Visualization of business process architectures

Milde, T.

Award date:
2013

Eindhoven, September 2013

Visualization of Business Process Architectures

by

Thomas Milde

Final thesis for the Master of Science program in Business Information Systems
as defended on Friday 27th September 2013
by Thomas Milde, BSc, student identity number 0787581

Information Systems group
Department of Industrial Engineering and Innovation Sciences
Eindhoven University of Technology

Assessment committee:

Dr. ir. R.M. Dijkman	Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology
Ir. R.R.H.M.J. Goverde	Precedence BV
Prof. dr. ir. H.A. Reijers	Department of Mathematics and Computer Science, Eindhoven University of Technology

Abstract

As business process management spreads across most big enterprises and other organizations, they build process collections of hundreds of models. Managing these process collections as a whole poses a major challenge to those organizations. Business process architectures are an approach to structuring and understanding process collections. This report proposes visualization as a means of communicating about process architectures. Similar developments in literature about software architecture research are analyzed. Based on literature and input from practitioners, use cases and requirements for business process architecture visualization techniques have been identified. A prototypical visualization tool that serves a selection of these use cases has been implemented. Starting points for further research in the field of business process architecture visualization are pointed out.

Preface

This thesis is the result of my graduation project for the Master's program in business information systems at Eindhoven University of Technology (TU/e). The project was carried out at the information systems group at the department of industrial engineering and innovation sciences of TU/e. Input from a practical perspective was provided by Precedence B.V., Maastricht.

I would like to thank everyone who contributed to the project. First of all, I want to thank my first supervisor Remco Dijkman who suggested the topic for this project and supported me from its start until the finalization of this thesis. Furthermore, I want to thank Roy Goverde and Mark Cloesmeijer from Precedence for their valuable input and Roy Goverde for participating in the assessment committee. Thirdly, I would like to thank Hajo Reijers for his feedback on the draft of this thesis and his participation in the assessment committee.

Last, but not least, I would also like to thank my friends and family who encouraged me and gave me personal support when I needed it.

Thomas Milde
Eindhoven, September 2013

Contents

List of Figures

1 Introduction

In business process management, processes of a company are created, documented, improved and analyzed with the goal of improving efficiency, quality and other measures. A big number of tools allows modeling and analyzing individual process models and managing collections of these models. The process collection of a big organization can easily contain models for hundreds of business processes. For example, the SAP reference model which has been described in [3] contains more than 600 process models that may be relevant for running an enterprise. While each process itself may be well-documented and can be analyzed with good tool support, the relations between the business processes are often not well, if at all, documented and understood. Even a clear structure in the process collection may be missing. Research into business process architecture (BPA) tries to cope with that problem. In this thesis, techniques for the visualization of business process architectures are explored.

1.1 Business process architectures

Typical enterprises have a big number of processes involved in their primary as well as in their supporting activities. For stakeholders, it is important to keep an overview of these processes in order to fully understand the working of the enterprise. However, without a structure, it is hardly possible to keep this overview. Business process architectures try to address this problem [6] by introducing explicit relations between processes.

Most of the literature on business process architectures does not provide a clear definition of that term. Eid-Sabbagh et al. [6] give a formal mathematical definition of a process architecture, but also state that "a business process architecture is a collection of business processes and their interdependencies with each other". While the formal definition is suitable for formal analyses, it does not seem appropriate for this work, because a more intuitive definition of what comprises a process architecture is needed when developing a practical application. The latter definition is more intuitive, but lacks some important aspects. It does not mention that the processes are all part of the same organization. Organization may refer to a company, government entity or non-governmental organization as well as to a sub-unit or collaboration (e.g. of supply chain partners) of such entities. When applied correctly, a business process architecture contains all the processes of the organization in question. If it does not, then important dependencies may be missed and therefore the reliability of any conclusions derived from the process architecture is doubtful. The dependencies between processes are classified by [6] to be compositions (process to subprocess relationship), specializations, triggers or information flows. Among these dependencies, the composition can be seen as a particularly important one because it gives a hierarchical structure to the process architecture. However, not necessarily the entire hierarchical structure of a process architecture is represented by composition relationships between processes. There can also be groups of processes that are logically grouped although they are not the subprocesses of one common parent. These logical groupings need to be added to the definition. Considering these aspects, a business process architecture can be defined as follows:*A business process architecture is a structured collection of the business processes of an organization. It consists of the business processes, their relations and a hierarchical structure of processes and groups of processes.*

1.2 Motivation for the visualization of business process architectures

As mentioned before, a process architecture can consist of hundreds of processes. The processes have many relations among each other and understanding the whole process architecture may be a difficult task. In many fields, graphical representations are used in order to make complex information more understandable. For example, mechanical engineers use drawings and 3D-animations to design complex systems and communicate about them. Building architects and software engineers use similar tools.

Graphical notations that show all details of an architecture, like the one used in [6] can help to analyze the architecture, but may be too technical to get an overall insight into the working of the process. A visualization should help a business user to get an overall idea about the structure of the process collection. However, it should also provide a possibility to get more detailed information about the process architecture.

1.3 Goals

As stated in the previous section, a visualization is supposed to help the user understand the structure of and relations in the process architecture. Therefore, the main goal of this research is the *development of an easy-to-use and useful graphical visualization of process architectures.* The term *useful* refers to the requirement that fulfilling a certain task should be supported by the tool. The tasks are defined as use cases in section 3.1.

To visualize a process architecture, the relations between processes need to be known. Hence, a sub-goal is the *identification of relations between processes on the same hierarchical level or on different hierarchical levels.* The reference hierarchical levels emphasizes that there may be relations between any two processes in the process architecture. Furthermore, it was mentioned that the main intention behind the visualization is to ease the

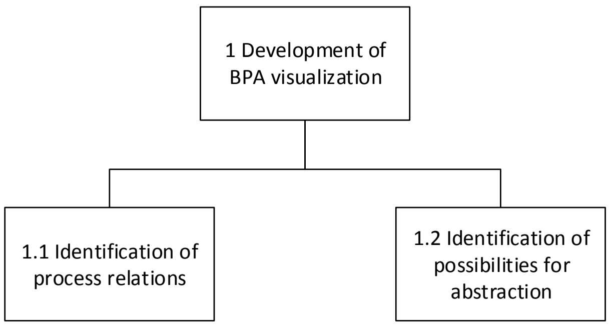

1 Development of BPA visualization
1.1 Identification of process relations
1.2 Identification of possibilities for abstraction

Figure 1.1: Goal tree of the research. The goals are stated in a shortened form.

understanding of a process architecture. To achieve this, there has to be an abstraction from details because a user cannot understand hundreds of objects and connections at the same time. Hence, another sub-goal is the *identification of possibilities to abstract from the details of a process architecture in order to obtain multiple levels of detail.* The goals can be seen in figure 1.1.

1.4 Research approach

The research approach is described by figure 1.2. The first step of the research was a literature study. The goal of the literature study was twofold: First, concepts about business process architectures (BPAs) needed to be identified, especially the relations that exist in process architectures. Secondly, visualization techniques had to be found. As visualization of BPAs is a new field, there is no literature about that exact topic. Literature from a neighboring field was consulted to compensate for that. The visualization techniques found were evaluated as a basis for drafting requirements for visualization techniques for BPAs.

In the second step, use cases for a business process architecture visualization tool and possible approaches to solving them have been identified. Using findings from the literature study, a number of use cases was identified and discussed with practitioners who regularly work on business process architectures. Their input was used to decide on the final use cases. Subsequently, a number of visualization techniques that may help to solve these use cases have been selected.

As a third step, a prototype was implemented that applies the findings from the first two steps and solves a selection of the use cases identified before.

The prototype was presented to the practitioners who initially gave input about the use cases and their feedback was collected. Furthermore, a test plan for a user test has been developed. Carrying out and evaluating the user test is left as future work.

1.5 Thesis structure

The rest of this report is structured as follows: Chapter 2 describes the literature study done in step one of the research. Chapter 3 describes the use cases that were found to be relevant and the requirements for visualization techniques addressing these use cases. Chapter 4 documents the architecture and implementation of the prototype. Chapter 5 describes the test plan for a large-scale test and the steps done as a preliminary validation. Chapter 6 presents results, limitations and suggests future work.

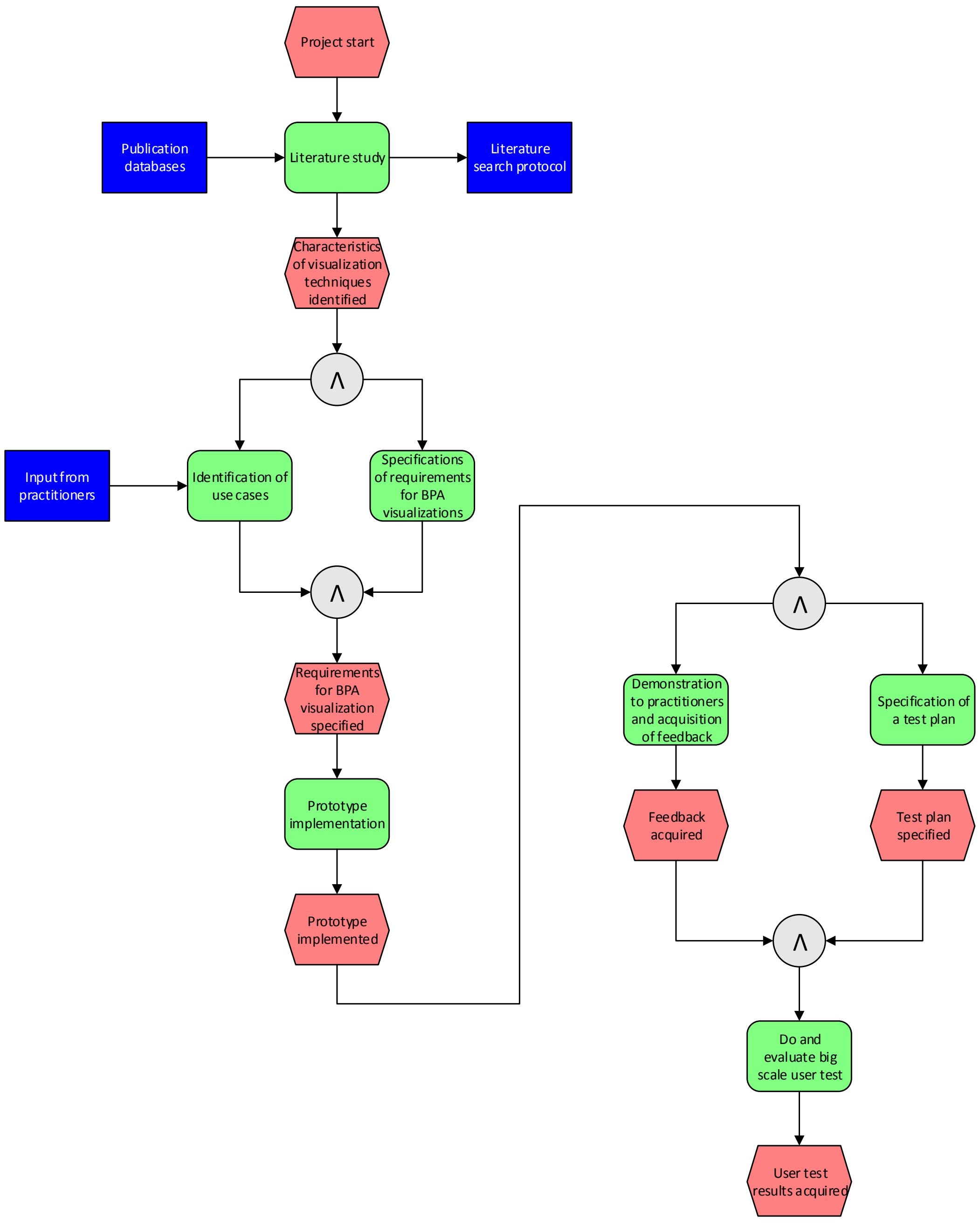

Project start
Publication databases
Literature study
Literature search protocol
Characteristics of visualization techniques identified
∧
Input from practitioners
Identification of use cases
Specifications of requirements for BPA visualizations
∧
Requirements for BPA visualization specified
Prototype implementation
Prototype implemented
∧
Demonstration to practitioners and acquisition of feedback
Specification of a test plan
Feedback acquired
Test plan specified
∧
Do and evaluate big scale user test
User test results acquired

Figure 1.2: Process diagram of the project

2 Visualization techniques for software architectures

This chapter presents a literature review about visualization techniques used in the related field of software architecture visualization. There is no published work yet in the field of business process architecture visualization and software architectures have parallels to BPA's which make it possible to draw conclusions from software- for business process architecture visualization. These parallels and the domain of software architecture visualization will be briefly introduced in the first part of the chapter. Subsequently, the methodology of the literature review is presented. After introducing the evaluation framework used for the literature study, the results are presented.

2.1 Software architecture visualization

A software architecture describes the structure of a software system on a level that is more abstract than source code. It can have any level of abstraction ranging from a high level view with a few modules representing big functional components down to a very detailed view that is close to the actual implementation. Just like a BPA, a software architecture can be hierarchical, but also contains other types of relations between modules. Big software systems have a complicated structure that yields problems for understandability. Visualization is used here as an aid to assist the software architect in understanding the structure of a system. As software architecture is a much more established field than business process architecture, much research has already been done into the visualization of software architectures. Visualization techniques can help in the general understanding of a software system's structure, but also in the analysis of concrete properties.

2.2 Methodology

In order to obtain literature for this study, the search engine Google Scholar has been used as well as Springer Link, the catalog of the library of TU Eindhoven and IEEE Xplore. The search term *Software architecture visualization* has been used in each of these sources. Papers were considered relevant if they present a visualization technique for software architectures or compare such visualization techniques. They were considered irrelevant if they were not about software architecture visualization or if they report only on implementation details rather than the actual visualization technique. After scanning the abstracts of the search results, 9 publications have been considered relevant. A full protocol of the literature search can be found in appendix A. The relevant publications have been analyzed using the framework presented in the next section.

The framework is a modified version of a general framework for data visualization introduced by Shneiderman[20]. The framework has been modified to fit the particular goals of this study: irrelevant aspects have been removed and additional ones have been added. An aspect is considered relevant if an implementation of this aspect from some domain is likely to provide an added value for the domain of business process architectures. Three criteria have been added to the general framework. Two of them are domain specific. They are considered relevant because the domains of software architecture visualization and BPA visualization have similarities that make the reuse of concrete elements of visualization techniques possible. A third element was added as an indicator of the quality of the visualization technique. The specific elements will be explained in section 2.3.

2.3 Framework for visualization techniques

In order to analyze and compare different types of software architecture visualization, criteria are needed for identifying the most important aspects of each visualization technique. The goal of this literature review is surveying the properties of software architecture visualizations in order to get a basis for the development of requirements for business process architecture visualizations. This means that the framework does not need criteria that facilitate a precise evaluation of every technique, but criteria that give an insight into the general ideas of the techniques. The framework used here is a modified version of the one presented in [20]. It consists of the aspects described below and summarized in table 2.1. The framework will be used further throughout this thesis the identify properties of visualization techniques.

2.3.1 Metaphor

A software architecture or business process architecture is an abstract concept which cannot be seen directly. Some primitives have to be found that graphically represent the elements of a software architecture. In some visualization techniques, a metaphor is used to ease the understanding of the abstract concepts. A metaphor is a comparison between the concepts of a software architecture and a concept in the real world.

It is interesting for this study to know about the metaphor used because a business process architecture is just as abstract as a software architecture.

Name	Question	Source
Metaphor	Does the visualization use a metaphor and, if so, which?	Domain-specific addition
Primary artifacts	Which primitives are used for visualizing the primary artifacts?	Domain-specific addition
Overview	How is a general overview over the data provided?	[20]
Zoom	Which possibilities are provided for zooming in and out in order to select the level of detail or overview?	[20]
Filter	Which possibilities and/or techniques are offered for selecting the displayed items other than by selecting the level of abstraction?	[20]
Details on demand	How is detailed information about items provided and which information is provided on demand?	[20]
History	Are functions implemented to undo actions done in the visualization? If so, how does it work?	[20]
Extract	Can the state of the visualization be exported in order to use it in other tools? If so, what exactly can be exported with which goal?	[20]
Evaluation	How have the authors evaluated their visualization technique?	Non-domain-specific addition

Table 2.1: Summary of the evaluation framework

2.3.2 Visualization of primary artifacts

The primary artifacts of a business process architecture are a (hierarchical) set of processes and the relations between those processes. In the domain of software architectures, these are a (hierarchical) set of modules/packages and their relations. Because these artifacts have no physical form, the way they are visualized can vary greatly. This means that the way how primary artifacts are visualized, is an important aspect in the design of a visualization technique for any of both domains. As solutions from the software architecture domain can likely be reused for business process architectures, the visualization of primary artifacts is a valuable aspect.

2.3.3 Overview

Understanding the overall structure of the architecture is one of the main concerns in as well software architecture visualization as BPA visualization. Hence, a visualization technique needs to provide an overview of the general structure of the architecture. That means that the user has the opportunity to see the big picture while details are omitted and may become visible following certain user interaction.

2.3.4 Zoom

Apart from understanding the big picture, users typically also want to see more details on parts of what they are seeing or they want to see a part more clearly. Therefore, providing the possibility of zooming in to areas of interest and out to see the bigger picture, can be a valuable addition to the user's experience.

2.3.5 Filter

If many items are displayed at the same time, the user may be distracted from the information they are actually interested in. Therefore, different ways of filtering can be provided to limit the amount of information that is being displayed. This criterion captures what possibilities for filtering exist in a particular visualization technique and how they are used.

2.3.6 Details on demand

While filtering serves to reduce the amount of information that is being displayed, details on demand means providing specific information when the user requests it. One visualization technique can provide details on demand in several ways. As specified by the name part "on demand", the provision of details always happens upon a specific user interaction.

2.3.7 History

When the user changes the settings of a visualization technique, they might want to undo and redo a recent change. If the change is not trivial to make, a history function, i.e. undo and redo provided by the software, may be valuable.

2.3.8 Extract

If a visualization technique has many settings or complicated interaction mechanisms, returning to a certain situation may require a significant effort. Therefore, a user may want to preserve the state of a visualization. Whether tools for software architecture visualization include such a functionality may influence the decision for whether or not to include it in a tool for BPA visualization.

2.3.9 Evaluation

Some publications only introduce a new technique, others compare multiple existing techniques or empirically evaluate them. This forms the third aspect of the framework: How is the effectiveness of the visualization technique evaluated? This aspect can give an idea of the effectiveness of the visualization technique. However, its importance in this study should not be overestimated. Because of the different target groups and different semantics of business process architectures, the effectiveness of a visualization technique applied to BPAs can vary significantly from the effectiveness of the same technique applied to software architectures.

2.4 Results

This section presents the results of applying the framework to each relevant publication.

3D visualization of software architectures[7]

Feijs and De Jong present a 3D visualization for software architectures based on usage relations between different components. Figure 2.1 shows an example of their visualization technique. Lego bricks represent modules. They are grouped according to the package they belong to. Lines between modules represent usage relations. The bricks are grouped in layers according to freely selectable criteria.

Metaphor: Different styles of Lego bricks are used to indicate different types of components

Primary artifacts: Components are shown as Lego bricks, connected with directed arrows (usage relations). Hierarchy is not directly visualized.

Overview: Initially, all modules and their relations are shown, arranged in multiple layers, as can be seen in figure 2.1

Zoom: The user can zoom in in order to see more details about a set of modules.

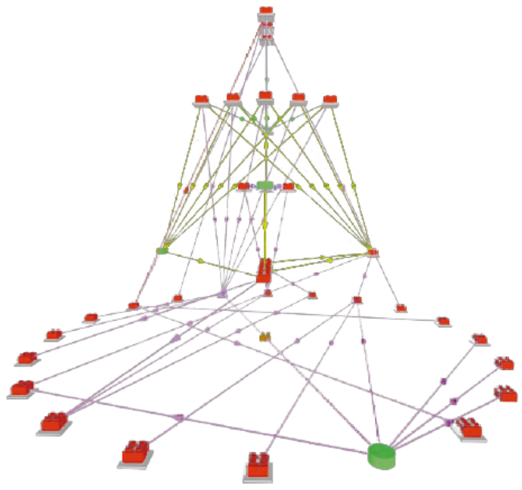


Figure 2.1: Overview in Feijs and De Jong's visualization, image from [7]

Filter: No additional possibilities for filtering are provided.

Details on demand: The authors mention only that the name of a module can be displayed on demand.

History: No such functionality is mentioned in the publication.

Extract: No such functionality is mentioned in the publication.

Evaluation: No evaluation is presented by Feijs and De Jong.

SHriMP views: an interactive environment for exploring Java programs[21]

Storey et al. present a visualization specifically for Java programs. They include multiple views and allow browsing the architecture. An example of their visualization can be seen in figure 2.2. In the picture, a class diagram can be seen on the top-left. Next to it is source code of a selected class and below it the Javadoc documentation for that class can be seen. The bottom-right corner shows a package structure.

Metaphor: No metaphor is used

Primary artifacts: Classes and packages are shown as rectangles, relations as lines between those rectangles.

Overview: As an overview, the top-level class and package structure of the software is shown.

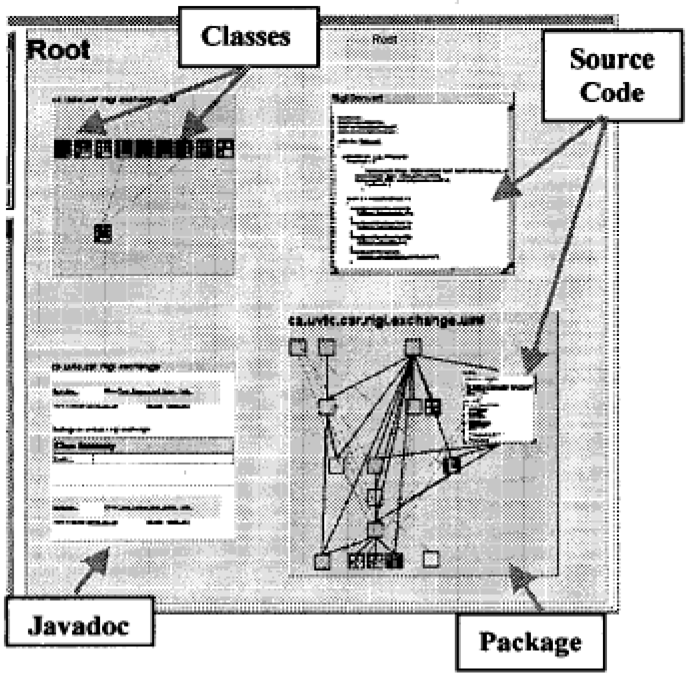


Figure 2.2: SHriMP views, image from [21]

Zoom: There are multiple types of zoom: Geometric zoom shows only a particular part of the whole architecture. Fisheye zoom emphasizes part of the architecture, but preserves the context information. The authors also mention semantic zoom, but this can better be classified as details on demand.

Filter: No possibilities for filtering are mentioned.

Details on demand: Zooming in with the "'semantic zoom"' provides further details in different views. For example, it can display documentation, package contents, or source code.

History: The authors do not mention history functions.

Extract: The authors do not mention any extraction functions.

Evaluation: No evaluation is presented by the authors.

EvoSpaces Visualization Tool: Exploring Software Architecture in 3D[1]

Alam and Dugerdil present a 3-dimensional visualization as a software city. Figure 2.3 shows an excerpt from that visualization. It shows several districts representing packages and buildings representing modules.

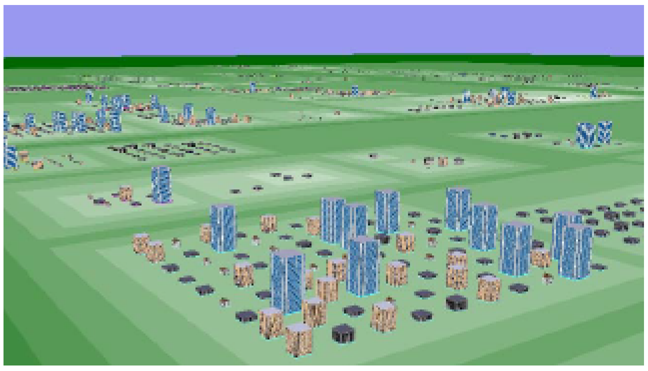


Figure 2.3: EvoSpaces, image from [1]

Metaphor: The architecture is visualized as a city consisting of districts with buildings and workers.

Primary artifacts: Modules are represented by buildings, packages by districts and methods as workers inside the buildings. Pipes between buildings represent relationships between modules.

Overview: A view of the whole city shows the structure of the software architecture. It omits the pipes (relations). The overview can be seen in figure 2.3.

Zoom: The user can zoom in to individual buildings in order to get information about their inner workings.

Filter: Dynamic information can be displayed by highlighting only those buildings that are involved in a particular execution trail.

Details on demand: When the user selects a module (building), the relations (pipes) of this module will be shown.

History: No history functions are mentioned.

Extract: No extraction functionality is documented.

Evaluation: No evaluation is presented.

Visualization-based analysis of quality for large-scale software systems[15]

Langelier et al. present a visualization technique that is aimed at analyzing particular quality criteria of a software architecture. Two examples of this visualization are shown in figure 2.4. The images show modules arranged by packages. Color, size and orientation of the modules indicate three quality criteria. Red and high buildings as well as buildings turned out of the direction of view indicate problems.

Metaphor: No metaphor is used. The design elements are justified by intuitive understandability.

Primary artifacts: Software modules are represented by three-dimensional blocks. These blocks vary in size, color and orientation. These three variables are controlled by one quality criterion each. Two different layouts (Treemap and Sunburst) are proposed for arranging the blocks. The package structure is shown by grouping of the blocks. Relations among modules are not directly shown.

Overview: The overview shows all modules grouped by packages.

Zoom: Geometric zoom is provided. The user can navigate in the three-dimensional world.

Filter: Two types of filters exist. The first one colors modules in red which have extreme values for particular metrics. The other one emphasizes all modules that have a particular relation with a selected module. The emphasis is achieved by removing the color from all other modules. This still preserves the context information.

Details on demand: No functionality for details on demand is provided, other than the possibilities to filter.

History: History functions are not documented in the publication.

Extract: No extraction functionality is mentioned in the publication.

Evaluation: The visualization techniques are evaluated in a user test, especially focusing on the layout techniques. The visualization proved useful in answering complex analysis tasks. It also became evident that the layout plays a major role.

Towards empirically validated software architecture visualization[14]

Knodel et al. describe an approach for the validation of software architecture visualization techniques and apply it to their own visualization tool SAVE. An example of the SAVE visualization is shown in figure 2.5. It shows the top-level package structure of a software.

Metaphor: No metaphor is used.

Primary artifacts: Components are shown as (nested) boxes and relations between the components are shown as arrows between those boxes.

Overview: As an overview, the tool provides a high level overview of the system, i.e. it shows the top-level modules and their relations.

Zoom: Geometric zooming capabilities are not documented.

Filter: Filtering functionality is not described by the authors.

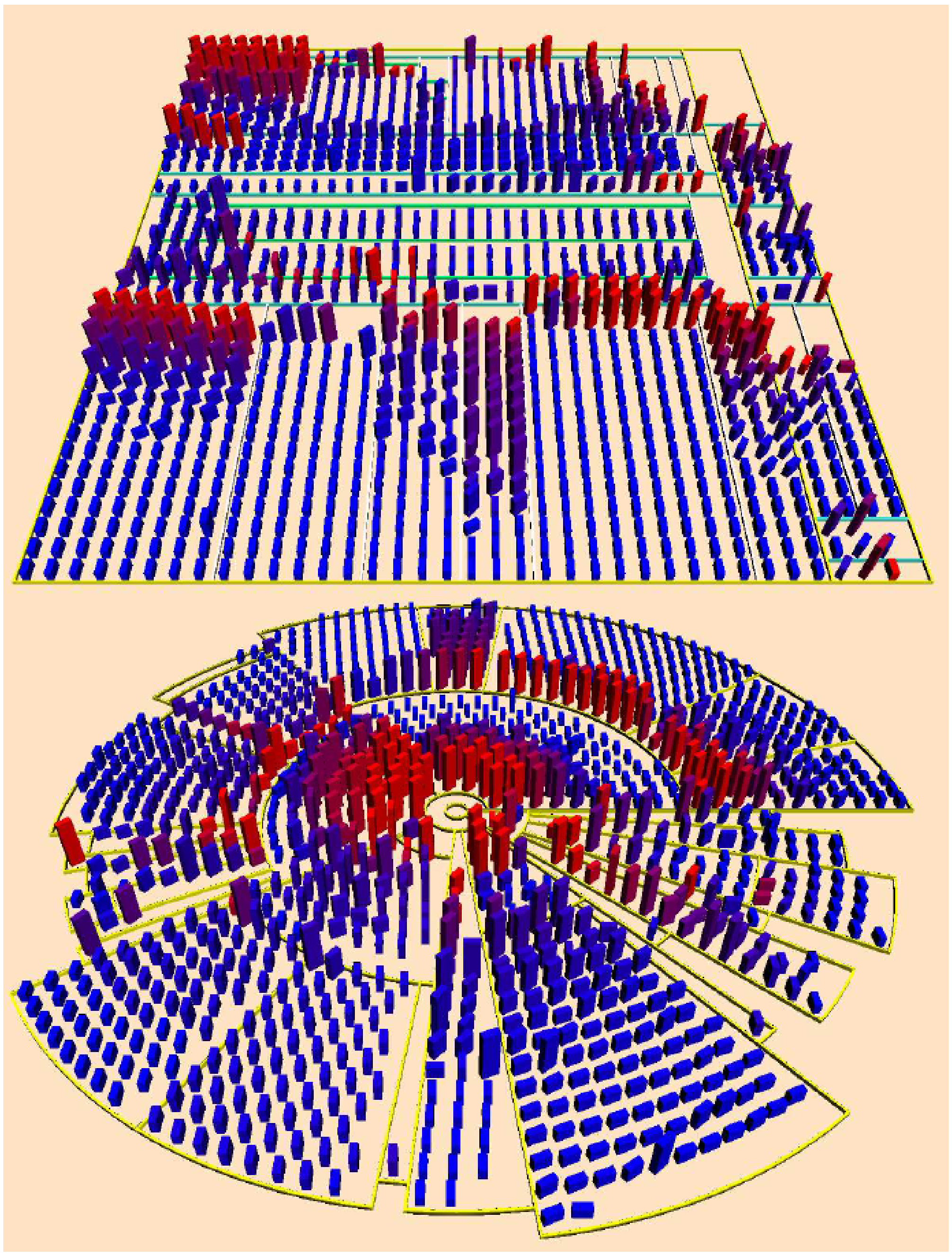


Figure 2.4: Visualization for analysis by Langelier et al., image from [15]

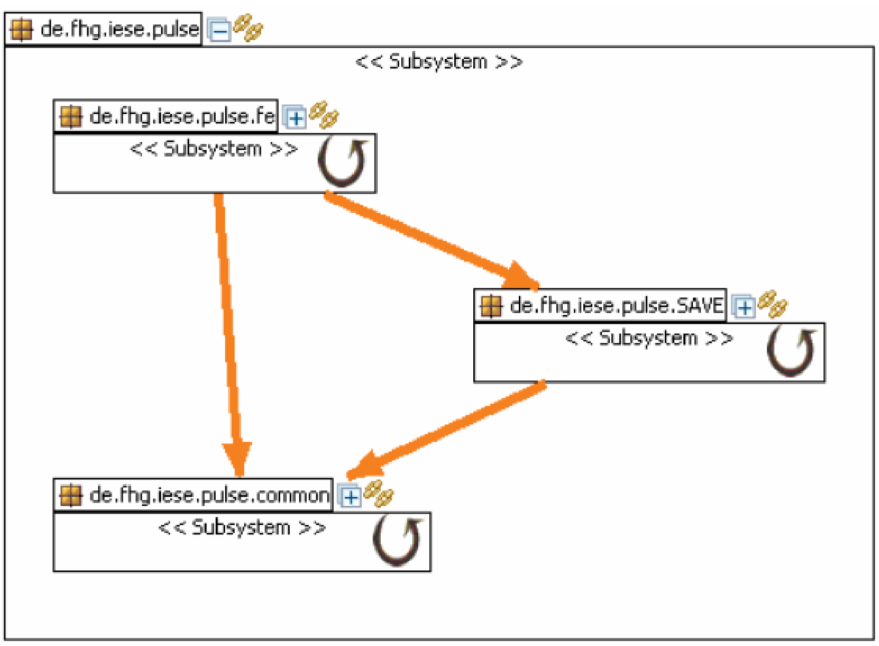


Figure 2.5: SAVE visualization, image from [14]

Details on demand: Packages can be opened to show their inner structure ("semantic zoom"). Symbols on the edges can be added in order to indicate analysis results.

History: No history functionality is documented.

Extract: No extraction functionality is mentioned in the publication.

Evaluation: Evaluation was done in several industrial projects as well as in a controlled environment. The visualization elements were iteratively refined based on the outcome of the evaluation.

Communicating Software Architecture using a Unified Single-View Visualization[18]

Panas et al. develop a visualization system that aims at unifying multiple stakeholder-specific views used in other tools. An example of the visualization can be seen in figure 2.6. It shows cities representing files, buildings representing functions and lines between cities and buildings representing relations between files and functions.

Metaphor: The software is visualized as a landscape consisting of multiple cities.

Primary artifacts: Buildings represent functions, cities represent files (modules). Different types of relations can be added as edges between the cities.

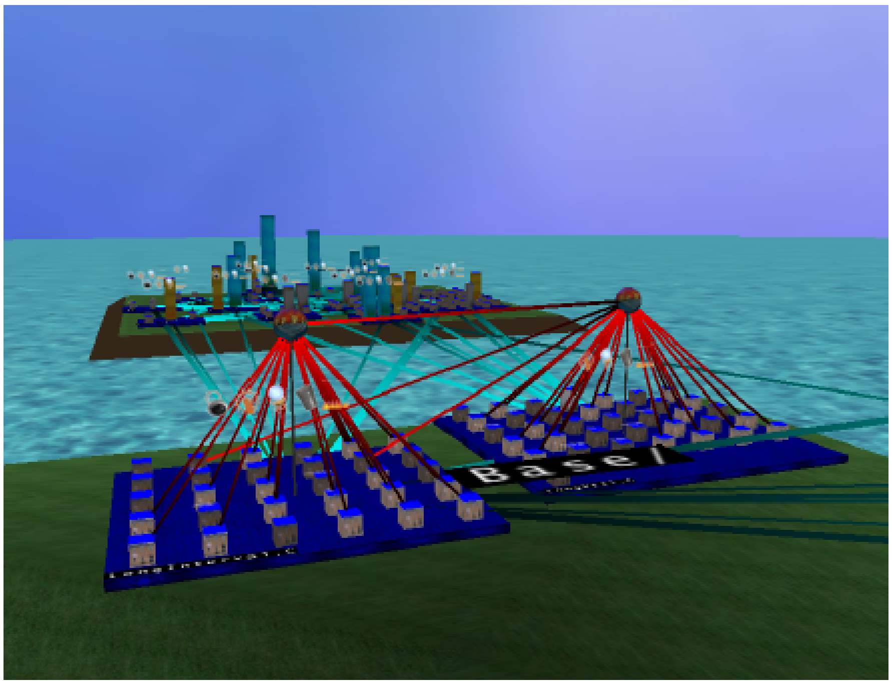


Figure 2.6: Software architecture visualization by Panas et al., image from [18]

Overview: As an overview, the complete landscape is shown. It can be seen in figure 2.6.

Zoom: Geometric zooming is available in the visualization tool.

Filter: No options for filtering are documented.

Details on demand: No details on demand are provided.

History: No history functions are documented.

Extract: No extract functions are documented.

Evaluation: No empirical evaluation is presented.

Towards pie tree visualization of graphs and large software architectures[19]

Samia and Leuschel present a so-called pie tree visualization with the aim of providing an exact visualization that is more comprehensible than a graph-based one. The emphasis is on visualizing the connections between individual modules. Figure 2.7 shows an example of the visualization. It shows a number of modules, each indicated by their name and pie charts that show its connections. Each pie chart shows either the incoming or the

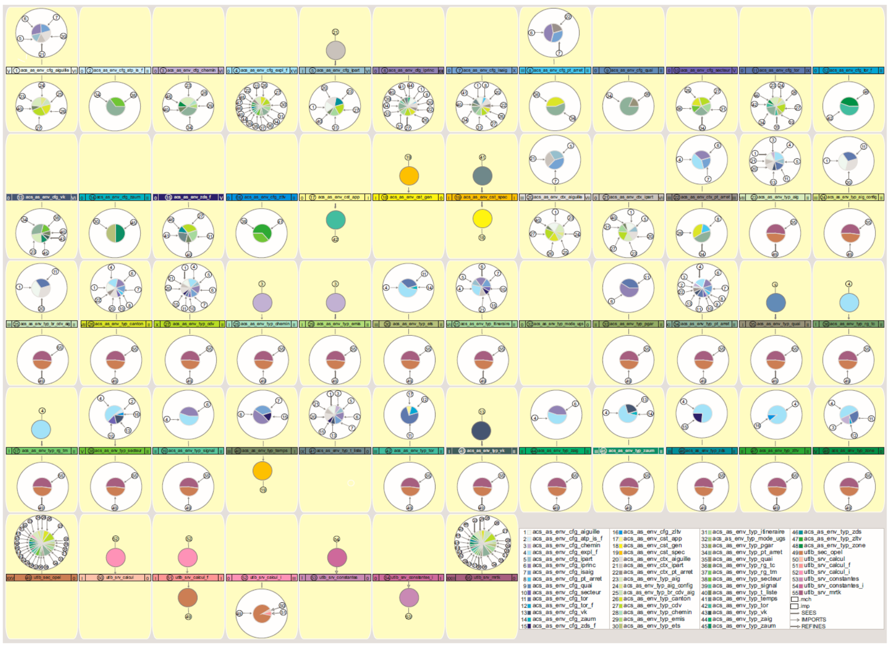


Figure 2.7: Pie chart visualization by Samia et al., image from [19]

outgoing connections of a module. The share of a connection in such a pie chart is based on properties of the connected module.

Metaphor: No metaphor is used.

Primary artifacts: Each module is shown by its name and two pie charts that show which incoming and outgoing relations the module has.

Overview: The visualization has only one view which can be seen in figure 2.7. However, it can not be considered an overview, because it does not give information about the overall structure that can be easily understood.

Zoom, Filter, Details on demand,History, Extract: Not applicable because the visualization provides only a single view.

Evaluation: No empirical evaluation is presented.

Navigating software architectures with constant visual complexity[16]

Li et al. describe techniques to keep the visual complexity of a graph-based visualization at a level that allows the user to understand the information provided. Figure 2.8 shows an example of this visualization. It indicates top-level modules as blocks and their relations as lines.

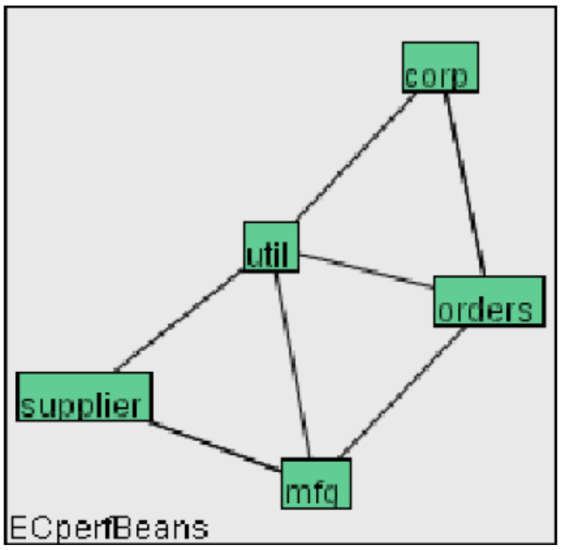


Figure 2.8: Overview in software architecture visualization by Li et al., image from [16]

Metaphor: No metaphor is used.

Primary artifacts: Modules are visualized as rectangles and relations as lines between those.

Overview: An overview is provided in the form of a visualization that shows top-level modules and their relations. It can be seen in figure 2.8.

Zoom: A fisheye view is provided that shows the structure of a module and additionally abstracts the rest of the visualization in order to decrease visual complexity.

Filter: An additional filtering option is not provided.

Details on demand: A semantic zoom is provided which opens up a particular module to show its inner structure.

History: History functionality is not documented.

Extract: Extraction functionality is not documented.

Evaluation: An evaluation of the effectiveness of the algorithms is presented. However, no validation of the practical usefulness for fulfilling particular tasks is presented.

Software architecture visualization: An evaluation framework and its application[11]

In their paper, Gallagher et al. present an evaluation framework and its application to six visualization tools. While [11] itself does not contain sufficient information about the visualization techniques of these tools, it serves as a source of references.

Source	Metaphor	Primary artifacts	Overview	Zoom
[7]	Lego	Bricks and arrows	All modules and relations	Geometric
[21]	None	Rectangles and lines	Top-level structure	Geometric and Fish eye
[1]	City	Buildings and pipes	All modules, no relations	Geometric
[15]	None	Modules as blocks, no relations	All modules shown	Geometric
[14]	None	Boxes and arrows	Top-level modules and relations	None
[18]	Landscape with cities	Cities (modules) and edges (relations)	Complete landscape	Geometric
[19]	None	Name label (modules) and pie charts (relations)	Single view (all modules and relations)	None
[16]	None	Rectangles and lines	Top-level modules and relations	Fisheye

Table 2.2: Summary of the results of the literature survey (continued in table 2.3)

2.5 Conclusions about software architecture visualization

Multiple visualization techniques have been analyzed in the previous sections using the framework presented in 2.3. Tables 2.2 and 2.3 summarize the results found for each visualization technique and each aspect of the framework. The tables show that for some aspects there are tendencies in the literature reviewed. Each aspect is analyzed below and possible implications for business process architecture visualization are discussed.

2.5.1 Metaphor

The visualizations from [1] and [18] use a city metaphor and the visualization from [7] uses Lego bricks as a metaphor. The rest of the visualization techniques do not use a real-world metaphor. The three techniques that use metaphors do not present a validation of their work. Therefore, there is no conclusive evidence that a metaphor should be used; it is not clear whether the usage of a metaphor will actually make it easier to understand

Source	Filter	Details on demand	History	Extract	Evaluation
[7]	None	Display module name	No	No	None
[21]	None	Semantic zoom	No	No	None
[1]	Showing execution trail	Relations shown on demand	No	No	None
[15]	Based on metrics or relations	None	No	No	User test
[14]	None	Semantic zoom, analysis results on edges	No	No	Industrial projects and controlled environment
[18]	None	None	No	No	None
[19]	None	None	No	No	None
[16]	None	Semantic zoom	No	No	Only verification, no validation

Table 2.3: Summary of the results of the literature survey (continued from table 2.2)

the visualization. While a metaphor may help the user to relate the content to something known, it may also distract them. Hence, a clear conclusion about metaphors cannot be drawn.

2.5.2 Primary artifacts

All but one visualization technique shows the software components as some kind of blocks. Depending on the metaphor, these can be buildings, lego bricks, simple rectangles, etc. Connections are usually shown as some sort of line or arrow. Again, the exact representation depends on the metaphor. The pie chart visualization technique by Samia et al.[19] shows that blocks and lines are not necessarily the only solution to the problem. In most of the visualization techniques, the blocks, i.e. the components, appear to be more central than the arrows, i.e. the dependencies. The pie-chart visualization, in contrast, puts a visual emphasis on the connections by showing them in pie charts while the components themselves are only shown as labels. What should be more central depends on the use case of the visualization technique. Different use cases may require different visualization techniques. However, as there is a majority of visualization techniques that uses block-and-line based primitives, it can be assumed that this approach should be considered for business process architecture visualizations as well.

2.5.3 Overview

There are two main ways of providing an overview. The first one is showing only the top-level packages. The second one is a zoomed-out view of all components. The first possibility provides an overview with less visual complexity due to a lower number of elements. It hides most details in the overview but makes it possible to understand the top level of the architecture. On the contrary, the second option provides more details in the overview. The bigger an architecture becomes, the less feasible the second option becomes. Understanding a graphic containing hundreds of elements is hardly possible for a human observer. Hence, the first option – showing an overview by displaying only the top-level elements – can be recommended for BPAs.

2.5.4 Zoom

A zoom functionality is usually provided and can therefore be seen as one of the basic functionalities of a software architecture visualization. It can be concluded that it should also be present in a business process architecture visualization. Some publications also refer to "semantic zoom" functionalities, but this type of functionality is not an actual zoom, but provides details on demand and is therefore discussed in the according section.

2.5.5 Filter

Among the software architecture visualization techniques analyzed here, filtering is not widely applied. Some filters based on metrics are present and another one filters in a way that only components related to one particular component are shown. Generally, there is not much attention for filtering techniques that mainly aim at reducing the visual complexity. The filters present mainly aim at giving insight into a particular aspect of the data. The reason for this is not evident from the literature. Visual complexity is recognized as a problem, hence filtering might still be considered as a solution that allows the user to select the right level of detail and amount of information they see.

2.5.6 Details on demand

Many visualization techniques provide a "semantic zoom", i.e. a possibility to see the inside of a component. This is an important way of providing details on demand because it allows keeping the overview simple while still providing the user with the option to see all details. Some visualization techniques show relations only on demand and others show some textual information, quality indicators, etc. on demand.

2.5.7 History and Extract

None of the visualization techniques analyzed here provides history or extract functionalities. While these functionalities are included in the general framework for visualizations[20], it can be assumed that they are not a primary concern for software architecture visualization techniques.

2.5.8 Evaluation

Among the publications that present an evaluation, there are two different ways of evaluating. The first one which has been applied more, is running a user test in a controlled environment. In such tests, the users are given assignments and their performance is used as an indicator of the quality of the visualization technique/tool. The other evaluation is the application in an industrial project. In this type of evaluation, the success of the project and satisfaction of those involved can be used as an indicator for the quality of the tool/technique.

2.5.9 Summary

In the previous paragraphs, several conclusions about software architecture visualizations have been drawn and possible implications for the visualization of business process architectures have been pointed out. As the use cases and users of BPA visualizations are different from those for software architecture visualization, the reliability of these results still needs to be proven. However, they can serve as a starting point in the development of BPA visualizations.

3 Specification

After concepts about the visualization of software architectures have been identified in the previous chapter, requirements for a business process architecture visualization tool have been specified. This chapter presents these requirements divided into two parts. The first part presents use cases for such a tool which have been identified in collaboration with practitioners. The second part focusses on more specific requirements for BPA visualization techniques that have been identified based on the findings from the previous chapter as well as the use cases presented in the first part of this chapter.

3.1 Use cases

As the goal of this research is helping business experts understand the structure of a process collection, the starting point are companies that have an (almost) complete repository of business processes modeled in a (semi-) formal notation, but are uncertain about the relations of the processes and the structure of the collection. The use cases are situations in which a user needs to understand the structure of the process architecture in order to fulfill a certain task.

3.1.1 Method for use case elicitation

Based on the motivation for developing a business process architecture visualization as well as on BPA literature, an initial set of usage scenarios of business process architectures has been identified. These scenarios were not yet fully specified use cases. They have been presented to two directors of Precedence, a Dutch consulting company focusing on process architectures. They were asked for three types of feedback:

- Would a tool that supports these scenarios help in their daily work?
- Which changes should be made to the scenarios?
- In which other ways would they want to use a visualization of business process architectures?

Taking their feedback into account, a set of final use cases has been developed. In the following section, first the initial usage scenarios will be presented, then the feedback and finally the final use cases.

3.1.2 Initial usage scenarios

The initial usage scenarios were based on the motivation for visualizing BPA's given in section 1.2 and on other research.

Understanding the structure of the architecture

As stated in the introduction, a business process architecture is not always easy to comprehend because it contains a big number of models and relations. Doing any work based on a business process architecture requires a basic understanding of that BPA. Hence, the first usage scenario focuses on getting a basic understanding of the structure of a business process architecture. That means being able to distinguish the elements of a business process architecture on a high level, understanding which high-level elements contain what processes, and identifying the relations between these elements.

Analysis of the quality of a process architecture

Eid-Sabbagh et al. [6] state as one of the questions to be answered with business process architectures, whether processes in relation to each other are sound. This question implies that a user would like to find out whether there are problems in the process architecture. Whereas the goal of visualization is not a formal mathematical analysis, patterns and anti-patterns for right or incorrect structures in a process architecture may be seen in a graphical visualization.

Implications of a local change

Whereas the previous two scenarios cover understanding and analyzing the process architecture as is, processes also evolve. It is important to understand the impact on the process architecture of changes done to a single process. Hence, analyzing the impact of a local change is considered as a further usage scenario. A local change may also be changing the boundaries of a process. Determining a process' boundaries is stated as one of the questions that can be asked to a process architecture in [6].

Path across multiple processes

Also according to Eid-Sabbagh et al. [6], another question that can be answered using business process architectures, is which path a client needs to follow as a whole, i.e. possibly using more than one process. This question can possibly be answered by a visualization and is therefore considered as one of the initial usage scenarios.

3.1.3 Feedback by practitioners

In the first meeting with one of the practitioners, the main message was that the most interesting usage would be the first one presented previously: A visualization of the process architecture itself that could aid the understanding of the structure, would have the most added value for his daily work.

The second person made it clear that he usually works with companies that already have a clear structure in their process repository that is well understood by these companies. But he also emphasized that there are other companies that have a repository of business process models but no clear structure. The reason for this can be that they modeled their processes to get a certain certification. He indicated that helping the companies in the latter situation develop a consistent structure for their process architecture, might be a possible growth market. Therefore, a visualization should help in developing a new structure. Also a comparison between an existing structure and a recommended structure would help. For companies that do not have a consistent structure yet, a visualization of the customer's path across multiple processes would also be helpful. It was pointed out that finding the impact of a local change is already possible with the tools they use.

3.1.4 Final use cases

Taking into account the initially suggested usage scenarios as well as the feedback from practitioners, the following final use cases have been developed. They are summarized in figure 3.1.

Understanding the structure of the architecture

The first scenario, *Understanding the structure of the architecture* has been confirmed as useful at least for companies that do not have a well developed view on their processes yet.

The prerequisite for this use case is a repository of processes that are captured in a formal or semi-formal notation. The repository should be comprehensive, i.e. it should cover most of the processes of the organization. If this is not the case, the structure cannot be correctly represented. The user wants to have the following interactions:

1. The tool displays the top-level structure of the process architecture. The user can see the top-level processes and how they are related. There is an option for the user to see details about these elements, such as the name or characterizing terms.
2. The user can select a process to refine it by showing its internal structure. This can be repeated a number of times to reveal the hierarchical structure of the process collection. The depth of the hierarchy depends on the individual process collection.

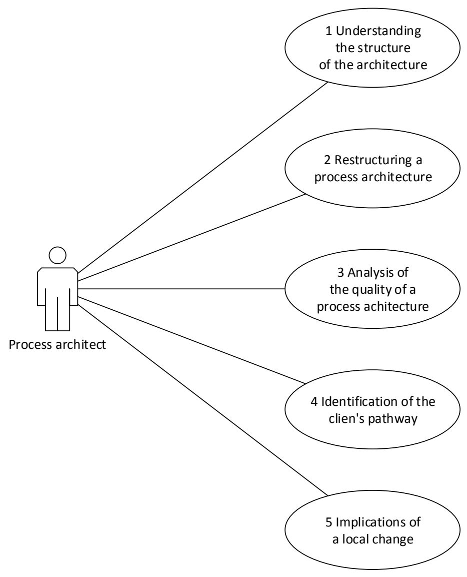

1 Understanding the structure of the architecture
2 Restructuring a process architecture
3 Analysis of the quality of a process achitecture
4 Identification of the clien's pathway
5 Implications of a local change
Process architect

Figure 3.1: Final use cases

3. The user can get details about connections between processes, such as the type of the connection. The types of relations depend on which relations are identified by the tool. Direct dependencies are a basic type.

The success of this use case can be measured by the user's perception of the usefulness and ease of use of the tool when answering questions about the structure of the process architecture.

Restructuring a process architecture

A process collection may have a defined hierarchical structure, but as processes are added, changed, and removed over time, this structure may not accurately represent the actual process collection anymore. Also, it was suggested in the practitioner's feedback that companies with an incomplete view on their process collection, would profit from a suggestion about the structure of the process architecture. In order to restructure the process architecture, the process architect would use certain guidelines. A tool that analyzes the content of the process collection and shows a possible restructuring can help them in defining a new structure.

As for the first use case, a complete repository of all the business processes of the organization is needed. The following interactions are relevant for this use case:

1. The tool displays a possible hierarchical structure that has been derived based on the processes in the process collection and may or may not take into account the

current hierarchical structure. The new structure is derived automatically by the tool based on the relations of processes among each other and/or the labels inside the processes.

2. The user can compare the current structure with the structure proposed by the tool in order to identify the proposed changes.

The perceived usefulness and ease of use when fulfilling tasks related to restructuring a process architecture, can be used to measure the success of this use case. Such tasks are judging the quality of aspects of the current structure as well as actually developing a new structure for the process architecture.

Analysis of the quality of a process architecture

Although this scenario was not pointed out as one of the most important by the practitioners, they have nevertheless seen an added value in it.

The structure of a process architecture may reveal faults in the way the organization works. For example, circular dependencies among multiple processes may lead to a deadlock. A clear visualization of the process architecture may help to identify such undesirable situations. Furthermore, a comparison between the actual architecture and a planned architecture can help a user to judge the quality of the current process architecture.

The precondition for this use case is the same as for the previous one. The following interactions are desired:

1. The tool displays the structure of the business process architecture on different levels of abstraction and indicates possible problems.
2. By selecting one of the indicated problems, the user can see more information about this problem, e.g. a description of the situation and the possible consequences.

As with the previous use cases, evaluation can be based on the perceived usefulness and ease of use when working on related tasks. The tasks in this use case involve finding possible problems in the process architecture.

Identification of the client's pathway

Especially for companies that do not have a good organization of their process collection yet, it was pointed out that a graphical identification of the client's pathway would be useful.

Even though a company has its processes documented, it may still be unclear which processes a client has to go through in order to get a certain service or achieve a particular goal under the given circumstances. By visualizing the dependencies among processes, it can be seen which of them may be relevant for the particular client. A similar approach may be taken for analyzing which processes are involved in a particular business action, like the development of a new product.

For this use case, all processes relevant to the customer have to be captured in a (semi-) formal way in a repository. The use case includes the following interactions:

- The user selects an activity or state in a process and the tool displays all pathes that the customer can take to reach that point of a process.
- The user selects an activity or state in a process and the tool displays all pathes that the customer can take from this point in the process onwards.

The tasks related to this task are finding the paths to and from certain states in the process collection. Again, perceived ease of use and usefulness should be measured.

Implications of a local change

When adding a new process to a process architecture or removing an existing one, other processes may need to be adapted. Therefore, it is essential to know which relations exist between processes or which need to be established.

For this use case, a process repository containing all relevant processes is necessary. As it is not known beforehand which processes are important, the repository should be as complete as possible. Interactions for this use case are:

- The user selects a certain process and sees which other processes it is related to.
- When the user selects one of the processes that the first one is related to, details about the relation, such as the type (e.g. working on the same data) is shown.

In this use case, relevant tasks for measuring how well the use case is fulfilled, consist of identifying processes that are affected by a certain change in a single process. Again, perceived ease of use and usefulness should be measured.

3.2 Visualization and abstraction techniques

In section 2.3 a framework has been developed for assessing visualization techniques in order to collect input for the visualization of business process architectures. The same framework is now used to specify the requirements for a BPA visualization. The framework element *Evaluation* is not covered, because it is not a property of the visualization techniques itself. The requirements presented in the following sections are based on the conclusion presented in section 2.5.

3.2.1 Metaphor

As stated in section 2.5, there is no evident preference for or against using a metaphor. Hence, the decision for or against using one cannot be made entirely objectively based on the data present. Not using a metaphor typically is a smaller implementation effort because easier primitives can be used to visualize a business process architecture. A metaphor may also increase the visual complexity because it adds details to the visualization that do not directly transport a meaning, e.g. the water in the visualization by Panas et al.[18]. Based on these considerations, it is suggested not to use a metaphor for the BPA visualization.

3.2.2 Primary artifacts

As concluded in section 2.5, a graph-like structure is applicable for most cases. Hence, a business process architecture visualization tool should have at least one visualization that represents architecture elements (processes and groups) as nodes of a graph and the relations between them as the edges of the graph. When trying to understand the general structure of a process architecture, the user may also want to focus on the relations between processes. Therefore, a visualization technique that puts a higher emphasis on the relations should also be provided.

3.2.3 Overview

A business process architecture visualization needs to provide an overview of the top-level structure of the BPA in the beginning. This enables the user to understand the overall structure before they delve into more detailed views. Concretely, this means that in the beginning the top-level elements and/or relations are shown. As argued in section 2.5, providing an overview that shows all elements at the same time, is not useful for a big process architecture with hundreds of elements.

3.2.4 Zoom

A geometric zoom functionality can help the user to get a better view on particular parts or to get an overview. It should be provided wherever it is appropriate, i.e. for any visualization technique that can be zoomed. "Semantic zoom" is considered in the section about details on demand.

3.2.5 Filter

In many of the software architecture visualization techniques covered in the related work chapter, no filtering options are provided. However, filtering is a very valuable addition. Several filters for BPA visualizations may be useful and should be considered in the implementation of a BPA visualization tool:

- Filtering relations to reduce visual complexity: It is anticipated that the number of relations in the process architecture is very high and may make it hard to understand its structure. Hence, a BPA visualization tool should provide the user options to filter edges so to show only those important to the user.
- Filtering elements to reduce visual complexity: A process architecture does not only contain many relations, but also many processes and groups of processes. Filters should be provided that reduce the number of elements in a way that makes it easier for the user to understand the process architecture.
- Showing only connections of a certain node: To understand the connections of a certain element of the process architecture, a user might want to highlight the element and its connections. By selecting an element and applying this filter, the element, all the elements it is connected to and the connections should be highlighted. Highlighting can be done by completely removing all other objects, by reducing the visibility of other objects and/or by applying a different style to the highlighted objects. While removing other objects reduces the visual complexity, it also removes context information which may be helpful for the user to understand the graphic.

3.2.6 Details on demand

As concluded in section 2.5, there should be a "semantic zoom", i.e. a way to open up the top-level elements seen in the beginning to show their contents. Furthermore, details on demand can be provided by displaying additional (textual or other) information when an object gets selected.

3.2.7 History and Extract

Conforming the findings from software architecture visualizations, history and extract functionalities are not considered necessary.

3.3 Summary

In this chapter, five use cases have been presented that can be covered by a business process architecture visualization tool. Furthermore, the visualization framework used before has been applied to specify requirements for the visualizations used in such a tool. These requirements do not fully describe a software that visualizes business process architectures, but lay a foundation for such a tool. BPA visualization is a completely new field and therefore a complete specification of a tool does not seem appropriate based on the limited information available. An over-specification would also limit the possibilities to explore techniques that may or may not be beneficial.

4 Architecture and Implementation of the prototype

Based on the requirements described in the previous chapter, a prototype has been implemented[1] which covers selected use cases by means of visualization and analysis techniques. In this chapter, the functionality of the prototype will be introduced and its implementation will be described on an abstract level. Details of the implementation needed for further developing it can be found in appendix B.

4.1 Functionality of the prototype

In section 3.1 five use cases for a business process architecture visualization tool have been presented. Because of the limited time that was available for developing it, the prototype focusses on the first and second use case, *Understanding the structure of the architecture* and *Restructuring a process architecture*. This selection of use cases is highlighted in figure 4.1. The first use case is the most important one because it allows the user to understand the BPA and therefore is the basis for working with it. The second use case was chosen because it is the starting point for structuring an unstructured process collection or changing an existing BPA. The rest of the use cases is also considered relevant, but left as future work. The software architecture of the prototype has been designed to make it easy to extend it in the future in order to cover more use cases.

To support *Understanding the structure of the architecture*, at least one visualization technique needed to be implemented that covers the requirements specified in section 3.2. As finding visualization techniques for BPA's is one of the goals of this research, it is preferable to provide more than only one visualization technique. As stated in the requirements, there should be a visualization technique that represents a business process architecture as a graph. Also according to 3.2, it does not need to use a metaphor. Hence, the first visualization technique shows the BPA as a graph with no real-world metaphor. In the remainder, this technique is referred to as the *block-based visualization*. Details about it are explained in section 4.1.1.

Also mentioned in section 3.2 was that a visualization technique with a higher emphasis on the relations should be provided as well. One possibility to achieve this was presented in [19]. However, their suggested pie tree visualization does not work well for large

[1]The prototype can be downloaded at `http://bpav.thomasmilde.com`

architectures because it provides only a single view. Because the data about connections is shown by diagrams that can only be interpreted using the legend, it is also hard to get an overview in this visualization technique. Another technique for emphasizing relations between elements uses a circular layout. This technique is used in various fields, for example in brain connectivity visualization [13]. It is referred to by several names, but in this work the term *Connectogram* as suggested by [13] is used.

To address the need for filtering in order to reduce the visual complexity, a number of filters specific to an individual visualization technique have been created. These are explained together with the according visualization technique. Moreover, a visualization technique-independent filter has been implemented. It satisfies the need for reducing the number of nodes in the visualization by grouping them into "artificial" nodes containing a set of groups and/or processes that are strongly interconnected. This *Automatic grouping* is explained in detail in section 4.1.2.

To support the use case *Restructuring a process architecture*, it is necessary to provide a technique that creates a new structure for the process architecture based on the processes it contains. This *automatic clustering* is introduced in section 4.1.2. Furthermore, the use case states that a comparison between the automatically created structure and the original structure should be possible. Therefore, a *comparison table* is provided which shows the quantitative overlap between top-level elements of the original architecture and groups of the automatically generated architecture.

To provide the user with a known means of navigating the process architecture, a *tree view* and a *search function* have been added. The tree view shows the hierarchical structure of the process architecture without the relations. It is explained in section 4.1.1. The search function allows the user to search architecture elements based on their names.

The prototype naturally needs data to be visualized. Hence, it must provide a possibility to import a process architecture. For the first prototype, a big process collection with a clear structure was needed. As a copy of the SAP reference model was available to the author and because it is of sufficient size and has a clear structure, this process collection was chosen as the data to be used in the prototype. The SAP reference model does not explicitly specify relations between processes. To identify these relations, a *dependency detection* has been implemented. It detects direct dependencies between processes based on labels and is explained in detail in section 4.1.2.

Figure 4.2 provides an overview of how the functionalities support the use cases. In section 4.1.1, the following four visualization techniques will be explained:

- Block-based visualization
- Connectogram visualization
- Tree view
- Comparison table

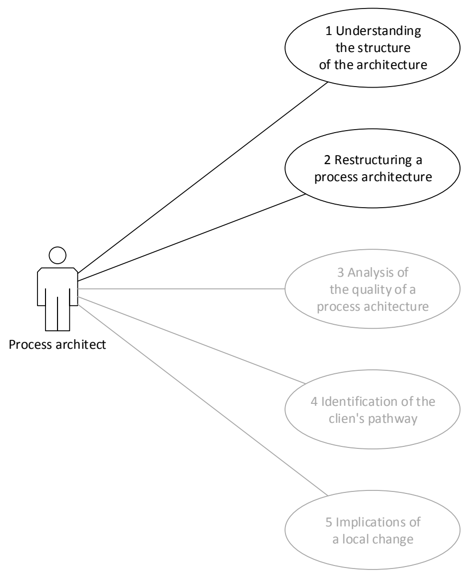

1 Understanding the structure of the architecture
2 Restructuring a process architecture
3 Analysis of the quality of a process achitecture
4 Identification of the clien's pathway
5 Implications of a local change
Process architect

Figure 4.1: Use cases for business process architecture visualization. The use cases in gray have not been selected for the prototypical implementation.

The visualization framework will be applied again to show in how far the visualization techniques match the requirements described in section 3.2. Table 4.1 summarizes the application of the visualization framework. The comparison table is not a graphical visualization of a single process architecture but rather a comparison mechanism and therefore the visualization framework is not applicable to it and was not applied to the comparison table.

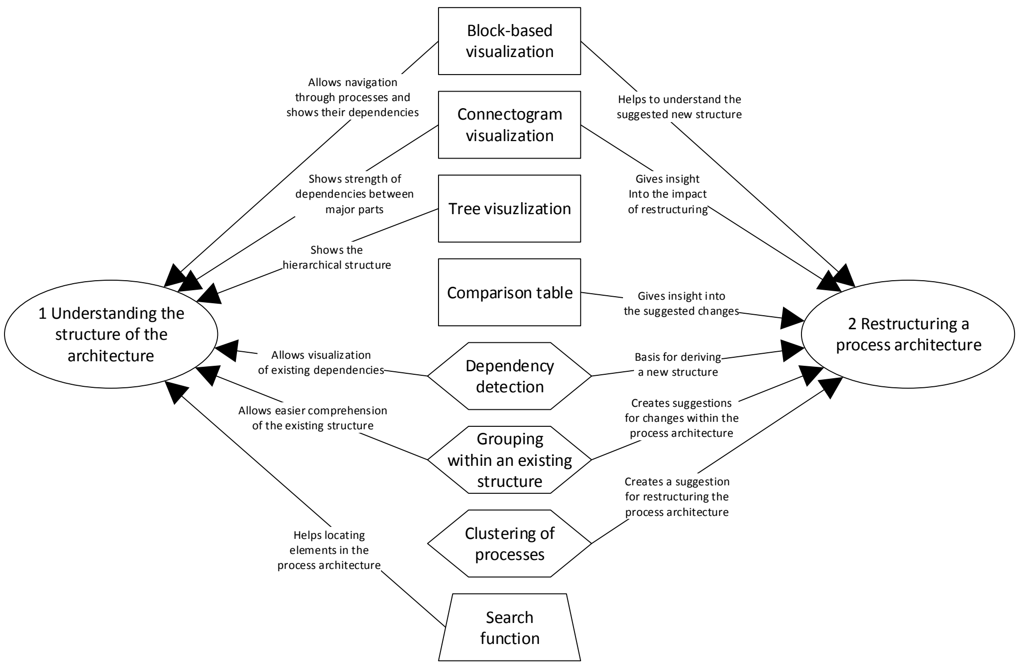
In section 4.1.2 the following three analysis techniques will be explained:

- Dependency detection
- Automatic grouping
- Clustering

4.1.1 Visualization techniques

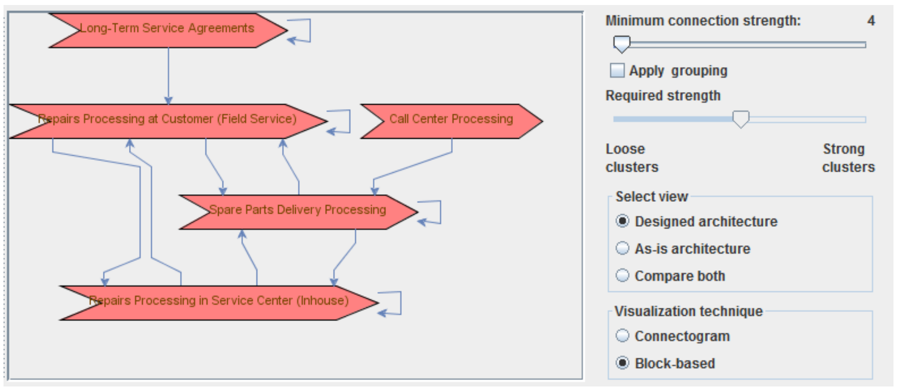
Block-based visualization

The block-based visualization of the architecture shows each element of the architecture as a node. Dependencies among processes are shown as directed edges between these nodes. An example of such a visualization can be seen in figure 4.3. The example shows the five processes involved in customer service. All of these processes have sub-



Block-based visualization
Connectogram visualization
Tree visuzlization
Comparison table
Dependency detection
Grouping within an existing structure
Clustering of processes
Search function
1 Understanding the structure of the architecture
2 Restructuring a process architecture
Allows navigation through processes and shows their dependencies
Shows strength of dependencies between major parts
Shows the hierarchical structure
Allows visualization of existing dependencies
Allows easier comprehension of the existing structure
Helps locating elements in the process architecture
Helps to understand the suggested new structure
Gives insight Into the impact of restructuring
Gives insight into the suggested changes
Basis for deriving a new structure
Creates suggestions for changes within the process architecture
Creates a suggestion for restructuring the process architecture

Figure 4.2: Relation between functionalities and use cases. Use cases are shown as ellipses, visualization techniques as rectangles, analysis techniques as hexagons and other functionalities as trapezoids.


Long-Term Service Agreements
Repairs Processing at Customer (Field Service)
Call Center Processing
Spare Parts Delivery Processing
Repairs Processing in Service Center (Inhouse)
Loose clusters
Strong clusters

Minimum connection strength: 4

☐ Apply grouping

Required strength

Select view
- ● Designed architecture
- ○ As-is architecture
- ○ Compare both

Visualization technique
- ○ Connectogram
- ● Block-based

Figure 4.3: Block-based visualization of the *Customer Service* group of the SAP reference model

processes, but this cannot be seen directly in the block-based visualization. It is possible to show the content of any of the processes by double-clicking it. The edges between the processes may represent multiple dependencies each. For example, the arrow from *Call center processing* to *Spare parts delivery processing* may indicate a direct dependency of these two elements as well as a dependency of an element inside *Spare parts delivery processing* on an element inside *Call center processing*. On the right of the figure, a setting "Minimum connection strength" can be seen which indicates that only those edges will be shown that represent at least a certain number of dependencies, four in the example.

Figure 4.4 shows an excerpt of the application of the same visualization technique to another part of the reference model. It is a part of the top level of the reference model and automatic grouping has been applied. The double-arrow shapes indicate that the nodes do not represent processes, but logical groups. Each group consists of processes and/or other groups.

The block-based visualization can be applied to the original process architecture as well as to the automatically clustered structure. It can also be used to show both next to each other.

The visualization framework can be filled in as follows for the block-based visualization:

Metaphor: No metaphor from the real world is used. The information is represented as a graph.

Primary artifacts: Processes are represented as blocks in the form of value-chain arrows and groups of processes are indicated with two overlapping value-chain arrows. Relations are indicated as lines with an arrow on one end.

Overview: In the overview, only the top-level elements, i.e. the processes and groups on the highest level and the relations between them, are shown.

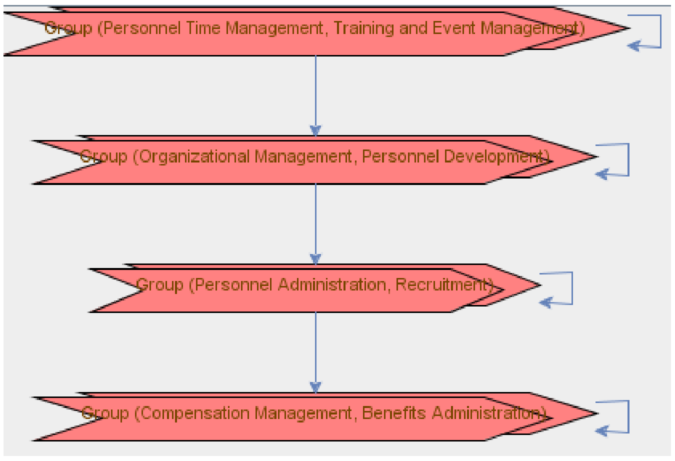

Group (Personnel Time Management, Training and Event Management)
Group (Organizational Management, Personnel Development)
Group (Personnel Administration, Recruitment)
Group (Compensation Management, Benefits Administration)

Figure 4.4: Excerpt from the block-based visualization of the top-level of the SAP reference model after automatic grouping has been applied

Zoom: Zoom is not yet provided due to technical difficulties. It should be added as a future development.

Filter: A filter for relations is provided that shows only relations that have at least a minimum strength which can be set by the user.

Details on demand: The user can open each node to see its "content", i.e. the processes, groups and relations it contains or its process model. This can be referred to as "semantic zoom".

History: No history function is provided. All interactions with this visualization technique can be made undone by other user interactions, e.g. setting back the minimum edge strength to its previous value.

Extract: No extract functionality is provided. The interaction mechanisms of this visualization technique are not very complex and therefore a particular state of the visualization can be restored rather easily.

Connectogram visualization

The connectogram visualization focuses on visualizing how strongly the top-level parts of the process architecture are connected to each other. It shows each top-level element of the process architecture as a segment of a circle. These segments are subdivided into segments for each sub-element, but this sub-division is not explicitly visible. The connections between different elements are bundled per top-level element which makes it easier to identify that connections that exist on a high level while dropping some detail about individual dependencies. Figure 4.5 shows an example of a connectogram.

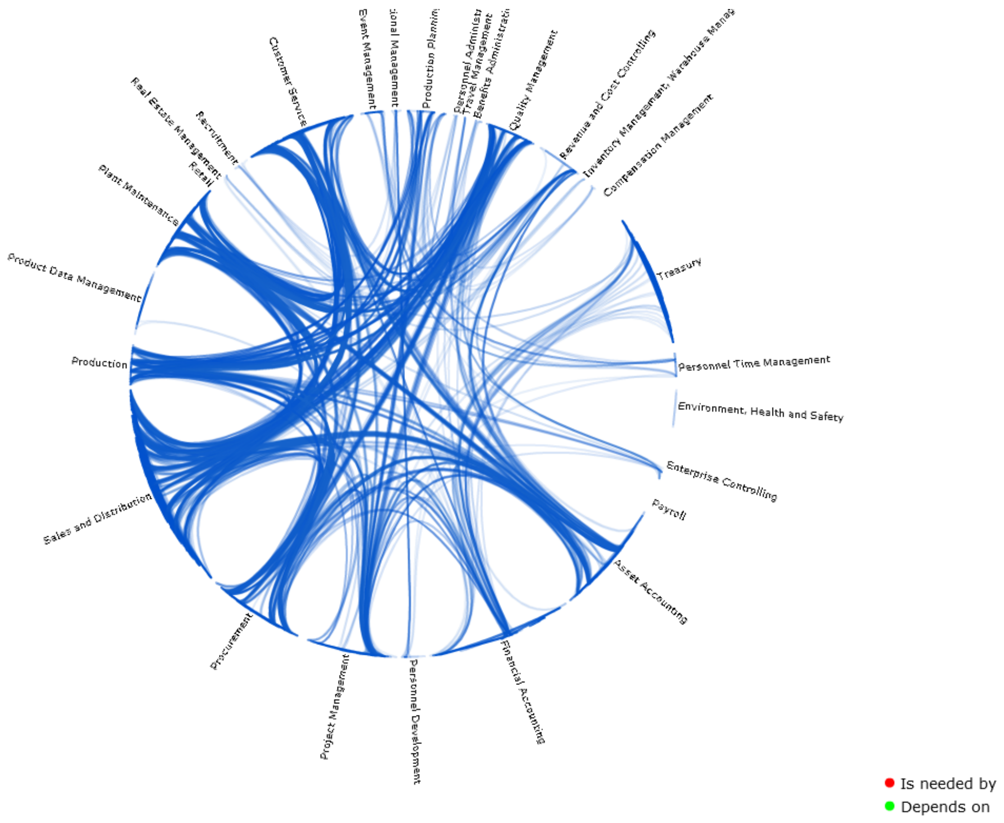


Figure 4.5: Top level of the reference model visualized as a connectogram

For the connectogram, the visualization framework can be filled in as follows:

Metaphor: No real-world metaphor is used. The process architecture is shown as a circle.

Primary artifacts: Processes and groups are shown as circle segments. Each segment is sub-divided into smaller segments that represent its children, but only the elements on the current top level are labeled and therefore explicitly visible. Relations are shown as bundled lines inside the circle that connect the respective elements of the circle.

Overview: In the overview, all elements are represented in the circle but only the top-level elements are labeled. To the user, only the top level elements are clearly visible

Zoom: No zoom functionality is provided.It would likely not have a big added value because the visualization is always fitted to the screen and does not have many details that would profit from an inspections at a higher zoom-level.

Filter: No filters are provided.

Details on demand: When pointing at an element, its relations and the elements it is related to are highlighted. By clicking on an element, this one can be opened, i.e. selected as the new top level that is represented by the circle as a whole.

History: History functionality is not provided due to the simple interaction mechanisms that do not make undo and redo functionalities necessary.

Extract: Extract functionality is also not present due to the simple interaction mechanisms that allow to easily return to a certain state of the visualization.

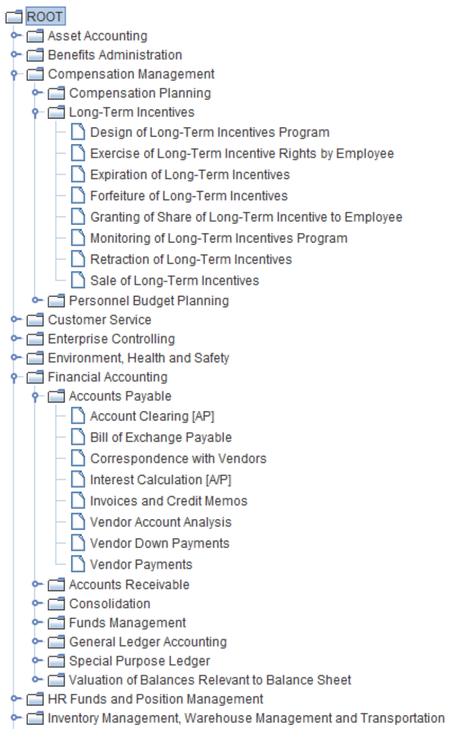
Tree visualization

The tree visualization shows only the hierarchical structure of the process architecture. It is mainly intended to ease navigation of the process architecture: The tree makes it possible to easily see which node is being visualized at the moment and where it is located in the hierarchical structure as well as to select a node to be visualized. An excerpt of the tree visualization can be seen in figure 4.6.

The tree visualization provides the following values for the visualization framework:

Metaphor: No real-world metaphor is used. The representation is similar to the tree-view provided by file system browsers.

Primary artifacts: Elements and groups are represented as folders or entries (branches or leafs) depending on whether they have children or not. Relations are not represented.



- ROOT
  - Asset Accounting
  - Benefits Administration
  - Compensation Management
    - Compensation Planning
    - Long-Term Incentives
      - Design of Long-Term Incentives Program
      - Exercise of Long-Term Incentive Rights by Employee
      - Expiration of Long-Term Incentives
      - Forfeiture of Long-Term Incentives
      - Granting of Share of Long-Term Incentive to Employee
      - Monitoring of Long-Term Incentives Program
      - Retraction of Long-Term Incentives
      - Sale of Long-Term Incentives
    - Personnel Budget Planning
  - Customer Service
  - Enterprise Controlling
  - Environment, Health and Safety
  - Financial Accounting
    - Accounts Payable
      - Account Clearing [AP]
      - Bill of Exchange Payable
      - Correspondence with Vendors
      - Interest Calculation [A/P]
      - Invoices and Credit Memos
      - Vendor Account Analysis
      - Vendor Down Payments
      - Vendor Payments
    - Accounts Receivable
    - Consolidation
    - Funds Management
    - General Ledger Accounting
    - Special Purpose Ledger
    - Valuation of Balances Relevant to Balance Sheet
  - HR Funds and Position Management
  - Inventory Management, Warehouse Management and Transportation

Figure 4.6: Excerpt of the tree visualization

Aspect	Block-based visualization	Connectogram visualization	Tree visualization
Metaphor	No real-world metaphor. Visualization as graph	No real-world metaphor. Visualization as a circle	No real-world metaphor, but similarity to directory trees
Primary artifacts	Blocks and arrows	Elements as circle segments; relations as bundled lines	Elements as folders and entries; relations not shown
Overview	Graph view of the top-level elements	All elements and relations shown, but only top-level elements labeled.	Fully collapsed tree (except root node)
Zoom	-	-	-
Filter	Minimum edge strength	-	-
Details on demand	Semantic zoom	Semantic zoom and relation highlighting	Opening/closing of branches
History	-	-	-
Extract	-	-	-

Table 4.1: Visualization framework applied to the visualization techniques of the prototype

Overview: The overview consists of the tree structure which is fully collapsed except for the root node.

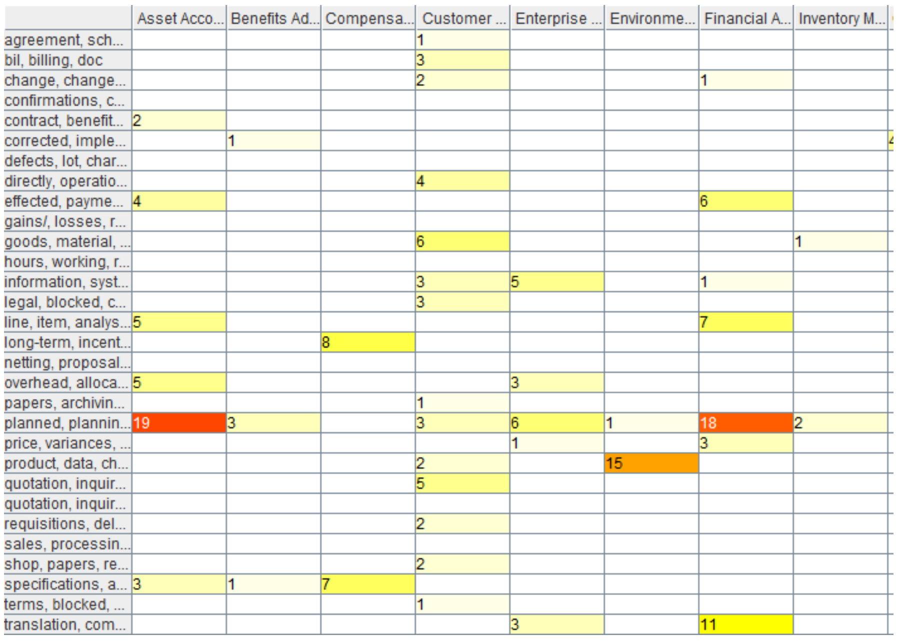
Zoom: No zoom functionality is provided. Zooming would probably not provide considerable added value because the tree does not have details that need to be seen at a higher zoom-level and would become illegible at significantly lower zoom-levels.

Filter: No filters are provided.

Details on demand: Nodes of the tree can be expanded or collapsed to adjust the level of detail.

History: History functionality is not provided and not needed due to the simple interaction mechanism.

Extract: Extract functionality is not provided and not needed due to the simple interaction mechanism.



	Asset Acco...	Benefits Ad...	Compensa...	Customer ...	Enterprise ...	Environme...	Financial A...	Inventory M...	
agreement, sch...				1					
bil, billing, doc				3					
change, change...				2			1		
confirmations, c...									
contract, benefit...	2								
corrected, imple...		1							4
defects, lot, char...									
directly, operatio...				4					
effected, payme...	4						6		
gains/, losses, r...									
goods, material, ...				6				1	
hours, working, r...									
information, syst...				3	5		1		
legal, blocked, c...				3					
line, item, analys...	5						7		
long-term, incent...			8						
netting, proposal...									
overhead, alloca...	5				3				
papers, archivin...				1					
planned, plannin...	19	3		3	6	1	18	2	
price, variances, ...					1		3		
product, data, ch...				2		15			
quotation, inquir...				5					
quotation, inquir...									
requisitions, del...				2					
sales, processin...									
shop, papers, re...				2					
specifications, a...	3	1	7						
terms, blocked, ...				1					
translation, com...					3		11		

Figure 4.7: Excerpt of the comparison table

Comparison table

In addition to the visualization techniques, a table view is provided to compare the original structure of the process architecture to the structure resulting from automatic clustering. Each column of the table represents a top-level entry of the original architecture and each row represents a top-level element of the automatically clustered architecture. Each cell states the number of processes that both of these elements have in common, i.e. both of them have a sub-element which represents that process. An excerpt of a comparison table can be seen in figure 4.7. The colors are chosen to support the numerical contents of the cells. Cells that would contain the value zero are left blank for a better overview.

As the comparison table is not a visualization of a single process architecture, but rather a display of a comparison technique, the visualization framework has not been applied to it.

4.1.2 Analysis techniques

To support the visualization techniques and add functionality necessary for the selected use cases, analysis techniques have been implemented. They are described in detail in the sections below.

Dependency detection

A process collection that is loaded into the tool might in principle contain information about the dependencies among processes. Otherwise, or if the information about dependencies might be inaccurate, an automatic detection of dependencies is needed. To be able to analyze dependencies, (semi-)formal models of the processes are needed. The SAP reference model is modeled as EPCs. This standard is well suited for analyzing dependencies because events/statuses have to be explicitly modeled. If one process creates a certain event as a result and another process can be started by the same event, a dependency between these two processes is indicated. The only information known of events are their labels. Therefore, identity is based on similar labels. The choice for similarity over exact equality of labels has been made in order to accommodate slight variations in naming styles and/or typing mistakes.

Grouping within an existing structure

A process architecture that comes with an existing hierarchical structure may be hard to understand if the number of elements and connections on one level is very high. It has been observed that in such cases often a number of elements can be identified that have a particularly strong connection among each other. This suggests that these elements form a logical group. Hence, the prototype offers an automatic grouping that can help the user understand the structure of the process architecture by reducing the visual complexity.

There is a single parameter that controls the grouping: the required density, i.e. the strength of connections within the group. The density of a group is calculated by the formula $\frac{\sum_{edges\ in\ group} strength\ of\ edge}{maximum\ number\ of\ edges}$. The strength of an edge is a number between 0 and 1 which depends on the number of dependencies this edge represents.

Clustering of processes

A process collection may lack a hierarchical structure. An automatic clustering can introduce the missing structure based on the content of the processes. The automatic clustering also helps to evaluate whether the chosen hierarchical structure is the right one. In the prototype, the Weka software[12] is used to automatically cluster all processes into a new structure. The difference between this clustering and the grouping described before is that the clustering creates a completely new structure while the grouping preserves the structure as it was defined and refines it by adding additional layers.

The automatically generated clusters need names for the user to identify them. In the prototype, the three most significant words found within the labels of the processes in the cluster, are chosen as names. Most significant words are those that occur frequently within the cluster and not very frequently in the process collection as a whole.

4.2 Architecture of the implementation

Apart from the functions as specified and explained before, there are always non-functional requirements that also have an influence on the architecture of the software. As stated in the beginning of this chapter, the current prototype only supports two out of five use cases. Therefore it is important that adding additional functionality and changing existing functionality are as simple as possible. Hence, extensibility and adaptability were particularly important. As it is a research prototype, such aspects as performance and scalability are not a primary concern. However, to be able to test a variety of visualization and analysis techniques, to support more use cases in the future and to try different user interaction mechanisms, adaptability and extensibility are important. In particular, there are the following requirements:

1. It must be possible to add new visualization techniques and new analysis techniques. The effort involved in adding a new technique should be minimized.
2. Visualization and analysis techniques do not directly depend on each other, i.e. a visualization technique does not rely on the presence of a particular analysis technique or the other way around.
3. The control flow of the application can be changed without modifying visualization or analysis techniques.
4. Different data sources for process architectures can be added.

The first two requirements ensure that future research with different analysis and visualization techniques can be incorporated into the prototype as easily as possible. The third requirement makes sure that it is possible to support new use cases without changing the complete application, because adding a new use case likely requires adding new behavior to the application. The fourth requirement aims at allowing to test the prototype with other process architectures in the future.

The requirements mentioned before indicate that the visualization techniques, analysis techniques and import components should be as self-enclosed as possible. They should be components on their own that can be exchanged easily. A model-view-controller (MVC) architecture provides these advantages by separating the three main components model, view and controller. The MVC architecture style has been used as a starting point for the architecture of this application.

The implementation is split into five packages as can be seen in figure 4.8. The *controller* package keeps the status of the application and controls the data and control flow. It is the single place that needs to be adapted to change the control flow of the application, for example to support new or changed use cases. The *model* package contains data structures to represent a business process architecture, i.e. the central data artifact of this application. All other components interact with the model. User interface components are defined in the *ui* package. The visualization components are in the sub-package

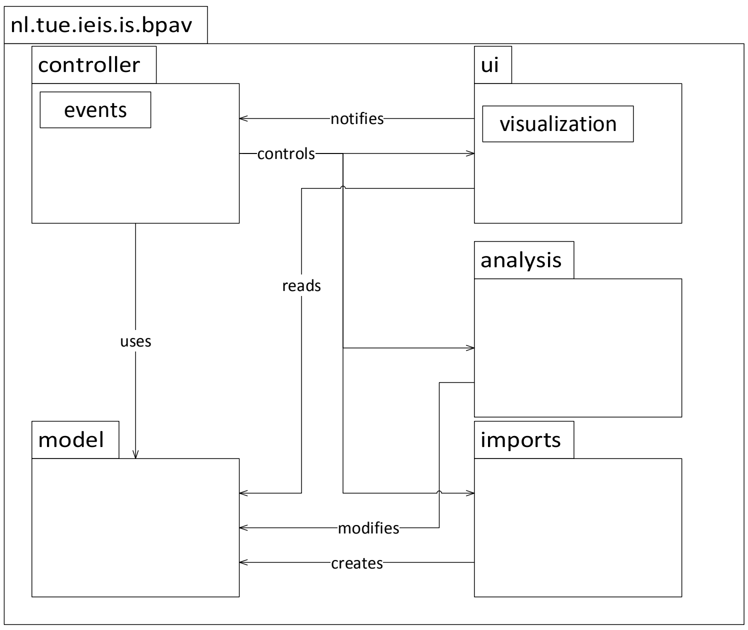


Figure 4.8: Architecture of the prototype

visualization to isolate them as much as possible from the rest of the application. The *analysis* package contains tools for analyzing and transforming the model and the *imports* package contains routines for loading data from an external source. Both of these packages encapsulate their respective functionalities in a way that allows to add and change functionality rather easily. The following paragraphs describe each package in more detail.

4.2.1 The *controller* package

The *controller* package was introduced in order to keep the status of the application saved in one central location. This makes it easier to understand and change the overall control flow of the application than if the control is scattered around the system.

It is possible to implement all control flow in a single controller class. However, this would result in a single big and powerful class which is considered bad style in software engineering because it is hard to understand and maintain such a class. As extensibility and adaptability are important, it was decided to split the control into multiple classes. Each controller class is responsible for a certain phase of the application's life cycle, e.g. *MainController* for starting up the application and delegating to other controllers and *ExplorationController* for visualizing the process architecture and allowing the user to navigate through it.

The sub-package *events* contains classes of objects that are used for the communication between the controllers and the user interface. It contains notifications which are sent from the user interface to the controller upon user input as well as updates which are sent from the controller to the user interface when a change in the visualization needs to be performed. Both types of events are typically lightweight classes that hold a small amount of data and do not contain application logic. The type of event indicates the type of user interaction that has been done or the type of update that needs to be performed. Compared to directly invoking specific methods, sending events allows an easier extension of functionalities of the controller as well as the user interface/visualization components.

4.2.2 The *model* package

The *model* package contains data structures needed for storing all information the application uses about a business process architecture, including the (hierarchical) structure of the architecture, the names and relations of processes and the process models if available. Furthermore, the package contains the functionality needed to efficiently access and manipulate these data structures.

4.2.3 The *ui* package

The *ui* package contains all user interface components. The *MainForm* is the main window of the application which can hold all of the other components. The *visualization* sub-package contains the components (panels) that are used to graphically represent the process architecture. There are several components for different types of visualization. The controllers decide, according to the input of the user, which components are to be used.

4.2.4 The *analysis* package

The *analysis* package contains techniques for analyzing and changing the model. More concretely, there are three analysis tools. The class *DependencyAnalysis* computes dependencies among the processes in a process collection. That means it computes the relations between processes that are characteristic for the process architecture. The *Clusterer* uses the external library Weka[12] to compute a clustering, i.e. a structure, of the process architecture, based on these relations. The class *GreedyGrouping* implements a possibility of grouping the children of a certain node in order to simplify the structure by introducing an additional layer in the process architecture.

4.2.5 The *imports* package

The imports package provides the functionality needed to load a process collection from an external source. At the moment, the only import loads the SAP reference model including the pre-defined structure as well as the EPC models of the processes in the reference model.

4.3 Summary

Based on the use cases and requirements in the previous chapter, a prototype has been developed. The prototype covers two out of five use cases which were considered the most important. Its functionalities are based on the requirements presented in the previous chapter. The prototype was software-architecturally designed in a way that allows it to be extended by future research.

5 Validation

The prototype that has been developed, is a new piece of technology and therefore its practical relevance has to be evaluated. The tool is practically useful only if it is likely to be used in practice. Therefore, the likeliness of the tool being used, i.e. its acceptance by users, is the criterion that needs to be evaluated. More precisely, the functionalities of the tool need to be evaluated and the evaluation should be done in the context of the use cases they are intended for.

This chapter presents the setup for a user test that still needs to be executed. The validation has not yet been done because the previous steps of the process, i.e. identifying use cases and requirements and implementation, took longer than planned. The validation of the prototype is still an important step that should be carried out as future work and therefore this test plan is presented as a basis for executing a user test. The first section of this chapter describes the fundamentals of the test and the second section explains its exact setup.

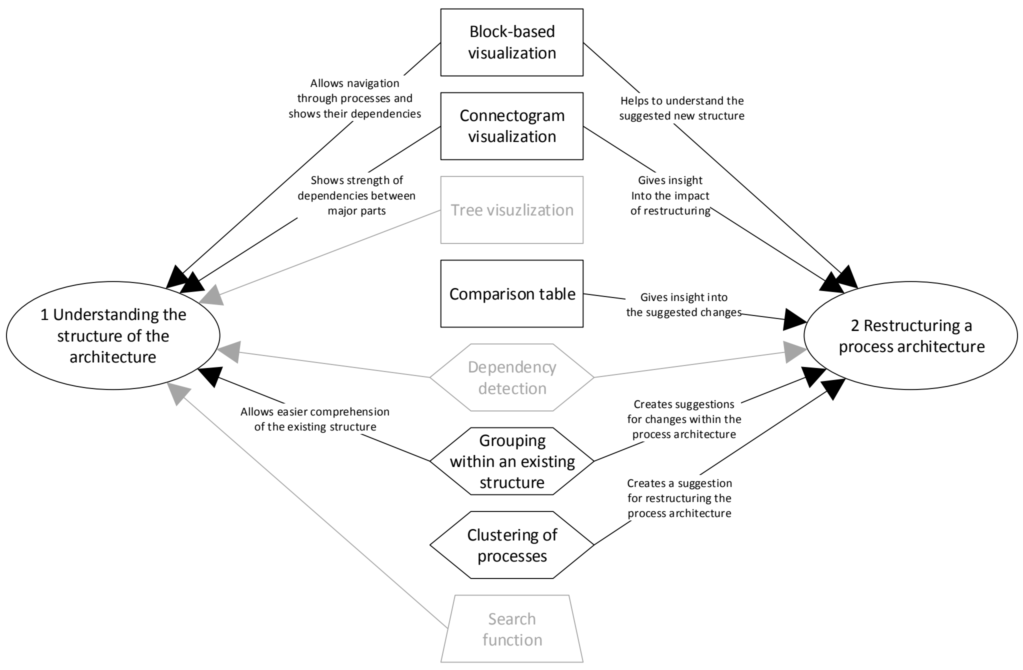
5.1 Framework for the test

This section gives the foundation for the test setup based on the functionalities and use cases of this tool and on theories about the acceptance of technology.

5.1.1 Functionalities and use cases to test

In chapter 4, an overview of the functionality of the tool and its intended relation to the use cases has been presented. The overview given in figure 5.1 which has been seen in that chapter already, gives a starting point for the evaluation. *Tree visualization* and *Search function* are present in current industry standard tools already and therefore cannot present an improvement by themselves. For that reason, they are not explicitly evaluated (indicated in gray). *Dependency detection* cannot be used in an isolated manner and most of the functionalities of the prototype depend on it. Hence, testing it separately is hardly possible. It is shown in gray as well to indicate that it is not included in the test.

That leaves eight functionality-use case pairs that need to be evaluated. However, the functionalities are not completely independent: Grouping and clustering rely on a vi-



Block-based visualization
Connectogram visualization
Tree visuzlization
Comparison table
Dependency detection
Grouping within an existing structure
Clustering of processes
Search function
1 Understanding the structure of the architecture
2 Restructuring a process architecture
Allows navigation through processes and shows their dependencies
Shows strength of dependencies between major parts
Allows easier comprehension of the existing structure
Helps to understand the suggested new structure
Gives insight Into the impact of restructuring
Gives insight into the suggested changes
Creates suggestions for changes within the process architecture
Creates a suggestion for restructuring the process architecture

Figure 5.1: Relations between functionalities and use cases that need to be evaluated separately: The functionalities *Tree visualization*, *Dependency detection* and *Search function* (shown in gray) will not be evaluated directly. Use cases are shown as ellipses, visualization techniques as rectangles, analysis techniques as hexagons and other functionalities as trapezoids.

Nr.	Use case	Functionality
1	1	Block-based visualization
2	1	Connectogram visualization
3	1	Grouping and block-based visualization
4	1	ARIS Explorer and Designer
5	2	Clustering and block-based visualization
6	2	Clustering and connectogram visualization
7	2	Clustering and comparison table
8	2	ARIS Explorer and Designer

Table 5.1: Combinations of functionalities and use cases that need to be tested

sualization technique in order to present results to the user. Grouping will be tested with the block-based visualization and clustering with the block-based visualization, connectogram and comparison table. To reduce the number of tests, the combination of clustering and grouping will not be tested.

To establish the added value of the functionalities, a comparison to an industry standard tool (the reference tool) is necessary. The reference tool for this test will be ARIS. The SAP reference model which is used in the prototype, has originally been modelled in ARIS. Hence, ARIS can be seen as the base-line tool to which the new prototype should be compared. The relevant functionalities of ARIS are the Explorer and the Designer which provide a navigation tree and the possibility to view process models and their properties. In the rest of this chapter, ARIS refers to the functionalities Explorer and Designer of ARIS. Table 5.1 summarizes the tests that have to be done.

5.1.2 The Technology Acceptance Model

A multitude of theories about the acceptance of IT systems exists. A popular one is the Technology Acceptance Model (TAM) by Fred D. Davis [4]. In contrast to newer models like the TAM2 [22] or the Unified theory of acceptance and use of technology (UTAUT)[23], it only focuses on the factors *perceived usefulness* and *perceived ease of use*, not taking into account environmental factors like social influences. This is an advantage when evaluating a new technology without a specific social and economical environment. Furthermore, TAM2 and UTAUT are more complex models which would significantly increase the effort for the user test because more values would need to be measured when using them. They might provide better results but in order to limit the effort of the user test, the TAM was chosen as the basic model for this user test.

The TAM presents a set of six statements each for perceived usefulness and perceived ease of use. The user is asked to evaluate each statement on a likeliness scale with seven steps (extremely likely, quite likely, slightly likely, neither likely nor unlikely, slightly unlikely, quite unlikely, extremely unlikely). The following statements are given to determine the perceived usefulness:

1. Using (system) in my job would enable me to accomplish tasks more quickly.
2. Using (system) would improve my job performance.
3. Using (system) in my job would increase my productivity.
4. Using (system) would enhance my effectiveness on the job.
5. Using (system) would make it easier to do my job.
6. I would find (system) useful in my job.

The following statements are given to determine the perceived ease of use:

1. Learning to operate (system) would be easy for me.
2. I would find it easy to get (system) to do what I want it to do.
3. My interaction with (system) would be clear and understandable.
4. I would find (system) to be flexible to interact with.
5. It would be easy for me to become skillful at using (system).
6. I would find (system) easy to use.

In both of these lists, the name of the system that was evaluated in [4] has been replaced by *(system)*. Other than that, the statements are literally taken from [4].

5.1.3 Adaption of TAM for the user test

The final design for the user test is based on the functionalities and use cases presented in section 5.1.1 and the TAM as presented in the previous section. In the statements about perceived usefulness, the TAM refers to the job of the person filling in the questionnaire. As each test should be executed in the context of a certain use case, this reference to the job is replaced by the respective use case. In place of the system, the functionality as indicated in table 5.1 is filled in. This results in a final list of 48 statements. The statements for the first combination of use case and functionality can be found in table 5.2. The lists of statements for the other seven cases can be created by replacing the name of the functionality and/or the description of the use case. Therefore, the lists of all eight combinations are not included here.

The statements about perceived ease of use do not refer to the user's job. This indicates that the usability does not depend on the task that the user tries to fulfill. Therefore, the evaluation of the perceived ease of use is based solely on the functionality/tool, not on the use case. The functionalities are the same as in the evaluation of the perceived

1. Using the block-based visualization for understanding the structure of the business process architecture would enable me to accomplish tasks more quickly.
2. Using the block-based visualization would improve my performance in understanding the structure of the business process architecture.
3. Using the block-based visualization for understanding the structure of the business process architecture would increase my productivity.
4. Using the block-based visualization would enhance my effectiveness in understanding the structure of the business process architecture.
5. Using the block-based visualization would make it easier to understand the structure of the business process architecture.
6. I would find the block-based visualization useful for understanding the structure of the business process architecture.

Table 5.2: List of statements concerning the perceived usefulness for using the block-based visualization for understanding the structure of the architecture

usefulness and also ARIS is included as a reference tool again. The list of statements is modified similarly to the first one. This results in a list of 36 questions. The questions for the first functionality are listed in table 5.3. Again, the rest of the statements can be created by replacing the name of the functionality/tool and therefore is not reproduced here.

5.2 Setup of the test

This section describes the practical setup of the test. Firstly, a target group is proposed and secondly the steps of the user test session are explained.

5.2.1 User group

The users in the test should be familiar with the topic of business process management and should have at least a basic understanding of the concept of a business process architecture. An individual user will give the most reliable results if he or she practically works with business process architectures on a regular basis. However, in a user test the number of participants is very important and it would be very hard to find sufficiently many practitioners from the BPA field who are willing to participate in the user test because participating in that test does not directly contribute to their work. Therefore, it is proposed to let the students of the Business Process Management course taught at

1. Learning to operate the block-based visualization would be easy for me.
2. I would find it easy to get the block-based visualization to do what I want it to do.
3. My interaction with the block-based visualization would be clear and understandable.
4. I would find the block-based visualization to be flexible to interact with.
5. It would be easy for me to become skillful at using the block-based visualization.
6. I would find the block-based visualization easy to use.

Table 5.3: List of statements concerning the perceived ease of use of the block-based visualization

TU Eindhoven participate in this test. Towards the end of the course, they have sufficient knowledge to qualify as test participant and earlier experiments carried out with students of this course got high numbers of participants (more than 100). The results for each individual user will be less meaningful than for an individual practitioner but the high number of participants strongly increases the significance of the test result.

5.2.2 Steps of the user test

In the beginning, the tools are introduced to the participants and the rest of the test session is outlined. After that the tests for perceived usefulness are done. At the end, the perceived ease of use is tested.

Test introduction

The first step is an introduction to the prototype as well as to the reference tool. The functionalities of the prototype should be shown in a demonstration of five minutes, not including the time the prototype needs to start. Approximately the same amount of time should be spent to introduce the functionalities of ARIS. The exact steps for the introduction of the tools are listed in appendix C.1

Test of the perceived usefulness

To start, the first use case is introduced to the user. To help them get a concrete idea about the use case, they are presented with a number of questions related to that use case. The users are not asked to fill in answers for these questions. Examples for such questions are listed in appendix C.2. Subsequently, the test for each functionality is done. For this, the user is asked to explore the capabilities of the tool in working on that use case. The list of example questions is still accessible. After the user has had time to explore the tool's capabilities concerning the use case, they are asked to answer the list of six questions for the functionality-use case combination that they just tested. After that, the test continues in the same way for the next functionality and the next use case.

Test of the perceived ease of use

After all questions concerning the perceived usefulness have been answered, the user is presented with the questions about the perceived ease of use. First, the questions for the perceived ease of use of the block-based visualization are presented, then the test continues with one functionality after the other. The user is not explicitly asked to use the tool again because they already have experienced using it. However, access to the tool does not need to be denied to the user for they might need to remind themselves about the way the functionality works.

5.3 Limitations

The test plan as presented here, has a number of limitations. Firstly, as mentioned before, the test will be carried out among students who usually lack practical experience in working with business process architectures. This limits the reliability of conclusions about the usefulness of the tool. Conclusions about the usability of the tool are less affected because the judgment of usability does not depend on domain knowledge as much.

The second important limitation is that a research prototype is being compared to an industry-strength tool. This may be a disadvantage for the prototype because it was created with less attention for factors like reactiveness (fast reactions upon user input), error-freeness and an interface that looks attractive.

Furthermore, the user test measures the users' opinion about the usefulness and ease of use instead of measuring these characteristics directly. This can be seen as a limitation because the users' opinion may of course differ from objective values. However, whether such a tool will be of any practical use depends on whether it is accepted by its potential users and they typically base their decisions on their opinion rather than objective measures.

5.4 Current status of the evaluation

As mentioned above, the user test has not yet taken place. The tool has been presented to one of the practitioners who gave input on the use cases. The functionality was demonstrated to him in detail, i.e. all existing functionalities were shown and their intended use was explained. Subsequently, he was asked:

- whether he thinks that the tool would be helpful in his work
- what the most significant added value of the tool would be for him
- what should be improved to increase the tool's value

The answers indicated the following:

- The tool, if it was developed to a sufficient level of maturity, would be helpful fur understanding and restructuring process repositories of big enterprises.
- The biggest added value of the tool is the support it provides in getting an overview over the structure of the process repository and the way it can show the impact of restructuring the architecture.
- More types of relations should be taken into account by the tool, such as shared IT-systems, KPI's and products.
- The tool should be integrated in a process management software because its functionality is too narrow and attracts a too small user group to sustain as an independent software.

The suggestions for further development are taken into account in the suggested future work, section 6.3.

5.5 Summary

A test plan for a user test has been presented. A test according to the plan will establish values for the perceived usefulness and perceived ease of use of the tool. The perceived usefulness indicates whether the users think that the functionality of the tool is chosen appropriately to support the use cases it is supposed to support. The perceived ease of use shows the users' opinion about the usability of the tool. The test has not yet been carried out because of delays in earlier phases of the project.

Presenting the tool to a practitioner who works in the field of business process architectures, has lead to overall positive feedback and a number of attention points for future development and research.

6 Conclusions

The motivation for this research was the fact that big process collections are hard to comprehend because of the high number of processes and relations between these processes that they contain. Visualization has been proposed as a possible solution to this problem. As a starting point for developing requirements for a business process architecture (BPA) visualization tool, literature from the field of software architecture visualization has been studied. From the results of this literature study, requirements for a BPA visualization tool have been concluded. Subsequently, use cases for BPA visualization have been developed based on literature about business process architectures and input from practitioners. Five use cases have been found.

In the next step, a prototype of a BPA visualization tool has been developed. Realizing all five use cases within the scope of the project was not feasible and therefore two important use cases have been selected for the tool to support. The prototype offers multiple visualization techniques and a number of analysis techniques to support the visualization. Finally, a test plan has been proposed for the validation of the prototype. The proposed user test has not yet been executed because of delays in the previous steps of the project.

6.1 Results

The main goal of this research was the *development of an easy-to-use and useful graphical visualization of process architectures*. Two sub-goals have been identified: *identification of relations between processes on the same hierarchical level or on different hierarchical levels* and *identification of possibilities to abstract from the details of a process architecture in order to obtain multiple levels of detail.*

The main goal has been addressed by the development of a prototype. It addresses two of the five use cases that have been identified for the visualization of business process architectures. The usefulness and usability have not yet formally been tested.

The first sub-goal, the identification of relations in the business process architecture, has been addressed by a dependency detection in the prototype. That dependency detection focusses entirely on direct dependencies between processes based on event labels. The existence of more types of relations is known to the author. For example, the exchange of messages and common use of shared resources are further relations in a process architecture that can influence the performance of processes.

The second sub-goal, abstraction to obtain multiple levels of details, has been addressed in the prototype in multiple ways. The first technique for abstracting from details is showing only a single level of the hierarchical structure at a time. This technique abstracts by not showing all elements of the architecture but also by displaying relations on an abstract level. Secondly, there is a possibility to abstract from the relations in a process architecture by only showing strong edges. This second abstraction technique allows the user to select an appropriate level of detail. A third technique of abstraction is the automatic grouping. It builds on the first technique, only showing one level of the hierarchy at a time, and extends it by adding new dynamic levels of hierarchy. The user can also influence the level of detail using this technique: Not only can they select whether or not to use it, but there is also a possibility to influence the way how groups are created.

6.2 Limitations

This section outlines limitations in the developed prototype as well as in the research approach.

6.2.1 Limitations of the prototype

The prototype only addresses two out of five use cases that have been developed in section 3.1. This choice was made in order to be able to address two use cases in detail instead of only touching on all five. Furthermore, the prototype only works with a single type of relations although there are more types of relations in a process architecture. The relations are identified based on event labels which may lead to false-positives, i.e. the identification of relations that actually are no relations.

Except for these functional limitations, the prototype may still contain errors because it has not been tested rigorously. The graphical design of the prototype does not live up to the standards of current commercial software. Moreover, starting up the prototype and certain actions within it take longer than most users would be willing to accept.

6.2.2 Methodological limitations

In the first step of the research, literature from the field of software architecture visualization has been studied. This choice was made because that field is closely related to business process architecture visualization. However, studying more fields might have improved the results.

The use cases were developed based on literature from the field of business process architectures and the input of practitioners but also influenced by the own insights of the author. The questions from literature on which the use cases have been based, were not selected rigorously enough to guarantee the completeness of the use cases. Furthermore, bringing in the own insight of the author may have negatively influenced the correctness of the use cases, but it was tried to moderate that effect by discussing the suggested use cases with practitioners.

The prototype has not yet been subjected to a user test which means that it is not certain yet that it fulfills its goal of making business process architectures more understandable and helping with redesigning them.

6.3 Future work

The research presented in this work forms a foundation for further investigation into the visualization and analysis of business process architectures. The following sections point out future fields of work.

6.3.1 Validation

As pointed out before, a user test still needs to be carried out. This is seen as the most important future step because it will give feedback about the quality of the work done until now. The user test will give results for perceived usefulness and perceived usability of each tested functionality.

If a functionality gets a high score for perceived usefulness, it means that this functionality is a valuable addition that should be kept and further developed. A low score in that criterion points at the opposite. It does not necessarily mean that the functionality is useless and should not be considered anymore at all but it calls for an investigation on the reasons for the low perceived usefulness.

A high score for perceived ease of use indicates that the users perceive the interaction mechanisms as intuitively usable. The mechanisms should be further developed in the same direction. A low perceived ease of use indicates that the interaction mechanisms are not intuitive. Findings from human technology interaction should be taken into account to redesign the interaction mechanisms.

6.3.2 Requirements and use cases

To improve the requirements that have been found and increase the confidence in them, literature from more fields than only software architecture visualization should be studied.

A comprehensive overview of questions that can be answered with business process architectures, should be created by means of surveying published literature about business process architectures. This can help finding more use cases.

6.3.3 Types of relations

At the moment, the only type of relation that is represented in the prototype, are direct dependencies. There are several ways in which processes can interact or be related:

Document or Information exchange: Processes can run in parallel and exchange information. In some cases, this may be captured by events in EPCs, but in other cases or other modeling notations, further analysis may be necessary to identify these relations.

Roles and People: Processes and activities that are executed by the same people or people with the same roles, influence each other in a way that is not always obvious. Two processes that share human resources, may negatively influence each other's performance.

IT-systems and other resources: Just like the shared use of human resources, also the shared use of other resources may impact processes' performance. Moreover, shared use of an IT-System, production machinery, etc. may point to the processes fulfilling similar functions.

Products and services: Processes related to the same product/service are not necessarily part of the same group in the hierarchical structure, but they still form a logical group throughout the organization. When making changes to a product or service, it is essential to identify the processes that are affected.

By incorporating these relations into the visualization and analysis, the picture of the process architecture becomes more complete. This also means that the number of relations will grow significantly and more possibilities for filtering will be needed.

6.3.4 Coverage of more use cases

The current prototype only focuses on two out of five use cases identified in chapter 3. The rest of the use cases should be supported by a future version of the tool for it to increase the value for the user.

Analysis of the quality of a process architecture

There are two major steps in realizing this use case. The first one is the identification of patterns that indicate problems. These patterns can be derived from the practical experience of experts in the field of BPAs and from research results in the area. The second step is the automatic recognition of these patterns. This step requires a good formal representation of each of the patterns.

Identification of a client's pathway

Supporting this use case mainly requires an additional visualization technique. The dependencies required to support it are already analyzed by the current prototype.

Implications of a local change

The additional types of relations listed above may help in identifying the impacts of a local change. Once they are incorporated, this use case mainly requires a way of presenting the processes affected by a change.

6.3.5 Improvement of the prototype to a higher maturity

At the moment, the tool opens only one format of a business process repository. In order to be more useful, it should be able to open different file formats. To improve the workflow of using the tool, it should finally be integrated in BPM tools. This would enable the user to use the functionality of the current prototype and any future improvements in the environment that they usually use for modelling and organizing their business processes. Furthermore, a rigorous test for errors in the prototype should be carried out in order to improve the reliability of the tool. Moreover, measures should be taken to improve its performance.

6.4 Summary

The goals set in the beginning have been partially achieved and points for future improvement have been pointed out. A user test is the next major step in the development of business process architecture visualizations.

A Literature study protocol

This appendix presents a protocol for the literature study presented in short in chapter 2. The first part describes details on the search strategy. In the second part, all search results are listed.

A.1 Search strategy

The literature study was carried out using a number of online search engines for scientific resources. Each search result was checked for its match with the inclusion and exclusion criteria. Those publications that matched the inclusion criteria and did not match the exclusion criteria, are discussed in chapter 2.

Considering the high numbers of search results that are typically found for a certain query in the resources used, it is not feasible to assess every single result for its relevance. However, these search engines sort the results in a way that the most relevant results are the first ones. This means that it is legitimate to stop evaluating the results after ten irrelevant results in a row. Hence, results that followed ten irrelevant results in a row, were considered not to be found.

A.1.1 Search resources

The first resource used in the literature search was the search engine *Google Scholar*. It searches the databases of most major publishers and therefore is a good starting point for finding relevant literature. Furthermore, the search tool *Focus* of the library of TU Eindhoven was used. It covers all online and offline resources available to that library. After these publisher-independent resources had been consulted, the search tools of some important publishers have been used to complete the search. *Springer Link* (Springer) and *IEEE Xplore* (IEEE) have been searched. All the content covered by the last three resources is also searched by *Google Scholar*. However, they were consulted to avoid results that depend on only a single search resource.

A.1.2 Search terms and inclusion and exclusion criteria

The search term used for all the search engines was "Software Architecture Visualization" (without the quotes). Results were considered relevant if they met at least one of the inclusion criteria and none of the exclusion criteria. These criteria were checked based on the abstract of the publication. If it was clear from the title of a publication that it is not relevant, then the abstract was not consulted.

The following inclusion criteria were used:

- The publication introduces and describes a visualization technique for software architectures.
- The publication describes one or more existing visualization techniques for software architectures.

The following exclusion criteria were used:

- The article describes technical details of a visualization techniques, but not an overall visualization technique.
- The article describes a specific application of software architecture visualization and not the visualization technique itself.

As the goal of this literature study is the identification of principles in software architecture visualization that can possibly be reused for business process architecture visualization, descriptions of visualization techniques are needed. Implementation details are not the subject of this work and neither are specific tasks and workflows in software engineering.

A.2 Results of the literature search

This section shows the results per search engine. The tables show the name, authors and year of each publication and indicate whether it is relevant or not. If the relevance column is filled with *Duplicate*, this means that the publication is relevant but has already been found in another resource.

A.2.1 Google Scholar

Address of the resource: `http://scholar.google.com`

Title	Authors	Year	Relevance
A classification and comparison framework for software architecture description languages	N Medvidovic, RN Taylor	2000	No
Pattern Oriented Software Architecture: On Patterns and Pattern Languages	F Buschmann, K Henney, DC Schmidt	2007	No
Scenario-based analysis of software architecture	R Kazman, G Abowd, L Bass, P Clements	1996	No
Software architecture: foundations, theory, and practice	RN Taylor, N Medvidovic, EM Dashofy	2009	No
Software architecture visualization: An evaluation framework and its application	K Gallagher, A Hatch, M Munro	2008	Yes
Playing detective: Reconstructing software architecture from available evidence	R Kazman, SJ Carrière	1999	No
ArchJava: connecting software architecture to implementation	J Aldrich, C Chambers, D Notkin	2002	No
Abstractions for software architecture and tools to support them	M Shaw, R DeLine, DV Klein, TL Ros	1995	No
A software system for interactive and quantitative visualization of multidimensional biomedical images.	RA Robb, DP Hanson	1991	No
Visualization of areas of interest in software architecture diagrams	H Byelas, A Telea	2006	No
The software architecture of a real-time battlefield visualization virtual environment	S Julier, R King, B Colbert, J Durbin	1999	No
Structural manipulations of software architecture using Tarski relational algebra	RC Holt	1998	No
Software architecture reconstruction: A process-oriented taxonomy	S Ducasse, D Pollet	2009	No

Title	Authors	Year	Relevance
Using visualization for architectural localization and extraction	D Jerding, S Rugaber	1997	No
Architectural styles and the design of network-based software architectures	RT Fielding	2000	No
Tool support for architecture analysis and design	R Kazman	1996	No
A Network Software Architecture for Large Scale Virtual Environments.	MR Macedonia	1995	No
3D visualization of software architectures	L Feijs, R De Jong	1998	Yes
An open graph visualization system and its applications to software engineering	ER Gansner, SC North	2000	No
Gase: visualizing software evolution-in-the-large	R Holt, JY Pak	1996	No
Shrimp views: An interactive environment for exploring java programs	MA Storey, C Best, J Michand	2001	Yes
Towards empirically validated software architecture visualization	J Knodel, D Muthig, M Naab, D Zeckzer	2006	Yes
SAVE: Software architecture visualization and evaluation	S Duszynski, J Knodel, M Lindvall	2009	No
A software architecture reconstruction method	GY Guo, JM Atlee, R Kazman	1999	No
Symphony: View-driven software architecture reconstruction	A van Deursen, C Hofmeister, R Koschke, L Moonen, C Riva	2004	No
Visualization-based analysis of quality for large-scale software systems	G Langelier, H Sahraoui, P Poulin	2005	Yes
Software evolution observations based on product release history	H Gall, M Jazayeri, R Klösch, G Trausmuth	1997	No
The weather research and forecast model: Software architecture and performance	J Michalakes, J Dudhia, D Gill, T Enderson, J Klemp, W Skamarock, W Wang	2004	No

Title	Authors	Year	Relevance
Secrets from the monster: Extracting Mozilla's software architecture	MW Godfrey, EHS Lee	2000	No
A relational approach to support software architecture analysis	L Feijs, R Krikhaar, R Van Ommering	1998	No
On the use of visualization to support awareness of human activities in software development: a survey and a framework	MAD Storey, D Čubranić, DM German	2005	No
A hybrid process for recovering software architecture	V Tzerpos, RC Holt	1996	No
Pattern visualization for software comprehension	R Schauer, RK Keller	1998	No
An open software architecture for virtual reality interaction	G Reitmayr, D Schmalstieg	2001	No
Architecture-based runtime software evolution	P Oreizy, N Medvidovic, RN Taylor	1998	No
Object-oriented software development	A Eliëns	1995	No
Beyond the Renderer: Software Architecture for Parallel Graphics and Visualization.	TW Crockett	1996	No

A.2.2 TU Eindhoven library

The search tool *Focus* of the library of TU Eindhoven was used. It can be found at `http://library.tue.nl/focus/`

Title	Authors	Year	Relevance
3D visualization of software architectures	"L Feijs, R de Jong"	1998	Duplicate
Towards empirically validated software architecture visualization	"J Knodel, D Muthig, M Naab, D Zeckzer"	2006	Duplicate
A Systematic Analysis of Software Architecture Visualization Techniques	Z Sharafi	2011	No

Title	Authors	Year	Relevance
Extreme Scaling of Production Visualization Software on Diverse Architectures	"H Childs, D Pugmire, S Ahern, B Whitlock, M Howison, Prabhat, G Weber, W Bethel"	2011	No
Software architecture visualization: An evaluation framework and its application	"K Gallagher, A Hatch, M Munro"	2008	Duplicate
Connecting research and practice: an experience report on research infusion with software architecture visualization and evaluation	"M Lindvall, WC Stratton, DE Sibol, C Ackermann, WM Reid, D Ganesan, D McComas, M Bartholomew, S Godfrey"	2012	No
Visualization of software architecture graphs of Java systems: managing propagated low level dependencies	"L Schrettner, L Fülöp, R Ferenc, T Gyimóthy"	2010	No
EvoSpaces Visualization Tool: Exploring Software Architecture in 3D	"S Alam, P Dugerdil"	2007	Yes
Weka?STPM: a Software Architecture and Prototype for Semantic Trajectory Data Mining and Visualization	"V Bogorny, H Avancini, BC de Paula, CR Kuplich, LO Alvares"	2011	No
Texture-based visualization of metrics on software architectures	"H Byelas, A Telea"	2008	No
Visualization of areas of interest in software architecture diagrams	"H Byelas, A Telea"	2006	No
A new software architecture for parallel computation and visualization	DM Butler	1992	No
Coverity Architecture Analyzer to Deliver Advanced Visualization of Software Systems: Product Now Leverages Coverity's Patented Software DNA Map	Anonymous	2008	No
Software visualization	S Diehl	2005	No
Project visualization for software	KT Hansen	2006	No

Title	Authors	Year	Relevance
Highly Configurable Software Architecture Framework for Acquisition and Visualization of Biometric Data	J Stelovsky	2007	No
The visualization handbook	"CR Johnson, CD Hansen"	2005	No
Visualization Criticism	"R Kosara, F Drury, LE Holmquist, DH Laidlaw"	2008	No

A.2.3 Springer Link

Address of the resource: `http://link.springer.com`

Title	Authors	Year	Relevance
Connecting research and practice: an experience report on research infusion with software architecture visualization and evaluation	"M Lindvall, WC Stratton, DE Sibol, C Ackermann, WM Reid, D Ganesan, D McComas, M Bartholomew, S Godfrey"	2012	No
Highly Configurable Software Architecture Framework for Acquisition and Visualization of Biometric Data	J Stelovsky	2007	No
Understanding Architecture through Structure and Behavior Visualization	"D Heuzeroth, W Löwe"	2003	No
An experiment on the role of graphical elements in architecture visualization	"J Knodel, D Muthig, M Naab"	2008	No
Measurement and visualization of the architecture of an adult tree based on a three-dimensional digitising device	"H Sinoquet, P Rivet"	1997	No
A Visualization System for the Comfort Analysis of Modular Architecture: A Case Study	"D Kim, S Lee, SA Kim"	2012	No
Model Generalization and Methods for Effective Query Processing and Visualization in a WebService/Client Architecture	"M de Vries, P van Oosterom"	2007	No

Title	Authors	Year	Relevance
Visualization of the Implications of a Component Based ICT Architecture for Service Provisioning	"R Wagenaar, M Janssen"	2002	No
GVis: A Java-Based Architecture for Grid Enabled Interactive Visualization	"Y Zhao, W Chen, Y Qiu, J Shi"	2004	No
An object-oriented architecture for applications of scientific visualization and mathematical modeling	"VA Semenov, PB Krylov, SV Morozov, OS Tarlapan"	2000	No
On porting software visualization tools to the web	"M D'Ambros, M Lanza, M Lungu, R Robbes"	2011	No
Visualization of Scientific Data for High Energy Physics: Basic Architecture and a Case Study	CE Vandoni	1994	No
Three-Dimensional Visualization of the Molecular Architecture of Cell–Cell Junctions In Situ by Cryo-Electron Tomography of Vitreous Sections	"A Al-Amoudi, AS Frangakis"	2013	No
Grapheur: A Software Architecture for Reactive and Interactive Optimization	"M Brunato, R Battiti"	2010	No

A.2.4 IEEE Xplore

Address of the resource: `http://ieeexplore.ieee.org/`

Title	Authors	Year	Relevance
Software Architecture Visualization: An Evaluation Framework and Its Application	K Gallagher, A Hatch, M Munro	2008	Duplicate
A Systematic Analysis of Software Architecture Visualization Techniques	Z Sharafi	2011	Duplicate
SAVE: Software Architecture Visualization and Evaluation	S Duszynski, J Knodel, M Lindvall	2009	Duplicate
SoftArchViz: A Software Architecture Visualization Tool	AP Sawant, N Bali	2007	No

Title	Authors	Year	Relevance
Let's enforce a simple visualization rule in Software Architecture	BH Wu	2011	No
Communicating Software Architecture using a Unified Single-View Visualization	T Panas, T Epperly, D Quinlan, A Saebjornsen, R Vuduc	2007	Yes
An Integrated Approach of AHP-GP and Visualization for Selection of Software Architecture: A Framework	KD Babu, ; P Govindarajulu, AR Reddy, ANA Kumari	2010	No
Understanding Software Architectures by Visualization–An Experiment with Graphical Elements	J Knodel; D Muthig; M Naab	2006	No
An extensible and integrated software architecture for data analysis and visualization in precision agriculture	L Tan; R Haley, R Wortman, Q Zhang	2012	No
High-level static and dynamic visualisation of software architectures	J Grundy, J Hosking	2000	No
The software architecture of a real-time battlefield visualization virtual environment	S Julier, R King, B Colbert, J Durbin, L Rosenblum	1999	No
A Framework for Software Architecture Visualisation Assessment	K Gallagher, A Hatch, M Munro	2005	No
A model and software architecture for search results visualization on the WWW	O Alonso, R Baeza-Yates, R.	2000	No
Software Visualization in the Context of Service-Oriented Architectures	S Eicker, T Spies, C Kahl	2007	No
Extreme Scaling of Production Visualization Software on Diverse Architectures	H Childs, D Pugmire, S Ahern, B Whitlock, M Howison, M Prabhat, G Weber, W Bethel	2011	No
Towards pie tree visualization of graphs and large software architectures	M Samia, M Leuschel	2009	Yes

Title	Authors	Year	Relevance
Architecture of a software system for scientific data visualisation	M Kolodnytsky, A Kovalchuk	2001	No
EvoSpaces Visualization Tool: Exploring Software Architecture in 3D	S Alam, P Dugerdil	2007	Duplicate
An Architecture to Support Model Driven Software Visualization	RI Bull, MA Storey, JM Favre, M Litoiu	2006	No
Extreme Scaling of Production Visualization Software on Diverse Architectures	H Childs, D Pugmire, S Ahern, B Whitlock, M Howison, Prabhat, GH Weber, EW Bethel	2010	No
The SAVE Tool and Process Applied to Ground Software Development at JHU/APL: An Experience Report on Technology Infusion	WC Stratton, DE Sibol, M Lindvall, P Costa	2007	No
Browsing and searching software architectures	S Elliott Sim, CLA Clarke, RC Holt, AM Cox	1999	No
Technology Infusion of SAVE into the Ground Software Development Process for NASA Missions at JHU/APL	WC Stratton, DE Sibol, M Lindvall, P Costa	2007	No
Tool support for reverse engineering multi-lingual software	TD Hendrix, JH Cross, LA Barowski, KSMathias	1997	No
Navigating software architectures with constant visual complexity	W Li, P Eades, SH Hong	2005	Yes
Developing an approach for analyzing and verifying system communication	WC Stratton, DE Sibol, M Lindvall, C Ackermann, S Godfrey	2009	No
An architectural connectivity metric and its support for incremental re-architecting of large legacy systems	RJ Bril, A Postma	2001	No
Visualizing metrics on areas of interest in software architecture diagrams	H Byelas, A Telea	2009	No

Title	Authors	Year	Relevance
Visualization as an aid for assessing the mission impact of information security breaches'	A D'Amico, S Salas	2003	No
A Study on the Application of the PREViA Approach in Modeling Education	M Schots, CSC Rodrigues, C Werner, L Murta	2010	No
Model-Based Validation & Verification Integrated with SW Architecture Analysis: A Feasibility Study	I Morschhauser, M Lindvall	2007	No
A two-phase process for software architecture improvement	R Krikhaar, A Postma, A Sellink, M Stroucken, Cverhoef	1999	No
Software evolution based on software architecture	H Hua	2004	No
Software impact analysis in a virtual environment	SA Bohner, D Gracanin	2003	No
OSIRIX: An Open Source Platform for Advanced Multimodality Medical Imaging	OR Faha	2006	No

B Documentation of the prototype

This appendix is an addition to chapter 4 and provides more in-depth information about the implementation of the prototype. The target audience of this information are those who want to further develop the prototype or who are interested in the details of the implementation. It is recommended to first read the general descriptions in chapter 4 before consulting this appendix.

In the first sections, an overview of the most important interaction patterns in the prototype will be given. After that, some important classes will be described in more detail. For an in-depth description of all classes, including their fields and methods, the API documentation (Javadoc) should be consulted. At the end of this appendix, an overview of the external libraries used in the prototype, is given.

B.1 Interaction sequences

B.1.1 Starting of the application

When the application is started, the first class in action is *MainController*. It coordinates the whole startup phase of the application as can be seen in the sequence diagram shown in figure B.1. It creates a *ProcessArchitecture* object, i.e. the object that holds the central model for the whole application. Subsequently, it creates an *ImportController* object whose responsibility it is to fill the model from an external source.

The *ImportController* uses an *SAPImporter* object to load an EPML-export of the SAP reference model. For reading the EPML data, an EPC implementation by Remco Dijkman is used. As the export of the reference model does not contain explicit dependencies between processes, they need to be detected which is done by an instance of *DependencyAnalysis*. This step completes the import of the model and control is passed back to the *MainController*. The internal workings of *SAPImporter* and *DependencyAnalysis* are not shown in the sequence diagram. They are discussed in sections B.2.5 and B.2.4.

Subsequently, the application form, an instance of *MainForm* is created. Control over the form is taken by a newly created *ExplorationController*, i.e. a controller that allows navigating the process architecture using different visualizations. The *MainController* finishes its work by making the form visible and adding a listener that will do some finalizing work when the window is closed in order to ensure a clean exit of the application.

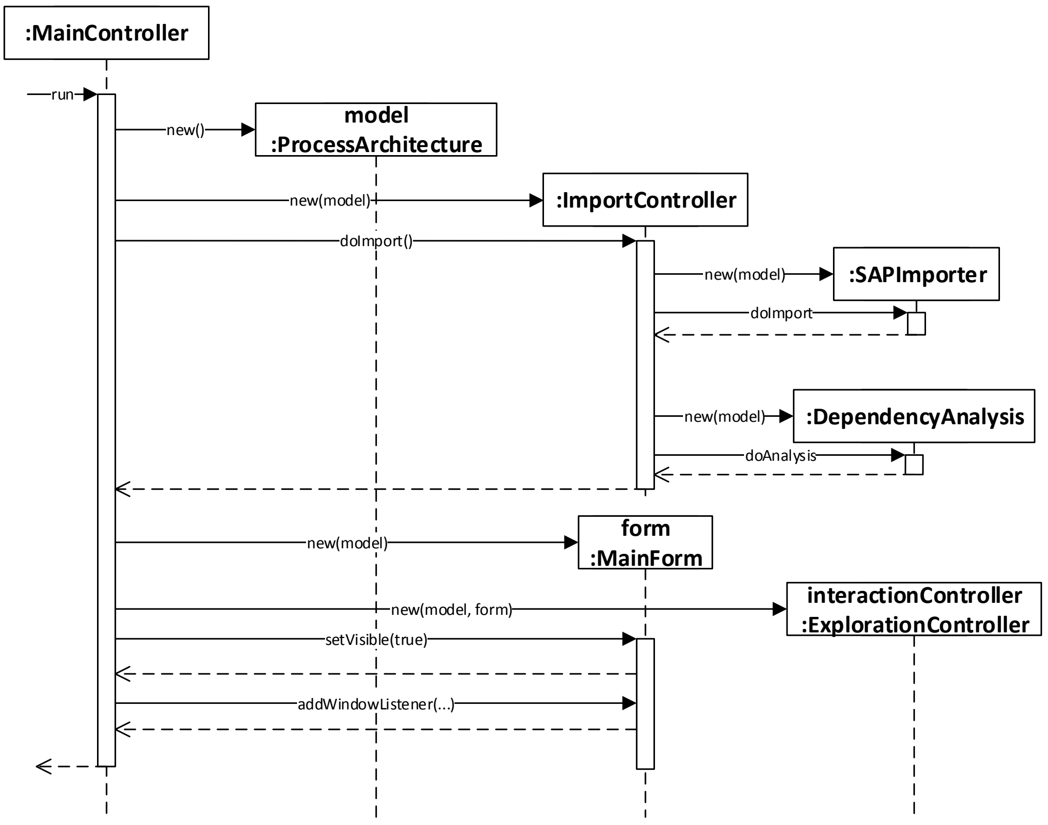
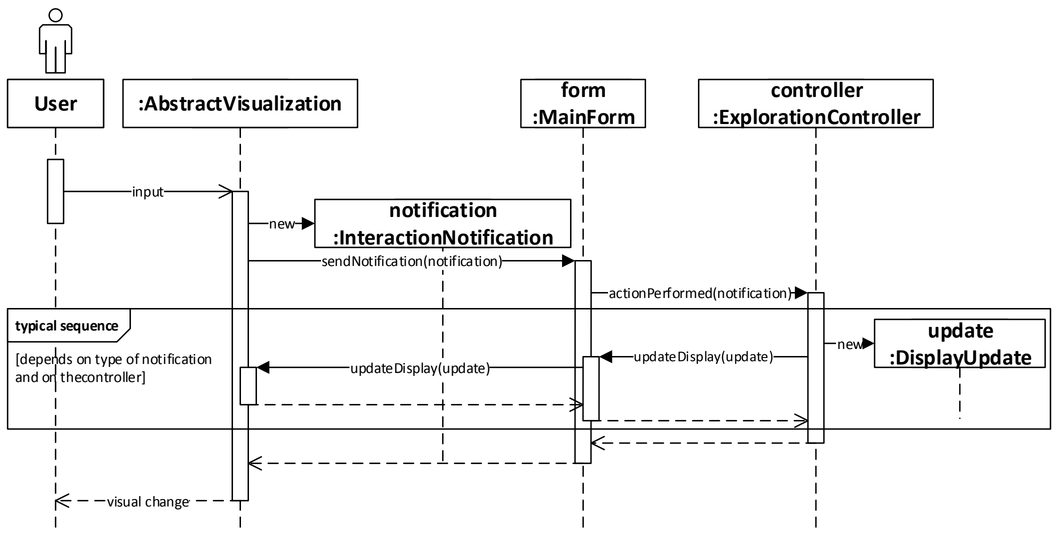


Figure B.1: Interaction sequence during startup of the application



User
:AbstractVisualization
form :MainForm
controller :ExplorationController
input
new
notification :InteractionNotification
sendNotification(notification)
actionPerformed(notification)
typical sequence
[depends on type of notification and on thecontroller]
new
update :DisplayUpdate
updateDisplay(update)
updateDisplay(update)
visual change

Figure B.2: Interaction sequence upon user input

B.1.2 Handling of user interactions

While the application is running, the user has the possibility to interact with several user interface components. All of them, except a separate search window, are arranged on the *MainForm*. All user interface components interact with the controller via the *MainForm*. An interaction sequence is shown in the sequence diagram in figure B.2. The interface component that the user interacts with, may be an instance of a subclass of *AbstractVisualization*. When it receives user input, it may do some internal processing, but it should not directly change its state upon user input. This would undermine the principle that the state of the application is changed only by the controller. Violation of that principle may lead to unexpected behavior and inconsistencies between different user interface components.

The user interface component creates a *notification*, i.e. an instance of a subclass of *InteractionNotification*. This object will be sent to the *form* which sends it on to the *controller*. This indirection decreases the number of connections between user interface and controller.

The actions of the *controller* upon receiving the *notification* depend on the type of the *notification*, on the data it carries and on the state of the controller. The controller may initiate changes in the model, change its state, exchange UI components, run an analysis, etc. Typically, the last action is sending an update to the user interface to display any changes to the user. The diagram shows the interaction sequence for sending such an update. The *controller* creates an instance of a subclass of *DisplayUpdate*. That is an object that indicates the sort of update through its type and typically holds a small amount of data related to the update. The *controller* sends the *update* to the *form*. The *form* sends it to all visualization components that are currently active.

Each visualization component decides on its own what to do upon receiving an update. The actions typically depend on the type of *update* and the data it holds. Some visualization components will ignore certain types of updates because they are not relevant for the specific visualization technique. For example, an *EdgeThresholdUpdate* will be ignored by the *TreeVisualization*: the *EdgeThresholdUpdate*sets a minimum strength for edges to be displayed, but the *TreeVisualization* does not display edges at all, hence the update is irrelevant. When a visualization component handles an update, it may cause an *InteractionNotification* to be sent because some change is done that may also have been done by a user. Such notifications might cause infinite loops of notifications and updates if not taken care of. Therefore, the *ExplorationController* does not handle two notifications at the same time, i.e. as soon as it receives an *InteractionNotification*, it sets an indicator that will prevent it from processing another one. The indicator is only removed once the *actionPerformed* method has been completed.

B.2 Class structure

Figure B.3 which has already been shown in section 4.2, displays the top level package structure of the prototype. In the following sections, the most important classes from these packages are explained.

B.2.1 The *controller* package

As described before, the controllers are objects that coordinate all actions in the application and control the status of the program. At the moment, there are three controllers, as can be seen in figure B.4: the *MainController*, the *ImportController*, and the *ExplorationController*. Apart from those three classes, there is an interface *InteractionController* and there are two enumeration classes for specifying visualization techniques and comparison techniques. The events package contains the notifications and updates sent between controllers and the user interface.

The functionality of the *MainController* has been comprehensively covered in section B.1.1: it controls the start of the application. The functionality of *ImportController* has been covered in that same section: it coordinates *SAPImporter* and *DependencyAnalysis* to load the model. *ExplorationController* and the *events* package will be covered in the following two paragraphs.

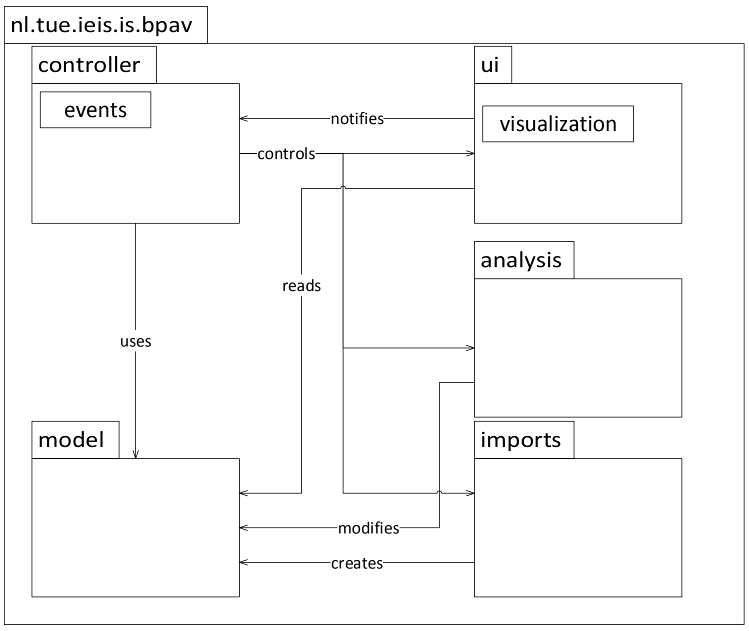


Figure B.3: Top-level package structure

The *events* package

As explained before, there are two different types of events: notifications and updates. Notifications are classes of objects that can be sent from user interface components to the controller in order to inform it about user input. Updates go the other way around, i.e. they can be sent from the controller to the user interface in order to trigger display changes. All notifications are subclasses of the abstract *InteractionNotification* class. All updates are subclasses of *DisplayUpdate*. Table B.1 lists all notifications with a short summary of their meanings. Table B.2 lists all updates with short summaries of their meanings. Some updates have a meaning that is very similar to a certain notification, e.g. *EdgeThresholdUpdate* is very similar to *EdgeThresholdChangeNotification*. Nevertheless, they are different classes. The controller autonomously decides whether it will directly translate such a notification into its corresponding update or do other processing steps.

The *ExplorationController* class

ExplorationController allows the user to navigate through a process architecture using multiple visualization techniques. It does so by processing notifications sent from the user interface components and sending updates to the visualization components.

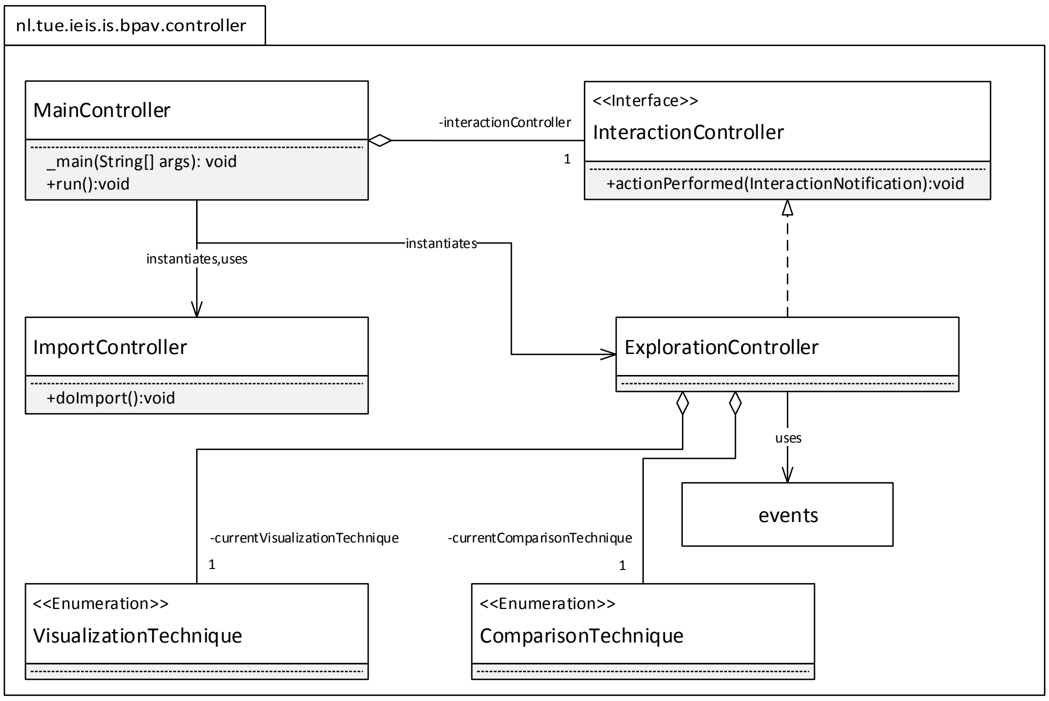


Figure B.4: Class diagram of the *controller* package

Class name	Explanation
ComparisonTechniqueChangeNotificiation	A new technique for comparing two models (currently table or two graphics) has been selected.
CouplingThresholdNotification	The minimum coupling for automatic grouping has been set or automatic grouping has been switched off.
EdgeThresholdChangeNotification	The minimum strength for an edge to be displayed has been changed.
ModelSelectionNotification	The model to be displayed (currently original model, automatically clustered model, or a comparison of both) has been selected.
NodeSelectionNotification	A node has been selected.
SearchResultSelectionNotification	A search result has been selected.
VisualizationTechniqueSelectionNotification	The main visualization technique (currently connectogram or block-based visualization) to be used has been selected.
ZoomNotification	The zoom level has been changed.

Table B.1: List of all notifications

Class name	Explanation
DisplayNodeUpdate	The current root node has changed. The content of that node should be shown.
EPCModelUpdate	The EPC model contained in this update should be displayed.
EdgeThresholdUpdate	Change of the minimum strength for an edge to be displayed
HighlightUpdate	Contains a set of nodes that should be highlighted and indicates whether any previous highlighting should be cleared.
ModelChangeUpdate	Indicates that the *ProcessArchitecture* has changed and transports the new model.
ZoomUpdate	The zoom level should be changed to the level stated in the update. Note that currently no visualization component supports zooming.

Table B.2: List of all updates

Notification class	Action
ComparisonTechniqueChangeNotificiation	Sets the current comparison technique and adds the appropriate component to the *MainForm*. If not done before, runs automatic clustering.
CouplingThresholdNotification	Activates automatic grouping with the set minimum coupling inside groups or deactivates automatic grouping, depending on the notification data.
EdgeThresholdChangeNotification	An *EdgeThresholdUpdate* is sent.
ModelSelectionNotification	Changes the currently used visualization technique or switches to comparison mode. Replaces the appropriate UI components in the *MainForm*. If the new selection is comparison, then it runs the automatic clustering if not done before.
NodeSelectionNotification	If the selected node is a leaf node with an associated EPC: shows that EPC. Otherwise, sends a *DisplayNodeUpdate* with the selected node. If the selected node is *null*, the node in the *DisplayNodeUpdate* will be the parent of the current root node or the root of the model if the current root has no parent.
SearchResultSelectionNotification	If the selection is empty, sends an empty *HighlightUpdate* (clear highlighting). Otherwise sends a *DisplayNodeUpdate* with the parent of the selected result (if the result has a parent) and a *HighlightUpdate* with the selected result node.
VisualizationTechniqueSelectionNotification	Changes the current visualization technique. Replaces the appropriate components in the MainForm.
ZoomNotification	Sends a *ZoomUpdate* with the desired zoom level.

Table B.3: Actions of the *ExplorationController* by notification type

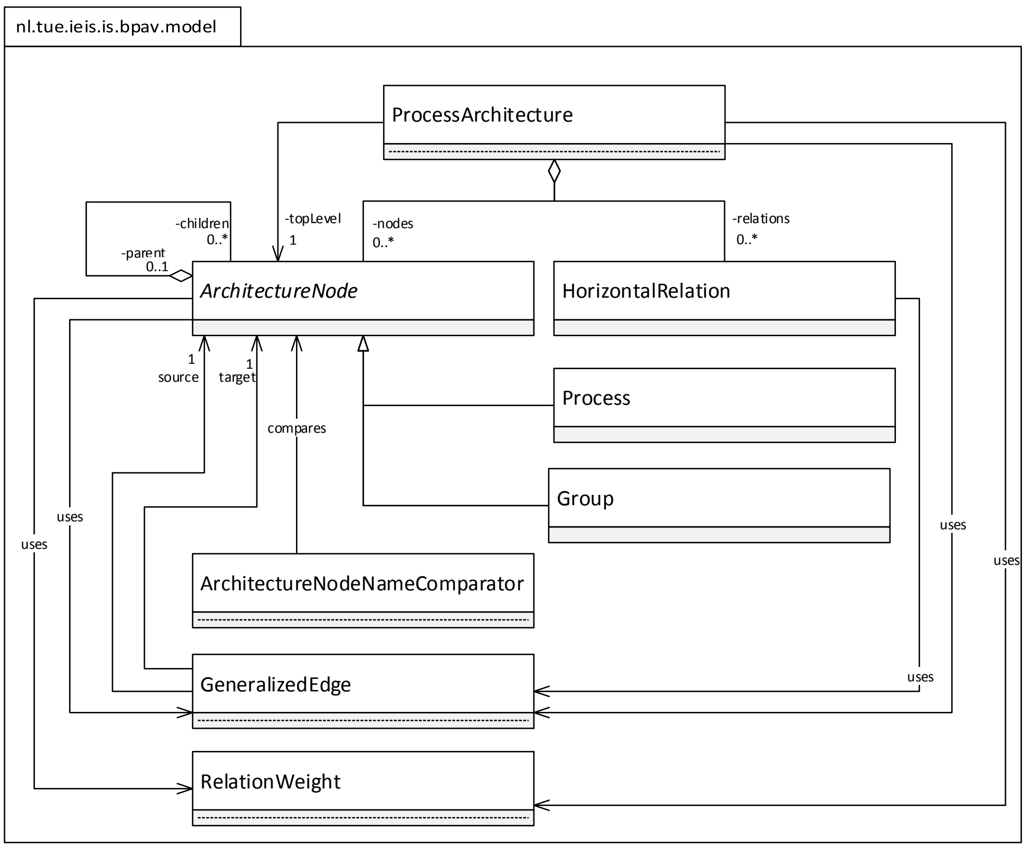


Figure B.5: Class diagram of the *model* package

B.2.2 The *model* package

The *model* package contains classes for representing a business process architecture. Its classes are shown in figure B.5.

ProcessArchitecture represents a business process architecture and contains *ArchitectureNode*s and *HorizontalRelation*s, i.e. architecture elements (processes and groups) and dependencies. The name *HorizontalRelation* emphasizes that hierarchical relations are not represented as relations here, but as a tree structure: Every *ArchitectureNode* can have children, i.e. other *ArchitectureNode*s that are below it in the hierarchy.

Except for these classes that directly represent the BPA and its elements, the *model* package contains three more classes. *ArchitectureNodeNameComparator* is used for sorting *ArchitectureNode* objects in alphabetical order.

GeneralizedEdge is an abstract representation of a relation. It represents an edge as visible in a visualization that only shows one hierarchical level at a time. Each relation belongs to exactly one *GeneralizedEdge*: If the relation's source and target are the same, then that node is also source and target of the *GeneralizedEdge*. Otherwise, source and target are two different nodes that have the same parent node. The source node of the *GeneralizedEdge* is an ancestor of the source node of the relation and the target node of the *GeneralizedEdge* is an ancestor of the target node of the relation.

RelationWeight is used for assigning a weight from 0 to 1 to a *GeneralizedEdge*. The weight depends on the number of relations that the *GeneralizedEdge* represents and is calculated using a polynomial function. The parameters of the function are chosen such that the strongest edge in the model gets weight 1 and 70% of the edges get weights between 0 and 0.95. This system avoids giving very low weights to most edges because of a few outliers. The exact values have been chosen empirically and are not necessarily optimal.

B.2.3 The *ui* package

The *ui* package contains all graphical components. Its classes are shown in figure B.6. *MainForm* is the main window of the application. It holds most of the graphical components. It holds one or two visualization components at a time and also contains the *ConfigurationPanel*. The *SearchForm* is a separate window that shows search results.

The subpackage *visualization* contains all visualization components. All of them are subclasses of *AbstractVisualization* which itself is a subclass of *JPanel*. *JGraphXVisualization* is an abstract class for visualization classes that use the library JGraphX[17].

B.2.4 The *analysis* package

The *analysis* package contains classes that analyze and/or change the model. Its classes can be seen in figure B.7. The classes *ProcessGroup*, a group of processes, and *WeightedUndirectedEdge* are helpers used by *GreedyGrouping*.

The *Clusterer* class

Clusterer is used to restructure the process architecture. To achieve this, it uses the library Weka[12]. It uses a KMeans clusterer to divide the processes into groups based on the labels in their EPC models.

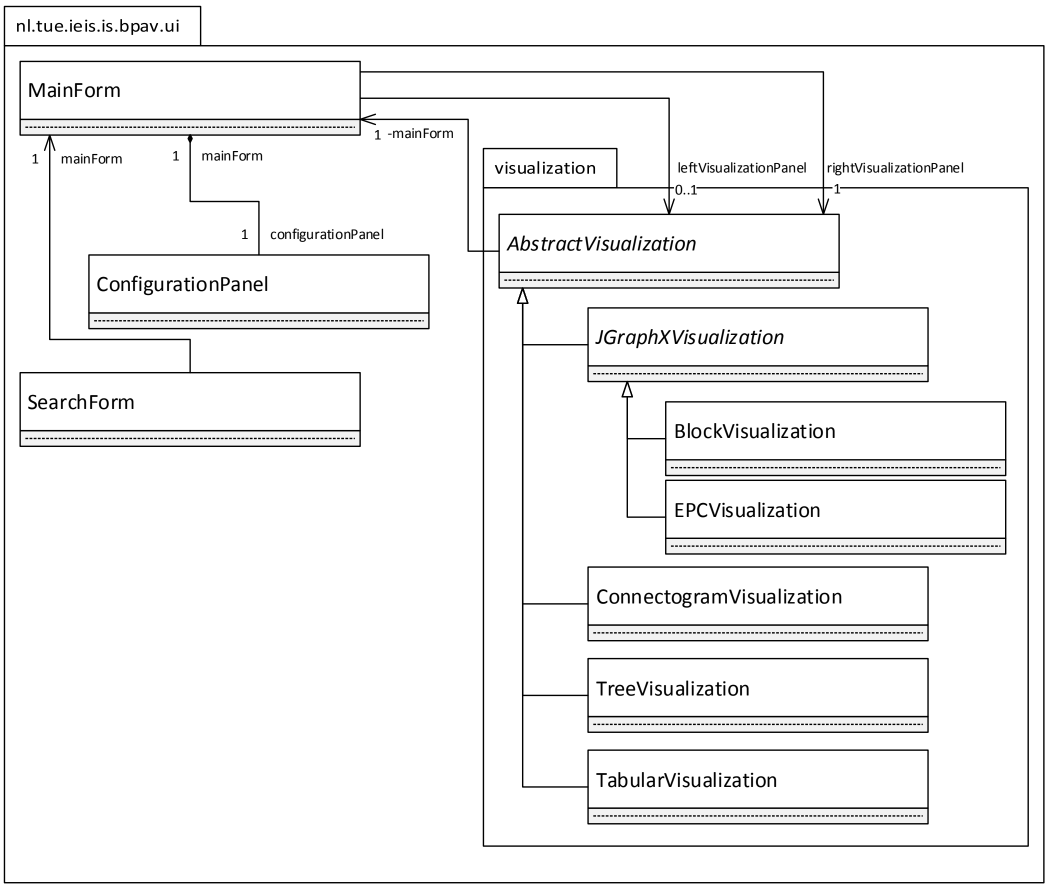


Figure B.6: Class diagram of the *ui* package

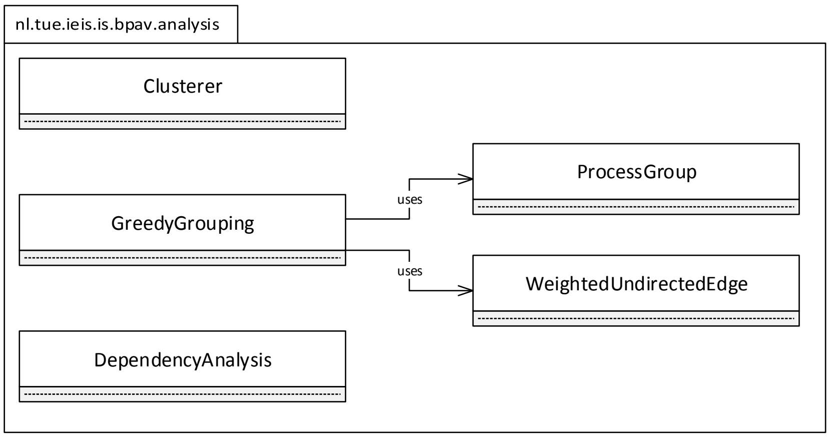


Figure B.7: Class diagram of the *analysis* package

The *DependencyAnalysis* class

DependencyAnalysis detects relations between processes. To do so, it compares labels of all final events in the EPC models (i.e. events without outgoing edges) to labels of initial events (i.e. events without incoming edges) in other EPC models. If two labels have a string edit distance of less than 5, then they are considered to constitute a relation.

The *GreedyGrouping* class

GreedyGrouping is used to restructure a limited number of elements. The class tries to form groups of elements based on a greedy algorithm. The groups have to have at least a certain inner coupling which can be specified as a parameter.

B.2.5 The *imports* package

The *imports* package contains classes related to loading a business process architecture from an external source. As the only external source that can currently be used, is an EPML export of the SAP reference model, there are only two classes in the package. They can be seen in figure B.8. The class *SAPImporter* reads a separate file that contains information about the hierarchical structure of the reference model and subsequently loads all EPC models for the processes in the reference model. For loading all models from an EPML file, the class *EPMLBulkImporter* is used. It utilizes an EPC implementation that only imports one model per time from a file. By using the *EPMLBulkImporter* the program avoids reading the EPML file multiple times.

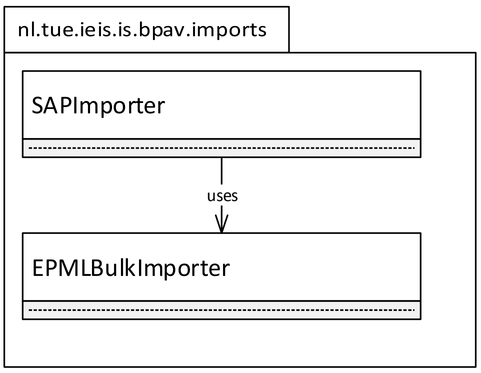


Figure B.8: Class diagram of the *imports* package

B.3 Libraries

Multiple libraries are used at several points in the software. The list below lists all the libraries used and their purpose. The Java standard library is not included in the list. Other versions of the libraries than the ones stated here have not been tested.

Apache Commons IO 2.4[9]: Used for copying a file to a temporary location in class *ConnectogramVisualization*

Apache OpenNLP 1.5.3[8]: Used for tokenization for the search function in class *ProcessArchitecture*

DJ Native Swing[5]: Used for embedding a flash component in *ConnectogramVisualization*

EPC implementation by Remco Dijkman (unpublished): Used for importing and storing EPC models in multiple packages.

JGraphX 2.0.0.1[17]: Used for drawing graphs in *JGraphXVisualization* and its subclasses.

JSON-Java[2]: Used in the *model* package for creating a JSON representation of the model to transfer it to the flash component in *ConnectogramVisualization*.

SWT 4.3[10]: Required in combination with DJ Native Swing for embedding a native component in *ConnectogramVisualization*.

Weka 3.6.9[12]: Used for clustering in class *Clusterer*

C User test

C.1 Steps of the tool introduction

The introduction to the prototype consists of the following steps:

1. Show the top-level structure of the process architecture with the block-based visualization like it is shown when the tool has just started.
2. Change the value for *minimum connection strength* to show the effect of this setting
3. Double-click an element to show its content. Click on the root node of the tree view to show the top level again.
4. Enable the automatic grouping and point out a group. Disable the automatic grouping again.
5. Select the connectogram and wait until it is shown. Point at an element to see how its connections are highlighted.
6. Select *As-is architecture* instead of *Designed architecture* and let the users see the resulting connectogram.
7. Select *Compare both* and then *Table* and show the resulting table to the users.

The person presenting the tool should comment everything that he or she does, especially say which element is being clicked. This presentation shows every functionality that is to be evaluated and therefore allows the user to see the basic interaction mechanisms that they need during the test.

The number of relevant functionalities in ARIS is smaller. They are introduced as follows:

1. After ARIS is started, go to the *Explorer* and navigate to the *Main group* of the database containing the SAP reference model.
2. Open up an element within the *Main group* in the navigation tree. Double-click on a model within that element to show its EPC model in the *Designer*.
3. Open another model using the *Explorer tree* within the *Designer*.
4. Click on *Explorer* to return to the *Explorer*.

C.2 Questions for the introduction of the use cases

C.2.1 Use case 1 – Understanding the structure of the architecture

The goal of this use case is understanding the overall structure of the process architecture. This involves understanding the elements and relations on the top level of the process architecture as well as on lower levels. It also involves understanding the hierarchical structure itself. Hence, there are two questions that focus on the top level structure of the process architecture, two questions that focus on relations at a lower level and two questions that focus only on the hierarchical structure. The concrete architecture elements have been chosen to yield question of medium complexity whose answer is not immediately obvious. The following questions are suggested for this use case:

- Does *Customer Service* depend on *Asset accounting*?
- Does *Compensation Management* depend on *Benefits Administration* or the other way around?
- Does *Call Center Processing* depend on *Long-Term Service Agreements*?
- Does *Currency translation* depend on *Consolidation of Investments* or the other way around?
- Is *Employee transfer* a process within *Personnel Administration*?
- Does *Master Data Processing Product safety* contain the group *Protocol*?

C.2.2 Use case 2 – Restructuring a process architecture

This use case aims at supporting the user in creating a new structure for the process architecture. This involves finding problems in the current process collection and designing a new one. The questions and tasks are intended to steer the user towards using the capabilities of the tool to fulfill such tasks. Redesigning a whole process architecture cannot be done within the limited time of a user test. Therefore, more indirect questions are asked:

- Which groups in the process architecture have little coherence?
- Should the groups *Customer service* and *Sales and distribution* be merged into a single group? Consider comparing the modeled structure with the automatically mined as-is structure to answer this question.
- Should *Production Planning and Procurement Planning* be split into multiple groups? Consider comparing the modeled structure with the automatically mined as-is structure to answer this question.

Bibliography

[1] S. Alam and P. Dugerdil. Evospaces visualization tool: Exploring software architecture in 3d. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, pages 269 –270, oct. 2007.

[2] Douglas Crockford. Json-java: A reference implementation of a json package in java. https://github.com/douglascrockford/JSON-java, June 2013.

[3] Thomas A. Curran, Gerhard Keller, and Andrew Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model.* Enterprise Resource Planning Series. Prentice Hall, 1998.

[4] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):pp. 319–340, 1989.

[5] Christopher Deckers. The dj project native swing. http://djproject.sourceforge.net/ns/index.html, June 2013.

[6] Rami-Habib Eid-Sabbagh, Remco Dijkman, and Mathias Weske. Business process architecture: Use and correctness. In Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 65–81. Springer Berlin / Heidelberg, 2012.

[7] Loe Feijs and Roel De Jong. 3d visualization of software architectures. *Commun. ACM*, 41(12):73–78, December 1998.

[8] The Apache Software Foundation. Apache opennlp. http://opennlp.apache.org/, August 2013.

[9] The Apache Software Foundation. Commons io. http://commons.apache.org/proper/commons-io/, June 2013.

[10] The Eclipse Foundation. Swt: The standard widget toolkit. http://www.eclipse.org/swt/, June 2013.

[11] K. Gallagher, A. Hatch, and M. Munro. Software architecture visualization: An evaluation framework and its application. *Software Engineering, IEEE Transactions on*, 34(2):260 –270, march-april 2008.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD*

Explorations Newsletter, 11(1):10–18, 2009.

[13] Andrei Irimia, Micah C. Chambers, Carinna M. Torgerson, and John D. Van Horn. Circular representation of human cortical networks for subject and population-level connectomic visualization. *NeuroImage*, 60(2):1340 – 1351, 2012.

[14] Jens Knodel, Dirk Muthig, Matthias Naab, and Dirk Zeckzer. Towards empirically validated software architecture visualization. In *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 187–188, New York, NY, USA, 2006. ACM.

[15] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. Visualization-based analysis of quality for large-scale software systems. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 214–223, New York, NY, USA, 2005. ACM.

[16] Wanchun Li, P. Eades, and Seok-Hee Hong. Navigating software architectures with constant visual complexity. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 225 – 232, sept. 2005.

[17] JGraph Ltd. Jgraph. http://jgraph.com/jgraph.html, May 2013.

[18] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc. Communicating software architecture using a unified single-view visualization. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 217 –228, july 2007.

[19] M. Samia and M. Leuschel. Towards pie tree visualization of graphs and large software architectures. In *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, pages 301 –302, may 2009.

[20] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343, 1996.

[21] M.-A. Storey, C. Best, and J. Michand. Shrimp views: an interactive environment for exploring java programs. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 111 –112, 2001.

[22] Viswanath Venkatesh and Fred D. Davis. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, 46(2):186–204, 2000.

[23] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. User acceptance of information technology: Toward a unified view. *MIS Quarterly*, 27(3):pp. 425–478, 2003.