

Representing Requirement Relationships

Ipek Ozkaya

Software Engineering Institute, Carnegie Mellon University
ozkaya@sei.cmu.edu

Abstract

Capturing structural relationships between requirements is essential for effective visualization. Existing techniques such as UML, SysML, and goal-oriented requirement engineering present a relationship focused approach in visualization; however, a commonly shared understanding of requirement-based relationships has not yet emerged. Considerations in structural and visual representation of requirement relationships are discussed in the paper, drawing issues from commercial tools, UML, SysML requirements diagram and goal-oriented requirement engineering. Lessons learned from author's own experience of developing requirement engineering tool support with graph-based techniques are summarized. Suggestions for areas of investigation for practical and better use of requirement relationships via visualization are provided.

1. Introduction

Visualization methods specifically targeting requirement engineering have not yet reached maturity where it is possible to analyze the outcomes for reducing semantic complexity, capturing the engineer's mental image, assisting bridging requirements with other design artifacts, and navigation in the rich and crowded requirement information space. This paper emphasizes understanding requirement relationships and basing efforts in universally capturing such relationships as a critical focus for requirement engineering visualization. The benefits of this approach can be better capturing repeating requirement specification patterns, using relationships for navigation and browsing, overcoming semantic gaps along with better mapping of requirements with possible design fragments and test specifications.

It is cognitively hard to refer to a complete list of requirements; therefore, the tendency is to think of

general relationships between them: Which requirements can be derived from another? Which are directly affected by a change? Which are independent from the rest? How functional requirements and quality attributes are related to each other? What kinds of usage scenarios exist? An external representation that makes it easy to see structural relationships and dependencies between requirements can potentially assist managing requirement engineering complexity and address such questions.

The success of using visualization techniques in earlier phases of design in other engineering disciplines is partly a result of the earlier treatment of relationships between requirements with a corresponding visual representation. Bubble diagramming in civil architecture is an example to this [1]. Requirement analysts, referred to as architectural programmers in civil architecture, do not just provide a list of spaces that the building should include, but lay them out in a bubble diagram, annotating the diagram with given or implicit relationships between the requirements pertaining to spaces. Patterns emerge as a result of this early treatment and recognition of repeating dependencies originating from requirements [2]. Software requirement engineering visualization can also benefit from understanding the relationships between requirements earlier in the software development life-cycle and how they can best be structured with a robust underlying data structure. Emphasizing requirement relationships via visualization can assist in capturing such requirement patterns and matching them with their corresponding satisfactory design and architecture fragments.

The paper is organized as follows. In Section 2, the use of requirement relationships in commercial tools, Unified Modeling Language (UML), Systems Modeling Language (SysML), and goal-oriented requirement engineering are summarized. Section 3 presents lessons learned in capturing requirement relationships computationally. This work was conducted as part of a research project for developing

tool support in computer-aided requirement traceability in design [3]. Conclusions and potential future work are summarized in Section 4.

2. Requirement Relationships

Commonly used requirement-focused relationships within the requirement engineering research and practitioner community can be listed as:

- *Traceability relationships*, which assist tagging change dependencies;
- *Message exchange relationships* provide an implicit method of capturing multiple requirements in use case scenarios;
- *Dependency relationships* are used in capturing how requirements may be realized by other software artifacts such as classes or components.

2.1. Commercial tools

The widely understood and shared mental image of class relationships of the object-oriented designs do not yet exist in the domain of requirement engineering, which makes coming up with effective visualization techniques be limited to information navigation as represented by commercial requirement engineering tools. Tools specifically designed for requirement engineering provide information management capabilities with in-built databases, document templates and functionalities to extract requirement information from existing documents. Current approaches of capturing requirement relationships visually are limited to representing traceability relationships and capturing requirement attributes. IBM-Rational Requisite Pro, Telelogic Doors, Borland Caliber are typical examples of such tools. In these, the internal representation of a requirement is a textual description. They rely on a back-end database support to store requirements. Such databases do not use any common data model across applications; hence do not capture repeating specification patterns and the relationships observed.

The relational database support assists capturing an internal structure within requirements as they are specified, such as grouping by a predetermined classification like features or use cases or requirement attributes such as owner. Using the relational data tables, it is possible to view requirements in matrices, tables or trees. This approach eases navigation and search capabilities; however, when it comes to utilizing requirements information in other design tasks there isn't much assistance as to how to interpret the semantic information better. The only structural relationship captured is a traceability relationship,

which is best viewed with a matrix following traced-to and -from arrows.

2.2. Scenarios

Scenario-based requirement specification methods assist in providing a larger semantic context to circumvent the problems observed in unstructured lists of requirements. Use case analysis [4] and quality attribute scenarios [5] are successful examples in applying scenario-based requirement elicitation and specification techniques. In a scenario-based approach visual representation is challenging because the atomic elements of the requirements are often not uniquely identifiable. Use case analysis circumvents this problem by close coupling requirements in scenarios with object-oriented designs.

UML has become the *de facto* visual representation technique, especially in representing detailed design and architecture, despite many of its drawbacks [6]. UML is also utilized in representing requirements as an extension of use case analysis. The scenario view of interaction diagrams and the ability to depict state transitions with actions within states make UML a viable tool in adoption for requirement engineering visualization. Interaction diagrams focus on message exchange view of the requirements problem ignoring fundamental aspects of quality attribute requirements. Similarly, state chart diagrams separate the discrete states, to each of which a different quality attribute scenario maybe applicable for. This makes it hard to capture a well defined overall requirement abstraction. While some of the object-oriented design relationships that can be captured with UML may hold for requirement representations as well, detailed design concepts are not always the most viable abstraction level for requirement capture and visualization.

Consider a case where the engineer needs to represent a flexibility requirement of replacing commercial-off-the-shelf components of the product:

The engineers replace the commercial database with one by another vendor in less than two weeks.

A usage scenario is not the best way to capture this requirement visually. Considerations in visualizing this requirement would lie more in understanding its relationship to other requirements and architecture. Which components encapsulate change? What other requirements influence this requirement, such as what kind of a dependency exists between achieving this requirement and performance? An effective visual representation for capturing this requirement should capture the types of dependencies that could occur between other requirements.

2.3. SysML Requirements Diagram

SysML is a visual modeling language based on UML, specifically addressing specification, analysis and design of systems of systems, including software [7]. SysML includes a *requirements diagram* as well. The requirements diagram offers a relationship view of requirements, most of which address dependencies and allocation of requirements in other software artifacts, such as design components. Requirements are envisioned to be depicted in graphical, tabular or tree structure formats. Reference [7] presents the syntax and details of the requirements diagram in SysML. This section focuses only on the relationships captured in this diagram.

SysML captures atomic requirements as a requirement stereotype of UML class. The relationships in SysML requirements diagram are grouped under dependency relationships as copy, satisfy, derive, verify, and refine. Copy relationship creates a read only copy of the requirement text for reuse management. Derive relationship is for organizing subordinate dependencies, for example a requirement may be derived from a business goal. Satisfy and verify relationships are aimed at bridging requirements with design elements and test cases respectively. Refinements are those relationships where a requirement adds more information onto the base one. Derive, verify and copy relationships inherit their properties from UML's trace construct. The visual representations of these relationships are as stereotypes on the dashed arrow.

While the specific relationships to bridge requirements between systems architecture and testing artifacts may prove to be helpful, for the requirements diagram to scale a better break down of a requirement stereotype is needed. A requirement even as a UML class stereotype is captured by the long text that represents a requirement. A closer look into this stereotype may reveal for example better decomposition of requirements as scenarios of usage and quality. Better visual guidance about how multiple requirements relate to each other to make up a scenario would also be valuable to understand. Opportunities may exist also in creating project taxonomies if attributes on a requirement take special form, such as specifying list of types of a requirement for organization.

Verify, satisfy, and trace relationships in the requirements diagram focus on relating requirements to other software design artifacts, such as test cases. This assumes only a direct dependency to commonly occur between a requirement and a design element. When one-to-many and many-to-many relations exist the

diagrams become cluttered with lines reducing legibility. In addition, when relating requirements in this level of detail the integrity of the underlying models also needs to be considered. One of the challenges in representing design information with a visual abstraction is the inevitable gap that may occur between the visual model that the designer views and its conformance with the underlying model. The problems occurring as a result of this gap has been previously analyzed in the context of UML diagrams [8].

The requirements diagram does not offer a higher level of abstraction to support requirement navigation. The current visualization constructs would inevitably result in several diagrams with a lot of elements, in a way representing most of the problems observed in requirement texts in a graphical format. This ultimately calls for robust tool support.

2.4. Goal-oriented requirement engineering

Goal-oriented requirement engineering is a formal approach for mapping business goals to requirements and architectural design [9]. Requirement entities and relationships are captured with the goal-oriented requirement language (GRL). The approach is well established with extensions in representing use case maps and user requirement notation [10]. The formal representation facilitates notation-based mapping from requirements to other representations, such as state-charts or component-connector representations of a system; hence move from requirements to design phases [11]. In goal-oriented requirement engineering a client's business goal may be decomposed in different ways and may lead to different functional requirements depending on the decomposition. The different ways of decomposing the goal leads to an alternative search space for solution exploration.

Goals are modeled in terms of *hard goals* or *soft goals*. Functional goals that do not lend themselves to ambiguity can be modeled as hard goals, such as "Withdraw Money". Goals that are qualitative in nature are characterized as soft goals, when they do not have a clear criterion, such as "Withdraw Money Fast".

Goal models use relationships extensively. AND/OR relationships with obvious semantics where AND depicts inclusion of both goals whereas OR depicts a choice is an example. Others as means-ends, decomposition, contribution, correlation, and dependency are represented in GRL [12]. These relationships allow for fine tuning in relating functional requirements, quality attributes, and goals to each other. The formal representation of requirements as goals facilitates automated generation of other software

artifacts such as a feature model or high-level architecture; however, the visual constructs do not overcome the semantic gaps observed with text and natural language based representation. The hard versus soft goal differentiation provides a possible way of categorizing and representing both functional and quality attribute requirements simultaneously, but the quantitative information such as how fast the withdraw money should execute is not readily available visually. Browsing goal-oriented diagrams is also not intuitive. Only with robust tool support practical effectiveness can be measured.

3. Structural visualization with a graph

Our previous work in implementing computer-aided design tool support in structuring requirements for visualization and traceability focused on extracting relationships in textually represented requirements. *DesignTrack*, the tool outcome of this effort, is implemented as an Eclipse plug-in. It utilizes a directed acyclic graph for structuring requirements. The emerging graph can be viewed in xml, spread sheets or trees to inspect the relationships [3]. In *DesignTrack*, each requirement is represented by a node and weighted-edges encapsulate different types of relationships between the requirements. Requirement text is parsed for extracting association hierarchy, indexing, classification, dependency, and implication relationships between requirements. The process is semi-automated with user interaction. References [13] and [14] further review technical challenges observed during implementation and how these relationships were identified.

Our effort highlighted several axioms specifically focused on capturing requirement relationships:

- *Making the distinction between composite-requirements and atomic- requirements is possible by using relationships:* An atomic-requirement is a requirement that can not further be broken down into other requirements in a requirement statement. There can be several requirement concepts that are atomic in nature such as a scenario or a constraint. The analogy with a class representation can be made here. A class is composed of different attributes and functions which allow for communication and reference outlets to that class. For a requirement there is no such accepted analogy. Stereotyping a requirement as a class as done in SysML requires this analogy to be further investigated. A requirement defined as an atomic requirement can later participate in generation of a new requirement by means of relationships. A composite-requirement is a requirement statement

made of atomic requirements and the relationships between them. Representing atomic requirements may better assist relating crosscutting requirements, such as all user transactions should be completed by a certain time.

- *View generation can be based on capturing the semantics of different relationships:* For example, association and hierarchy relationships can assist in generating a feature model view. Current requirement engineering practice often resorts to use case analysis for feature modeling when needed. However, when use cases do not encapsulate feature dependencies well this practice results in awkward application of use case models, resulting in ineffective use case definitions.
- *Relationships assist in navigation:* In a graph, it is possible to view requirements within accessibility paths from a given requirement node. Moreover, it is possible to generate partial models for navigation purposes by limiting or allowing viewing certain relationships. Classification and indexing further assists in creating containers for quick access and search.
- *Patterns specific to requirements emerge as a result of repeated use of relationship in requirement structuring.* There are requirements that appear as part of all the other requirements such as optimum transaction performance, or those that may be idle such as look and feel of the interfaces. Visualization assists in detecting these cases when requirements are related to each other with semantics.

These axioms can also be observed in the methods reviewed in section 2. Navigation is the most explicitly and commonly extractable axiom, while the rest often remain as implicit concerns. Other axioms can be added to these as well. However, what is missing is a common understanding of what constitutes a top priority indispensable axiom in respect to requirement representation. The disperse variations of semantically similar requirement relationships in different methods and tools supports this observation.

4. Conclusion and Future Work

The review presented here of the most commonly used techniques for requirement specification with visualization focus reveals a top priority concern in requirement engineering visualization. There are different schools of work with emphasis on capturing requirements relationships for concerns of business process modeling, traceability, better relating architectures and requirements and business goals, change management, generative computing, and the

like. However, a common and agreed upon understanding of what is most viable to capture visually in requirements does not exist. While navigation is a concern, better navigating through requirements does not guarantee their better analysis and use. On the other hand, overloading the formal and semi-formal representations can create undesired burdens on users.

Requirement engineering visualization is an area where a work redesigns in how a requirement engineer captures requirements would be required [15]. Future work should look into issues pertaining to relating requirements to each other and other software artifacts for practical use of requirement information.

- *Understanding the mental model best applicable for visual requirements:* How do engineers structure requirements as they work through the elicitation and specification process? How are relationships of requirements used? How they are typically captured? Who would produce and use the visual representations?
- *Visually detectable behavior of requirements:* Are there structural relationships between requirements that can be visually captured and mapped them with corresponding architectural patterns? Merely laying requirements in UML class stereotypes and treating them equivalent to how a class behaves in object-oriented design only moves the scalability and consistency issues to a visual context. Semantic relationships between requirements should be studied. Clustering and idle nodes are examples to visually detectable behavior of requirements. In a graph if many requirements span out of a requirement node that calls out for special analysis and attention as well. Could relationships help in capturing problem patterns, such as layers of requirements or compositions of requirements?
- *Capturing traceability at a higher level of abstraction:* The most common business case for requirement traceability is managing the costly impact of change management. This mostly pertains to issues observed in the implementation level. How can traceability at a higher level of abstraction, starting from business goals and moving to architecture representation be captured visually? How can relationships help? Would it suffice to rely solely on traced-to traced-from relationships?
- *Tool support:* What kind of tool support is optimum for requirement engineering visualization? What are typical user tasks that tools can assist in improving the use of requirement information throughout the software development life-cycle?

5. References

- [1] Pena, W. and S. Parshall, *Problem Seeking: An Architectural Programming Premier*, Washington AIA Press. 2001.
- [2] Alexander, C. *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, MA, 1964.
- [3] Ozkaya, I. *Requirement-driven design: assistance for information traceability in design computing* Ph.D. Thesis, School of Architecture, Carnegie Mellon University, Pittsburgh, PA, 2005.
- [4] Armour, F. and G. Miller: *Advanced Use Case Modeling: Software Systems*. Addison-Wesley, 2001.
- [5] Bass, L, P. Clements, R. Kazman, *Software Architecture in Practice*, Addison Wesley, 2003.
- [6] Clements, P. et al., *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, 2002.
- [7] *SysML Specification* v1.0, OMG 2006.
- [8] B. H. C. Cheng, R. Stephenson, and B. Berenbach, "Lessons Learned from Automated Analysis of Industrial UML Class Models (An Experience Report)". *MoDELS 2005*, pp. 324-338
- [9] A van, Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective", *Proceedings of 22nd International Conference on Software Engineering*, 2000.
- [10] D Amyot and G. Mussbacher, "URN: Towards a New Standard for the Visual Description of Requirements". *3rd SDL and MSC Workshop (SAM02)*, LNCS 2599, Aberystwyth, U.K., 2002, pp. 21-37.
- [11] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. "Towards Requirements-Driven Autonomic Systems Design" *ICSE 2005 Workshop on Design and Evolution of Autonomic Application Software*, St. Louis, Missouri, USA, 2005.
- [12] Goal-Oriented Requirement Language Description <http://www.cs.toronto.edu/km/GRL/>. Last viewed June 2006.
- [13] I. Ozkaya and O. Akin, "Requirement-driven design: assistance for information traceability in design computing", *Design Studies*, Vol: 27, Issue 3, 2006, pp. 381-398.
- [14] I. Ozkaya and Ö. Akin, "Use of requirement traceability in collaborative design environments", *CoDesign*, Vol: 1, Issue 3, 2005, pp. 155– 167.
- [15] Beyer, H. and K. Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufman, 1998.