# D3TraceView: A Traceability Visualization Tool

Gilberto A. de A. Cysneiros Filho
Department of Statistics and Informatics
Federal Rural University of Pernambuco
Recife, Brazil
g.cysneiros@gmail.com

Andrea Zisman
Centre for Research in Computing
The Open University
Milton Keynes, UK
andrea.zisman@open.ac.uk

***Abstract*— Software traceability is the ability to relate artefacts created during the life cycle of software development. Traceability is fundamental to support several activities of the software development process such as impact analysis, software maintenance and evolution, verification and validation. Despite the importance and advances in the software traceability area, traceability practice is still a challenge. One of these challenges is concerned with the visualization of traceability information. In this paper, we present D3TraceView, a traceability visualization tool that allows displaying traceability information in different formats depending on the purpose of use of traceability information. The tool supports different types of queries related to the use of traceability information. We use an Air Traffic Control Environment multi-agent system to demonstrate the use of the tool.**

***Software Traceability; Information Visualization; D3.js***

## 1. Introduction

Requirements traceability has been defined as the "*ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)*" [6][1]. This definition has been extended to incorporate the notion of software traceability [7] that encompasses and interrelates any uniquely identifiable software engineering artifact to any other[1].

The importance of traceability in the software development life-cycle has been endorsed by several standards for quality management and process improvement (CMMI, SPICE). Traceability is fundamental to support various activities of software development such as impact analysis, reuse, verification and validation; program comprehension; maintenance; and software evolution. Despite its importance, there are still several significant challenges associated with traceability, as described in [7]. In addition, in [3] the authors present several compelling areas of research that need to be addressed in order to support the challenges. One of these areas is concerned with the visualization of trace data. At the moment traceability information can be difficult to use since little attention has been given to the issue of presenting and visualizing traceability information.

As outlined in [12], the most common way to show traceability information in practice remains the use of matrix. Other ways to show traceability relations are tree browser explorer, lists, and hyperlinks views. As presented in the empirical study in [12], matrices and graphs are good to support management tasks; hyperlinks were preferred to support implementation and testing tasks. Matrices are adequate to provide an overview of the traces, and graphs could facilitate navigation through the traces. However, these forms of individually used visualizations have limitations to present traceability relations and assist with their use. The appropriate view of traceability information should be related to the task to be performed with the relations [22]. However, users are not always able to decide the forms of visualization that are most appropriate for the information they have, and for the tasks they need to use traceability information.

In this paper, we describe D3TraceView, a traceability visualization tool that provides different types of visualizations to users depending on the data to be visualized, the role of the user, and the task to which the user needs to perform with the relations. The tool is based on requirements identified on a previous work from one of the authors [5], and requirements defined in [16].

The main goal of the tool is to provide several types of visualization formats for specific trace data. The tool receives as input software artifacts (software models) and their respective traceability relations, and produces different views of the relations in various formats such as tables, matrices, radials, trees, sunbursts, etc. The tool also contains a repository of predefined queries related to the use of traceability relations. These queries are created to help users to execute several software activities, namely (i) software inspection (verification), (ii) maintenance and evolution (program comprehension), (iii) consistency checking of the models, and (iv) impact analysis. The tool was developed with Data-Driven Document (D3) JavaScript library (D3.js) [1].

The remaining of this paper is structured as follows. Section 2 provides a description of the D3TraceView tool. Section 3 explains the implementation aspects of the tool.

[1] Unless necessary to make a distinction, in this paper, we will use the term "traceability" to refer to both requirements and software traceability.

DOI reference number: 10.18293/SEKE2017-038

Section 4 discusses related work. Finally, Section 5 presents some conclusions and directions for future work.

## 2. D3TRACEVIEW TOOL

The goal of the D3TraceView is to be a visualization tool that provides interactive and context-specific traceability usage. In order to illustrate the tool and describe its main functionalities, we use examples of artefacts created during the development of a multi-agent system implementing an Air Traffic Control Environment (ATCE). The ATCE system was developed using *i** framework [23] to model the organizational environment; Prometheus methodology [19] to model system specification, architectural design, and detailed design phases; and JACK programming language [11] to implement the code. The traceability relations were generated using the rule-based traceability tool called RETRATOS [4], proposed by the authors of this paper.

D3TraceView was developed based on the challenge described in [13], in which traceability tools should be developed having user-friendly ways of formulating predefined and ad-hoc traceability queries, involving traces, artifacts, and all their relationships. Based on this concept, we developed D3TraceView to support a set of queries to help users to execute activities as proposed by OpenUP process [18]. Examples of these activities are: (i) assess results, (ii) software inspection, (iii) consistency checking, and (iv) change impact analysis.

The queries shown in this paper are concerned with artifacts of a multi-agent system, given the ATCE application we used to illustrate the tool. Examples of these queries are: (a) What are the i* actor elements that originate Prometheus goal elements? (b) What JACK agent elements are affected by changes in SD goal elements? However, other types of queries related to different activities, stakeholder's roles, and types of software artifacts could be specified and easily incorporated in the tool.

Apart from predefined queries, users can also analyze artifacts and trace relations browsing various visualization formats provided by the tool. This is possible by the use of mechanisms to zoom into some other data, as well as filtering retrieved data, and expanding and collapsing elements (e.g., matrix, table, radial, tree, sunburst).

### *A. Roles and Activities*

Based on OpenUP process [18], we have defined several roles and activities to be supported by the tool. The OpenUP is an agile process for the development of software systems. We decided to base D3TraceView tool on OpenUP process due to: support for the whole system process development; being freely available; support for several roles that define the behavior and responsibilities of an individual, or a set of individuals in a development team; and (iv) support for development tasks.

The stakeholder's roles adopted by the tool are: (i) user/customer; (ii) business analyst; (iii) architect; (iv) designer; (v) project manager, and (vi) developer. The different activities currently implemented in the tool are: (a) assess results, (b) software inspection (verification), (c) maintenance and evolution (program comprehension), (d) consistency checking, and (e) change impact analysis. Table 1 shows a mapping of these activities and roles.

TABLE 1. MAPPING OF ACTIVITIES AND ROLES

| Activities | Stakeholder's Role |
|---|---|
| Assess Results | Project Manager |
| Software Inspection | Business Analyst |
| Maintenance and Evolution | Architect, Designer, Developer |
| Consistency Checking | Architect, Designer, Developer |
| Change Impact Analysis | Business Analyst |

### *B. Files, Queries and Visualization Formats*

The screen in Figure 1 shows the main menu of options of D3TraceView. As shown in Figure 1, the user can (i) provide inputs about the software artifacts and traceability relation files to be used by the tool (option Files), (ii) choose pre-defined queries and visualize trace data in different formats that are associated with the queries (option Queries), and (iii) visualize traceability data in various different formats (option All Views). Figure 2 shows part of the screen to upload software artifacts to be used.

When a user chooses option "Queries" from the screen in Figure 1, the screen shown in Figure 3 is presented to the user displaying the activities supported by the tool. After selecting one of these activities the tool presents a set of pre-defined queries for the respective activity. Figures 4 to 8 show the screens with some of the queries for each of the various activities. For each activity, the user can select one or more queries. In addition, for some queries, the user can decide on a relevant artifact from a menu option. For example, as shown in Figure 6, in the case of the query "Who are the i* actors that originates Prometheus goals", the parts concerned with i* actors and Prometheus goals can vary depending on the artifacts being considered.

For each query, the tool presents its results as different graphs that are more appropriate to display the results of the query. However, it is possible to choose from a different type of visualization format (e.g., a visualization format to which the user is more familiar), by following the "All Views" option in Figure 1.

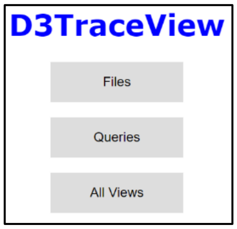


Figure 1. Main Menu

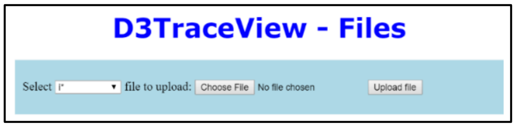


Figure 2. Upload of software artifact files

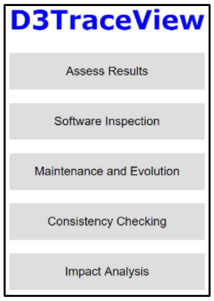
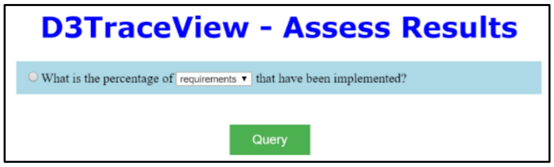
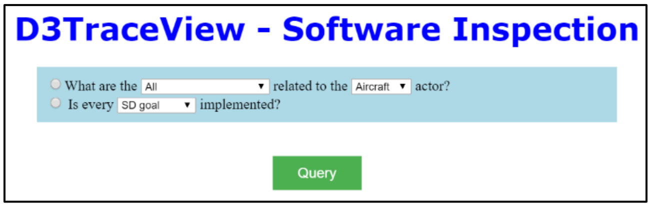


Figure 2. Menu of Activities



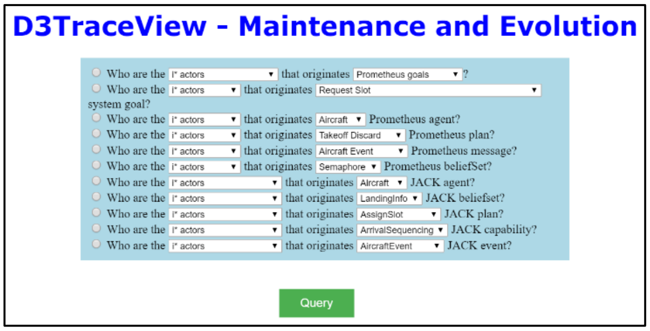
○ What is the percentage of requirements ▾ that have been implemented?

Figure 4. Assess Results



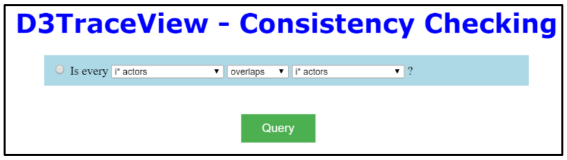
○ What are the All ▾ related to the Aircraft ▾ actor?
○ Is every SD goal ▾ implemented?

Figure 5. Software Inspection



○ Who are the i* actors ▾ that originates Prometheus goals ▾ ?
○ Who are the i* actors ▾ that originates Request Slot ▾ system goal?
○ Who are the i* actors ▾ that originates Aircraft ▾ Prometheus agent?
○ Who are the i* actors ▾ that originates Takeoff Discard ▾ Prometheus plan?
○ Who are the i* actors ▾ that originates Aircraft Event ▾ Prometheus message?
○ Who are the i* actors ▾ that originates Semaphore ▾ Prometheus beliefSet?
○ Who are the i* actors ▾ that originates Aircraft ▾ JACK agent?
○ Who are the i* actors ▾ that originates LandingInfo ▾ JACK beliefset?
○ Who are the i* actors ▾ that originates AssignSlot ▾ JACK plan?
○ Who are the i* actors ▾ that originates ArrivalSequencing ▾ JACK capability?
○ Who are the i* actors ▾ that originates AircraftEvent ▾ JACK event?

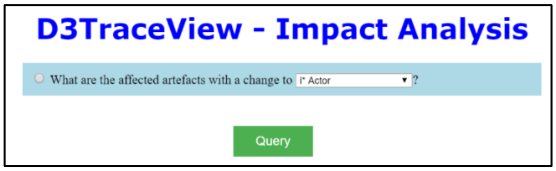
Figure 6. Maintenance and Evolution



○ Is every i* actors ▾ overlaps ▾ i* actors ▾ ?

Figure 7. Consistency Checking

Table 2 shows, for each pre-defined query, the types of visualization formats that the tool suggests. In Table 2, the identifiers of the queries follow the order in which the queries appear in the respective Figures 4 to 8. The types of visualization formats associated with the various queries were defined based on a comparative study presented in [12] and [17], and the available formats in D3 [1] that are adequate to present relations.



○ What are the affected artefacts with a change to i* Actor ▾ ?

Figure 8. Impact Analysis

In the following we discuss the different types of queries associated with each activity and their respective visualization formats used to present the results of these queries.

TABLE 3. TYPES OF VISUALIZATION FORMATS BY QUERIES

| Query | Type of Visualizations |
|---|---|
| **Assess Results** | |
| Q1 | Bar graph, Gauge |
| **Software Inspection (Verification)** | |
| Q1 | Table, Tree, Radial, Sunburst |
| Q2 | Table, Tree, Radial, Sunburst, Matrix |
| **Maintenance and Evolution (Program Comprehension)** | |
| Q1 to Q11 | Table, Tree, Radial, Sunburst, Matrix |
| **Consistency Checking** | |
| Q1 | Table, Tree, Radial, Sunburst, Matrix |
| **Impact Analysis** | |
| Q1 | Tree, Radial, Sunburst |

**Assess Results.** In the OpenUP process [18], this activity is concerned with assessing the results of an iteration in the development process, and determining the success or failure of this iteration in order to plan a subsequent iteration. For this activity, the queries are concerned with the percentage of implemented artifacts. In this case, the tool presents the results as bar graphs or as gauge display.

Bar graphs are used to display and compare the number or frequency for discrete categories of data. Therefore, bar graphs are useful to present the percentage of certain element types. For example, in the case of a query concerned with the percentage of SD goals that has been implemented in the ATCE system, the SD goals, SD tasks, and system goals are the discrete categories of data. The results of this query can be shown as the number of total elements implemented with respect to the number of total elements, as shown in Figure 9.

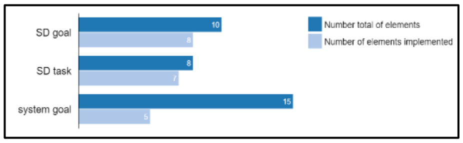


Figure 9. Bar graph view

Gauge displays are used to show a single discrete number representing the percentage. It is a progress bar with a circular, flat design. The gauge marker changes position within this range. Figure 10, shows the single discrete number representing the percentage in a gauge display type of graph.

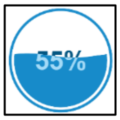
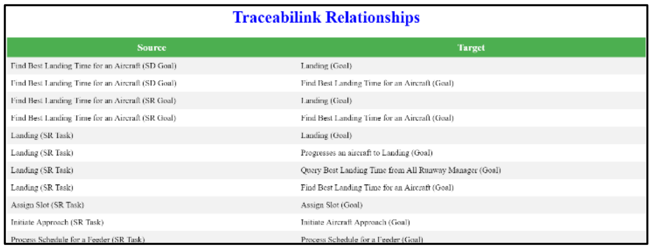


Figure 10. Liquid fill gauge display

**Software inspection (verification).** Software verification is the collection of methods used to determine if a software system is correctly built. For this activity, D3TraceView provides queries that compare artifacts of the system in terms of different relation types (e.g., implemented by, overlaps, achieves).

As shown in Figure 5, one of the queries is concerned with artifacts that are related to a certain type of element (in this case a certain type of actor). Another question is concerned with the case that verifies if a certain artifact is implemented in the system. The types of artifacts can vary depending on the documents (models) used for representing the system (e.g., i*, Prometheus, or JACK elements,); or other elements depending on the system. In these types of queries, the visualization technique shows traceability information involving specific artifacts.

As shown in Table 2, D3TraceView provides four visualization types for the first query namely table, tree, sunburst, and radial views. Tables are typical visualization formats to present simple lists of data (e.g. list of Prometheus goals). The disadvantage of using tables is that they do not show an overview of the dataset and the hierarchical information of traceability relations. Tables should also not be used to display large amount of data. Figure 11 shows a table with elements related to actor Aircraft.



Traceabilink Relationships

| Source | Target |
|---|---|
| Find Best Landing Time for an Aircraft (SD Goal) | Landing (Goal) |
| Find Best Landing Time for an Aircraft (SD Goal) | Find Best Landing Time for an Aircraft (Goal) |
| Find Best Landing Time for an Aircraft (SR Goal) | Landing (Goal) |
| Find Best Landing Time for an Aircraft (SR Goal) | Find Best Landing Time for an Aircraft (Goal) |
| Landing (SR Task) | Landing (Goal) |
| Landing (SR Task) | Progresses an aircraft to Landing (Goal) |
| Landing (SR Task) | Query Best Landing Time from All Runway Manager (Goal) |
| Landing (SR Task) | Find Best Landing Time for an Aircraft (Goal) |
| Assign Slot (SR Task) | Assign Slot (Goal) |
| Initiate Approach (SR Task) | Initiate Aircraft Approach (Goal) |
| Process Schedule for a Feeder (SR Task) | Process Schedule for a Feeder (Goal) |

Figure 11. Table view

Tree views are graphical representations to support hierarchical information display, as shown in Figure 12. In this case, each element can have a number of sub-elements. An element can be expanded to show sub-elements, if any exist, and collapsed to hide sub-elements. Tree views are very intuitive, largely used by software development tools, and allow elements to be shown without the need for scrolling the screen.

Sunburst is another type of graphical representation where nodes are drawn on adjacent rings representing a tree structure, as shown in Figure 13. In this representation, each child of a node with depth n is represented in the ring n + 1 on the same radian space as its parent(s). Sunburst performs better on large amounts of nodes than traditional tree view representations. Tree view representation grows rapidly in the vertical direction if many branches are expanded. In the case of sunburst, the nodes are distributed uniformly in all directions. In sunburst, the color of each item corresponds to an attribute of the item. In D3TraceView, the color in a sunburst view represents an element type (e.g. Prometheus goal), while the width of the representation of each element shows the importance of the element (i.e. number of sub-elements associated with the element). In addition, in the tool, sunburst view is dynamic, in which it is possible to click on an element and zoom information about this element, and it supports a large amount of data.

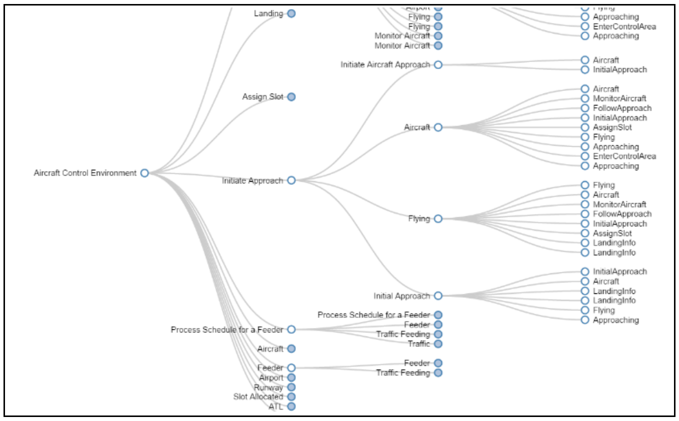


Figure 12. Tree view

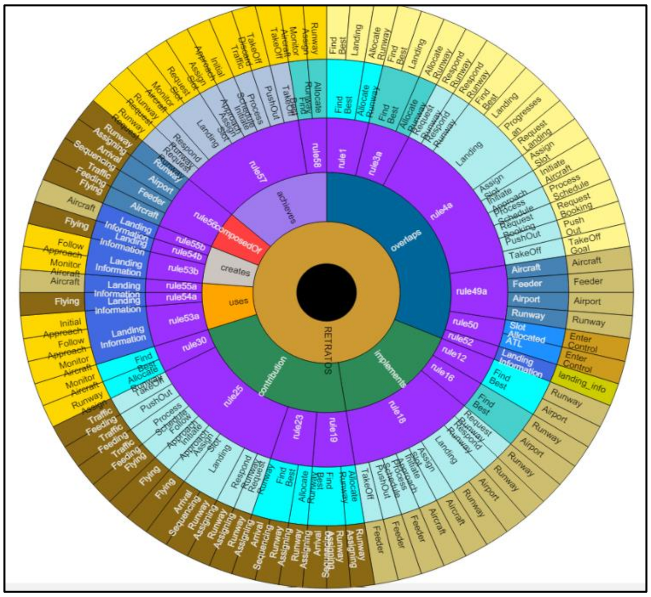


Figure 13. Sunburst view

Radial representation like Reingold-Tilford is a mixture of sunburst and tree views that do not use information of colors and spaces, as shown in Figure 14. For the second type of query, D3TraceView provides five visualization formats, namely list, matrix, sunburst, radial, and tree view. Figure 15 shows the matrix format, while all the other formats have been shown above.

**Maintenance and evolution (Program comprehension).** As shown in Figure 6, the pre-defined queries to support this activity are concerned with gathering a better understanding of the software. More specifically, the queries identify the artifacts in a certain software development phase that are related to artifacts in subsequent phases in the software development process. For example, i* actors (requirements phase) that are related to Prometheus goals (design phase). For these queries D3TraceView suggests five visualization types (see Table 2). The graphs for these visualization types are similar to the ones shown above.

**Consistency Checking.** Based on Figure 7, the queries to support this activity are concerned with the visualization of different types of relationships between various types of artifacts. Examples of these relationship types are: overlaps, depends, contributes, uses, achieves, creates, and composes. Similar to the above activities and queries, the results of these queries can also be visualized in the five different visualization types shown above.

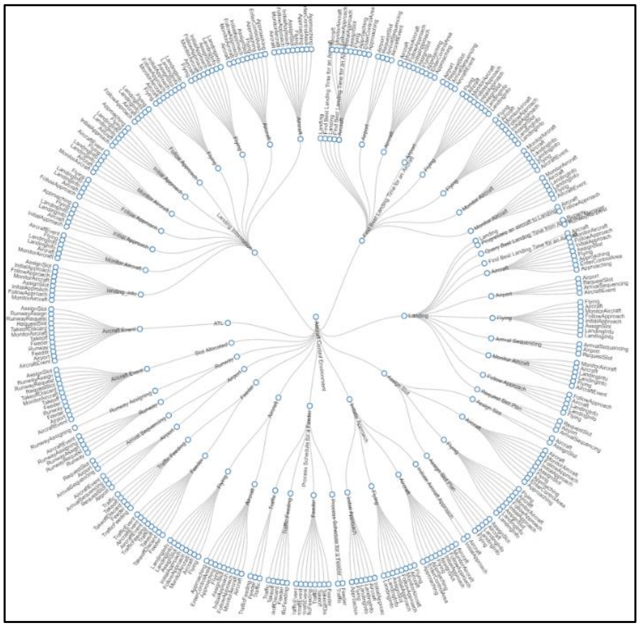
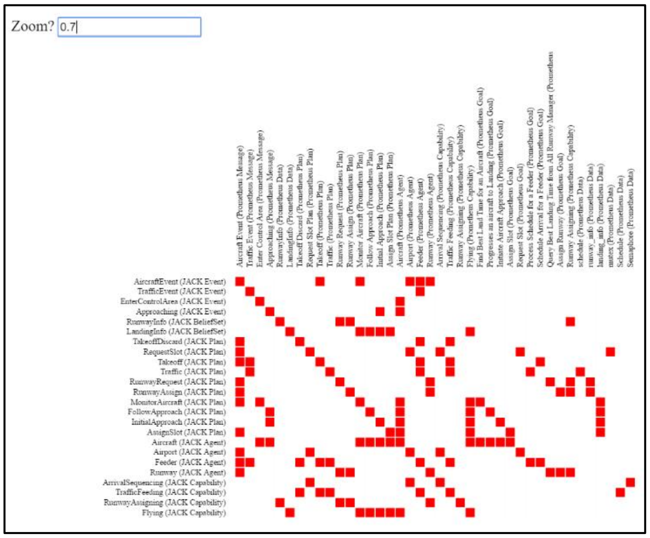


Figure 14. Reingold-Tilford tree view



Zoom? 0.7
AircraftEvent (JACK Event)
TrafficEvent (JACK Event)
EnterControlArea (JACK Event)
Approaching (JACK Event)
RunwayInfo (JACK BeliefSet)
LandingInfo (JACK BeliefSet)
TakeoffDiscard (JACK Plan)
RequestSlot (JACK Plan)
Takeoff (JACK Plan)
Traffic (JACK Plan)
RunwayRequest (JACK Plan)
RunwayAssign (JACK Plan)
MonitorAircraft (JACK Plan)
FollowApproach (JACK Plan)
InitialApproach (JACK Plan)
AssignSlot (JACK Plan)
Aircraft (JACK Agent)
Airport (JACK Agent)
Feeder (JACK Agent)
Runway (JACK Agent)
ArrivalSequencing (JACK Capability)
TrafficFeeding (JACK Capability)
RunwayAssigning (JACK Capability)
Flying (JACK Capability)

*Figure 15. Matrix view*

**Impact Analysis.** In this activity the queries are concerned with identifying the consequences of a change in the software, or identifying the artifacts that need to be changed. The tool suggests four visualization types, as described in Table 2.

## 3. IMPLEMENTATION

D3TraceView has been implemented as a web tool using D3.js library [1]. Figure 16 shows an overview of the main components of the tool. As shown in Figure 16, CASE tools represent the various tools that could be used to develop the software models. For our ATCE system, the case tools are Prometheus Design Tool (PDT), TAOM4E (for i* models), and JACK Intelligent Agent. The Traceability Generator represents tools that assist with generation of traceability relations (e.g., RETRATOS [4]). Both the software models and the traceability links are inputs to D3TraceView. The tool requires the links and the models to be represented in XML format, or in JSON format. The Generator component of D3TraceView generates the results associated with the various queries in the different visualization formats available in the D3 library.

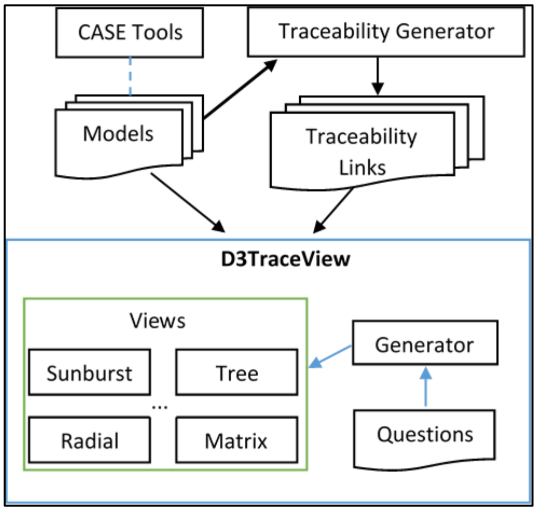


Figure 16. D3TraceView Component Diagram

## 4. RELATED WORK

Information visualization has been recognized as an important area of research that focuses on the use of visualization techniques to help people understand and analyze data [9]. It applies design principles, human perception, and color theory to present data. For several years, traceability information used to be visualized in terms of matrices and tables. However, a matrix is a two-dimensional view of a multidimensional information space (several software artifacts, with different levels of granularity, and various traceability relations connecting the artifacts). Moreover, when using a matrix it is difficult to navigate through the relationships. Another significant limitation is the lack of scalability [12]. In certain projects where the number of relationships is large, it becomes more difficult to visualize specific relationships. More recently, some other works have been developed to support information visualization of traceability information [16] [21] [10][20][2] [14] [15].

Marcus et al. [16] developed a prototype tool called TraceViz that is integrated with Eclipse IDE. TraceViz has three main areas to display: elements (source and target) in a hierarchical way; traceability relationships for a chosen element; and information of the properties and browsing history of a particular traceability relationship. It does not support traditional visualizations techniques such as matrix, tree, and radial. The technique in [21] is based on a graph-based representation with rings showing project artifacts and nodes representing relationships. The various types of relationships are indicated by the use of specific colors.

Similarly, Heim et al. [10] proposed an approach based on graphs representing requirements as nodes and relationships as vertices. However, the use of graph-based approaches to visualize traceability information are less intuitive to display relationships and do not scale well. In the case of large amounts of data it is difficult to understand the view as graphs, in which only a limited set of elements can be displayed. Merten et. al [17] use sunburst and netmap visualization techniques to show traceability relationships. These techniques were implemented as plugins for the Redmine project management tool. Thommazo et al. [20] present an approach to automated generation of requirement traceability matrix. In this work, traceability relationships can also be shown though a graph-based visualization. Chen et al. [2] present an approach that combines treemap and hierarchical tree visualization techniques to provide a global structure of traces and a detailed overview of each trace.

The above approaches are limited to support one or two types of visualization formats. D3TraceView supports several visualization formats and it has been developed in a way that it can easily accept other types of visualization formats. Moreover, our tool also allows for querying traceability data for different activities.

Several works have been suggested to support trace queries. A visual trace modeling language (VTML) was proposed in [14] that allows users to create queries using UML class diagrams and constraints. Maletic and Collard [15] propose a Trace Query Language (TQL) which can be used to write trace queries as XPath expressions for artifacts represented in XML format. The creation of queries by users is not an easy task. D3TraceView provides pre-defined questions, which can be extended in the tool.

## 5. CONCLUSION AND FUTURE WORK

In this paper, we presented a traceability visualization tool called D3TraceView that supports various visualization formats such as matrix, list, tree, radial, gauge, sunburst, table, and bar view. The tool is built using D3.js library and applies information visualization principles to present traceability relations. Currently, we are extending the tool with an editor to support the specification of queries. We are also evaluating the tool in terms of its usability and support for the various software development activities in the tool.

## REFERENCES

[1] Bostock, M., Ogievetsky, V. and Heer, J. 2011. D3: Data-Driven Documents. In *Proceedings of the IEEE Trans. Visualization & Comp. Graphics*.

[2] Chen, X., Hosking, J. and Grundy, J. 2012. Visualizing traceability links between source code and documentation. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*.

[3] Cleland-Huang, J., Gotel, O., Hayes, J., Mäder, P. and Zisman, A. Software traceability: trends and future directions. *Proc. Future of Software Engineering*, 2014.

[4] Cysneiros, G. and Zisman, A. 2008. Traceability and completeness checking for agent-oriented systems. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing*.

[5] Cysneiros, G. and Lencastre, M.. Towards a Traceability Visualisation Tool. 2012. In *Proceedings of the 2012 Eighth International Conference on the Quality of Information and Communications Technology* (QUATIC '12).

[6] Gotel, O. and Finkelstein, A. 1994. An Analysis of the Requirements Traceability Problem. 1994. In *Proceedings of the First IEEE International Conference on Requirements Engineering* (ICRE'94).

[7] Gotel, O., Cleland-Huang, J., Zisman, A., Hayes, J., Dekhtyar, A., Mäder, P., Egyed, A., Grünbacher, P., Antoniol, G. and Maletic, J. 2012. Glossary of Traceability Terms (1.0). In Cleland-Huang, J., Gotel, O. and Zisman, A. editors, *Software and Systems Traceability*, Springer.

[8] Gotel, O., Cleland-Huang, J., Hayes, J., Zisman, A., Egyed, A., Grünbacher, P., Antoniol. G. 2012. The quest for Ubiquity: A roadmap for software and systems traceability research. In *Proceedings of the 2012 20th IEEE International Requirements Engineering Conference (RE)*.

[9] Heer, J., Bostock, M., and Ogievetsky, V. A tour through the visualization zoo. *Commun. ACM* 53, 6 (June 2010), 59-67.

[10] Heim, P., Lohmann, S., Lauenroth, K. and Ziegler, J. 2008. Graph based visualization of requirements relationships. In *Proceedings of the 2008 Requirements Engineering Visualization*.

[11] Howden, N., Rönnquist, R., Hodgson, A. and Lucas, A. 2001. JACK intelligent agents - Summary of an agent infrastructure. In *Proceedings of the 5th International Conference on Autonomous Agents*.

[12] Li, Y. and Maalej, W. 2012. Which traceability visualization is suitable in this context? a comparative study. In *Proceedings of the 18th International Conference on Requirements Engineering Foundation for Software Quality*.

[13] Mäder, P. 2013. Interactive Traceability Querying and Visualiation for Coping with Development Complexity. In *CoRR*.

[14] Mäder, P. and Cleland-Huang, J. 2013. A visual language for modeling and executing traceability queries. *Softw. Syst. Model.* 12, 3 (July 2013), 537-553.

[15] Maletic, J. and M. Collard, L. 2009. Tql: A query language to support traceability. In *Proceedings of the 5th Workshop on Traceability in Emerging Forms of Software Engineering*.

[16] Marcus, A., Xie, X. and Poshyvanyk, D. 2005. When and how to visualize traceability links? In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*.

[17] Merten, T., Jueppner, D. and Delater, A. 2011. Improved Representation of Traceability Links in Requirements Engineering Knowledge using Sunburst and Netmap Visualizations. In *Proceedings of the 4th International Workshop on Managing Requirements Knowledge*.

[18] OpenUP. http://epf.eclipse.org/wikis/openup/

[19] Padgham, L. and Winikoff, W. 2004. *Developing Intelligent Agent Systems–A Practical Guide*, John Wiley & Sons.

[20] Thommazo, A., Malimpensa, G., Oliveira, T., Olivatto, G. and Fabbri, S. 2012. Requirements Traceability Matrix: Automatic Generation and Visualization. In *Proceedings of the 26th Brazilian Symposium on Software Engineering*.

[21] Voytek, J and Núnez, J. 2011. Visualizing Non-Functional Traces in Student Projects in Information System and Service Design. In *Proceedings of the Intl. Conf. Human Factors in Comp. Systems*.

[22] Winkler, S. 2008 On Usability in Requirements Trace Visualizations, In *Proceedings of the 2008 Requirements Engineering Visualization*.

[23] Yu, E. 1995. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.