Electronic Communications of the EASST
Volume 3 (2006)


# Proceedings of the Third Workshop on Software Evolution through Transformations: Embracing the Change (SeTra 2006)

## Generating Requirements Views: A Transformation-Driven Approach

Lyrene Fernandes da Silva, Julio Cesar Sampaio do Prado Leite

14 Pages

Guest Editors: Jean-Marie Favre, Reiko Heckel, Tom Mens
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# Generating Requirements Views: A Transformation-Driven Approach

**Lyrene Fernandes da Silva[1], Julio Cesar Sampaio do Prado Leite[2]**
[1]Federal University of Rio Grande do Norte (UFRN) - Brazil
[2]Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) - Brazil

**Abstract:** This paper reports the use of transformations based on XML to generate requirements views. A strategy to generate views is defined and scenarios and class diagrams are automatically created from a goal oriented model; the V-graph.

**Keywords:** Requirements models, views, traceability, transformations, XML.

## 1 Introduction

During software construction, we may use different types of models and languages such as: scenarios, requirements sentences, lexicons, component models, class diagrams, entity-relationship models, and activity diagrams. These different software representations are necessary because each one of them portrays a different and limited set of characteristics. This is done in order to decrease software complexity and help the engineer to focus on scoped problems.

However, because of the volatility of the requirements, it is necessary to be ready for changes, or evolution. Requirements evolution happens in two ways: during software development, changing from the high abstraction level to the implementation level, i.e., from the requirements to the code; and in order to make the model ready to attend new requirements or fix errors and omissions [19][23]. In both cases, knowing and managing the interactions between requirements (the traceability) is of fundamental importance. As a consequence it is important to decompose and modularize the concerns of the system for two main reasons: (a) the concerns indicate the coupling and cohesion among their components, and (b) it is important in order to analyze the impact of changes between requirements at the same abstraction level and requirements and software artifacts on different abstraction levels [28].

As defined in [33], traceability means the ability to find related requirements in a requirements specification, discovering: the source of the requirements (pre-traceability); the components that implement them (post-traceability); or requirements that affect each other [28]. Traceability is important to manage and to propagate changes in requirements, thus supporting software construction [15]. However, if many different models are used during the construction process, it is necessary to map the information from one kind of model to others, propagating changes, verifying correctness and conflicts among them.

In order to address this problem, in this paper, we present an approach based on transformations to create different views of requirements models. The transformations are defined by using rules and have been implemented in the context of XML (eXtensible Markup Language) technology, by means of XSLT transformations. We exemplify this approach by the creation of rules to transform a goal model, called V-graph, into models: scenarios [23] and class diagrams [4].

This approach was defined in the context of an aspect-oriented requirement modeling strategy [31][32]. In this context, we have considered that using views is extremely important because they can be an alternative way to separate concerns and decrease tangling and scattering problems that occur due to the tyranny of the dominant decomposition [34]. The V-graph model was used because it can represent both non-functional and functional requirements. Furthermore, it explicitly represents the positive and negative interactions between requirements in opposition to use cases, scenarios and requirements sentences which do not tackle the issue of requirements interference. Therefore, we can identify crosscutting concerns by analyzing the interactions among them. In this paper, we present our approach to generate views from V-graph, but we omit the details about "aspects" defined in [31][32] because of the limited space available.

The remainder of this paper is organized as follows. In Section 2, we present the context of this work, the concept of views and the V-graph. In Section 3, we present how we can generate different views by using transformations, the benefits of this strategy and the defined transformation rules to generate scenarios and class diagrams from V-graph models. In Section 4, we present a case study to illustrate the visualization mechanism. In Section 5, we cite some related work. Finally, in Section 6, we present the concluding remarks and guidelines for future works.

## 2 State of the Art

Considering software development as an evolutionary process, in which the design and the programming are based on the requirements definition and the requirements are continually changing, software evolution happens: during the development process by refining the models created during the requirements definition process into architecture models and code; and during each activity, by making a set of models better, generating different versions of the same model.

In both cases, it is necessary to be able to identify where changes impact, modify and propagate these changes for all the used models. The traceability (or mapping) between models can be supported by a transformation-driven approach [3]. Transformations are interesting in this case because they can be used to:

- Generate views – different total or partial models can be generated from a base model. Partial models can represent the system focusing on different concerns or different viewpoints. Total models can represent the information of a base model using another notation. In both cases, generating views helps software evolution.
- Facilitate analysis – by using different views, the engineer can focus on scoped problems, analyze and modify the modeling in a more effective way. Consistency and completeness checking are facilitated because transformations can generate models with the same information but changing the perspective to analyze it. A proper inspection mechanism, when in place, can point out errors and omissions between views.
- Propagate changes – changes made in a model can be automatically propagated to other models. This decreases the rework, increases the engineer's productivity and guarantees that all models are updated;

On the other hand, it is difficult to use a transformation-driven approach, because it is hard to map one model to others. This is not always possible, usually the concerns represented in a model are not represented in another model, some information can be omitted, and thus it is difficult to guarantee the consistency and completeness of these models.

In the requirements definition process, non-completeness and inconsistencies are more tolerable than in other activities because models created in this stage will be refined by future feedback. These models cannot have all of the information about the solution to the problem and they have to accommodate some conflicts and ambiguities from the domain because these conflicts have to be analyzed and resolved.

For us, the creation of views provides different perspectives of the same model, separating the crosscutting concerns in different ways and offering the requirements engineer different ways to analyze these concerns. In Section 2.1 we present the concept of views used in this work and in Section 2.2 we present the syntax and semantic of the V-graph.

## 2.1 Views

Views are representations of the overall architecture that are meaningful to one or more stakeholders in the system (IEEE St.1471). Using views is a way of separating different concerns in order to focus on one at a time. Views help understanding and elaborating solutions [33], therefore, they are necessary during all the development process.

In the requirements engineering area, the words viewpoints, views and point-of-view are sometimes used with similar or different meanings [22]. In order to make the view concept clear, in [16], three categories of views are presented:

- Views as opinions (viewpoints) – in the social context, each stakeholder has his premises, priorities and experiences, and they use different ways to deal with the problems. Therefore, it is necessary to know how to compare and negotiate the different opinions or different ways of how to look at the things, for example, what is the opinion of the manager, of the seller and of the buyer about an e-commerce site?
- Views as services (concerns) – the idea of partitioning the system into a set of services that can be connected in different ways provides component-based development. For example, a component for payment of bills, another for security, among others;
- Views as models (perspectives) – in the context of software engineering many techniques based on languages have been proposed in order to partially portray a system, such as entity-relationship models, use cases diagrams and sequence diagrams. Therefore, it is important to detect consistency and completeness problems among these models.

These categories of views are not disjointed categories. Usually, we use models (perspectives) to represent services (concerns) from the point of view of one or more stakeholders (viewpoint). Furthermore, models, by definition, make some types of information explicit and hide others, so we can have models focusing on functions, data, sequence of activities and so on.

## 2.2 V-Graph

V-graph is a type of goal model [36]. Goal models represent the functional and non-functional requirements through decomposition trees [25]. V-graph, see Figure 1(a), is defined by goals, softgoals, tasks and the following decomposition relationships – contribution links (and, or, make, help, unknown, hurt, break) and correlation links (make, help, unknown, hurt, break). Each element has a type and topics. The type defines a generic functional or non-functional requirement, for example, Security and Management. The topic defines the context of that element, for example, data and communication.

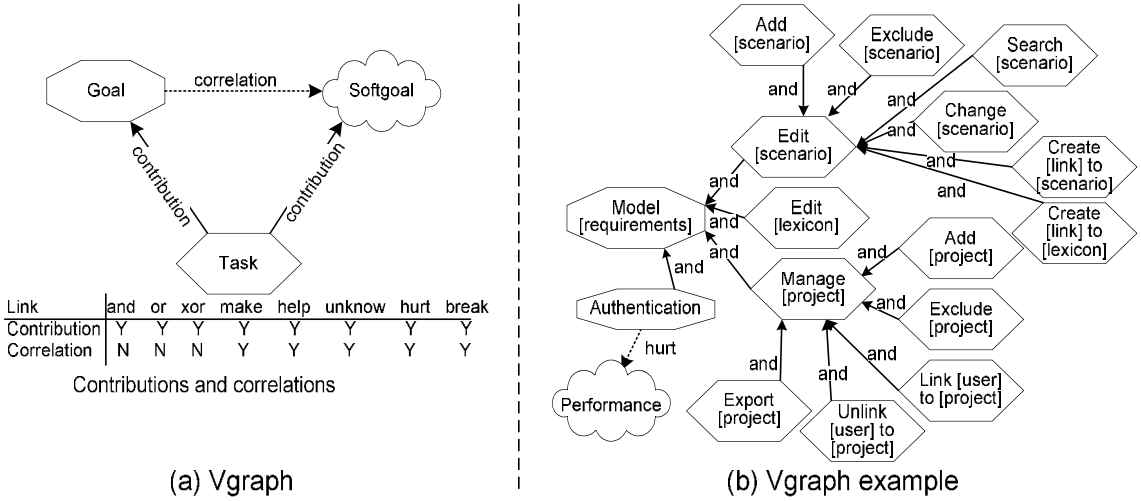



Figure 1. (a) V-graph and (b) a V-graph example

Figure 1(b) portrays a V-graph example. In such figure, we can observe that in order to achieve the goal “Model [requirements]” (“Model” is the type and “requirements” is the topic) the goals “Edit [scenarios]”, “Edit [lexicon]” and “Manage [project]” have to be achieved. The relationships among these four goals are contribution links. The contribution links represent a hierarchy between root (a more abstract element that is father of subelements) and children (an operationalization that can be a leaf or another root of the tree).

V-graph is an interesting model to represent requirements because with it we can consider requirements at three abstraction levels (softgoals, goals and tasks). This is important because in the same model we can represent reasons and operations, the context and how each element contributes to achieving the system goals. Furthermore, there are important results in goal modeling, concerning: how to analyze obstacles to the satisfaction of a goal [18]; how to qualitatively analyze the relationships in goal models; how to analyze variability [14]; how to analyze conflicts among goals through a propagation mechanism of labels [13]; how to identify aspects in goal models [36]; how to derive a feature, state and component model from goal models [30]; and how to provide goal reuse [24] – this last work mentions a composition mechanism used to integrate a goal model and a reusable goal model from a library.

## 3 Using Transformations to Generate Requirements Views

Views have been used in different activities of software construction because they help the developer to delimit the scope of a problem and thus, its complexity. Therefore, the developer can analyze the correctness and completeness of one concern or a set of concerns at a time. During the requirements definition process, as well as during the design process, i.e., during the elaboration of solutions, it is important that developers be able to obtain different views from a base model in order to facilitate the analyses of the solutions created from different viewpoints and perspectives.

As we defined in Section 2, a view is a representation of the software architecture or of one part of the system's architecture, focusing on one or more concerns, by one or more stakeholders. Figure 2(a) presents, through a features model [8], the variability, considering views as models and as services. This feature model shows that a view represents one or more

services using one or more types of models (notation). Therefore, we can create views to the requirements focusing each concern separately (partial views) or in conjunction (total views), using different types of models.

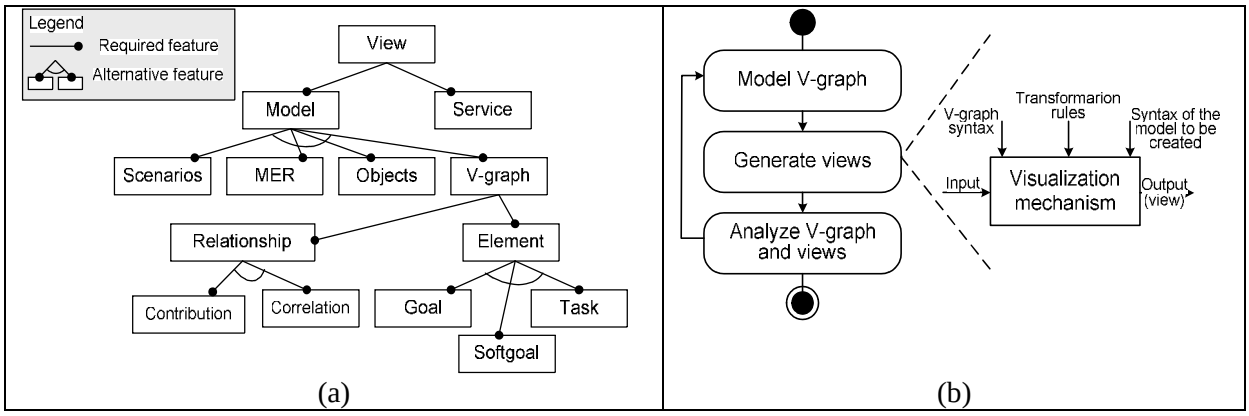



Figure 2. (a) Feature model representing the V-graph views and (b) Visualization mechanism

Therefore, we consider fundamental to have a visualization mechanism in order to facilitate the requirements modeling. An automatic mechanism to generate views can accelerate the modeling process because it decreases reworking and inconsistencies among different requirements models. Figure 2(b) summarizes a modeling process using a visualization mechanism to automatically generate views from V-graph. Such visualization mechanism consists on a transformation component that needs the following sets of information: the syntax and semantic of the source language (in our case, V-graph) and the target language (representation to be generated), and transformation rules. Next subsection deals with how to transform V-graphs into scenarios and class diagrams.

## 3.1 Transformation Rules

The V-graph is a representation where we can explicitly model functional and non-functional requirements using softgoals, goals and tasks. This is its dominant decomposition manner, an intentional-oriented decomposition. However, the hierarchy and the topics of the V-graph provide new perspectives of the system, based on, for example, situations and data. Furthermore, the relationships between goals, softgoals and tasks provide a perspective of interaction, or traceability. Using this knowledge, we define rules to transform the information from a V-graph into two different models: scenarios [23] and class diagram [4]. Therefore, we provide requirements views that help “combating” V-graph’s dominant decomposition.

**Transforming V-graph into Scenarios**

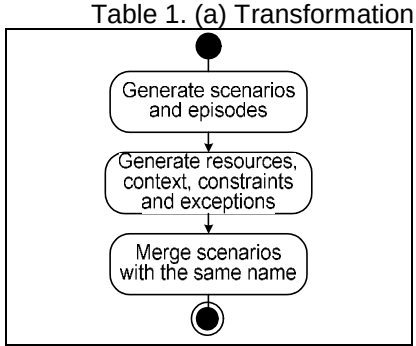
Scenarios are an interesting topic to the software engineering community [35][29]. The scenario-driven software development is based on the concept that using the problem's language (user’s domain) is really beneficial for the interaction and communication between users and developers. Scenarios are common situations to the users [31]. They have to take into account the usability, enable the comprehension of the domain and problem and help unifying criteria, the sorting of details and user training [7].

Each scenario describes, through semi-structured natural language, a specific situation of the application or of the domain, focusing on behavior. Scenarios can be detailed and used as design, in order to help programming. There are many representation proposals for scenarios, informal representations in free text [7] or formal representations [17]. We opted for an intermediate representation: it facilitates the comprehension using natural language and forces sorting of the information by using a well-defined structure, proposed in [23]. In order to transform the information in V-graph into a scenarios model some steps are followed:

1. First, we generate scenarios and episodes by using the goal tree hierarchy: each node (goals and tasks) of the tree that is not a leaf generates a scenario; each subgoal and subtask that is a leaf generates an episode of that scenario; and each subgoal and subtask that is not a leaf generates an episode with a reference (link) to a scenario; contributions "or" generate optional episodes.
2. Second, we generate resources, constraints and context. Each topic of goal and task that generated episodes without references to any scenario is a resource of that scenario; each softgoal negatively related (correlation or contribution) to goal/task that generated a scenario generates an exception into that scenario; each softgoal negatively related (correlation or contribution) to goal/task that generated a episode (without reference to any scenario) generates a constraint into that episode; each goal related (contribution) to task that generated a scenario generates the context into that scenario.
3. Next, we merge scenarios with the same title, because it is possible there be more than one goal or task with the same name in V-graph in order to facilitate the visualization of the tree of goals.

Table 1 summarizes the transformation process from V-graph to Scenarios and Section 4 presents an example of scenarios model generated by using this process.

Table 1. (a) Transformation process and (b) transformations: V-graph → Scenarios




**(a)**

| V-graph | Scenarios |
|---|---|
| Goals, tasks | Scenarios and episodes in accordance of the hierarchy of goals and tasks |
| Softgoals negatively correlated | Exceptions or constrainsts |
| Goals of tasks that generated scenarios | Context |
| Topics of goals and tasks that generated episodes without references to scenarios | Resources |

**(b)**

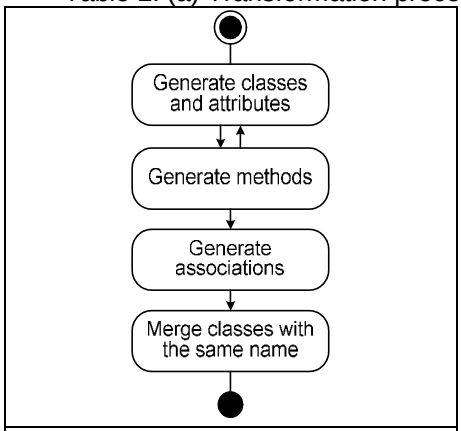
### Transforming a V-graph into a Class Diagram

Objects are executable entities, instances of a class that defines its attributes and services. Object or class models, usually, are used when we want to adopt the object-oriented paradigm in the design and programming activities. However, these models can be used during the requirements definition process to represent data and functionalities [33]. In this sense, we derived class diagrams from V-graph models. In order to transform the information in V-graph into a class diagram, we followed the general pattern described below.

1. First, we use the hierarchy of goals and tasks to define classes and attributes: each topic of goal/task that is not a leaf generates a class whose name is the topic name; and each topic of goal/task that is a leaf generates an attribute in class generated by its parent node.
2. Second, we define the methods of the classes: every goal/task is method of classes whose name is its topic; each goal/task that is a leaf generates a method into class generated by its parent node; if one of the methods refers any topic different of class name then this topic is an attribute of that class.
3. Next, we generate the relationships between classes: classes refer at least one similar method are associated; each correlation or contribution between elements that generate classes generates an association into the class diagram;
4. Finally, classes with same name are merged.

Table 2 summarizes the process to transform V-graph into Class Diagram and Section 4 presents an example of class diagram generated by using this process.

Table 2. (a) Transformation process and (b) transformations: V-graph → Class diagram




**(a)**

| V-graph | Class diagram |
|---|---|
| Topic | Classes and attributes in accordance of the hierarchy of goals and tasks |
| Goals, tasks | Methods |
| Contributions, correlations, and goals/tasks that refer similar topics | Associations |

**(b)**

## 3.2 Implementation

We implemented this strategy using XML (*eXtensible Markup Language*) and XSLT (*eXtensible Stylesheet Language Transformation*). V-graph syntax was defined using a DTD (*Document Type Definition*). The visualization mechanism, i.e., the transformations, was programmed in XSLT. Therefore, a V-graph model (in XML) is the input to the visualization component and the outputs are scenarios (in HTML) and class diagrams (in Dot). Figure 3 shows how we used XML and XSLT to implement our strategy. The Dot format and the GraphViz application [16] are used to create graphic representations to V-graph and class diagrams.

The choice for XML based transformations was due to the characteristics of our proposal, models written in XML, and due to the characteristic of our transformation rules. In our case, the transformations were localized and direct, and as such the XSLT mechanism was sufficient. Of course if more complex rules were necessary, we would have to use a more robust transformation platform. However it is important to stress that despite its simplicity, we

can easily implement different representations and layouts for the information described in XML with its structure defined in a DTD. Furthermore, XML can easily be read and changed for different applications.

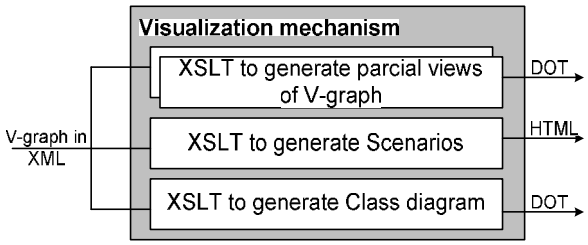



Figure 3. XML and XSLT used to transform requirements views

## 4 Case Study

This section presents part of our case study. The complete case study has four goal models: a goal model for an information system that helps writing scenarios and lexicon [32]; a goal model for Security; a goal model for Reliability; and a goal model for Persistence. The V-graph illustrated in Figure 1(b) is the part of this case study we have used in this paper to demonstrate our approach. On the right side of Figures 4 e 5 we show this same V-graph. The octagons are goals and the hexagons are tasks. Pointed links are correlations and the other links are contributions. Each goal and task has at least one Type and zero or more Topics (bracketed text). On the left side of Figures 4 e 5, we portray the created views: scenarios and class diagram.

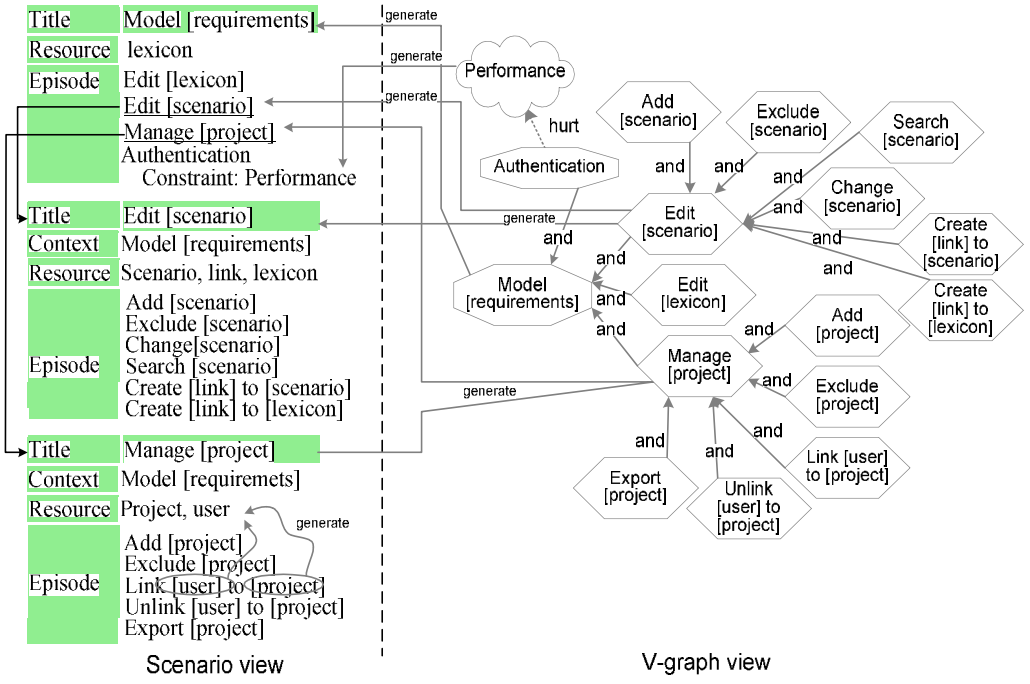



Figure 4. Example of the scenario view

Figure 4 portrays an example of scenarios generated from the V-graph. In this example, there are three scenarios with the titles “Model [requirements]”, “Edit [scenario]” and “Manage [project]”; they were derived from goals with similar names. In the scenario “Manage [project]”, there are two resources (project and user) generated from topics of their episodes; the context attribute is generated from the goal “Model [requirements]”; the attribute episode is generated based on the goals/tasks that decompose the goals “Model [requirements]”, “Edit [scenario]” and “Manage [project]”; and each underlined episode indicates a relationship with another scenario.

Figure 5 portrays an example of class diagram to the V-graph (in Figure 1b). Each goal/task that is not a leaf (and has topics) generates a class into the class diagram. Children of goal/task that do not generate classes (and have topics) generate attributes related to the class generated by their goals/tasks parent. Relationships between classes are generated from the contribution links between goals/tasks that generate those classes. In Figure 5 we can observe the classes “requirement”, “project” and “scenario”, their attributes and operations; the relationships between them are generated based on the relationships between tasks/goals whose topics are source to these classes.

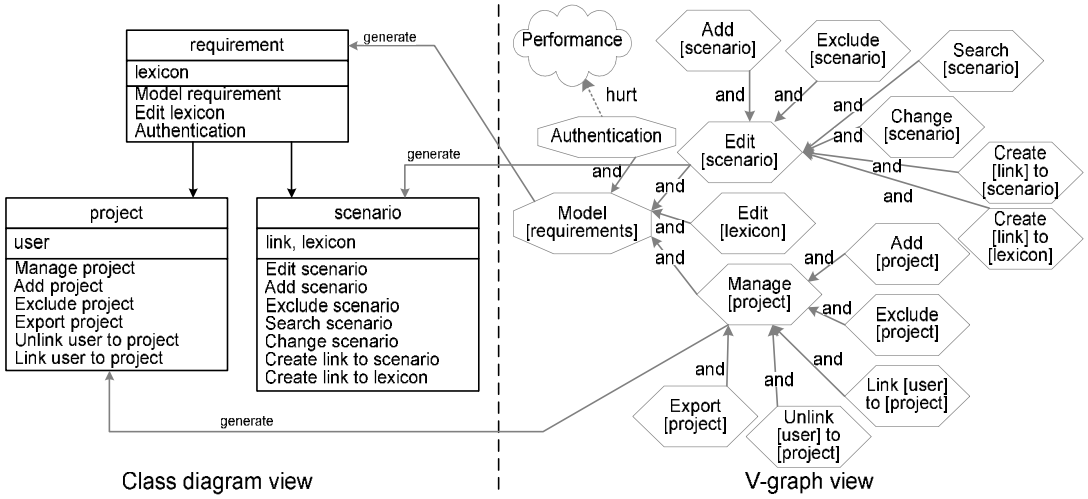



Figure 5. Example of class diagram

This case study is available in [32]. The complete modeling has 9 goals, 105 tasks, 12 softgoals, 7 correlations and 173 contributions. After the transformation process applied to this case study, 40 scenarios and 14 classes were generated into the scenarios view and class diagram view, respectively. Although the class diagram does not have the types of relationships and cardinality, the models created help the engineer to analyze the domain from the date perspective in opposition to the intentional perspective of the V-graph. Scenarios and class diagrams represent two other dominant ways to separate concerns, therefore by using our approach the engineers have these views without having to create them manually. Furthermore, any change made in the V-graph can be automatically propagated to the other models.

# 5 Related Work

Although using different types of models helps managing the complexity during software modeling, it causes tangling and scattering of concerns, making it difficult to maintain every model consistent and updated. Therefore, integration mechanisms are necessary in order to integrate services and models as well as to integrate opinions (conflicts resolution). This paper focused on the integration of models using a transformation-driven approach.

Software transformation has been a central topic in different software related areas. In the early seventies/eighties several researchers believed it to be central to the idea of automatic programming and several program transformation initiatives were initiated, notably the Irvine Transformation Catalogue [**11**]. Also in the area of software reuse, the idea of software transformation was particularly successful, for instance [**1**] and the approaches on product-line [**2**] and the Draco approach to software construction [**21**][**26**][**30**]. The use of transformations, in this type of context, requires a more powerful mechanism, since the control structure is not straightforward and a strict discipline to help the validation of the complex rewrite rules is necessary.

There are also many approaches less complex which have been used to transform requirements models into other requirements models or design models. Many of them are based on natural language that process or consider the structure of the source language to identify the constructs of the target language, such as:

In [6], a process to generate ontology from LEL is defined. This approach is based on transformations but it is only semi-automatic. In [5], an approach has been defined to generate activity diagrams from use cases and after that to transform these diagrams into Pres, a formal notation that permits verification. Therefore, this approach enables the enrichment of the use case model and the production of more precise and complete requirements. In [12], an automated approach to transform feature models into the class diagrams is defined.

In [9], a process to integrate RNFs and RFs is defined. This approach uses the constructs of MER, of class diagram and of lexicon extended language (LEL) [20], in order to make this integration. Such integration process is based on mapping the RNFs specified in LEL into the MER and into the class diagram. However, this mapping is not based on transformations, it is based on the analysis of the information in the LEL, MER and in the class diagram.

In [27], an integration framework of models (modeling methods) is defined. This framework determines the specification of: (1) style – defines the notation; (2) work plan – specifies the activities, strategies and processes to define a view; (3) domain – indicates the domain area; (4) specification – the development method; and (5) work report – indicates the state and history of the modeling. The information described in (1) e (2) is abstract information; they can be applied to every instance of the same type of model whereas information in (3), (4) and (5) of this framework is specific to each instance of the model. The main goal of this work is to give support for consistency checking among different models and managing inconsistencies, facilitating the reuse of information on how to map one representation into others. This framework inspired us to define informally the information described in (1) and (2) in order to specify the transformations from V-graph into scenarios and class diagrams, as we have shown in Section 3.2.

## 6 Final Remarks

In this paper, we present a visualization mechanism used to generate requirements models. This mechanism is transformation-driven. We created some transformation rules in order to automatically generate scenarios and class diagrams from the V-graph model. Using transformations during the requirements definition helps us make the trace among the different models used. It facilitates modeling because consistent models are generated and changes are automatically propagated. Consequently, these transformations help software evolution.

The results that we have had using this approach have been satisfactory because we consider that generated views help the engineer analyze the system. However, such views cannot be considered complete models, but initial models that help the engineers because they do not have to begin the modeling from scratch. Future work involves: making better transformation rules in order to obtain more detailed models; defining transformation rules to both directions, V-graph → (scenarios and class diagrams) and (scenarios and class diagrams) → V-graph; creating a verification mechanism to report inconsistencies and omissions into each type of view; and also tools are extremely necessary to support the edition of any of these models. Furthermore, it is necessary to plan experiments in order to validate our approach at the requirements definition stage and to evaluate what is the impact of using it during the entire software development process.

Currently, we are working on the definition of the transformation rules to generate the system architecture from the requirements definition. This work is part of our aspect-oriented approach to model requirements [31][32]. When taking crosscutting concerns into account, the visualization approach presented in this paper is equally important because using views is an alternative way to separate crosscutting concerns, facilitating the tasks of modeling, analysis, traceability and software evolution.

## 7 References


1. D. Batory, S. Dasari, B. Geraci, V. Singhal, M. Sirkin, J. Thomas. Achieving reuse with software system generators. In: IEEE Software, September-1995, pp. 89-94.
2. D. Batory, R. Lopez-Herrejon and P. Martin. Generating Product-Lines of Product-Families. In: Automated Software Engineering Conference, 2002.
3. I. Baxter. **Transformational Maintenance by Reuse of Design Histories**, Ph.D. Thesis, Information and Computer Science Department, University of California at Irvine, Nov. 1990, TR 90-36.
4. G. Booch, J. Rumbaugh and I. Jacobson. **The Unified Modeling Language User Guide**. Addison-Wesley, 1999.
5. R. Boudour and M. Kimour. Model Transformation for Requirements Verification in Embedded Systems, In: **Asian Journal Informational Technology**, 4 (11): 1012-1019, 2005.
6. K. Breitman1, J. Leite. Lexicon Based Ontology Construction. In: Lecture Notes in Computer Science 2940- Editors: C. Lucena, A. Garcia, A. Romanovsky, et al., ISBN: 3-540-21182-9, Springer-Verlag Heidelberg, February 2004, pp.19-34.
7. J. Carroll et al. d'etre: capturing design history and rationale in multimedia narratives. In: HUMAN FACTORS IN COMPUTING SYSTEMS (CHI94), Boston-USA, ACM Press, 1994, p. 192-197.

8. K. Czarnecki and U. Eisenecker. **Generative Programming: Methods, Tools, and Applications**, Addison-Wesley, 2000.
9. L. Cysneiros, J. Leite and J. Neto. A Framework for Integrating Non-Functional Requirements into Conceptual Models. **Requirements Engineering Journal**, Vol. 6, No. 2, p. 97-115, 2001, Springer-Verlag London Limited.
10. Draco - Software Reuse, Domain Analysis and Draco Information. Available at: http://www.bayfronttechnologies.com/l02draco.htm. Accessed on: Mar, 7th, 2006.
11. M. S. Feather. A Survey and Classification of some Program Transformation Approaches and Techniques. In IFIP 87, pages 165-195, 1987.
12. F. García, M. Laguna, Y. González-Carvajal and B. González-Baixauli. Requirements variability support through MDD and graph transformation. Submitted to Elsevier Preprint, 2005.
13. P. Giorgini, J. Mylopoulos, E. Nicchiarelli and R. Sebastián, Reasoning with goal models, Proceedings of the 21st International Conference on Conceptual Modeling, 2002, pp. 167-181.
14. B. Gonzáles, M. Laguna and J. Leite, "Visual variability analysis with goal models", Proceedings of IEEE International Symposium on Requirements Engineering (RE'04), Japan, 2004, pp. 38-47.
15. O. Gotel, and A. Finkelstein. An analysis of the requirements traceability problem. In: PROC. OF THE FIRST INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING (ICRE'94), IEEE Conputer Society Press., 1994. p. 94-101.
16. GRAPHVIZ. Available at: http://www.graphviz.org/. Accessed on: Mar, 7th, 2006.
17. P. Hsia et al. Formal Approach to Scenario Analysis. **IEEE Software**, vol. 11, No. 2, 1994. p. 33-41.
18. A. Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering", IEEE Transaction Software Engineering, 26(10):978–1005, 2000.
19. M. Lehman. Laws of software evolution revisited. **Lecture Notes in Computer Science**, Vol. 1149, 1996. p.108-120.
20. J. Leite and A. Franco. O Uso de Hipertexto na Elicitação de Linguagens da Aplicação. In: ANAIS DE IV SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 1990. p. 134–149.
21. J. Leite, **M. Sant'Anna, F. Gouveia.** Draco-PUC: A Technology Assembly for Domain-Oriented Software Development, International Conference on Software Reuse 1994.
22. J. Leite. Viewpoints on Viewpoints. In: ACM Joint Proceedings of the SIGSOFT'96 Workshops, ACM Press, 1996. p. 285-288.
23. J. Leite et al. Enhancing a requirements baseline with scenarios. In: PROC. OF THE THIRD IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING (RE'97), IEEE Computer Society Press, 1997. p. 44-53.
24. J. Leite, Y. Yu, L. Liu, E. Yu and J. Mylopoulos, "Quality-Based Software Reuse", Proceedings of the CAiSE 2005-LNCS 3520, 2005, pp. 535-550.
25. J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach", IEEE Transactions on Software Engineering, 18(6):483–497, June 1992.
26. J. Neighbors. The Draco Approach to constructing Software from reusable components. In: IEEE Trans. on Software Engineering, vol.SE-10, No.5, pp.564-574, September-1984.

27. B. Nuseibeh. Crosscutting requirements. In: PROC. OF THE 3RD INTERNATIONAL CONF. ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD 2004), Lancaster-UK, 2004. p. 3-4. ISBN:1-58113-842-3.
28. W. Robinson, S. Pawlowski and V. Volkov. Requirements Interaction Management. **ACM Computing Surveys**, Vol. 35, No. 2, 2003. p. 132-190.
29. C. Rolland et al. A proposal for a scenario classification framework. **Journal of Requirements Engineering**, Vol. 3, Springer Verlag, 1998. p. 23-47.
30. M. Sant'anna, J. Leite and A. Prado. A Generative Approach to Componentware. In: Proc. of the Workshop on Component-based Software Engineering, ICSE'20, Kyoto, Japan, April 1998.
31. L. Silva, J. Leite. An Aspect-Oriented Approach to Model Requirements. In: RE'05 DOCTORAL CONSORTIUM in conjunction on the 13th IEEE International Requirements Engineering Conference, Paris-France, 2005.
32. L. Silva. **An Aspect-Oriented Strategy to Model Requirements**. Rio de Janeiro, 2006. 220p. PhD Thesis on Software Engineering - PUC-Rio. In Portuguese.
33. I. Sommerville. **Software Engineering**, Ed. 6th, Addison- Wesley, 2000.
34. P. Tarr, et al. "N Degrees of Separation: Multi-Dimensional Separation of Concerns". In: PROC. OF THE 21st Int'l Conf. on Software Engineering (ICSE'99), 1999. p. 107-119.
35. K. Weidenhaupt et al. Scenario Usage in system development: current practice. **IEEE Software**, Vol. 15, No. 2, 1998. p. 34-45.
36. Y. Yu, J. Leite and J. Mylopoulos, "From goals to aspects: discovering aspects from requirements goal models", Proceedings of IEEE International Symposium on Requirements Engineering (RE'04), Japan, 2004, pp. 38-47.
37. Y. Yu, J. Mylopolous, A. Lapouchnian, S. Liaskos and J. Leite, "From stakeholder goals to high-variability software design", Internal report, 2005.
**38.** L. Zorman. Requirements Envisaging through utilizing scenarios – REBUS. 1995. Ph.D. Dissertation, University of Southern California.