(ITEA 3 – 17003)

PANORAMA
Boosting Design Efficiency for Heterogeneous^3 Systems

Deliverable: D 4.1

Requirements and Use-Cases for assessing systems in early design phases

Work Package: 4

Result visualization and assessments

Task: 4.1

Use-Case Analysis of assessment scenarios

Document Type:	Deliverable	**Classification:**	Public
Document Version:	Revision 1	**Contract Start Date:**	2019-04-01
Document Preparation Date:	2020-01-31	**Duration:**	2022-03-31

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

EUREKA

History

Rev.	Content	Resp. Partner	Date
0.1	Initial structure added	Lukas Krawczyk	2019-06-11
0.2	Initial Draft on Use-Cases	Fabian Kneer Lukas Krawczyk	2019-08-16
0.3	Added Overview of Visualization Frameworks	Fabian Kneer Harald Mackamul	2019-11-12
0.4	Added Visualization Techniques	Fabian Kneer	2019-12-06
0.5	Use-Cases and Visualization Background on Safety Analysis	Magnus Cruz	2019-12-23
1.0	Introduction, Summary, Conclusion, and final corrections	Lukas Krawczyk	2020-01-24

Contents

List of Figures

List of Tables

Summary

This deliverable is the first deliverable document of the PANORAMA Work Package 4 ”Result visualization and assessments”. It contains the results of task T4.1 ”Use-Case Analysis of assessment scenarios” which aims at defining the requirements and use cases for the visualizers and assessment techniques that will be developed within WP4. While visualizers are required to provide an easily understandable representation of the results from WP3, such as details on the energy consumption of system aspects, performance characteristics or safety/security relevant information, assessment techniques will be required to support e.g. decisions or comparing large amounts of results.

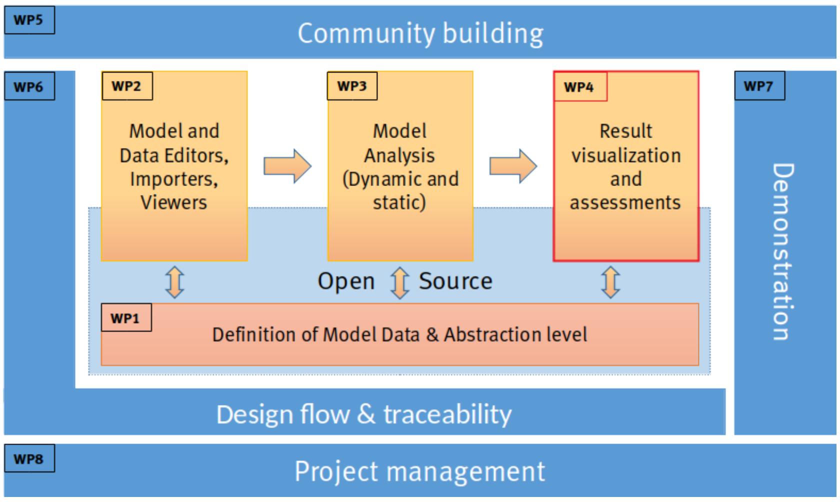

WP5
Community building
WP6
WP2
Model and Data Editors, Importers, Viewers
WP3
Model Analysis (Dynamic and static)
WP4
Result visualization and assessments
WP7
Demonstration
Open Source
WP1
Definition of Model Data & Abstraction level
Design flow & traceability
WP8
Project management

Figure 1: PANORAMA Work Package Structure including their relation to WP4

In order to put this document into context, the relation between WP4 (red box) and the other work packages is illustrated in Fig. 1. The project is organized into two groups of work packages: The technical solutions to the project objectives are developed by WP1-4 (colored in red / orange), whereas WP5-8 cover management, packaging, and dissemination related topics (colored in blue).

1 Introduction

The complexity of future solutions for mobility results in intricate and unforeseen impact of product and project decisions on systems level, even in late development phases. Especially software development in mobility, one of the main factors for innovation and value creation – and costs, must be evaluated also in system context. To cope with this fact, the early assessment of design decisions is a key factor for success.

The objective of WP4 is to enable this assessment by providing methods, processes and tools for visualizing and assessing the results of analysis and simulation (cf. Fig 1.1).

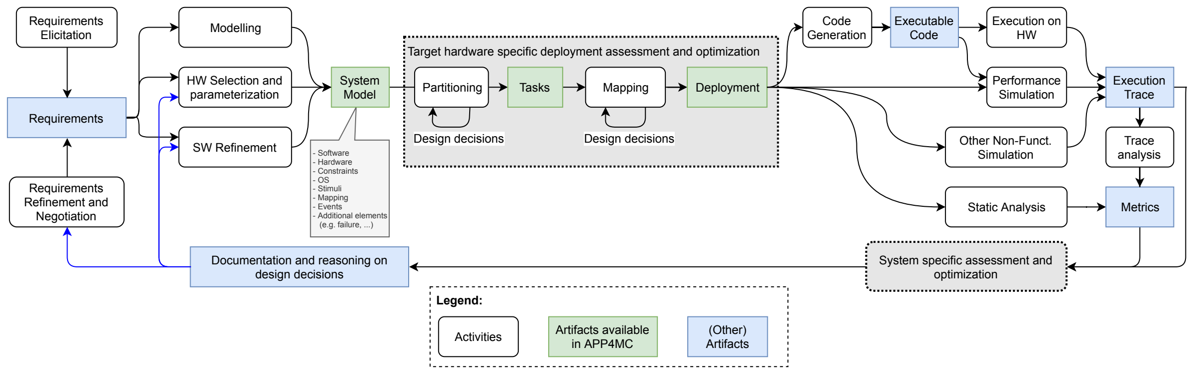


Figure 1.1: High-Level overview of a typical design flow within mobility systems development

While visualizers are required to provide an easily understandable representation of the results from WP3, e.g. details on the energy consumption of system aspects, timing and performance characteristics, or safety/security relevant information, assessment techniques will be required to support e.g. decisions or comparing large amounts of results. As a first step towards the early assessment of design decisions, this document deals with visualizing results by e.g. presenting charts, diagrams, and other visual approaches to users in order to allow visually assessing results produced in this as well as the other technical work packages.

The remainder of this document is structured as follows. Chapter 2 presents the use cases along with the inherent requirements that become subject of the developed methods, processes and tools. They are categorized into four groups and described following the use case pattern presented in the Appendix. Visualization techniques along with general information on graphically representing data as well as the assessment and visualization of failure propagation are discussed in Chapter 3. It summarizes a variety of charts and graphs along with practical examples on their application in industrial use cases, such as Fault Tree Analysis (FTA). In order to support those use cases, Chapter 4 identifies and analyzes frameworks, which provide means to show appropriate charts, diagrams, and other visual approaches while being compatible with the open source nature of the project. Finally, Chapter 5 concludes this document.

2 Use Cases

This chapter presents the use cases related to visualizing and assessing analyses results. The focus lies on the representation of analyses results from internal methods and processes but also the results of external analysis tools. The visualizations that are part of the following use cases are used for e.g. comparing different deployment alternatives and getting a comprehensive overview of the variety of the results produced by the tool chain. The assessment is based on different results of internal and external analysis and is supported by the visualizations.

From a high level perspective the use cases are as followed:

- UC-4-ATB: Application Temporal Behavior (see Table 2.1)
- UC-4-ME: Mapping Evaluation (see Table 2.2)
- UC-4-DA: Deployment Alternatives (see Table 2.3)
- UC-4-SA: Safety Analysis (see Section 2.3.1)

The relationship among the use cases, actors, and different systems is shown in Figure 2.1.

The presented use cases in this chapter are following the use case pattern presented in the appendix 5. In the remainder of this chapter, the high level use cases are presented followed by a detailed view on specific use cases.

Table 2.1: Use Case 4-ATB - Application Temporal Behavior

Section	Content
ID	UC-4-ATB
Name	Applications Temporal Behavior
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	A developer wants to inspect the behavior of tasks executed on a specific system configuration that is sufficiently described in an early design phase in terms of an AMALTHEA system model.
Trigger	The developer executes the tool for the given AMALTHEA system model.
Actors	Developers, System
Preconditions	A valid AMALTHEA system model containing a sufficient amount of details on the hardware, software, stimuli, and deployment is available.
Postconditions	The task behavior is visualized on the screen.
Result	Verification of analytic approaches by visually comparing the tool's output with the results of the analytic approach.

Section	Content
Main scenario	1. Developer selects a model and requests to visualize the task behavior (see EX1) 2. System verifies the model for compatibility (see EX1) 3. System displays the model metrics for informational purpose (see AS1) 4. Developer specifies visualization parameters (see AS1) 5. Developer overrides settings within the model (see AS1) 6. System shows a summary of the configuration 7. System visualizes the task behavior
Alternative scenario	AS1 The developer skips the subsequent steps and continues to the task behavior visualization 1. Return developer to Main scenario step 7
Exception scenario	EX1 The application has been invoked without a model file or with an incorrect model file. 1. System notifies user that an error has occurred. 2. Application is terminated and the system returns to the state before the application has been launched.
Relations to other Use Cases	-

Table 2.2: Use Case 4-ME - Evaluation of the Hardware and Software Mapping

Section	Content
ID	UC-4-ME
Name	Mapping Evaluation
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	A developer wants to analyze the mapping between the hardware structure and the software system of a specific system configuration that is sufficiently described in an early design phase in terms of an AMALTHEA system model.
Trigger	The developer starts the tooling for the hardware and software mapping evaluation.
Actors	Developer, System
Preconditions	A valid AMALTHEA system model containing the relevant information of hardware, software, and deployment is available.
Postconditions	-
Result	Visualization of the deployment of software to the given hardware platform.

Section	Content
Main scenario	1. A developer runs the evaluation of Hardware and Software mapping on a system model (see AS1 and EX1). 2. System represents the mapping metrics for information purpose. 3. System visualize the hardware structure with the deployment of the software represented in the mapping model. 4. The system represents the deployment and mapping metrics (see AS2). 5. A developer evaluates the mapping between hardware and software.
Alternative scenario	AS1: The System does not contain a mapping model. 1. The system starts the mapping editor by visualizing the hardware structure of the system. 2. A developer adds the mapping model to the system model. 3. Continue Main scenario Step 2. AS2: Dynamic information available 1. A developer starts the animation of the mapping metrics. 2. The system visualize the mapping metrics in a kind of animation. (e.g. utilization of the cores or memory consumption).
Exception scenario	EX1 The application has been invoked without a model file or with an incorrect model file. 1. System notifies user that an error has occurred. 2. Application is terminated and the system returns to the state before the application has been launched.
Relations to other Use Cases	-

Table 2.3: Use Case 4-DA - Deployment Alternatives

Section	Content
ID	UC-4-DA
Name	Deployment Alternatives
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	A developer wants to compare different deployment alternatives resulting out of an e.g. a DSE approach that has been optimizing a specific system configuration towards a given optimization goal. The System is sufficiently described in an early design phase in terms of an AMALTHEA system model.
Trigger	Internal or External invocation of the Tool
Actors	Developers, System
Preconditions	Multiple deployment alternatives have been created and are ready to be displayed.
Postconditions	-

Section	Content
Result	Visualization of multiple deployment alternatives.
Main scenario	1. Select deployment alternatives (DA) that should be compared with each other. 2. Developer executes the tool in order to compare the selected DAs. 3. Select output mode for assessing DAs. 4. Comparison of DAs will be visualized according to selected mode.
Alternative scenario	AS1: Automatic / External invocation: 1. A tool delivering deployment alternatives, e.g. for architecture optimization, finishes its execution and provides multiple deployment alternatives 2. Continue from **Step 3** of the main scenario.
Exception scenario	EX1 The application has been invoked with less then 2 deployment alternatives or deployment alternatives do not match the expected input. 1. System notifies user that an error has occurred. 2. Application is terminated and the system returns to the state before the application has been launched.
Relations to other Use Cases	-

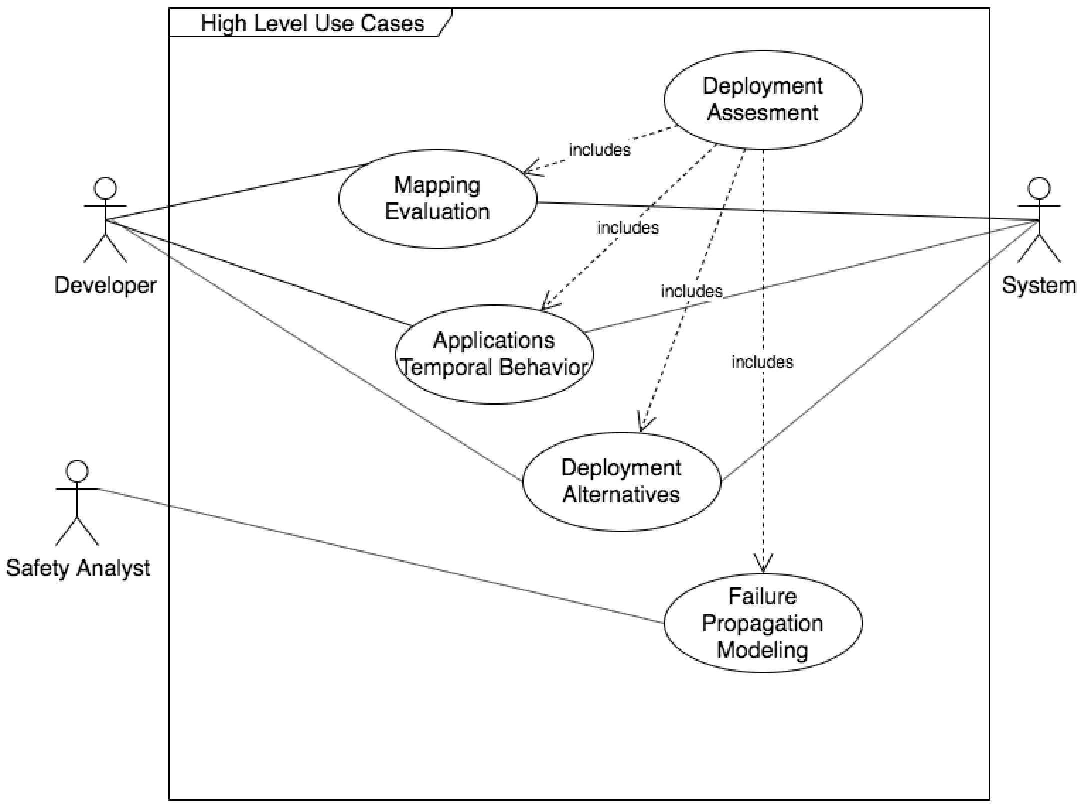


Figure 2.1: Overview of the high level Use Cases

2.1 UC-4-ATB: Applications Temporal Behavior

This section presents a detailed view on the use case *UC-4-ATB Applications Temporal Behavior*. Figure 2.2 shows the relationship between the low level use cases. The main task is represented in the use case *Task Execution* (see Table 2.7). In this use case, the results of the scheduling is represented. The use case *Metrics* (see Table 2.4) is used to analyze and interpret the deployment metrics for a concrete system configuration. It helps to interpret the visualization of the scheduling. The scheduling of *UC-4-ATB-4* is extended by the *Event Chain* use case (see Table 2.5). The extension is used to define and visualize event chains between artifacts of the deployment, e.g. tasks, runnables, and labels. The last extension is the use case *Communication* (see Table 2.6). In this extension the communication between the artifacts are analyzed and visualized. It is used, for example, to identify the need for synchronization.

The remainder of this section presents the detailed description of the four internal use cases of *UC-4-ATB Application Temporal Behavior*.

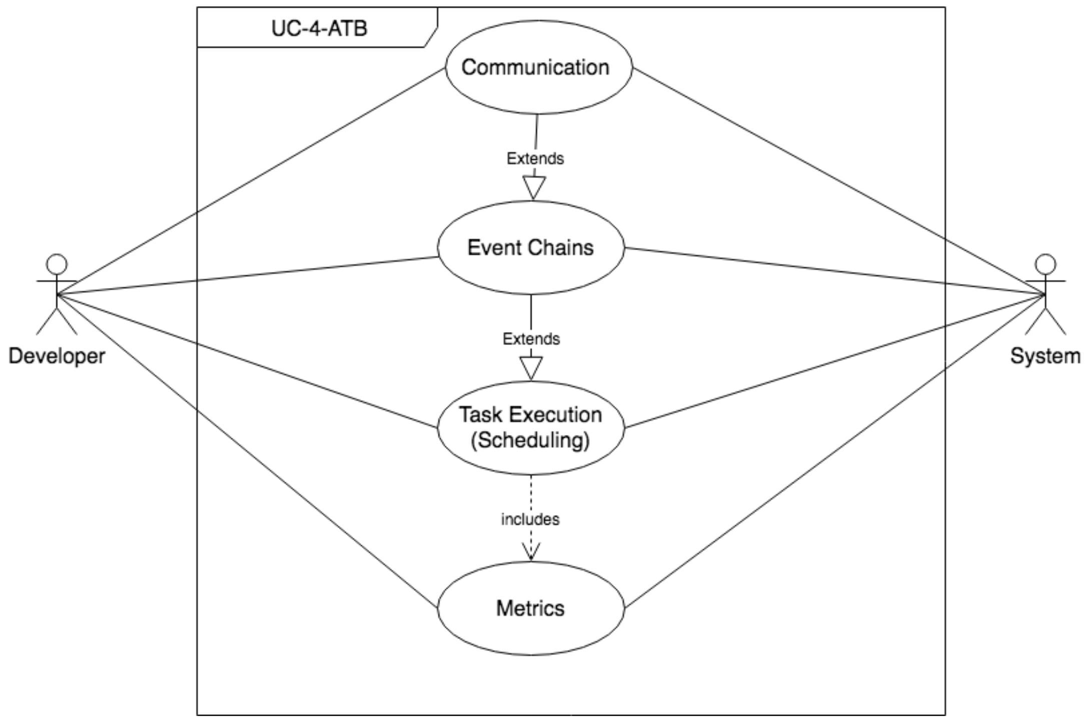


Figure 2.2: Detailed Overview of the Use Case UC-4-APT Applications Temporal Behavior

2.1.1 UC-4-ATB-1 Metrics

Table 2.4: Use Case Description of UC-4-ATB-1 Metrics

Section	Content
ID	UC-4-ATB-1
Name	Metrics

Section	Content
Author	Dortmund University of Applied Sciences and Arts
Priority	Low
Description	A developer inspects the deployment metrics for a concrete system configuration that is sufficiently described in an early design phase in terms of a model.
Trigger	The developer executes the task visualizer for a concrete system model.
Actors	Developer, System
Preconditions	AMALTHEA model that contains a valid mapping model.
Postconditions	
Result	Visualization of early (rudimentary) analysis results.
Main scenario	1. System displays the model metrics for informational purpose 2. Developer specifies visualization parameters 3. Developer overrides settings within the model 4. System shows a summary of the configuration
Alternative scenario	-
Exception scenario	-
Relations to other Use Cases	-

2.1.2 UC-4-ATB-2 Event Chains

Table 2.5: Use Case Description of UC-4-ATB-2 Event Chains

Section	Content
ID	UC-4-ATB-2
Name	Event Chains
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	A developer wants to analyses the event chains of a system model to identify end-to-end latencies, over-sampling, under-sampling, and more.
Trigger	The developer executes the task visualizer for a concrete system model and opens the event chain visualization.
Actors	Developer, System
Preconditions	System model that contains a valid mapping model. The system model contains global event Chains (Optional) (see AS1)
Postconditions	-

Section	Content
Result	Verification of analytic approaches by visually comparing the determined results with the tool's output.
Main scenario	1. System displays the model metrics for informational purpose (see UC1.1 Metrics) 2. System visualizes the task behavior (see UC1.4 Task Execution). 3. The developer defines a custom event chain (see AS1). 4. Developer select the labels that will be inspected. 5. The developer set the position of every label in the chain. 6. The system represents the event chain visualization.
Alternative scenario	AS1. The system model contains global event chains. 1. The system displays the event chain visualization.
Exception scenario	-
Relations to other Use Cases	UC-4-ATB-4 TaskExecution (see 2.7), UC-4-ATB-1 Metrics (see 2.4))

2.1.3 UC-4-ATB-3 Communication

Table 2.6: Use Case Description of UC-4-ATB-3 Communication

Section	Content
ID	UC-4-ATB-3
Name	Communication
Author	Dortmund University of Applied Sciences and Arts
Priority	Medium
Description	A developer wants to analyses the communication between tasks, runnables, and labels to identify the need for synchronization, identify communication flows, and more.
Trigger	The developer executes the task visualizer for a concrete system model.
Actors	Developer, System
Preconditions	AMALTHEA model that contains a valid mapping model.
Postconditions	
Result	Verification of the communication of a system.
Main scenario	1. System displays the model metrics for informational purpose (see UC-4-ATB-1 Metrics) 2. System visualizes the task behavior (see UC-4-ATB-4 Task Execution). 3. The system represents the event chain visualization (see UC-4-ATB-2 Event Chains). 4. The Developer analyses the visualized results to identify communication flows, the need for synchronization.

Section	Content
Alternative scenario	-
Exception scenario	-
Relations to other Use Cases	UC-4-ATB-1 Metrics (see 2.4), UC-4-ATB-2 Event Chains (see 2.5), and UC-4-ATB-4 TaskExecution (see 2.7)

2.1.4 UC-4-ATB-4 Task Execution (Scheduling)

Table 2.7: Use Case Description of UC-4-ATB-4 Task Execution (Scheduling)

Section	Content
ID	UC-4-ATB-4
Name	Task Execution (Scheduling)
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	A developer wants to inspect the behavior of tasks for a concrete system configuration that is sufficiently described in an early design phase in terms of a model.
Trigger	A Developer executes the task execution visualization tool for a concrete system model.
Actors	Developer, System
Preconditions	AMALTHEA model that contains a valid mapping model.
Postconditions	-
Result	Verification of analytic approaches by visually comparing the determined results with the represented task execution behavior.
Main scenario	1. System displays the model metrics for informational purpose (see UC1.1 Metrics) 2. Developer specifies visualization parameters 2.1. The developer selects one of the three scheduling algorithms (Preemptive RMS, Cooperative EDF, Preemptive EDF). 2.2. Developer sets a time unit for the scaling. 2.3. The developer selects one of the three simulation modes (Normal, Dataflow, Extended Dataflow). 3. System shows a summary of the configuration 4. System visualizes the task behavior. 4.1. The system displays the scheduling of the task on the cores in the main view. 5. The developers analyses the task of a core. 5.1. The system displays a detail view on the tasks that run on the selected core.

Section	Content
Alternative scenario	-
Exception scenario	-
Relations to other Use Cases	UC-4-ATB-1 Metrics

2.2 UC-4-ME: Hardware and Software Mapping Evaluation

This section shows a detailed view on the use case *UC-4-ME Hardware Software Mapping*. In Figure 2.3 the relationship between the three low level use cases is shown. The main contribution lies in the use case *Structural Visualization of Deployment* (see Table 2.9). The use case describes, how hardware and software model artifacts are visualized to produce a visual overview on the deployment. The visualization is extended by the use case *Visualization of Mapping Metrics* (see Table 2.8). The static visualization of hardware and software is extended by adding specific metrics. There are some animations available for dynamic content related to specific metrics. Example for metrics are utilization of the cores or memory consumption. The third use case *Mapping Editor* (see Table 2.10) is included in the UC-4-ME-2. The use case describes the visualization of the hardware and the process to map the software artifacts on a given hardware configuration.

The remainder of this section presents the low level use cases of *UC-4-ME Mapping Evaluation* in more detail.

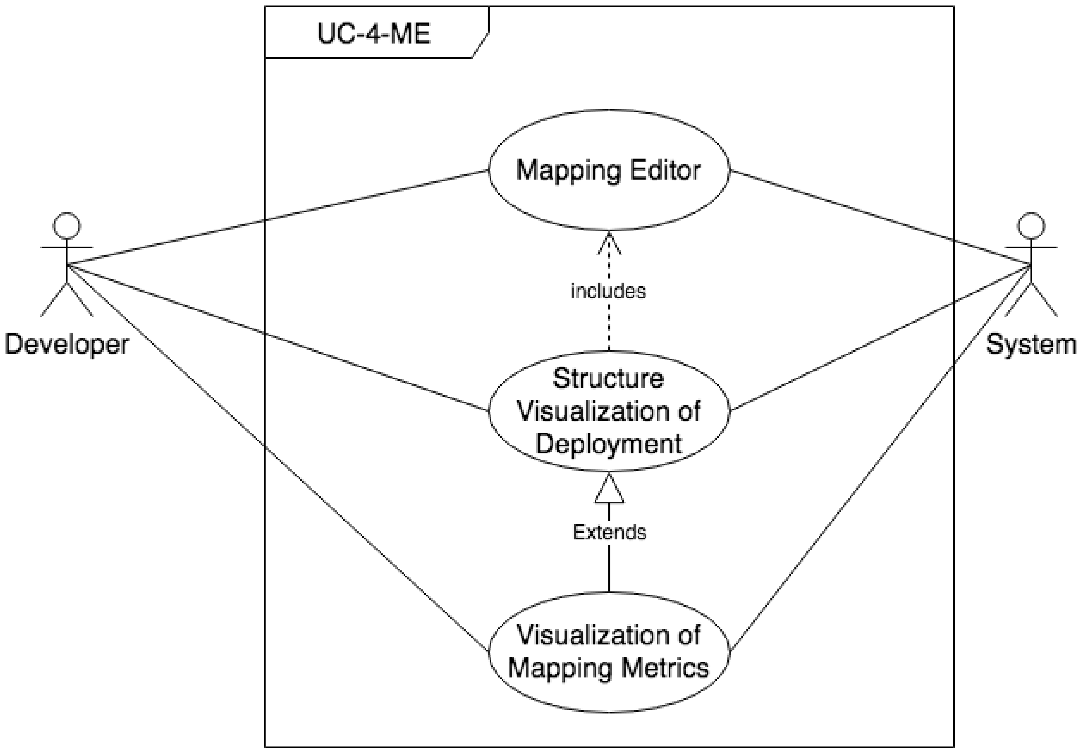

UC-4-ME
Mapping Editor
includes
Structure Visualization of Deployment
Extends
Visualization of Mapping Metrics
Developer
System

Figure 2.3: Detailed Overview of the Use Case UC-4-ME Mapping Evaluation

2.2.1 UC-4-ME-1 Visualization of Mapping Metrics

Table 2.8: Use Case Description of UC-4-ME-1 Visualization of Mapping Metrics

Section	Content
ID	UC-4-ME-1
Name	Visualization of Mapping Metrics
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	For an analyzing purpose, the metrics of the mapping between hardware and software are visualized. The visualization is static for a specific time or an animation for dynamic information, such as a trace. The represented metrics can consist of and are not limited to: utilization of the cores, the memory consumption, energy consumption.
Trigger	
Actors	Developer, System
Precondi-tions	A valid system model containing the relevant information of hard-ware, software, and deployment is available.
Postcondi-tions	
Result	Visualization of the early analysis results for the mapping between hardware and software.
Main scenario	1. The System visualize the mapping between hardware and software. (see UC-4-ME-2 Structural Visualization of Deployment) 2. The User wants to see the first analyses results. 3. The System visualized the static information of the metrics (see AS1) 3.1. The System extends the visualization of the memory by adding a view for memory consumption. 3.2. The System extends the visualization of the cores by adding a view for utilization of a core. 3.4. The System extends the visualization of the deployment by adding a view for energy consumption.
Alternative scenario	AS1: 1. The User wants a dynamic visualization for a trace. 2. The System visualizes the metrics by extending the currents views with information about memory consumption, utilization of a core, and energy consumption. 3. The User starts the animation. 4. The System updates the new views for the metrics in a given time interval to establish an dynamic animation of the metrics.
Exception scenario	-

Section	Content
Relations to other Use Cases	UC-4-ME-2 Structural Visualization of Deployment (see 2.9)

2.2.2 UC-4-ME-2 Structural Visualization of Deployment

Table 2.9: Use Case Description of UC-4-ME-2 Structural Visualization of Deployment

Section	Content
ID	UC-4-ME-2
Name	Structural Visualization of Deployment
Author	Dortmund University of Applied Sciences and Arts
Priority	High
Description	Visualize the mapping of software components to the hardware structure of a system and handling the complexity of the deployment.
Trigger	A developer selects a model that contains a valid mapping model and starts the mapping visualization.
Actors	Developer System
Preconditions	A valid system model containing the relevant information of hardware, software, and deployment is available.
Postconditions	-
Result	A visualization of the hardware components and the connected software artefact's on the hardware components.
Main scenario	1. Developer selects a model and requests to visualize the deployment 2. System verifies the model for compatibility (see AS1) 3. System displays the model metrics for informational purpose 4. Developer specifies visualization parameters 7. System visualizes the structure of the deployment.
Alternative scenario	AS 1: The system model does not contain mapping model. 1. The system notifies developer about missing mapping model and offers to open mapping editor 2. Developer wants to add mapping model. 3. The system opens the mapping model editor instant of the deployment visualization (see UC-4-ME-3 Mapping Editor).
Exception scenario	-
Relations to other Use Cases	Use Case UC-4-ME-3 Mapping Editor (see 2.10)

2.2.3 UC-4-ME-3 Hardware and Software Mapping Editor

Table 2.10: Use Case Description of UC-4-ME-3 Mapping Editor

Section	Content
ID	UC-4-ME-3
Name	Mapping Editor
Author	Dortmund University of Applied Sciences and Arts
Priority	low
Description	A Developer wants to edit the system model to add the mapping between a hardware structure and a software model.
Trigger	The evaluation tooling was started without a system model that contains a valid mapping model.
Actors	Developer, System
Preconditions	A valid system model containing the relevant information of hardware and software.
Postconditions	
Result	A valid system model that contains all relevant information for hardware, software, and the mapping between them.
Main scenario	1. The system represents the current hardware structure. 2. A developer inspects a hardware component. 3. The system gives an overview of software components that are already mapped to the hardware component. 4. A developer opens the adding windows. 5. The system visualize the software components that could be added to the hardware component. 6. A developer selects the software components. 7. The system adds the components to the hardware structure. 8. As long as the mapping model is incomplete, continue with Step 2.
Alternative scenario	-
Exception scenario	-
Relations to other Use Cases	

2.3 UC-4-SA: Safety Analysis

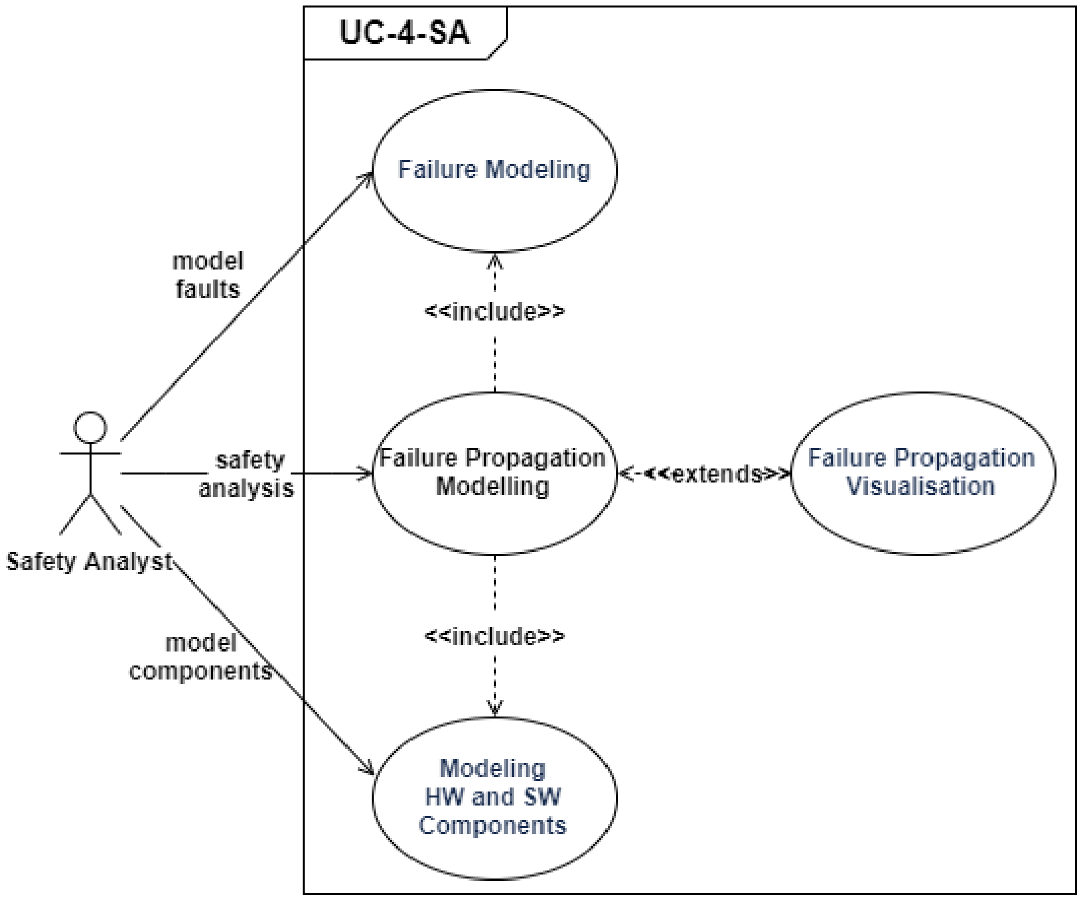


Figure 2.4: Overview of the Safety Analysis Use Cases

2.3.1 Failure Propagation Modelling

Table 2.11: Failure Propagation Modelling

Section	Content
ID	UC-4-SA-1
Name	Failure Propagation Modelling
Author	Critical Software
Priority	Medium
Description	Safety Analyst models and analyzes the dangers, barriers, and failures in heterogeneous systems. In addition to mapping hardware and software configuration, we understand the possible output for visualizing the failure propagation modes of the modelled system.
Trigger	Safety Analyst uses tools to model faults and simulate propagation statically.
Actors	Safety Analyst
Preconditions	The list of safety requirements and critical system components.
Postconditions	FTAs and FMEAs run on propagation models.
Result	Model view of relationships between functions, failure modes, and software components.
Main scenario	**Step 1:** Modeling Hardware and Software Components (see AS1) 1. The developer a system model with hardware and software components. 2. Map the relationships between software and hardware processes. 3. Briefly write down the list of critical system functions. **Step 2:** Failure Modeling 1. List the failure modes grouped by functions. 2. Associate the possible components (software, hardware) of the fault source. 3. Associate faults with the possible components through which they can propagate. **Note:** It is possible to create one or more fault propagation models for the same system architecture.
Alternative scenario	**AS1:** If the hardware and software components model already exists: 1. Import the template into the current project. 2. Continue from **Step 2** of the Main scenario.
Exception scenario	-
Relations to other Use Cases	**UC-4-SA-2** (Table 2.12)

2.3.2 Failure Propagation Visualization

Table 2.12: Failure Propagation Visualization

Section	Content
ID	UC-4-SA-2
Name	Failure Propagation Visualization
Author	Critical Software
Priority	Medium
Description	Safety Analyst builds FTA and FMEA views from the relationships between critical functions and software/hardware components that will be impacted by failure modes directly or indirectly. After this mapping, the expected output will be one or more graphs with the system fault propagation models..
Trigger	Safety Analyst uses tools to model the fault propagation view.
Actors	Safety Analyst
Preconditions	System Components Model and Failure Model validated.
Postconditions	One or more graphs of the fault propagation models displayed.
Result	Visualization of fault propagation models for systems that allow static impact analysis due to security failures.
Main scenario	1. Select the model containing the hardware and software components. 2. Select the model with the critical functions and system failure modes. 3. Define the relationships between the components and the functions you want to view. 4. Select the output mode for the fault propagation model. (see EX1)
Alternative scenario	-
Exception scenario	**EX1:** Application does not display fault propagation model because there are inconsistencies: 1. The system displays the list of problems in the model that prevent failure mode from being displayed. 2. Viewing is stopped.
Relations to other Use Cases	**UC-4-SA-1** (Table 2.11)

3 Visualization Techniques

This chapter discusses the techniques for visualizing data coming from e.g. analysis results in order to support assessment. Section 3.1 provides a general overview on existing techniques that can be used for e.g. large amounts of data. It summarizes the variety of graphs, charts, plots, and other visualization techniques along with proper examples on how those can be applied for assessing analysis results during the development of embedded systems in e.g. the automotive or avionic domain. Finally, Section 3.2 extends this description by an in-depth discussion of assessment techniques for safety based on a failure propagation use case.

3.1 Data Visualization

The visualization of data is an important part of every analysis. Data is gathered or produced during an analysis and has to be filtered and represented in a way that a person can understand and explore the information.

There are five steps that need to be considered during the visualization of data (see [Maz09] for more information):

1. Define the problem: The problem defines for what a visualization is used. It could be the representation of a hypotheses, finding new information in a data set, or communicate information between people.

2. Examine the nature of the data: There are different types of data and every type is suitable for different visualizations. The data could be quantitative (e.g. integers), ordinal (days of a week), or categorical / nominal (city names).

3. Number of dimensions: The number of dimensions is important for choosing the visualization. The dimensions can be independent or dependent. The dependent variables vary and their behavior is analyzed compared to the independent variables.

4. Data structures: The data can be linear structured (e.g. tables), temporal (changing over time), spatial or geographical (e.g. a map), and network (e.g. relationships between entities).

5. Type of interaction: A visualization can be static (e.g. figure), transformable (a user can control the process of date visualization, e.g. change the scale), or manipulable (the user can modify parameters of the visualization, e.g. zoom on details).

Fig. 3.1 shows a summary of important variables that need to be considered during the process of visualize data.

Different examples of graphical elements that can be used to visualized data are represented in Fig. 3.2. The different elements can be combined to represent different characteristics of data. The color could be used to identify different variables or categories in a set of points.

Problem	Data type	Dimensions	Data structure	Type of interaction
Communicate	Quantitative	Univariate	Linear	Static
Explore	Ordinal	Bivariate	Temporal	Transformable
Confirm	Categorical	Trivariate	Spatial	Manipulable
		Multivariate	Hierarchical	
			Network	

Figure 3.1: Variables to consider when designing visual representations [Maz09]

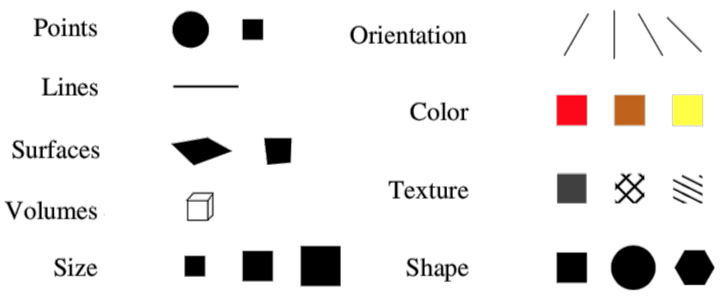


Figure 3.2: Example of graphical elements [Maz09]

In Fig. 3.1 the effectiveness of graphical elements or variables for the visualization of the three types of data quantitative, ordinal, and nominal is shown. For example the length is very good to represent quantitative data (like a set of Integers) but is less effective to represent ordinal (weak days) or nominal data. Colors should be used to separate the different categories of nominal data, but is less effective for quantitative data.

The next section represents different charts that are used to visualize the different types of data sets.

3.1.1 Charts

This section gives an overview of different charts and what kind of data is suitable for a specific chart.

In a **scatter plot** different variables are visualized. Every data point is represented as a point between the axis. The scatter plot helps to identify correlations between two different variables or identify clusters and outliers. Fig. 3.3 shows an example of a scatter plot. The plot represents the results of an optimization for a combustion system. The goal of the optimization was to achieve similar soot and NOx emissions to that of the baseline case but with a 10% fuel consumption improvement.

The **line chart** is used to visualize quantitative data as a position on quantitative scale. The points are connected to form a line- or curve-segment. Points between to data points can be interpolated. The interpolation helps to visualize trends, locale structures, and the general distribution of the data. The line chart helps to visualize groups of data points with a continuous domain. One line chart can consists of more than one data visualization. The different variables need to be on the same scale to be integrated in a single line chart. The numbers of combined line charts should not be higher then 3 or 4. Fig. 3.4 shows an example for a line chart diagram illustrating the periodic activation pattern of three tasks over time scale of 20ms.

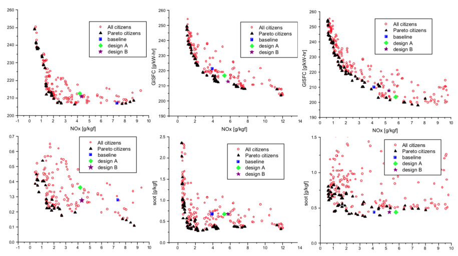


Figure 3.3: Example of a Scatter Plot

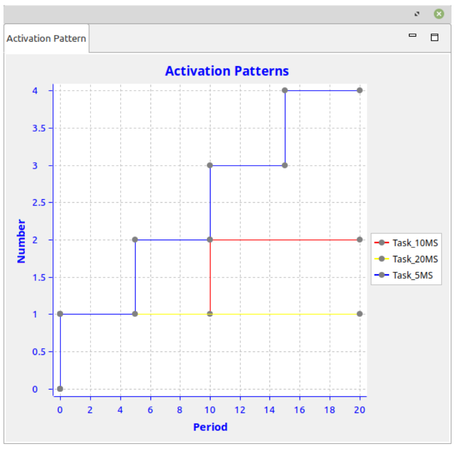


Figure 3.4: Example of a Line Chart, illustrating the number of triggering events (activations) for a given task set.

A **bar chart** can be vertical or horizontal. The data is represented as a bar instead of a line. It can be used to represent nominal, discrete, quantitative, and dependent data. Normally, the horizontal axis contains the independent dimension and the vertical axis the associated dependent dimension. The Integration of several bar charts in a single chart is possible to represent several variables. A 3D visualization in a 3D-coordination system can be used if a second independent variable is available as a third dimension. Fig. 3.5 shows an example of a bar chart indicating the number of memory accesses per task from a modern engine management system.

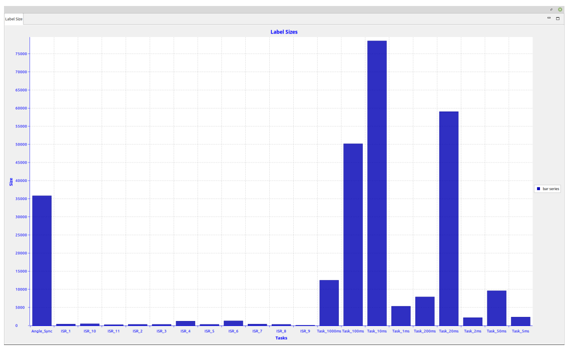


Figure 3.5: Bar Chart used for graphically representing the number of memory accesses for a given task set.

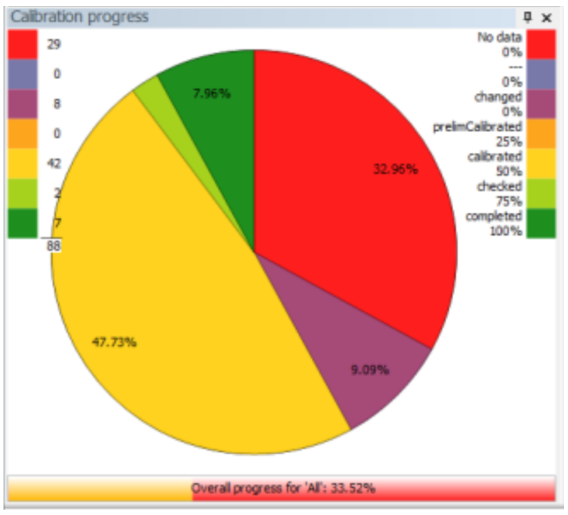


Figure 3.6: Example of a Pie Chart, Vector [Vec]

The **pie chart** is used to visualize quantitative characteristics over a nominal and independent variable. The characteristics a represented as different colored or textured segment of a circle.

A pie chart implies, that the segments can be summed up to a basic population. The size of a pie charts is important because in small circles it is difficult to compare the surfaces of segments. Fig. 3.6 shows a pie chart that indicates the current status of an automated calibration process, including information on how much work has already been completed (green area), along with the status of the remaining activities.

An **histogram chart** is a special bar chart or line chart. The chart is used to visualize the frequency of occurrence of a data point and not the data point itself. For the representation of quantitative data, the values are classified. Fig. 3.7 illustrates a bar chart histogram depicting the number of activation events for two tasks that were extracted from multiple traces with individual time spans. An example of a line chart histogram showing the distribution of e.g. the execution time of a task following a gaussian distribution is shown in Fig. 3.8.

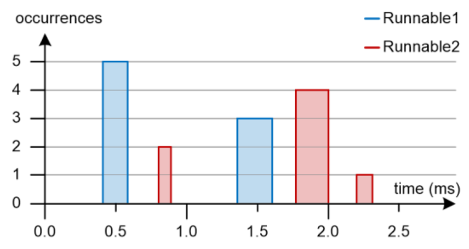


Figure 3.7: Column Histogram representing the number of activations of two tasks for a limited number of traces with different time spans [App]

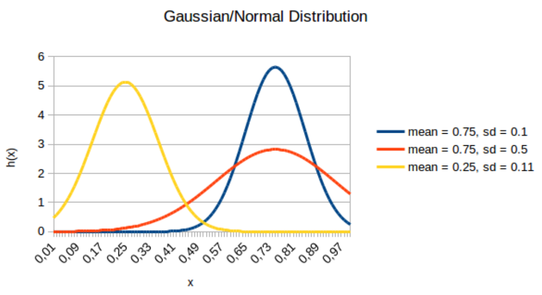


Figure 3.8: Line Histogram visualizing three Gaussian distributions with different values for mean and standard deviation [App]

Examples of different possible curve forms for a histogram are illustrated in Fig. 3.9 representing the following distributions:

- a) Normal distribution
- b) Bimodal distribution (indicates, that characteristics of two different populations exist)
- c) Multi-modal distribution (indicates, that characteristics of several different populations exist)
- d) skew distribution (positive)
- e) skew distribution (negative)
- f) Upset distribution (indicates a very concentrated distribution)
- g) Very flat curve (indicates values from different populations)
- h) cropped curve (indicates, that a part of the population is missing or was deleted)
- i) Upset distribution with a peak (indicates, that all elements after a specific threshold are combined)

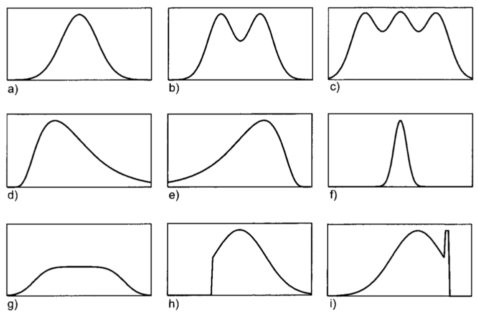


Figure 3.9: Possible Curve Forms of a Histogram, Schumann & Mueller [SM00]

The extension of a line or curve chart in the 3 dimensional space is the **surface chart**, represented in Fig. 3.10. Points in the 3 dimensional space are connected to build a surface. The surface structure provides information about the distribution and local trends, such as the fitness landscapes representing the quality of a solution space resulting from an optimization approach.

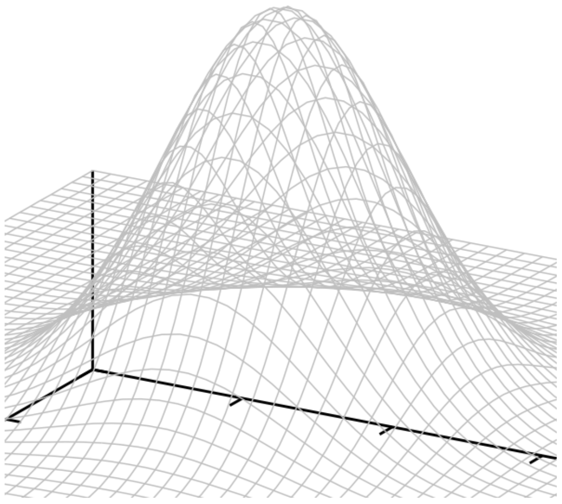


Figure 3.10: Surface Chart representing the fitness landscape of an optimization solution space [AM15]

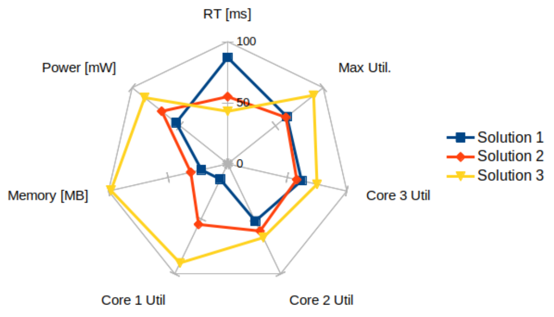


Figure 3.11: Spider Chart visualizing various metrics of three possible implementation candidates

A spider chart can be used to compare different strengths and weaknesses of multivariate data with three or more quantitative variables. The axis represent different characteristics. The values are represented by lines between the axis. An example of a spider chart comparing various quality metrics of three possible implementation candidates is illustrated in Fig. 3.11.

The parallel plot represents the different characteristics as vertical parallel bars and not as a circle like the spider chart. It is used to compare different strengths and weaknesses of multivariate data with three or more quantitative variables.

The **box and whisker plot** is used to visualize the distribution of a data set by categories. The median and first and third quartiles are represented in a box. The whiskers represent the minimum and maximum of a data set. Fig. 3.13 shows an example of a box and whisker

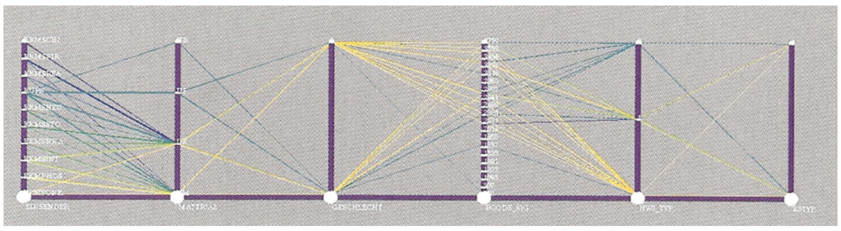


Figure 3.12: Example of a Parallel Plot, Schumann & Mueller [SM00]

plot. The plot represents the emission factors for individual plume analysis separated between periods with no influence from trucks (red) and periods with at least one passing truck (black). Horizontal lines represent the median values, boxes represent the 75th percentile and whiskers represent the 90th percentile.

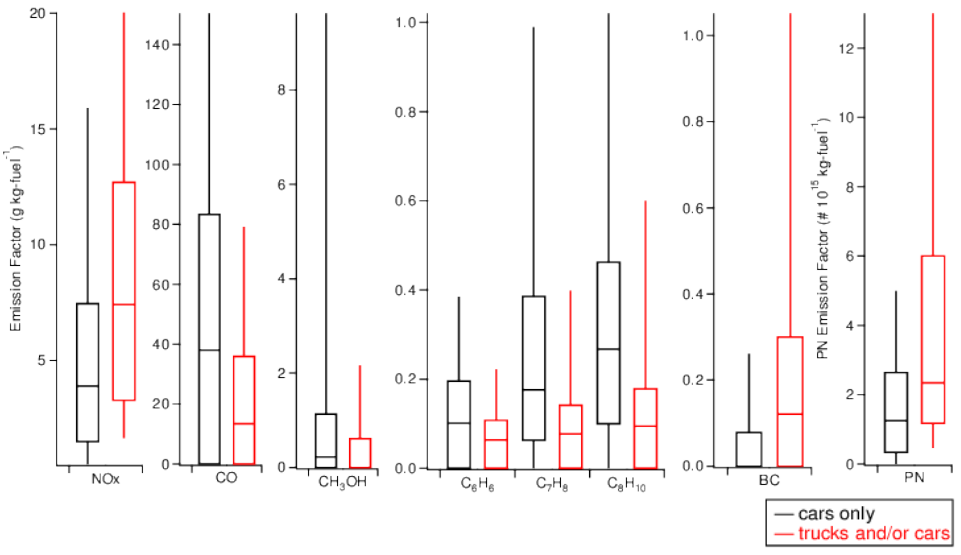


Figure 3.13: Example of a Box and Whisker plot, Wang et al. [WJZ+15]

The identification of the best chart to represent data depends on many different variables as mentioned in section 3.1. The Fig. 3.14 represents the "*chart chooser*" from Abella [Abe]. The chart chooser can be used as a starting point to choose a chart. Beginning in the center different ways based on the data types and attributes of the data a chart can be chosen .

3.1.2 Graphs

This section shows how information can be visualize using graphs. A graph is an abstract structure and can be used to represent any information that can be modeled as objects and the relationship between the objects. Objects are represented as nodes and relations between the objects as edges.

The first and major challenges for visualize structured information lies in the *representation* of a graph. The challenge is to visualize the graph covering most information but make it easy to read and to interpret. The second challenge is the *scaling* problem. The algorithm that are

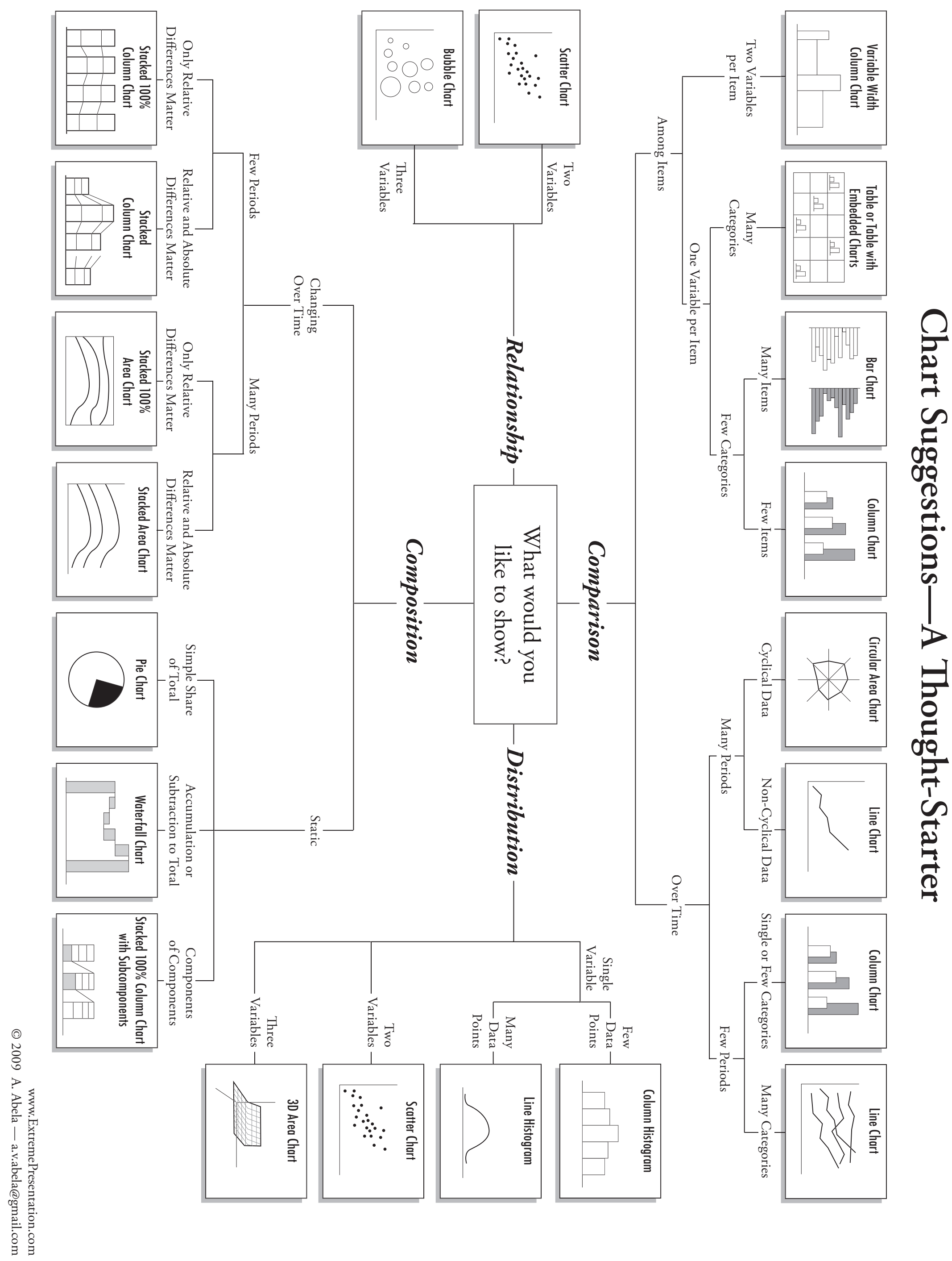


Figure 3.14: Chart Chooser, Abella [Abe]

used for the layout of the representation must be able to process the amount of information in a given time. Another scaling problem is the limited real estate of display area.

In a graph different information can be visualized by using colors, sizes, forms, or shapes for the nodes and edges.

Next, different graph types are described and some example graph layouts to handle the challenges regarding graph visualization

Graph Types

All Graphs have general attributes that classify their type. A graph consists of a nonempty set of nodes (vertices or points) and a set of edges that represents the relation between the nodes. The edges can be unweighted or weighted (nominal or ordinal quantitative). The weight is sometimes referred to as the *cost* of an edge. Examples for weights are a measure of length of a route, the energy required to move between to locations, etc. Some graphs can be traversed to form a path. This path consists of all traversed nodes and a sequence of edge to reach the nodes. A simple path has no repeated nodes within the path. In a cycle path the initial nodes is also the end node of the path. A graph without any cycle path is called acyclic.

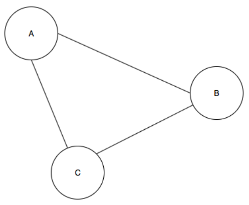


Figure 3.15: Example of an Undirected Graph

Undirected Graphs. An undirected graph is a graph, which only contains bi-directed edges. Fig. 3.15 shows an example of an undirected graph.

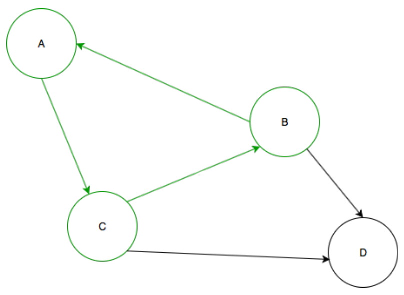


Figure 3.16: Example of directed cyclic graph

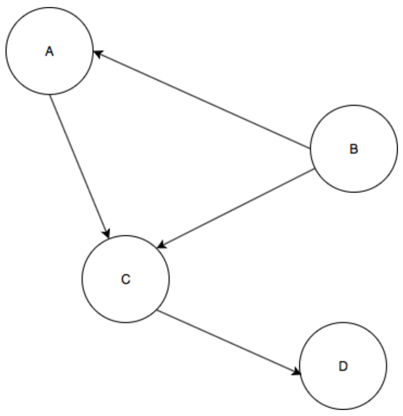
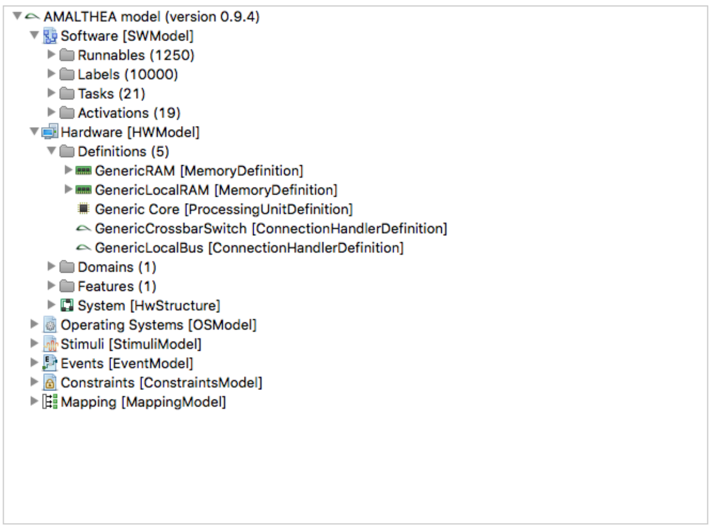


Figure 3.17: Example of directed acyclic graph

Directed Graphs. A directed graph hast direction for every edge. The direction is normally visualized by adding arrows to an edge. A path can be build by traversing between the nodes via the directed edges. As mentioned before, a distinction is made between directed cyclic graphs (see Fig. 3.16) and directed acyclic graphs (see Fig. 3.17).

Tree. A tree is a special type of graph. It contains no cycles, is usually directed, and has a single node as starting point, which is called root. The tree is a hierarchical structure that starts at the root node. The end nodes of a tree are called leaves. Fig. 3.23 shows and example of a tree. The tree represents an Amalthea system model including its nested sub-models (software, hardware, ...) and model elements.



```
▼ AMALTHEA model (version 0.9.4)
  ▼ Software [SWModel]
    ▶ Runnables (1250)
    ▶ Labels (10000)
    ▶ Tasks (21)
    ▶ Activations (19)
  ▼ Hardware [HWModel]
    ▼ Definitions (5)
      ▶ GenericRAM [MemoryDefinition]
      ▶ GenericLocalRAM [MemoryDefinition]
        Generic Core [ProcessingUnitDefinition]
        GenericCrossbarSwitch [ConnectionHandlerDefinition]
        GenericLocalBus [ConnectionHandlerDefinition]
    ▶ Domains (1)
    ▶ Features (1)
    ▶ System [HwStructure]
  ▶ Operating Systems [OSModel]
  ▶ Stimuli [StimuliModel]
  ▶ Events [EventModel]
  ▶ Constraints [ConstraintsModel]
  ▶ Mapping [MappingModel]
```

Figure 3.18: Example of Tree represented as Indented List

Network. A Network is a special type of directed graph. It has usually weighted edges, but in contrast to a tree, it has no topological restrictions. The graph in Fig. 3.19 shows a Network on Chip (NoC) as an example of a network graph. A NoC has a specific topology which describes the structure of the Network, in this case a mesh structure. The graph shows the connection between the cores via network interfaces and routers.

Graph Layout

There are many different layouts for the visualization of graphs. In the following paragraphs, three example layouts are described.

Radial. The nodes are arranged in circles around a focus node. It is usually used in an interactive visualization, where a user can choose the focus node. Fig. 3.20 shows an example for the radial layout.

Circular. In a circular layout, the nodes are arranged in a circles. The space between the nodes is usually evenly. Fig. 3.21 shows an example of the circular layout.

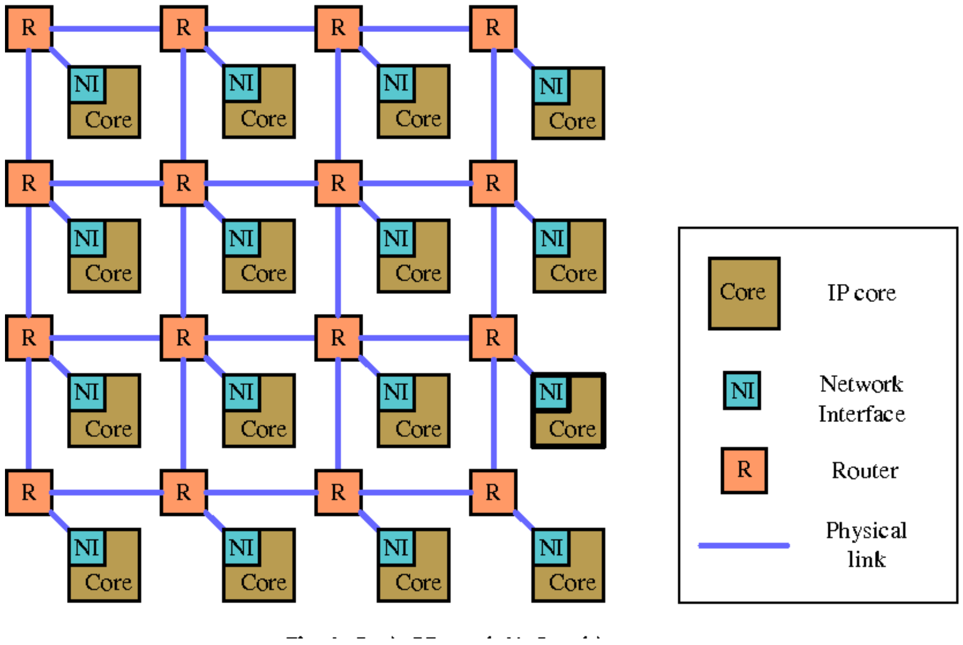


Figure 3.19: Example of a Network, Liu et al. [LGY12]

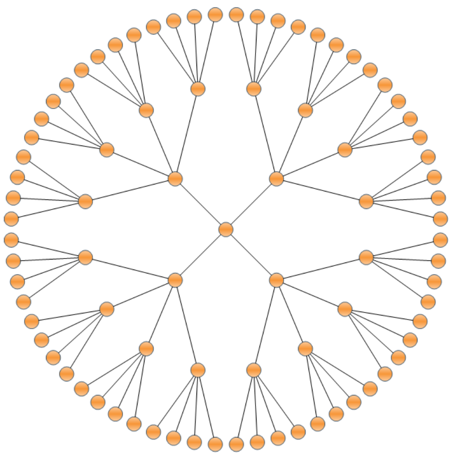


Figure 3.20: Example of Radial Graph Layout

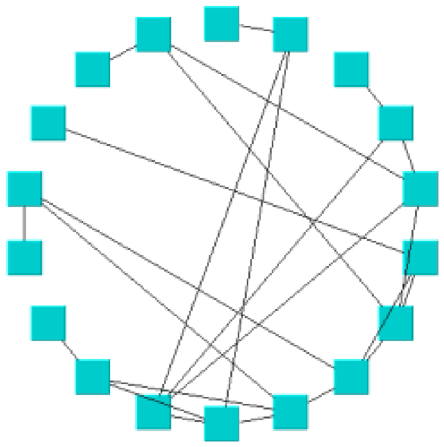
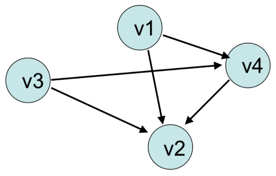


Figure 3.21: Example of Circular Graph Layout

Adjacency Matrix. The adjacency matrix represents a graph as matrix. The rows represent edges leaving the node and the columns represents edges entering a node. The structure is used for storing and processing a graph on a computer. Fig. 3.22 shows an example adjacency matrix.



	v1	v2	v3	v4
v1	0	1	0	1
v2	0	0	0	0
v3	0	1	0	1
v4	0	1	0	0

Figure 3.22: Example of an Adjacency Matrix

Tree Layouts. Trees are hierarchal structures with a root node. Every node can have multiple child nodes and with every refinement of a node the tree becomes larger. The following three layouts are examples to visualize trees. Fig. 3.24 shows a **note diagram**, which is a simple visualization of a tree. The starting point is the root node, which is refined vertically or horizontally layer for layer. A more complex visualization is the **tree-map** shown in Fig. 3.18. A tree-map is used to visualize a hierarchical structure. The different nodes are represented as nested rectangles. The size of the rectangles represents the value of the data element. A tree-map can be used to compare different structured information by the size of data elements. The last example in Fig. 3.23 shows a tree represented as **indented lists**. The Tree is build vertical and every new layer is indented on the horizontal. The examples in the Figure represents an Amalthea model used in PANORAMA.

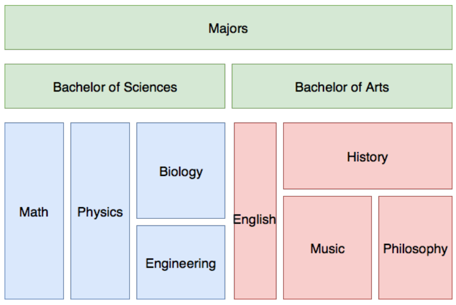


Figure 3.23: Example of Tree represented as Tree Map

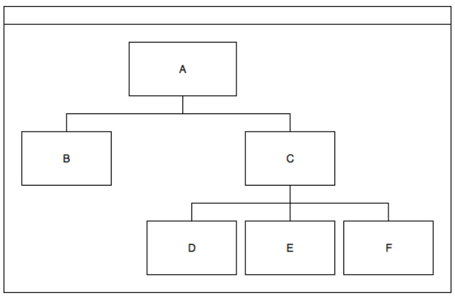
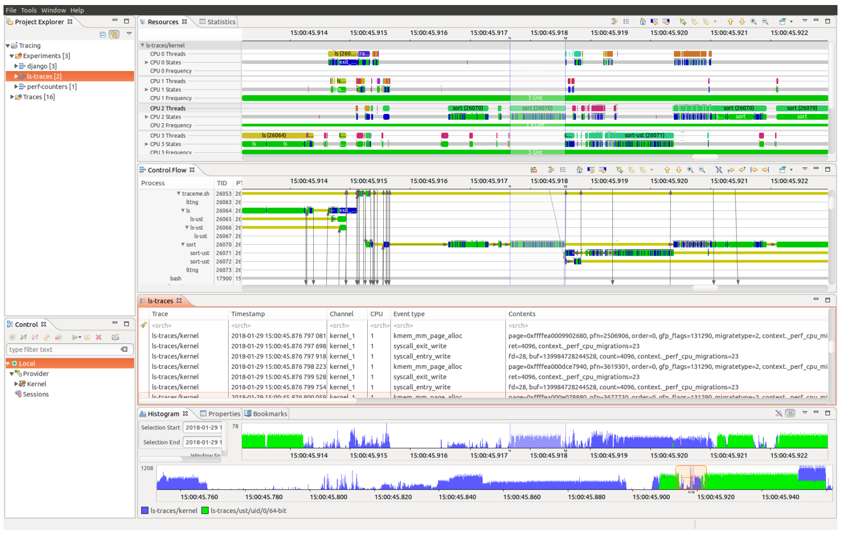


Figure 3.24: Example of a Node Tree

3.1.3 Tools for Assessment

A variety of tools exist that implement e.g. graphs for assessing analysis results, executions, dumps, application behavior, and other aspects that are relevant in developing multi- and many core systems. Two tools that integrate a variety of views, charts, graphs, diagrams, metrics, and various other visualization techniques to extract information from traces and logs is realized by Eclipse Trace Compass (cf. Fig 3.25) and the App4MC Task Visualizer (cf. Fig 3.26).

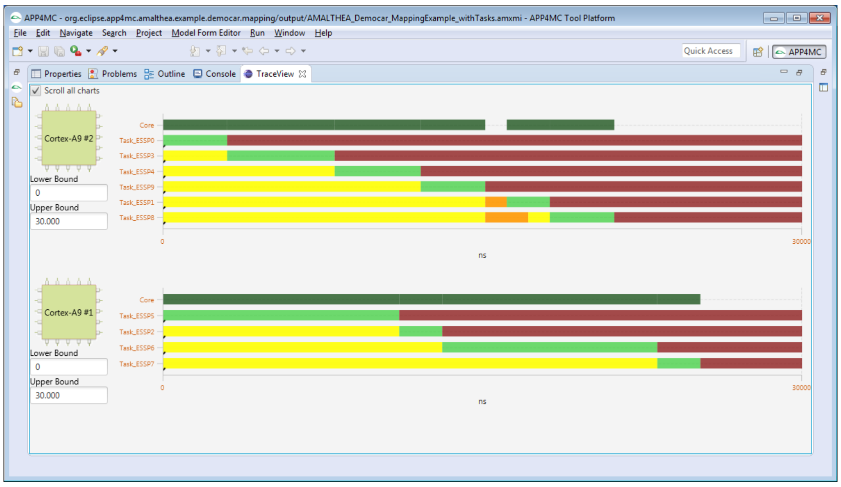
Eclipse Trace Compass [Ecl20] is a Java-based open-source tool that allows displaying and analyzing any kind of logs or traces. It provides support for a large number of trace formats,



Trace	Timestamp	Channel	CPU	Event type	Contents
ls-traces/kernel	2018-01-29 15:00:45.876 797 081	kernel_1	1	kmem_mm_page_alloc	page=0xffffea0009902680, pfn=2506906, order=0, gfp_flags=131290, migratetype=2, context._perf_cpu_mi
ls-traces/kernel	2018-01-29 15:00:45.876 797 698	kernel_1	1	syscall_exit_write	ret=4096, context._perf_cpu_migrations=23
ls-traces/kernel	2018-01-29 15:00:45.876 797 918	kernel_1	1	syscall_entry_write	fd=28, buf=139984728244528, count=4096, context._perf_cpu_migrations=23
ls-traces/kernel	2018-01-29 15:00:45.876 798 223	kernel_1	1	kmem_mm_page_alloc	page=0xffffea000dce7940, pfn=3619301, order=0, gfp_flags=131290, migratetype=2, context._perf_cpu_mi
ls-traces/kernel	2018-01-29 15:00:45.876 799 528	kernel_1	1	syscall_exit_write	ret=4096, context._perf_cpu_migrations=23
ls-traces/kernel	2018-01-29 15:00:45.876 799 754	kernel_1	1	syscall_entry_write	fd=28, buf=139984728244528, count=4096, context._perf_cpu_migrations=23

Figure 3.25: Example of Trace Compass usage for kernel analysis [Ecl20]

such as Common Trace Format (CTF), allowing to inspect Linux LTTng kernel traces as well as bare metal traces, GDB traces, and hardware traces. Especially its support for the Best Trace Format (BTF) for OSEK, along with features for e.g. Latency and Critical Path analysis along with Real-Time deadline investigation makes it especially applicable for the automotive domain.



APP4MC - org.eclipse.app4mc.amalthea.example.democar.mapping/output/AMALTHEA_Democar_MappingExample_withTasks.amxmi - APP4MC Tool Platform
File Edit Navigate Search Project Model Form Editor Run Window Help
Quick Access
APP4MC
Properties Problems Outline Console TraceView
Scroll all charts
Cortex-A9 #2
Lower Bound
0
Upper Bound
30.000
Core
Task_ESSP0
Task_ESSP3
Task_ESSP4
Task_ESSP9
Task_ESSP1
Task_ESSP8
0
30000
ns
Cortex-A9 #1
Lower Bound
0
Upper Bound
30.000
Core
Task_ESSP5
Task_ESSP2
Task_ESSP6
Task_ESSP7
0
30000
ns

Figure 3.26: Gantt-Chart illustrating the execution of 10 tasks on a dual-core ECU

The App4MC Task Visualizer [App] is a tool for visualizing the execution of tasks along with their states and state changes on the resp. executing cores. In order to execute the task visualizer, it is necessary to describe the overall system in terms of an AMALTHEA Model file. The minimal amount of information consists of a Software Model, Hardware Model, and Mapping Model denoting the specifying the deployment of software to hardware, allowing its usage in early design phases without any dine-grained knowledge of implementation details.

For an overview on additional tools that can also be used for assessing e.g. analysis results, such as the commercial INCHRON or Vector Tools, we refer to the State of the Art section in [PAN20].

3.2 Failure Propagation Visualization

As described in delivery D3.1, in the context of PANORAMA, we are also focusing on fault propagation modeling and analysis, specifically applying the FMEA and FTA techniques. The goal in this step is to create one or more views that represent FMEA, FTA as well as the propagation of modeled system failures. Figure 3.27 shows graphical examples of the outcome of the system safety analysis process.

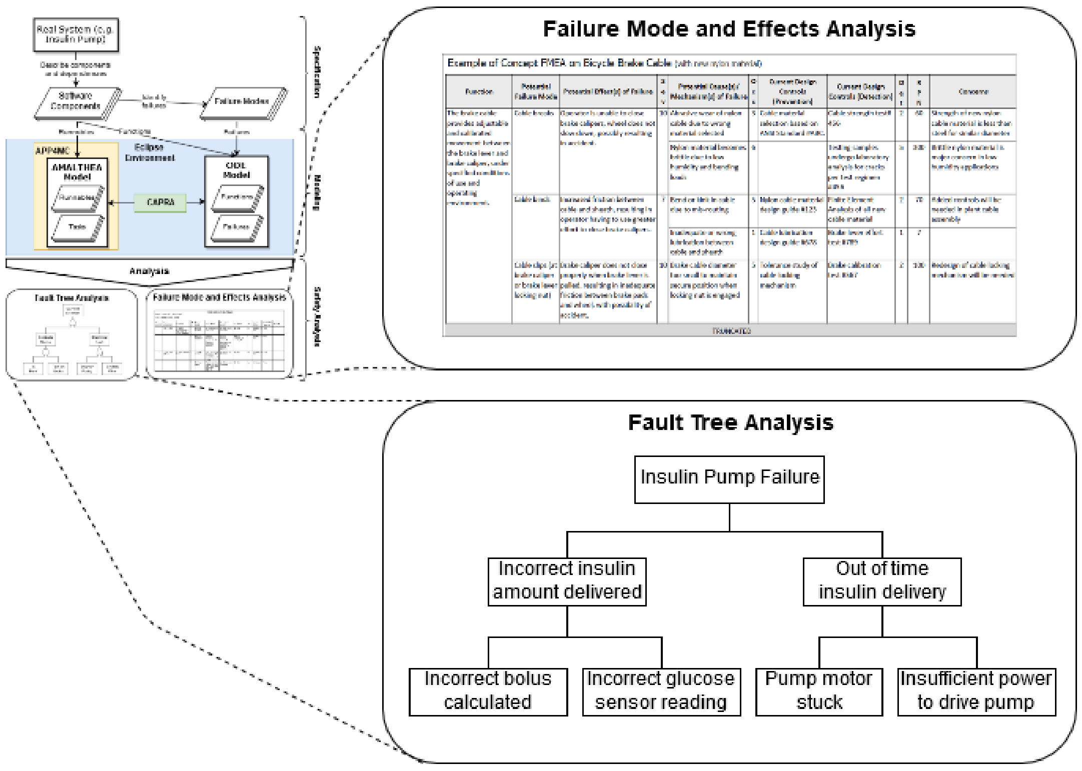

Real System (e.g. Insulin Pump)
Describe components and dependencies
Software Components
Identify failures
Failure Modes
Runnables
Functions
Failures
APP4MC
Eclipse Environment
AMALTHEA Model
Runnables
Tasks
CAPRA
ODE Model
Functions
Failures
Specification
Modeling
Analysis
Fault Tree Analysis
Failure Mode and Effects Analysis
Safety Analysis
Failure Mode and Effects Analysis
Example of Concept FMEA on Bicycle Brake Cable (with new nylon material)
TRUNCATED
Fault Tree Analysis
Insulin Pump Failure
Incorrect insulin amount delivered
Out of time insulin delivery
Incorrect bolus calculated
Incorrect glucose sensor reading
Pump motor stuck
Insufficient power to drive pump

Figure 3.27: System Safety Analysis Process Representation

3.2.1 Safety Modelling Process

In our Safety Analysis process proposal the cycle starts with the construction of the ODE and AMALTHEA models with the specification of Critical Functions, Failures and Runnables.

Figure 3.28 represents an ODE model for the Insulin Pump case study and Figure 3.29 refers us to the mapping of Runnables within the AMALTHEA model.

FTA Using the ODE FailureLogic model we can create an abstract view of an FTA through the FaultTree, FailureModel, Cause, and Failure classifiers as shown in Figure 3.30.

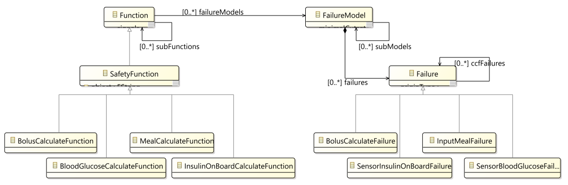
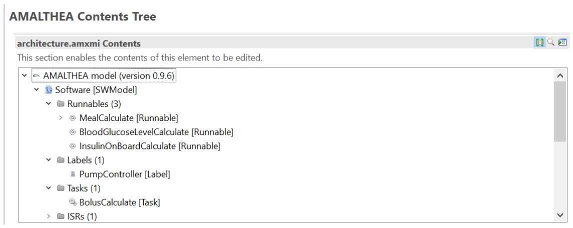


Figure 3.28: Insulin Pump Systems ODE Model



AMALTHEA Contents Tree

architecture.amxmi Contents

This section enables the contents of this element to be edited.

```
AMALTHEA model (version 0.9.6)
  Software [SWModel]
    Runnables (3)
      MealCalculate [Runnable]
      BloodGlucoseLevelCalculate [Runnable]
      InsulinOnBoardCalculate [Runnable]
    Labels (1)
      PumpController [Label]
    Tasks (1)
      BolusCalculate [Task]
    ISRs (1)
```

Figure 3.29: Insulin Pump Amalthea Software Model

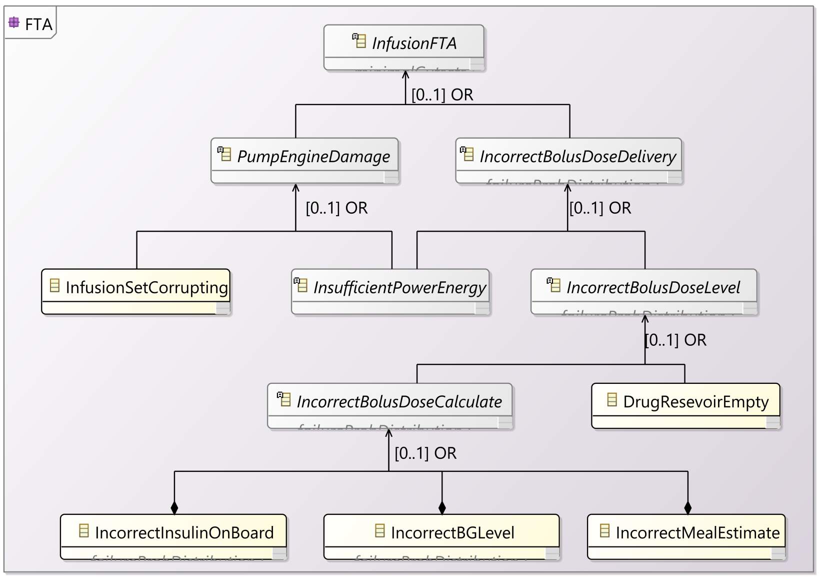


Figure 3.30: Insulin Pump - Fault Tree Analysis

FMEA Although an FMEA is commonly represented through a table for this work, our visualization approach consists of ODE model class diagrams with a list of failure modes. Figure 3.31 shows a generic case study of an Insulin Pump System.

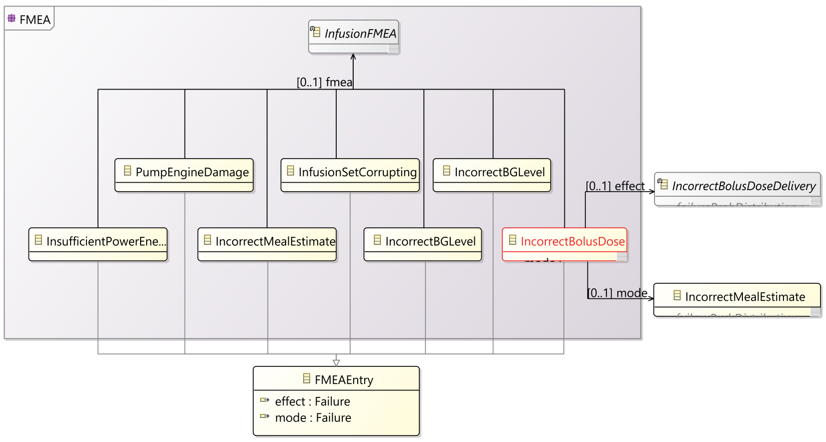


Figure 3.31: Insulin Pump - Failure Mode and Effects Analysis

3.2.2 Conclusion

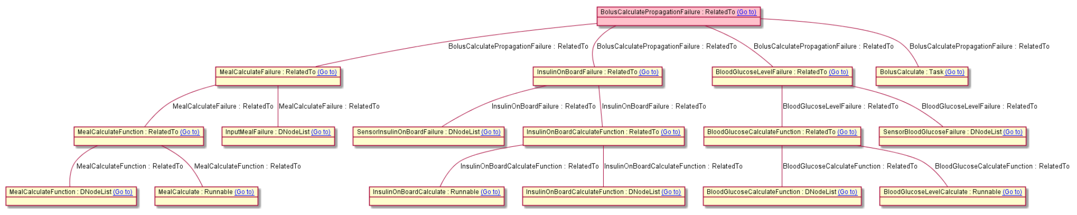

BolusCalculatePropagationFailure : RelatedTo (Go to)
BolusCalculatePropagationFailure : RelatedTo
MealCalculateFailure : RelatedTo (Go to)
InsulinOnBoardFailure : RelatedTo (Go to)
BloodGlucoseLevelFailure : RelatedTo (Go to)
BolusCalculate : Task (Go to)
MealCalculateFailure : RelatedTo
InsulinOnBoardFailure : RelatedTo
BloodGlucoseLevelFailure : RelatedTo
MealCalculateFunction : RelatedTo (Go to)
InputMealFailure : DNodeList (Go to)
SensorInsulinOnBoardFailure : DNodeList (Go to)
InsulinOnBoardCalculateFunction : RelatedTo (Go to)
BloodGlucoseCalculateFunction : RelatedTo (Go to)
SensorBloodGlucoseFailure : DNodeList (Go to)
MealCalculateFunction : RelatedTo
InsulinOnBoardCalculateFunction : RelatedTo
BloodGlucoseCalculateFunction : RelatedTo
MealCalculateFunction : DNodeList (Go to)
MealCalculate : Runnable (Go to)
InsulinOnBoardCalculate : Runnable (Go to)
InsulinOnBoardCalculateFunction : DNodeList (Go to)
BloodGlucoseCalculateFunction : DNodeList (Go to)
BloodGlucoseLevelCalculate : Runnable (Go to)

Figure 3.32: Insulin Pump - Failure Propagation Model

Our approach uses the APP4MC tool in conjunction with other Eclipse Environment plugins like Capra (Figure 3.32) to enable the construction and analysis of critical heterogeneous systems. Here our main focus is not on building an Amalthea model, but on the relationship of its components to the critical functions of a system for mapping and representing failure modes and their propagation analysis. For this, we use ODE component modeling. Importantly, at this time of prospecting, we still have models at a high level of abstraction. However, as future work, we will make possible a more definite view of modes and fault propagation through graphs and tables representing FMEAs and FTAs.

4 Overview of Visualization Frameworks

This chapter summarizes the major capabilities and specifications of common visualization frameworks. The goal is to provide an overview about the possible visualizations and first characteristics that can be used in further analysis and the decision process, i.e. which framework should be used to implement the use cases from Chapter 2 and the visualization techniques in Chapter 3.

The reminder of this chapter is divided into two sections. Section 4.1 presents the supported visualization techniques for every framework w.r.t. to the visualization techniques previously discussed in Section 3.1. A detailed summary on important characteristics, such as the license or specific requirements, are presented in Section 4.2.

4.1 Supported Visualization Techniques

As seen in Chapter 3, there are many visualization techniques that are important for the visualization of analyses and assessment results. Table 4.1 shows the overview of the analyzed frameworks and their support for the visualization techniques discussed in Section 4.1. The first group of columns presents a framework's support for the following charts:

- Scatter Plot
- Line Plot
- Step Chart
- Bar Chart
- Stack Series
- Pie Chart
- Histogram
- Bubble Chart
- Area Chart
- Spider Chart or Radar Graph

The second group of columns in Table 4.1 focuses on possible visualization techniques for complex structures. These structures are used to represent data and their relationships in a structured way. A well known example for such structures is the *Unified Modeling Language (UML)*. Besides common models like the UML, the table contains general structures like graphs and trees. Both structures consist of nodes and their relationship. While trees are used to represent data in a strict hierarchical structure, graphs are used to visualized models of networks.

The major difference among those two is that a tree always need to be connected and can never have loops.

The third and final group of columns shows different layouts for the visualization of a graph or a tree. The frameworks can automatically order the graph into the specified layout. For example, the radial layout is used to visualize trees. The layout starts with the root in the center of a circle and every new layer of the tree is ordered in a new circle around the preview layer.

Table 4.1: Overview of Frameworks w.r.t. supported Visualization Techniques

	2D										Common Graphs						Structures						
Framework	**Scatter Plot**	**Line Plot**	**Step Chart**	**Bar Chart**	**Stack Series Chart**	**Pie Chart**	**Histogram**	**Bubble Chart**	**Area Chart**	**Spider Chart or Radar Graph**	**UML**	**SysML**	**Gantt**	**Mindmap**	**WBS**	**Entity Relationship Diagram**	**Undirected Graph**	**Directed Graph**	**Cyclic Graph**	**Tree**	**Circular Layout**	**Radial Layout**	**Adjacency Matrix**
PlantUML											x		x	x	x	x	x	x	x	x			
SWTCharts	x	x	x	x	x			x	x														
Nebula Visualization Widgets	x	x	x										x										
mxGraph											(x)				x	x	x	x	x	x	x		
JGraphX											(x)				x	x	x	x	x	x	x		
Openchart2	x	x		x	x	x				x													
JGraphT																	x	x	x	x			
JCharts	x	x		x	x	x			x		(x)												
Chart2d	x	x	x	x	x	x			x														
JChart2D	x	x		x																			
JCCKit	x	x		x	x																		
GRAL Java Graphing	x	x	x	x	x	x	x	x	x	x			x										
JFreeChart	x	x	x	x		x	x						x										

Overview of Frameworks w.r.t. supported Visualization Techniques

	2D										Common Graphs						Structures						
Framework	**Scatter Plot**	**Line Plot**	**Step Chart**	**Bar Chart**	**Stack Series Chart**	**Pie Chart**	**Histogram**	**Bubble Chart**	**Area Chart**	**Spider Chart or Radar Graph**	**UML**	**SysML**	**Gantt**	**Mindmap**	**WBS**	**Entity Relationship Diagram**	**Undirected Graph**	**Directed Graph**	**Cyclic Graph**	**Tree**	**Circular Layout**	**Radial Layout**	**Adjacency Matrix**
FX-Diagram																	x	x		x			
hallvard plantUML											x												
Sprotty-theia																	x	x	x	x			
JavaFX - Charts	x	x	x	x	x	x		x	x														
Zest														x	x	x	x	x	x	x		x	
Gef														x	x	x	x	x	x	x		x	
Sirius											x	x			x	x	x	x	x	x			
Graphviz																	x	x	x	x	x	x	(x)
Axiis	x	x		x	x	x		x	x														
birdeye	x	x		x	x	x			x								x	x		x	x	x	
Flare	x	x		x					x								x	x		x	x	x	
Gephi																	x	x			x	x	
Improvise	x	x		x					x											x	x	x	
The InfoVis Toolkit	x	x		x	x															x	x		x
JIT JavaScript InfoVis Toolkit					x				x											x		x	
JUNG																				x	x		
NetworkX																					x		
R	x	x		x	x	x																	

Overview of Frameworks w.r.t. supported Visualization Techniques

	2D										Common Graphs						Structures						
Framework	**Scatter Plot**	**Line Plot**	**Step Chart**	**Bar Chart**	**Stack Series Chart**	**Pie Chart**	**Histogram**	**Bubble Chart**	**Area Chart**	**Spider Chart or Radar Graph**	**UML**	**SysML**	**Gantt**	**Mindmap**	**WBS**	**Entity Relationship Diagram**	**Undirected Graph**	**Directed Graph**	**Cyclic Graph**	**Tree**	**Circular Layout**	**Radial Layout**	**Adjacency Matrix**
Tulip	x																x	x		x	x		
VisAD - Visualization for Algorithm Development	x	x																					
Wilmascope 3D																		x					
Google Vis	x	x	x	x	x	x	x	x	x				x										
ELK																	x	x	x	x	x	x	
Xchart	x	x		x	x	x	x	x	x	x													
Deprecated Frameworks																							
LiveGraph																							
charts4j	x	x		x		x				x													

4.2 Major Characteristics

This section presents the major characteristics of the frameworks highlights these in Table 4.2. Similar to the previous section, the characteristics are used to identify if a framework's capabilities allow a feasible implementation of the use cases previously discussed in Chapter 2.

The first column of the table highlights a framework's support for additional layout algorithms. For example, the framework *Graphviz* offers many different layouts or layout filters to enhance the the automatic visualization of structures.

The second column represents the last update of the framework. For an faster overview the time stamp is color coded:

- Green: Last update was performed within the last 3 years
- Orange: Last update was performed more then 3, but lest then 8 years ago
- Red: Last update was performed more than 8 years ago

The third column shows the license of a framework.

- Green highlighted licenses are compatible to the Eclipse Public License (EPL) and henceforth can be directly integrated in an future implementation.
- Orange highlighted need further processing before an integration into an implementation.
- Unmarked licenses need to be further investigated during the deeper analysis of the frameworks.

The *Other Features* column contains a list of all additional features of a framework that provide helpful functionality. For example, the framework *Gephi* supports the import of CSV-files.

The *Requirements* column contains specific information about required frameworks or versions for an integration. These requirements could be the *JDK* version or third party frameworks like *graphiviz*, which is required for the layout.

For further investigation the *EPL and GPL Problem*[1] is highlighted in a column. These two licenses are at first not compatible, but there are some solution to handle the incompatibility with some extra effort.

Finally, the last column contains general comments for a framework.

Since the frameworks *LiveGraph* and *charts4j* are already identified as deprecated, we have marked those as unsuitable for an further implementation at the end of the table.

[1] `https://www.eclipse.org/legal/epl-2.0/faq.php#h.sfzscklic49g`

Table 4.2: Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
PlantUML	auto (top to bottom or left to right)	2019-07-14	GPL LGPL Apache EPL MIT license	Wireframe graphical interface Archimate diagram Specification and Description Language (SDL) Ditaa diagram Mathematic with AsciiMath or JLaTeXMath notation	Graphviz/Dotm v2.26.3	(x)	Eclipse Policy and Procedure for 3rd Party Dependencies
SWTCharts		2019-05-05	EPL 2.0		Java SWT		
Nebula Visualization Widgets		2019-08-01	EPL 1.0	Intensity Graph Thermometer Tank Progress Bar Meter Gauge Knob			
JCharts		2004-07-12	BSD license				
Chart2d		2002-09-28	LGPL			x	Eclipse Policy and Procedure for 3rd Party Dependencies
JChart2D		2015-01-9	LGPL		JDK 1.5.0	x	Eclipse Policy and Procedure for 3rd Party Dependencies

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
JCCKit		2005-01-05	LGPL v2		JDK 1.1.8	x	Eclipse Policy and Procedure for 3rd Party Dependencies
mxGraph	auto layout (editable by user) horizontal/ vertical CircleLayout CompactTreeLayout EdgeLabelLayout, FastOrganicLayout HierarchicalLayout OrganicLayout OrthogonalLayout ParallelEdgeLayout PartitionLayout StackLayout	2019-07-26	Apache 2.0		The JavaScript client requires Google Chrome 30 and later, Firefox 31 and later, Microsoft Internet Explorer 9.0 and later, Microsoft Edge 20 and later, Safari 6.2 and later, Opera 20 and later, the default Android browser in Android version 5.0 and later, or the default iOS browser in iOS 8.0 and later. The servers require Java 6.0 or later or .NET 3.5 or later. Using PHP 5.0 is informally supported. The I/O module for PHP requires libxml. The PHP examples do not support image export.		

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
JGraphX	horizontal/ vertical CircleLayout CompactTreeLayout EdgeLabelLayout, FastOrganicLayout HierarchicalLayout OrganicLayout OrthogonalLayout ParallelEdgeLayout PartitionLayout StackLayout	2019-07-26	BSD license				
Opencharl2		2009-08-31	LGPL v3			x	Eclipse Policy and Procedure for 3rd Party Dependencies
JGraphT	auto layout (editable by user)	2019-06-03	LGPL 2.1 EPL		JDK 1.8 GraphViz .dot	(x)	Eclipse Policy and Procedure for 3rd Party Dependencies
GRAL Java Graphing		2016-03-12	LGPL v3		JDK 1.6.0	x	Eclipse Policy and Procedure for 3rd Party Dependencies

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
JFreeChart		2017-11-05	LGPL		JDK 1.6.0	x	Eclipse Policy and Procedure for 3rd Party Dependencies
FX-Diagram	Autolayout with Graphviz	2018-01-28			JDK 1.8.0 Graphviz .dot (auto Layout) Eclipse 4.4		
hallvard plantUML	auto (top to bottom or left to right)	2019-03-08	EPL		PlantUML eclipse plugin		
Sprotty-theia		2019-10-15	EPL 2,0, GPL v2	The client is implemented in TypeScript, SVG is used for rendering, and CSS for styling, Eclipse Layout Kernel (ELK) for Layout	Eclipse Layout Kernel (ELK)	(x)	Eclipse Policy and Procedure for 3rd Party Dependencies
JavaFX - Charts		2019-03-11	GPL v2 with the Classpath exception		JDK 1.8.0	x	Eclipse Policy and Procedure for 3rd Party Dependencies
Zest	Spring Layout Algorithm Tree Layout Algorithm Radial Layout Algorithm Grid Layout Algorithm	2018-08-27			Part of GEF (Graphical Editing Framework)		
Gef		2019-06-19	EPL		graphviz .dot		

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
Sirius	automatic layout (can be customized by creator)	2019-07-10					
Graphviz	dot-filter for drawing directed graphs neato-filter for drawing undirected graphs twopi-filter for radial layouts of graphs circo-filter for circular layout of graphs fdp-filter for drawing undirected graphs sfdp-filter for drawing large undirected graphs patchwork-filter for squarified tree maps osage-filter for array-based layouts	2016-12-25	Common Public License Version 1.0				
Axiis		2010-03-06	MIT License				
birdeye			MIT License				
Flare		2019-01-24	BSD license				

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
Gephi	Force-based algorithms Optimize for graph readability	2017-09-24	CDDL 1.0 GPL v3	Input/Output Gephi can read the majority of graph file formats but also supports CSV and relational databases import. Spreadsheet import wizard Database import Save/Load project files	Java 7 and 8	x	Eclipse Policy and Procedure for 3rd Party Dependencies
Improvise		2014-09-07	GNU Affero General Public License		Java version 1.6		
The InfoVis Toolkit		2005-11-30	MIT License		Java 1.4		
JIT JavaScript InfoVis Toolkit		2011-05-05	BSD				
JUNG		2010-01-24	Berkeley Software Distribution (BSD) license		JDK 1.5 or newer Collections-Generic Cern Colt Scientific Library 1.2.0		
NetworkX		2019-04-01	3-clause BSD license				

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
R		2019-07-05	GPL			x	Eclipse Policy and Procedure for 3rd Party Dependencies
Tulip		2019-09-13	LGPL			x	Eclipse Policy and Procedure for 3rd Party Dependencies
VisAD - Visualization for Algorithm Development		2013-10-24					
Wilmascope 3D	Force-based algorithms	2003-10-17	LGPL	Force Directed Algorithm (FDA) for Layout	JRE 1.5.0 and Java3d 1.3.1.	x	Eclipse Policy and Procedure for 3rd Party Dependencies
Google Vis		2017-02-23	Creative Commons Attribution 4.0 License, Apache 2.0				

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
Eclipse Layout Kernel - ELK	Layered layout, Random Layout, Tree Layout, Forced layout, DOT Layout, Circo Layout, Box Layout	2019-12-18	EPL 2.0	Standard layout algorithms of ELK are in plain java and are ready to be used out of the box. There is also a possibility to provide custom algorithms ELK also provides eclipse based infrastructure to connect them to editors and viewers, to have a possibility of automatic layout if there is change in the diagram.	JavaFX/ GEF/ GMF		
Xchart		2020-01-11	Apache 2.0	Multiple Y-Axis charts Scatter charts, Donut charts, Stick charts, Dial charts, OHLC charts, Error bars, Logarithmic axes, Themes - XChart, GGPlot2, Matlab CSV import and export, Real-time charts	Java 8 and up		
Deprecated Frameworks							

Overview of Frameworks w.r.t. major characteristics

	Other					Eclipse Plugin Development	
Framework	**Layout**	**Last Update**	**Licenses**	**Other Features**	**Requirements**	**EPL & GPL Problem**	**Comments**
LiveGraph							Deprecated - Dead links on Web page for documentation
charts4j		2011-01-9	MIT License				Deprecated - Dead links on Web page for documentation

5 Conclusion and Outlook

As a first step towards the early assessment of design decisions, this document deals with visualizing results by e.g. presenting charts, diagrams, and other visual approaches to users in order to allow visually assessing results produced in this as well as the other technical work packages. It introduces and discusses both, high as well as low level use cases for the methods, processes and tools for assessment to be researched within PANORAMA. From a high level perspective, the derived use cases were grouped into four categories:

- UC-4-ATB: Application Temporal Behavior (see Table 2.1)
- UC-4-ME: Mapping Evaluation (see Table 2.2)
- UC-4-DA: Deployment Alternatives (see Table 2.3)
- UC-4-SA: Safety Analysis (see Section 2.3.1)

In order identify suitable techniques for visualizing data resulting from e.g. analysis, Chapter 3 has analyzed various visualization techniques and their applicability on selected use cases. It gave a short introduction into *Data Visualization* and provided several examples on how those techniques are or may be used in e.g. automotive and avionic systems development. Additionally, it described external tools that integrate such visualization techniques for e.g. assessing real-time systems. Finally, it discussed *Failure Propagation Visualization* that shows how the results of the safety relevant analysis aspects resulting from e.g. Work Package 3 can be visualized and assessed in Work Package 4.

In order to provide a basis for the subsequent implementation of the different visualization techniques, Chapter 4 identified 40 visualization frameworks and both, investigated and analyzed those w.r.t. their features, capabilities, and characteristics. For each of the these frameworks, its capabilities towards supported visualizations were identified and first characteristics for a selection process collected. During the collection of frameworks, two framework were already identified as deprecated.

What remains is to assign the collected frameworks to the identified use cases and evaluate the frameworks with a deeper analysis, which is subject of Work Packages 4's subsequent Task *"T4.2 - Evaluation of existing assessment and visualization approaches"* and its corresponding Deliverable *"D4.2 - Specification and concepts on system assessment for early design phases"*.

Appendix

1 Use Case Pattern

Table 1: Use Case Pattern with short Description

	Section	Content
1	ID	Unique identifier of the use case in the development project or program
2	Name	Name of the use case in the model (this name is shown in the use case diagram)
3	Author	Names of the Authors, that described the Use Case
4	Priority	Priority of a Use Case based on the used prioritization technique
5	Description	Short Description of the Use Case
6	Trigger	Event that triggers the execution of the Use Case
7	Actors	List of all actors, that are related to the Use Case
8	Preconditions	Preconditions that are fulfilled before execution of the Use Case
9	Postconditions	Set of postconditions that are fulfilled after execution of the Use Case
10	Result	Result of the use case, i.e., the added business value which is provided to the actors after execution of the use case
11	Main scenario	Normal sequence of activities (e.g. execution flow in 70% of all cases)
12	Alternative scenario	Set of alternative activities. Each alternative process also leads to a successful execution of the use case (e.g., in 30% of cases)
13	Exception scenario	Set of exception scenarios. These scenarios are executed when an exceptional situation occurs in the use case process. These scenarios ensure a controlled error and exception handling.
14	Relations to other Use Cases	Short description of the relationships to other Use Cases (e.g. extend, include, or generalization)

Bibliography

[Abe] A. Abela, *Chart suggestions - a tought-starter*, www.ExtremePresentation.com.

[AM15] A. Aleti and I. Moser, "Fitness landscape characterisation for constrained software architecture optimisation problems," Dec. 2015, pp. 11–20. DOI: 10.1109/ICECCS.2015.12.

[App] App4MC consortium, *App4mc documentation*, https://www.eclipse.org/app4mc/help/app4mc-0.9.6/.

[Ecl20] Eclipse Foundation. (Jan. 31, 2020). Trace compass, [Online]. Available: https://www.eclipse.org/tracecompass/.

[LGY12] F. Liu, H. Gu, and Y. Yang, "Dtbr: A dynamic thermal-balance routing algorithm for network-on-chip," *Comput. Electr. Eng.*, vol. 38, no. 2, pp. 270–281, 2012, ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2011.12.006. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2011.12.006.

[Maz09] R. Mazza, *Introduction to Information Visualization*, 1st ed. Springer Publishing Company, Incorporated, 2009, ISBN: 1848002181, 9781848002180.

[PAN20] PANORAMA Consortium, "Panorama d2.1 - state-of-the-art analysis on editors, viewers and transformationtools used in the automotive and avionic industry," ITEA3 Project PANORAMA, Deliverable, Jan. 31, 2020.

[SM00] H. Schumann and W. Müller, *Visualisierung: Grundlagen und allgemeine methoden.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, ISBN: 978-3-642-57193-0. DOI: 10.1007/978-3-642-57193-0_3. [Online]. Available: https://doi.org/10.1007/978-3-642-57193-0_3.

[Vec] Vector, *Canape*, https://assets.vector.com/cms/content/products/canape/Docs/CANape_ProductInformation_EN.pdf.

[WJZ+15] J. Wang, C.-H. Jeong, N. Zimmerman, R. Healy, D. Wang, F. Ke, and G. Evans, "Plume-based analysis of vehicle fleet air pollutant emissions and the contribution from high emitters," *Atmospheric Measurement Techniques*, vol. 8, pp. 3263–3275, Aug. 2015. DOI: 10.5194/amt-8-3263-2015.