements specifications. However, the diversity of notations and formats, as well as the existence of different levels of requirements description make requirements reuse difficult. In this paper we present a metamodel to integrate some different types of semiformal diagrams into a requirements reuse approach. The description of reusable elements constitutes the basis for benefiting from diverse representations of requirements in the development of specifications by reusing requirements.

1 Introduction

In general terms, software reuse can contribute to an increase on productivity. In particular, since requirements engineering triggers the software development process [22], requirements reuse can empower the software life cycle [5]. Different authors, such as Cybulski [5], Sutcliffe and Maiden [25], and Mannion [13], among others, have pointed out that reusing early software products and processes can have an impact on the life cycle from two basic points of view: (a) allowing the software development resources to be more profitable, and (b) promoting the reuse based development across the entire software process. Quality and productivity in software specifications are favored by using proven and validated requirements components (*requirements assets*). This has all been successfully proven when applied to specific domains in industrial software production [14, 7].

In order to approach software reuse from the early stages of requirements elicitation and analysis, an adequate framework should be established. This framework makes demands on the support of structures and tools. Reusing requirements, as does reusing any

other software product, requires the assets to be represented, classified, stored, selected and adapted [4, 12]. These activities face the difficulties in dealing with the diversity and complexity of the requirements and models documenting requirements as principal trade-off. The management of diverse models requires a high level of abstraction to describe the common features of the different models. Metamodeling [17], ontological models [1] and case-base reasoning [8] have been applied as higher abstract structures. All of these approaches bring, as trade-off, the need for specific tools for management and maintenance of descriptive models, but for the case of metamodeling the tools may be more readily available or more easily built than in the other cases. Metamodeling is a standard way of integrating different models [9, 17]. Metamodels based on a standard language, for example MOF (Meta Object Facility) [19], give us the advantage of being supported by a well known and accepted meta-metamodel.

This paper is aimed at proposing a metamodel for reusable requirements as a conceptual scheme to integrate semiformal requirements diagrams into a reuse strategy. The diversity of modeling techniques and the existence of different levels of requirements description make requirements reuse difficult. To face these difficulties we have taken two main actions. First, we constrain the scope of the study to six widely known techniques, which mainly focus on functional requirements: scenarios, use cases, activity diagrams, data flows, document-task and workflows. Second, we establish a reuse framework which consists of three steps: (a) fixing a Diagram Family Model, (b) specifying a new application from the Diagram Family Model, and (c) requirements engineering as a basis for requirements family evolution. The theoretical basis for this reuse framework is the requirements metamodel.

The rest of the paper is arranged as follows: Section 2 presents a metamodel for the integration of diverse requirements representations into assets. Section 3 shows the description of use cases and workflows under the proposed metamodel. Section 4 presents a framework for requirements reuse. Section 5 concludes the paper and focuses on future work.

2 Modeling Reusable Requirements

Addressing systematic requirements reuse requires a model for reusable requirements element, which is a kind of "Asset". Furthermore, due to the diversity of modeling techniques, a metamodel for requirements is needed.

2.1 The Reusable Requirements

We consider as reusable requirements those which are represented in *Requirements Representation Model*, this is a kind of *Asset*, as shown in Figure 1. There are three different kinds of Requirements Representations Models: *Formal* (presenting a rigorous semantic and syntax), *Semiformal* (models which make the agreement among stakeholders easier) and *Non-Formal* (expressions in natural language, videos, images, voice and sounds to constrain the system to be developed).

We are mainly intended at reusing Semiformal kind of diagrams from the requirements engineering process. This process is characterized by necessities for communication and agreement between different people related to the process. These necessities
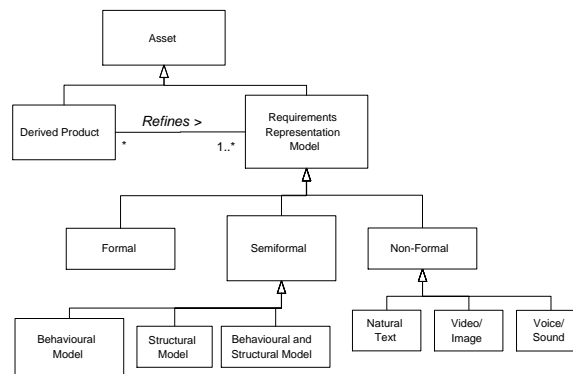
Fig. 1. The assets can be of two kinds: Requirements Representation Model and Derived Product.

promote the higher formality levels to be avoided in early stages of the software life cycle. As a consequence, formal models are not very common in early stages of the software life cycle. On the other hand, in despite of they are very useful in the requirements engineering process, non-formal diagrams show such ambiguity and lack of precision that it makes their reuse extremely difficult. Semiformal diagrams provide an intermediate formality level and constitute a fundamental product for the negotiation phase within the requirements engineering process. These semiformal diagrams are so important, it leads us to address their reuse. There are three kinds of semiformal models:

– Behavioural Models: These emphasize dynamic issues of the system such as allowing communication between the system and users to be modeled. The system behaviour is represented with activities or jobs which may be done by actors or subjects to fulfill certain goals or objectives under certain constraints and coordination mechanisms, e.g. Use Case Diagram and Data Flow Diagram.
– Structural Models: These provide a static view of the system as a components collection which interact with each other, or as a data set and corresponding constraint as support for the system, e.g. Class Diagram and Entity - Relationship Diagram.
– Behavioural and Structural Models: These incorporate both dynamic and static issues, e.g. Collaboration Diagram and object-oriented Petri nets.

Software requirements might be related to each other through traceability relationships. This means the requirements come together into a requirements specification document so that the application requirements can be better reflected. As an example, a use case diagram might be related to a class diagram. This traceability relationship between different diagrams is a central issue to be taken into consideration in the reuse context, and it is in practice often not managed. Although we do not address traceability relationships in this paper, the Requirements Representations Models come together the way is described in section 2.5. Besides, Requirements Representations Models are related to *Derived Product*. A Derived Product is any software product resulting from

refining a set of Requirements Representations Models, e.g. design elements which are derived from a requirements model.

With regard to software requirements data to be considered, we respect the Software Engineering Institute (SEI) recommendations. They classify the requirements from the point of view of the level of description. From this perspective there are two kinds of requirements [23]:

– C-Requirements: These mainly refer to the functional and non-functional issues that software products must fulfill, from the customer and user point of view. With regard to functional issues, there are typical diagrams in this category such as Use Cases, Workflows, Scenarios, and Activity Diagrams.
– D-Requirements: These refer to the functional and non-functional characteristics to be fulfilled by software products from the developer point of view. Typical diagrams in this category are State Machines, Interaction and Collaboration Diagrams.

Our requirements reuse framework is initially based on C-Requirements and semiformal representations. C-Requirements supply a higher abstract perspective than D-Requirements. Very specific details of software application included in D-Requirements make the integration and organization of requirements data, belonging to a set of domain applications, difficult. Semiformal diagrams specify the expected software services in such a way that they make the beginning of the requirements engineering process easier. Our reuse framework integrates D-Requirements, as well as design elements, by means of *Refines* relationship between Requirements Representations Models and Derived Product.

2.2 Requirements Metamodel

Because there are diverse modeling techniques among different development paradigms, we need a conceptual scheme to describe the requirements in the reuse framework. This conceptual scheme constitutes a Metamodel for different Requirements Representation Models (for simplicity, we shall call it the *metamodel*).

The central elements of the metamodel are the *Requirements Representation Model*, which describes different requirements diagrams, the *Modeling Unit*, which describes units that belong to the requirements diagrams, and the *Domain Objective*, which allows the Requirements Representation Model to be characterized, see Figure 2. In this metamodel, in its current development stage, there are different kinds of Modeling Units which correspond to Semiformal, Behavioural Requirements Representation Models. This metamodel also includes different relationships between modeling elements.

A Requirements Representation Model is related to at least one requirements Project. Each Requirements Representation Model is characterized by a Domain Objective which represents domain knowledge.

2.3 The Modeling Units

In Requirements Representations, which model the "Universe of Discourse", there are Modeling Units. There are six categories of Modeling Units which are instanced from the metamodel:
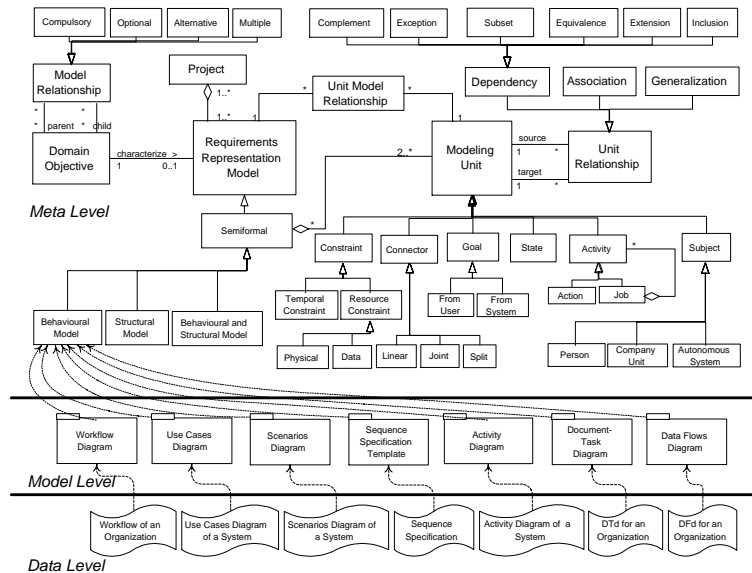
Fig. 2. Requirements Metamodel expressed in UML.

- Activity: This models a process which may be a job or an action. Job is an activity formed by other activities. Action represents an atomic activity, that is, one which is never subdivided.
- Subject: This is a person, company unit or autonomous system which is directly associated within and in charge of the activities.
- Goal: This represents the specific intentions of users and system in the context of interaction between users and systems.
- Constraint: This consists of information constraining the system functionality. It represents temporal constrains, or required resources for the system.
- Connector: This represents a criterion for semantic ordering of modeling units which can be of three kinds: Linear, Joint, or Split.
- State: This represents a dynamic situation (a period of time) during which an entity is satisfying some condition, performing some activity, or waiting for some event.

2.4   Relationships between Modeling Units

The *Unit Relationship* class allows the relationships between Modeling Units to be described in the metamodel. This relationship represents the links between elements inside a requirements diagram. Representing these links allows the content of different diagrams to be described and integrated in our reuse strategy. There are four kinds of relationships between Modeling Units in the metamodel:

- Dependency: This is a semantic relationship which establishes the changes in a Modeling Unit have an effect in another one. This is a six-kind relationship:

- Extension: It establishes an element to be defined in terms of another element definition.
- Inclusion: It establishes an element able to express its function in the context of another element.
- Equivalence: It establishes the content of participant elements as interchangeable without modifying the system services.
- Subset: It establishes the content of an element as a subset of the another element content.
- Exception: It specifies an abnormal situation which the system must become aware of.
- Complement: It specifies contents to be added to a described element.
- Generalization: This means some Modeling Unit is a sub-type of another one (the same as UML generalization / specialization).
- Association: This is a semantic relationship which specifies links among Modeling Units in a requirements diagram.

All these kinds of relationships between Modeling Units show a *direction* issue establishing that some elements act as a source while others act as a target. This direction characterization is shown in requirements diagrams in different ways, for example "requires", "produces", "starts", "executes", "performs", etc. Graphically, direction is represented in requirements diagrams by an arrow toward the element acting as the target.

2.5   The Domain Objectives and Model Relationships

The *Model Relationship* class describes the relationships between Domain Objectives. These relationships give us the basis for the integration of a body of domain knowledge which is related to Requirements Representation Models. Hence, these relationships allows the requirements to be sorted regardless the way they are modeled.

Model Relationship has a structural issue which determines the degree of association between two or more related diagrams. The structural issue is expressed by means of four kinds of Model Relationship:

- *Compulsory*: If the objectives are so strongly related that it is not possible to separately reuse them without resulting in a lack of sense in the domain.
- *Optional*: If the objectives are so weakly related that reusing the parent objective means non-mandatory reuse of the child objectives.
- *Alternative*: If there are choices to choose from. It means that reusing the parent objective implies choosing one and only one of the child elements.
- *Multiple*: If there are choices to choose from. It means that reusing the parent objective implies choosing at least one of the child elements.

2.6   The Unit Model Relationship

Some modeling units may have such a complexity that they need to be specified in another complete Requirements Representation Model. For example, a process in a Data Flow Diagram may be exploded in another Data Flow Diagram, or a use case may be specified as a Sequence Specification Template, as shown in Figure 3. These relationships are described as *Unit Model Relationships*.

3 Use Cases and Workflows under Metamodel

The metamodel allows several diagrams to be instantiated, thus the diagrams instances are integrated in our reuse framework. In this Section we show how Use Cases and Workflow Diagrams are exemplified as instances of Behavioural Model. For clearness of figures, while the Meta Level is expressed using UML, the Model Level has two particular characteristics:

– Each requirements diagram is represented as a "package", benefiting from its UML definition [20] as an "aggregation of classes".
– Relationships are represented as "UML associations" being labeled by the meta-class name which defines the name of the relationship.
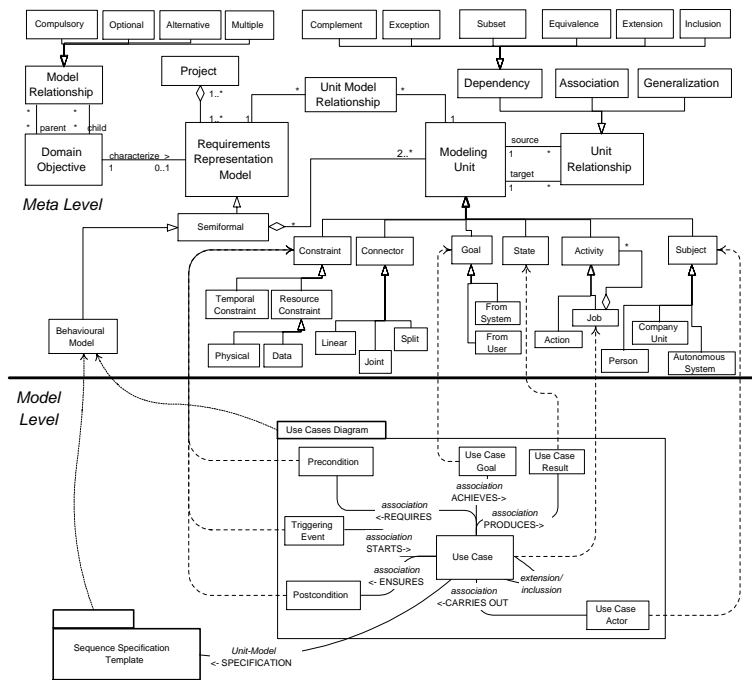

Fig. 3. Representation for Use Case Diagrams under metamodel.

3.1 The Use Cases Diagram

The Use Case Diagram is represented as an instance package of Behavioural Model, see Figure 3. The Use Case element is related to a Triggering Event, a Precondition, a

Postcondition, a Result, a Goal and an Actor which are instances of different kinds of Modeling Units. In addition, a Use Case shows Dependencies (Inclusion or Extension) with another Use Case.

According to specification of UML language [20], a Use Case describes a service that an entity provides independently from internal structural details. The service is described as a sequence being started by an actor. The Use Case has to show possible variants, for example alternative sequences and exceptional behaviour. Due to this, the use of natural language based structures to describe Use Cases, as has been proposed in [3, 6], is highly recommended. According to these proposals, we include in our metamodel the *Sequence Specification Template*. The Use Case class is related to the Sequence Specification Template through an instance of Unit Model Relationship.

3.2 The Workflow Diagram

The Workflow package is composed of Workflow Activities which are associated to Transition Information (*To* and *From* relationships). There are two kinds of these activities: Complex and Primary. The Complex Activity may be exploded in another Workflow Diagram. The Primary Activity also has Association Relationships with elements belonging to Workflow Participant, Workflow Application, and Workflow Relevant Data. Elements of the Transition Information class are joined to a Workflow Relevant Data class.

The structure of a Workflow Diagram is more complex, as regards the relationships between system activities, than the structure of a Use Case Diagram. Use Cases are composed of step sequences while the Workflow Process includes activity sequence issues. These issues in a Workflow Diagram allow the management of business resources to be represented. In contrast, the Use Case Diagram only represents possible interactions between actors and the system.

3.3 The Sequence Specification Template

Both the Use Cases and Workflows, as well as other modeling techniques, are related to the Sequence Specification Template in our requirements reuse strategy. A Use Case describes a service which is supplied for an entity. A Workflow Primary Activity also describes a service that a workflow process supplies. A service could be described as a complete sequence where actors and the system are involved. The Sequence Specification Template allows the stream of steps in Use Cases and Workflow Activities to be represented.

The Sequence Specification Template consists of Sequence Steps which are related to Sequence Indicator, as shown in Figure 5. There are two kinds of Sequence Steps: Communication Step and Operative Step. Each Step is joined to a Step Subject that may be the System itself (which corresponds to the system being modeled by Use Case Diagram or Workflow Diagram) or Unit Actor (which corresponds to one of the Actors of the Use Case or the Workflow Activity). The Step Subject is joined to the Sequence Step through three possible relationships: Carries Out, Sends or Receives.
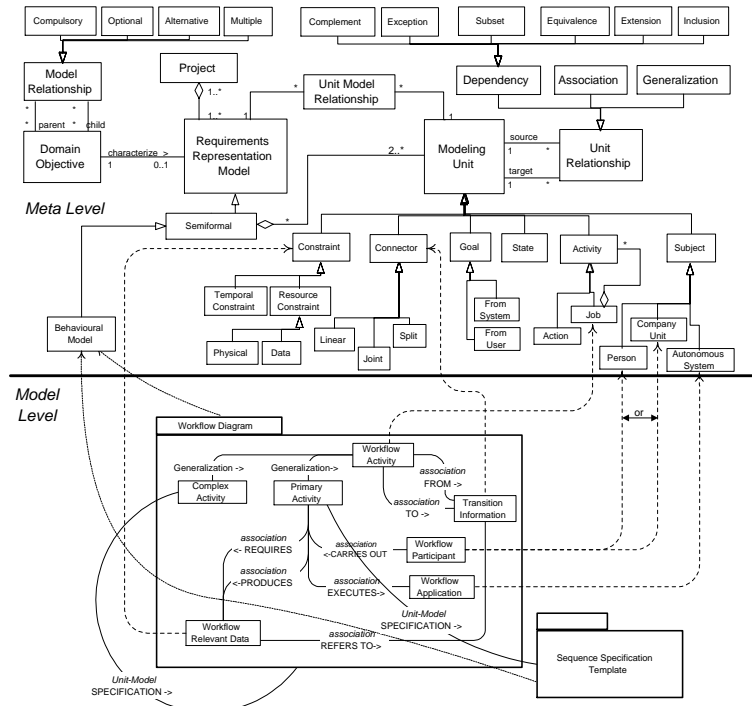
Fig. 4. Representation for Workflows under metamodel.

Each one of the Sequence Steps can be associated to an Exception. Both the Sequence Step and the Exception are associated to a Type which may be Atomic or Complex. The Atomic Type is an instance of the Action metaclass. The Complex Type is an instance of the Job metaclass. Finally, each Sequence Step could be associated to a Condition and to a Restriction, both of which are instances of the Constraint metaclass.

4   Framework for Requirements Reuse

The requirements metamodel we have presented underpins our requirements reuse framework. This framework is aimed at turning the requirements process into a reuse based approach. Different from several reuse approaches, for example FODA [11], PuLSE [2] and ODM [24], which face the reuse process by starting on the domain analysis, we propose fixing a requirements family from existing, semiformal diagrams on a domain. Sorting the existing specifications, to be reused in a product family approach, is an alternative on domains where usually there are not enough resources to afford a domain analysis from the scratch.

Figure 6 shows the three steps of our requirements reuse framework: (1) Fixing a Diagram Family Model, (2) Specifying a new application, and (3) Requirements Engi-
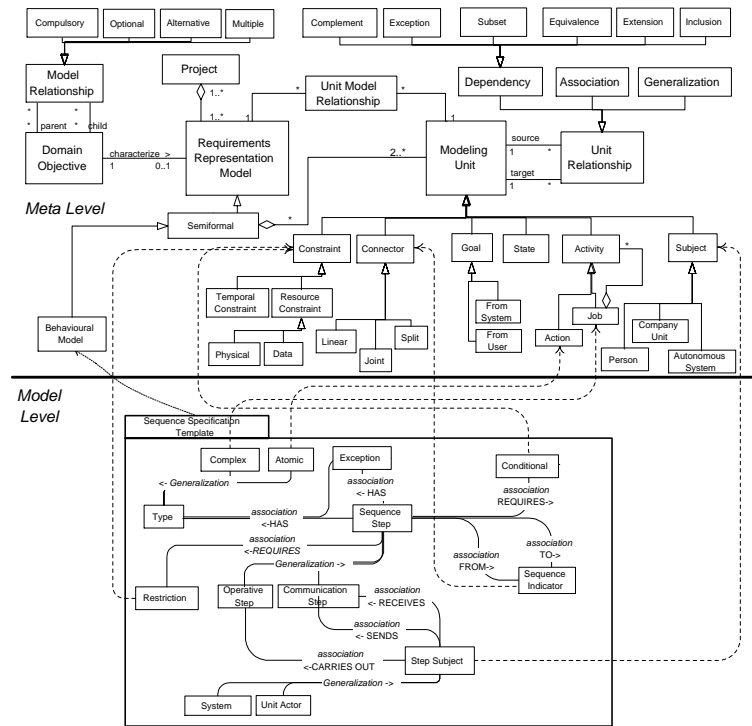
Fig. 5. Representation for Sequence Specification Template under metamodel.

neering. The output from the first step is the Diagram Family Model, which is used in second step to obtain the specification for a new requirements project. The requirements family could be expanded by taking new diagrams from the third step. The Figure also shows the second step feedbacks the Diagram Family Model, hence new projects are included into diagram family to make sure it contains all requirements for all member products.

4.1 Fixing a Diagram Family Model

The main step of our requirements reuse framework is sorting the diagrams as a family. Inputs for this step are previous documents of requirements specifications, as well as domain knowledge. The purpose in this step is obtaining groups of requirements by *identifying, describing, and storing* the diagrams as a family. Sorting the requirements is based on the stakeholders' points of view, which act as a guidance to obtain the diagram family, as has been used in [13]. Obtaining the Diagram Family Model is achieved as follows:

– *Identifying* the domain stakeholders and establishing a domain dictionary. Identifying the domain stakeholders allows the goals of each stakeholder to be elicited
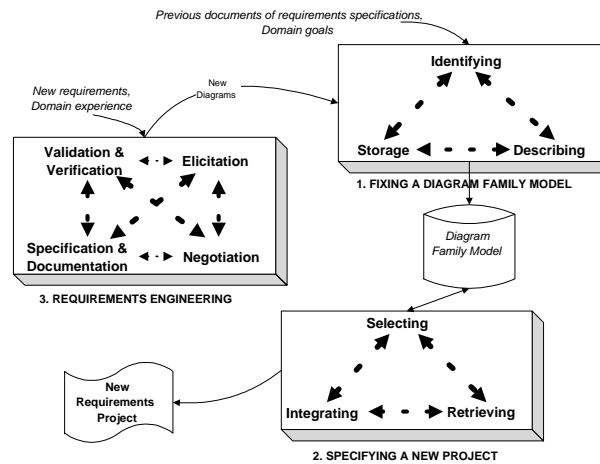
Fig. 6. Requirements reuse framework.

and arranged in a lattice structure by linking them with Model Relationships. This stage makes demands on the experience and domain knowledge to guide the decision making process to establish the lattice. The decisions include dealing with overlapping and conflicting goals as well as dependencies among goals.

– *Describing* the Domain Objectives by linking them to existing requirements diagrams. This stage is aimed at taking benefit from the existing requirements diagrams in domain applications, thus it could be a need for rewriting some diagrams through an iterative process which is supported by the tool we present in Section 4.4.

– *Storage* of Diagram Family Model in a repository which supplies the operative support to *create, manage and reuse* the diagrams. Foundation for this stage is the McClure's proposal [18] establishing the repository is *a tool for definition, storage, access and management of the information that describes together an enterprise and its software systems, during each one of the stages of software life cycle*. The Diagram Family Model contains data from the initial stages of software process.

The Diagram Family Model allows the commonalities and variabilities among requirements to be viewed [14, 7]. The *Optional*, *Alternative* and *Multiple* kinds of Model Relationship represents the variabilities. The *Compulsory* kind of Model Relationship represents the commonalities.

4.2   Specifying a New Project

Specifying a new project from the Diagram Family Model consists of making decisions about commonalities and variabilities in requirements. Commonalities and variabilities are inside the Diagram Family Model. Each Project belonging to the domain has a set of requirements. These requirements are chosen from the Diagram Family Model by

selecting from commonalities and variabilities. Hence, while specifying a new product there is a need for going over the objectives lattice in such a way that the adequate reusable elements are selected. This stage consists of three actions:

– *Selecting* the Domain Objectives that applies for the new project. Going over the objectives lattice, the user makes a decision about including the current objective in the new project. Selected objectives are added to new project and they are included in the document of requirements specification for the New Requirements Project.
– *Retrieving* the diagrams which are linked to the selected Domain Objectives. This stage implies the need for iterate with the Data Manager which access data from repository.
– *Integrating* the new requirements Project into Diagram Family Model. There is a need for feedback the repository so that the new Project is included. At this stage it is important to take into account the way to make sure about the logical consistency of the new Project.

4.3   Requirements Engineering

Specifying new projects can lead to including new requirements in Diagram Family Model. In such a case, new requirements are specified by requirements engineering process being composed of *elicitation, negotiation, specification & documentation, and verification & validation* activities [21]. The output of this process is a set of *New Diagrams* which are integrate into family. Because of these diagrams are specified when the Diagram Family Model has been built, their specification is addressed with a reuse compromise and being supported by our R^2 environment, which is presented in Section 4.4.

4.4   A Tool for Requirements Reuse (R^2)

The requirements reuse framework is supported by R^2, a prototype system which is implemented using relational tables (ORACLE) and Java language. This tool is composed by eight main elements with corresponding sub-elements: (1) *User Interface* including menu options and necessary views to support the interaction between reusers and the environment, (2) *Requirements Editor* which supplies needed functionality for creation and modification of requirements diagrams, (3) *Data Manager*, it allows the requirements information to be stored, classified, retrieved and updated, (4) *Repository*, which physically contains the information related to requirements diagrams regarding the conceptual scheme of our requirements metamodel, (5) *Consistency Analyzer*, it is a module which allows the information to be directed to and directed from external Petri net applications to address the consistency analysis of diagrams and projects, (6) *Lexicon Manager*, it allows the domain lexicon to be managed according the principles of circularity and minimal vocabulary that proposes Leite *et alt*. [15], (7) *Diagrams Translator*, it provides services to rewrite diagrams in such a way that they become expressed in other supported notation, and (8) *Repository Manager*, it is a module which allows the quality of requirements diagram which are stored in repository to be estimated.

Figure 7 shows the interface of the Requirements Editor in R^2. The identifier of project and their elements are on the left side of the window. Graphical notation corresponding to current element appears on the right. The Figure shows the selected diagram inside the so called project "Communicator". This editor carries out two essential functions in our requirements reuse framework. First, it allows the existing requirements diagrams to be restructured, if needed, before they become a part of Diagram Family Model. Second, the editor supplies functionality for diagrams maintenance, thus supporting the evolution of Diagram Family Model. Supporting diverse modeling techniques, as they are instantiated under metamodel, the Requirements Editor makes communication among stakeholders easier.
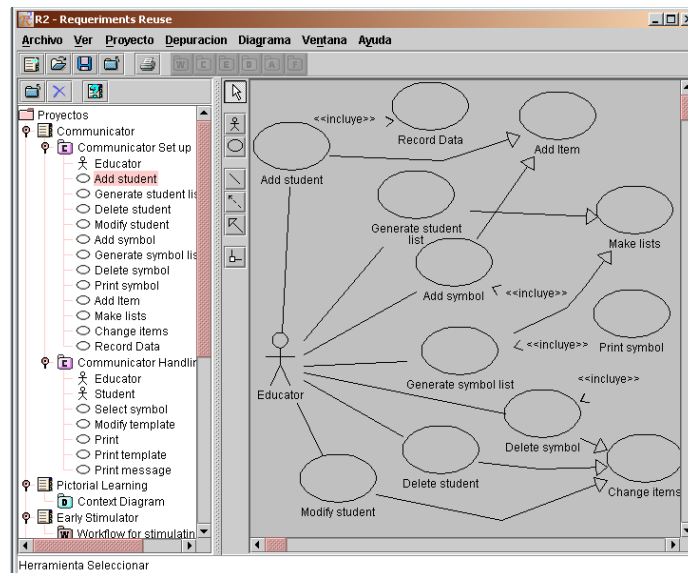

Fig. 7. Requirements Editor interface of R^2 environment.

5   Conclusions and Future Work

In the development of complex software systems the modeling techniques, showing stakeholder's viewpoints, are required [10]. Due to this, we have investigated the management of the richness of units and relationships, which are represented in diverse modeling techniques, to establish a requirements reuse framework. In this paper we propose a requirements metamodel that is based on the same language as UML. This metamodel describes useful modeling information on the requirements reuse context.

The contribution of this paper is a metamodel as theoretical basis to approach the requirements reuse in domains where the requirements information has been represented

in diverse semiformal models. For this reason, we have not addressed formal and non-formal models in this paper. Although informal models are widely used, their lack of formality and their complexity make reuse difficult. There are very interesting proposals for reusing non-formal representations [5] based on reusing requirements patterns. On the other hand, reusing formal representations has been successfully addressed [7], however formal based reuse requires having formal requirements specifications.

The metamodel allows every requirements element from the modeling techniques we have investigated to be identified as an object instance of some requirements meta-class. In this way, the metamodel supports for the integration of different Requirements Representations. Nevertheless, this integrating perspective of requirements diagrams is one building block in the way of turning the requirements process into a reuse based approach. Furthermore, as we claim in [16], empirical research shall be conducted to evaluate the results of applying requirements reuse in domains. Thus, our immediate work should apply our Diagram Family Model and our Requirements Reuse prototype system in generating new requirements specifications in a specific domain. We expect that reusing requirements on our reuse framework can make better requirements specification because of requirements engineers might focus on decisions about taken a set of requirements regardless the details of definition and documentation.

Acknowledgments

References

1. Raquel Anaya. *Desarrollo y Gestión de Componentes en el Marco de OASIS*. PhD thesis, Universidad Politécnica de Valencia, Spain, 1999.

2. J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. De-Baud. PuLSE: A methodology to develop software product lines. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99)*, pages 122–131, Los Angeles, CA, USA, May 1999. ACM.

3. A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, Boston, 2000.

4. Richard E. Creps, Mark A. Simos, and Rubén Prieto-Díaz. The STARS conceptual framework for reuse processes. In *Proceedings of STARS'92*, November 1992.

5. Jacob L. Cybulski. Patterns in software requirements reuse. Technical report, Department of Information Systems. University of Melbourne, July 1998.

6. Amador Durán. *Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información*. PhD thesis, Universidad de Sevilla, Spain, 2000.

7. Stuart R. Faulk. Product-line requirements specifications (PRS): an approach and case study. In *Proceedings of 5th IEEE International Symposium on Requirements Engineering*, pages 48–55, Toronto, Canada, August 2001. IEEE Computer Society.

8. F.J. García, J.M. Corchado, and M.A. Laguna. CBR applied to development with reuse based on mecanos. In *Proceedings of 13th International Conference on Software Engineering and Knowledge Engineering - SEKE'01*, pages 307 – 311, Buenos Aires, Argentina, June 13-15 2001. Knowledge Systems Institute.

9. R. Geisler, M. Klar, and C. Pons. Dimensions and dichotomy in metamodeling. In *Proceedings of 3th BCS-FACS Northern Formal Methods Workshop*. Springer-Verlag, Sep. 1998.

10. M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, and J. Vassiliou. Theory underlying requirement engineering: An overview of NATURE at genesis. *Proceedings of the IEEE International Symposium on Requirements Engineering*, 1993.

11. 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.

12. E. Karlsson, editor. *Software Reuse. A Holistic Approach*. Wiley Series in Software Based Systems. John Wiley and Sons Ltd, 1995.

13. Barry Keepence, Mike Mannion, Hermann Kaindl, and Joe Wheadon. Reusing single system requirements from application family requirements. In *Proceedings of the 21st International Conference on Software Engineering*, pages 453–462. ACM Press, May 1999.

14. J. Kuusela and J. Savolainen. Requirements engineering for product lines. In *Proceedings of ICSE 2000, IEEE*. IEEE Computer Society, 2000.

15. J. Leite, G. Hadad, J. Doorn, and G. Kaplan. A scenario construction process. *Requirements Engineering. Springer-Verlag London Limited*, 5(2000):38–61, 2000.

16. O. López, M.A. Laguna, and F.J. García. Reuse based analysis and clustering of requirements diagrams. In *Pre-Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02)*, pages 71–82, Essen, Germany, September 2002.

17. S. Mann and M. Klar. A metamodel for object-oriented statecharts. In *Proceedings of The Second Workshop on Rigorous Object-Oriented Methods*, May 1998.

18. Carma McClure. *Software Reuse Techniques: Adding Reuse to the System Development Process*. Prentice-Hall, 1997.

19. OMG. Meta object facility (MOF) specification. version 1.3.1. document OMG-MOF V1.3, November 2001. http://www.omg.org/.

20. OMG. Unified modeling language specification. Version 1.4. Technical report, Object Management Group Inc., September 2001. http://cgi.omg.org/docs/formal/01-09-67.pdf.

21. K. Pohl. Requirements engineering, an overview. Encyclopedia of Computer Science and Technology, Vol. 36, Marcel Deccer Inc., 1996.

22. Colette Rolland and Naveen Prakash. From conceptual modelling to requirements engineering. Technical Report Series 99-11, CREWS, 1999.

23. H. Dieter Rombach. Sofware specifications: A framework. SEI Curriculum Module. Technical Report SEI-CM-11-2.1, Software Engineering Institute, Carnegie Mellon University, January 1990.

24. 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.

25. A. Sutcliffe and N. Maiden. The domain theory for requirements engineering. *IEEE Transactions on Software Engineering*, 24(3):174–196, March 1998.