Improved Representation of Traceability Links in Requirements Engineering Knowledge using Sunburst and Netmap Visualizations

Thorsten Merten, Daniela Jüppner
Department of Computer Science
Bonn-Rhine-Sieg University of Applied Sciences
Sankt Augustin, Germany
thorsten.merten@h-brs.de
daniela.jueppner@smail.inf.h-brs.de

Alexander Delater
Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
delater@informatik.uni-heidelberg.de

***Abstract*—The representation of traceability links in requirements knowledge is vital to improve the general understanding of requirements as well as the relevance and consequences of relations between requirements artifacts and other artifacts in software engineering. Various visualization techniques have been developed to support the representation of traceability information, e.g. traceability matrices, graphs and tree structures. However, these techniques do not scale well on large amounts of artifacts and often do not provide additional functionality to present supplementary data. In this paper, we use Sunburst and Netmap visualizations as alternative visualization techniques. These techniques perform well even on large amounts of artifacts and traceability links. Moreover, they provide the ability to present derivative data. An implementation of the visualizations was developed in conjunction with a requirements plugin for the Redmine project management platform. In this paper, the applicability of Sunburst and Netmap visualizations for requirements engineering knowledge is illustrated by applying it to an example project and the results are compared to traditional visualization techniques.**

***Keywords*-component visualization, requirements, artifacts, knowledge-management**

I. Introduction

Requirements knowledge is usually captured in requirements artifacts, which are defined as “documented requirements” [1]. “Documented requirements” can be represented in various forms e.g. goals, scenarios, user profiles, use-cases etc. They are organized in requirements documents or requirements specifications. In turn, these documents are organized in sections or chapters.

However, additional traceability links between requirements artifacts can be added, especially when knowledge management (KM) tools are used. The knowledge about requirements artifacts and their traceability links to other artifacts is important for various tasks performed throughout the design and development of a software system [2]. Therefore, these tools incorporate various visualizations of traceability links between requirements artifacts to simplify understanding. Traditional approaches include traceability matrices as well as more advanced techniques like trees or graphs.

However, traditional approaches have various disadvantages. For example, most traceability links between requirements artifacts cannot be represented in a tree or traceability matrices can grow quickly to an enormous size, as N artifacts result in a NxN matrix.

In this paper, Sunburst and Netmap visualization techniques will be adopted for representing traceability links between requirements knowledge. Both visualizations provide an overview of the primary data’s locations in the knowledge base and display traceability links between requirements artifacts efficiently. Moreover, they perform well on large amounts of nodes [3]. For evaluation, the visualization techniques are applied on an example project, specified using the RE Plugin. The results are compared to other visualization techniques.

The remainder of this paper is structured as follows: Section 2 describes the motivation behind using Sunburst and Netmap and discusses related work on existing visualization techniques. Our approach is presented in detail in section 3 and is applied to an example in section 4. The results are discussed in section 5 and section 6 provides conclusion and highlights future work.

II. Motivation

Visualizing traceability links will directly support three main activities related to these links: recovering, browsing and maintenance of traceability links [4]. Browsing, in turn, supports the understanding of requirements knowledge. Generally, traceability links substantiate the meaning of artifacts as they are shown in their specific context. In the following, we will introduce different visualization techniques traditionally used for showing artifacts and their dependencies.

A. Lists

Lists can represent large amounts of similarly structured knowledge in a linear way. However, they are limited to a single dimension for arranging the information.

B. Matrices

A matrix is a two-dimensional grid that represents links between knowledge. One matrix can display traceability links between two sets of artifacts, such as requirements and design elements [5]. Even for non-technical users, traceability matrices are easy to understand and are used in simple scenarios, such as recording or checking the existence of single links between artifacts.

However, for real-world projects, traceability matrices can become very large and unreadable, especially if multiple layers of traceability are to be tracked [6]. Matrices can grow quickly and cannot display additional attributes. Often, more than one matrix is needed [1] to represent different types of connections within requirements artifacts.

C. Trees

Trees are the traditional way of presenting an overview of the requirements knowledge (see fig. 1). Established tools as IBMs RequisitePro or DOORS use these representation forms. In most tools the tree is embedded in the design pattern ”Two Panel Selector” described by Tidwell [7] to support efficient navigation.

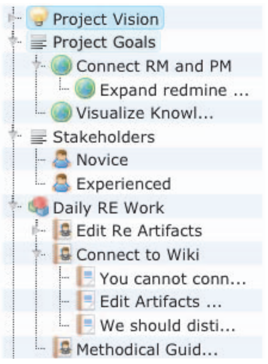


Figure 1. Requirement artifacts represented in a tree structure

Although being intuitive, trees lack the ability to present additional traceability links between requirement artifacts. Examples for these traceability links are rationale decisions, conflicts, decompositions for refinement, implementations and so on. Any trace beside the parent/child relation would extend the tree data structure towards a graph.

D. Traditional Graphs

If artifacts are interpreted as nodes and traceability links as edges, they can be visualized in a graph-based notation. In fact, a traceability matrix can be interpreted as the adjacency matrix of such a graph. Traceability graphs can provide an intuitive representation of relations between artifacts [8].

Despite all of these advantages, graphs with standard layout are often not as intuitive to ordinary users as matrices or simple references, as they can become huge and thus hard to understand [9]. Additionally, it is known that people tend to over interpret graphs. For example, nodes shown at a higher position in the graph tend to be recognized as more important [3].

As stated by Winkler et al., “scalable visualization techniques can help to manage the huge amounts of traces occurring in real-world projects” [5]. However, as described above, simplicity is the main advantage of lists, sections, matrices, graphs and trees, but they are not well suited for representing larger amounts of artifacts with multiple traceability links or do not scale well [4]. Thus, new visualization techniques are required to give an overview of requirements knowledge and its relations and to support for further analysis.

III. Approach

For a better understanding of our approach, the data model of the requirements repository will be explained first. Second, Sunburst and Netmap visualizations are presented and applied to an example requirements repository.

A. Data Model

The data model consists of requirements artifacts representing the primary knowledge and traceability links connecting artifacts. Each artifact has common properties like name, description or author. Requirements artifacts like goals, scenarios, use-cases, roles and so on are distinguished by their type property. The type is mapped to a color for visualization.

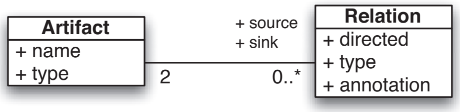


Figure 2. Data Model in UML2 Notation (simplified)

As for artifact types, color coded traceability link types are used to relate artifacts. A traceability link may be directed or undirected. As an example, a refinement from artifact A to artifact B should be modeled as a directed link. A conflict between artifact C and D might be undirected. A traceability link can be tagged to make it more expressive. For example, the refinement between A and B could be tagged as ”solution” to make clear that B is a concrete implementation for A. Fig. 2 gives an overview of the data model.

New artifacts are inserted into the requirements specification through the navigation tree as shown in fig. 1. The software transparently creates traceability links of the type parent/child. Fig. 3 displays some of these traceability link types: parent/child (blue, directed), dependency (orange) and conflict (red).

We use two visualization techniques to display the requirements artifacts and their traceability links in this paper. Firstly, a tree structure as in fig. 1 gets transformed in radial layout (Sunburst, see fig. 4). Secondly, a way of efficiently displaying and filtering traceability links is presented (Netmap, see fig. 3). Different colors per artifact type to facilitate distinction and recall of the types.

B. Sunburst Visualization

The Sunburst Visualization [10] is a graph, which nodes are arranged in a radial layout. Nodes are drawn on adjacent rings representing a tree structure. Each child of a node with depth n is represented in the ring $n + 1$ on the same radian space as its parent(s). The arrangement makes the graph more compact and provides a better user orientation as discussed below.

It can be seen in fig. 1 that a traditionally rendered tree, as used by most software products, grows rapidly in the vertical direction if many branches get expanded. It is comlicated to identify nodes on the same level of the tree. Furthermore, only a small part of the tree fits on a computer screen. By using Sunburst the tree gets transformed in a radial layout. The tree grows uniformly in all directions as shown in fig. 4 and 5. Nodes are displayed on adjacent rings representing the trees structure. The center represents the project (the root of the tree), whereas each ring is a structuring level of requirements artifacts (level).

The usefulness and understandability of this visualization has already been proven for the navigation in file system structures [10]. A major problem stated in [10] is that nodes on the outer circles tend to be small and may even become unreadable. The problem occurs for branches with many child nodes. However, the problem could be mitigated using a dynamic aspect of the infovis toolkit [11]: The toolkit allows the folding and unfolding of nodes, such that the user can hide branches, which are not of interest for the current research. Folded branches free space for other nodes, which get drawn bigger (see fig. 4 and 5). In our implementation, the visualization initially presents the whole specification. Afterwards users can focus on the information they are searching for. If a certain branch in the tree should be further researched regarding traceability links between requirements knowledge, a Netmap helps to gain more insights. This is presented in the next section.

C. Requirements knowledge in a Netmap

In difference to Sunburst, where nodes are drawn on adjacent rings representing a tree structure, the nodes in a Netmap [12] are segments of exactly one ring. Each node is represented equally to prevent misinterpretation of the nodes.

In the inner circle traceability links are drawn. Like the requirements artifacts, traceability link types are colored differently. Furthermore, directed traceability links are shown as arrows and undirected ones as hyper lines (curved lines) without an arrow.

Obviously, drawing all nodes on only one ring narrows the radial space for each node. Artifacts as well as the traceability link types need to be filtered by their type attribute to make the search for traceability links between requirements knowledge more efficient.

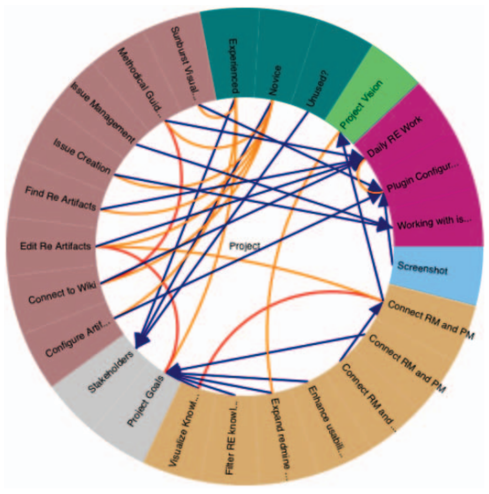


Figure 3. Netmap visualization of connected requirement knowledge

D. Filters and additional information

Our implementation enables to filter for artifact types as well as traceability link types. For example in fig. 3, seven node types are displayed whereas fig. 6 displays use-cases, user profiles and goals, only. From a users point of view, filters can be triggered using select boxes. Furthermore, subtrees of the specification can be visulized, which reduced the number of nodes significantly and helps focussing on certain specification parts.

Both visualizations are interactive. They display additional data about the requirements artifacts, like the full name or the description, when hovered with the mouse pointer as shown in fig. 6. This enables the user to get additional information without leaving the visualization.

When a node is clicked links to the artifacts editor as well as all related arefacts is displayed. Therefore, the visualization can be used for content navigation as well.

IV. Example

In this section both visualization types are applied to an example requirements specification. The specification is structured as shown in fig. 1. It defines a vision statement for the project. The next section consists of project goals. Thereafter user profiles get identified. The end of the specification consists of use-cases for these roles, which are themselves structured in work areas. The visualization will be used to answer the following questions regarding the specification:

1) To which extend are use-cases defined and where do they reside in the specification?
2) Is each use-case complemented with at least one user profile?

3) Have user profiles been identified which are not referenced anywhere in the specification?

Fig. 4 and 5 show how both visualization types help to achieve this. In fig. 4, Sunburst is used to show an overview of the specification. Due to the artifact type's color-coding, use-cases can easily be spotted in the upper region below the work areas. In fig. 5, we combined two steps. All artifacts, which are not of type work area, have been folded to free space for the use-cases. Then, the filter is broadened to show artifacts of type "use-case step". The "size" of use-cases can be derived through the number of its steps. Most use-cases are of roughly the same size, consisting of 3-7 steps. At the top-left of fig. 5 use-case "Edit RE Artifacts" consists of 10 steps giving it a major radial distance. Use-case "Methodical Guidance" at the top-right does not consist of any steps. It might be recommendable to check whether the large use-case is still understandable and if further specification is needed for the small use-case. Together, the above gives a comprehensive answer to question 1.

To answer questions 2 and 3, fig. 6 switches to the Netmap view. The figure uses the same data as fig. 3 whereas this view is filtered to display use-cases, goals and user profiles, only. Yellow lines show dependencies and red lines denote conflicts.

Concerning question 2, the Netmap view shows that each use-case besides "Issue Management" is dependent to at least one user. Question 3 can also be answered through the Netmap. The user profile "Administrator" at the right is not connected to any use-case. It should be checked if this profile is really needed in the specification or if important use-cases regarding this user profile are missing.

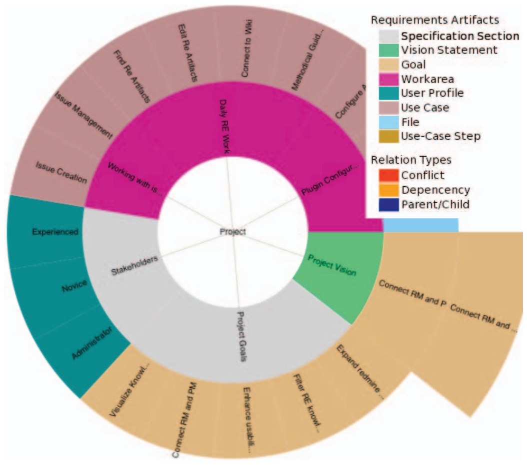


Figure 4. Tasks (unfolded)

Additionally, fig. 6 shows different goals and detailed artifact information for the goal "Visualize Knowledge". It can be seen that the use-case "Edit RE Artifacts" is connected to a goal, but the others are not. Different patterns in the visualizations are hints for various problems. Though

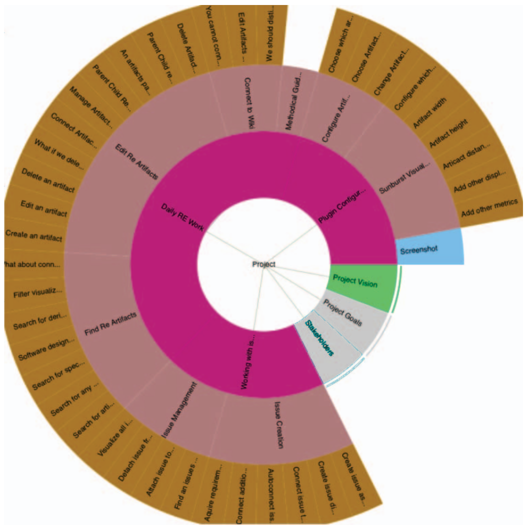
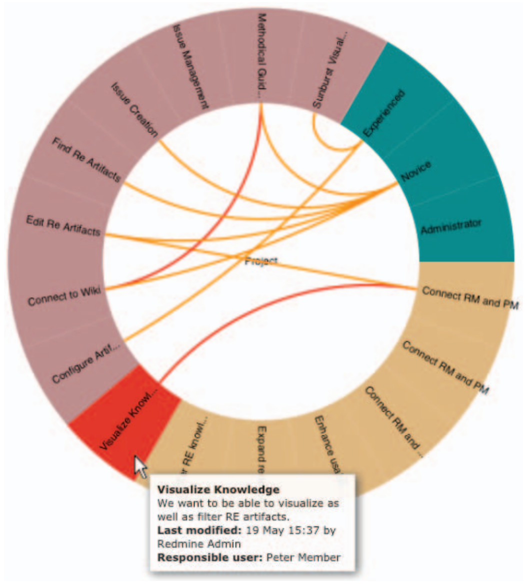


Figure 5. Subtasks (folded)



Visualize Knowledge
We want to be able to visualize as well as filter RE artifacts.
Last modified: 19 May 15:37 by Redmine Admin
Responsible user: Peter Member

Figure 6. Goals (filtered)

the above questions can be answered differently, e.g. using filters and database queries, these patterns deliver hints for broken or underspecified data, which should be analyzed and corrected before entering the final specification.

V. Discussion

Three exemplary questions have been answered by applying Sunbust and Netmap visualizations on realistic specification data. The questions could be answered using the combination of both visualization techniques. The questions exemplified in this paper are very basic and may also be answered differently, e.g. using complex database queries; as such the visualizations have to be evaluated in real life projects. However, though the first question can be answered using a traditional tree rendering as well, Sunburst saves space and the number of child nodes – in our case the number of use-case steps – is more comparable because of the radial layout. In traditional graph layouts, humans could hardly derive question 2 and 3. Graphs would become rather large with this amount of nodes and data filtering generally changes the way the graph is displayed when automatic layouts are used, all of which confusing the users. Questions 2 and 3 may be answerable using a matrix, but the drawbacks of matrices as represented above would become evident. Though the example questions can be answered differently, the potential of Sunburst and Netmap in RE could be demonstrated and it could be shown that humans can quickly detect patterns regarding a specifications quality.

Both visualization techniques are not very helpful without filters. Although they perform better than other graph layouts, they do not scale endlessly. Therefore, additional filtering techniques in combination with different visualizations should be further researched. The example uses several questions regarding specification quality and completeness. Different filters and filter settings may be related to different questions, that are of interest when browsing, editing or building a requirements knowledge base. On the other hand adding additional filters introduces problems; e.g. whenever the data is filtered for a certain part (branch) of the specification. Traceability links to other parts of the specification are not shown in the Netmap, since these nodes are not displayed on the ring.

VI. Conclusion and Future Work

In this paper, we have shown that Sunburst and Netmap visualizations are promising approaches for visualizing traceability links between requirements artifacts. They enable effective browsing of and intuitive navigation within the requirements knowledge base. The example questions in this paper are defined from a quality assurance perspective. In future work, we want to implement additional filters such that more questions can be answered using the presented visualizations. This way filter configurations for different stakeholder can be defined. Additionally, we want to combine both visualization techniques with other graph visualizations to alleviate some of the problems that get introduced by additional filtering techniques.

The visualizations presented in this paper have been implemented in a requirements plugin for the Redmine project management (PM) tool [13]. The RE plugin is developed in the BMBF project KoREM and uses the JavaScript Visualization Toolkit [11] to plot Sunburst and Netmap graphics. Redmine is web-based and consists of a bug-tracker, wiki, forum and other sources of PM knowledge. This infrastructure enables requirements and project management knowledge to be interwoven more tightly, as the RE Plugin is able to connect to Redmines knowledge management infrastructure. Thus, the integration enables us to visualize additional information together with requirements artifacts in the future.

Acknowledgment

This work is partly funded by the BMBF (German Federal Ministry of Education and Research) within the research project KoREM (http://www.korem.de). We thank everybody involved in the Javascript Visualization Toolkit project and at the Redmine project for providing his or her work open-sourced. The RE Plugin developed in KoREM will be returned to the community in a subsequent release.

References

[1] K. Pohl, ”Requirements Engineering: Fundamentals, Principles, and Techniques”. Springer Publishing Company, 2010.

[2] O. Gotel and A. Finkelstein, ”An analysis of the requirements traceability problem” in Software Change Impact Analysis, R. Arnold and S. Bohner, Eds. IEEE Computer Society Press, 1996.

[3] R. Spence, ”Information Visualization”, Addison Wesley, 2000

[4] A. Marcus, X. Xie and D. Poshyvanyk, ”When and how to visualize traceability links?” In TEFSE ’05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering (New York, NY, USA, 2005), ACM, pp. 56-61.

[5] S. Winkler and J. Pilgrim, ”A survey of traceability in requirements engineering and model-driven development”. Softw. Syst. Model. 9, 2010, pp. 529-565.

[6] E. Hull, K. Jackson, J. Dick, ”Requirements Engineering”. Springer, 2nd edition, 2005.

[7] J. Tidwell, ”Designing interfaces”, O’Reilly Media, Inc., 2006

[8] P. Heim, S. Lohmann, K. Lauenroth, and J. Ziegler. ”Graph-based visualization of requirements relationships”. In Proceedings of the 2008 Requirements Engineering Visualization, REV ’08, pages 51-55, Washington, DC, USA, 2008. IEEE Computer Society.

[9] I. Herman, G. Melancon, M.S. Marshall, ”Graph visualization and navigation in information visualization: a survey”. IEEE Trans. Vis. Comp. Graph. 06(1), 2000, pp. 24-43

[10] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald, ”An evaluation of space-filling information visualizations for depicting hierarchical structures”. International Journal of Human-Computer Studies, 53(5), 2000, pp. 663-694.

[11] N. G. Belmonte, ”The Javascript Visualization Toolkit”, http://thejit.org, 2011.

[12] Netmap Analytics Pty. Limited, http://www.netmap.com.au, 2007.

[13] J.-P. Lang, Redmine, http://www.redmine.org, 2011.