CHALMERS

Optimizing readability for complex UML diagrams generation

Master of Science Thesis in Computer Science and Engineering:
Algorithms, Logic and Programming Languages

MYRIAM ECONOMOU
ZEYNEP GÖKMEN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, 2015

Optimizing readability for complex UML diagrams generation

Myriam Economou
Zeynep Gökmen

Examiner: Gerardo Schneider
Supervisor: Rogardt Heldal

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, 2015

Abstract

Information visualization provides a way to convey key notions and concepts in a single image. It is a powerful tool if the visualization product is readable and has the appropriate level of information for the targeted audience. Readability means the ease with which the graphical representation is understood and processed by the viewer. Although several visualization techniques exist, this thesis work focuses on UML diagrams for software architectures' representation. Tools exist for automatic UML diagram generation from software architecture description but the outputs for complex diagrams are cluttered and unreadable.

The aim of this thesis is to provide a solution to generate automatically readable software architecture visualizations for a computer scientist's audience of all level. It is an effort to make progress in this research field by applying theoretical state-of-the-art research findings to a design study and evaluate how they perform. This work has been conducted for Ericsson (telecommunications Company) with the goal of improving the communication within and between teams as well as keeping their documentation updated in a simple manner.

The result of this thesis work is a prototype enabling the users to control several abstraction parameters and control diagram readability depending on the targeted audience. Experimentation and surveys have been conducted to determine the tool's best settings for the audience targeted during this study.

The prototype has been developed in Java and uses PlantUML as its UML diagram visualization engine.

Acknowledgements

We would like to express our sincere gratitude to our Ericsson supervisor Thorbjörn Larsson and university supervisor Rogardt Heldal for their support during our thesis work. We would also like to thank Chalmers researchers especially Truong Hu Quang for their helpful guidance. Besides them, our sincere thanks go to Fredrik Nilsson and his team at Ericsson Lindholmen, for the friendly working atmosphere they offered during the thesis work. Lastly, we would like to thank our families for their continuous encouragement throughout the thesis.

Table of contents

I. Introduction

1. Context

Information visualization is important in many domains and an active research area. The goal of information visualization is to create a readable graphical representation of a system**.** Ideally, the viewer should be able to discuss correctly key notions faster than if he had read a written description of them. If this is not the case (slow learning, incorrect concepts, misunderstandings, etc.) then the representation is of bad quality and missed its purpose. Successfully achieved, information visualization saves time, energy and sometimes money for both the person sharing the information and the one receiving it.

Readability is tightly related to the targeted audience as well as to the identification of the key concepts which should be understood by this audience. The same graphical representation might be easily readable for a group of persons while being hard to understand for another (e.g. House electrical installations map for an electrician vs for the house's owner). Additionally, if the representation is too simplified (abstracted), the targeted audience might misunderstand or miss the key notions that were set as goals to be understood. Diagram readability is particularly difficult to achieve when dealing with complex systems.

Computer systems are complex and dynamical as their structures change constantly especially when they are in their development phase. This makes visualization for such systems complicated. Luckily, automatic visualization tools are very valuable as they offer the possibility of keeping the systems' visualizations synchronized as the system evolves without much effort. The problem is that finding an automatic visualization tool producing readable diagrams at all time is challenging.

Ericsson is a telecommunication company in charge of setting up, developing and maintaining radio antennas for mobile devices. This thesis has been conducted at Ericsson's DURA department (Development Unit Radio) which is responsible for developing baseband processing functions for Ericsson's products. The DURA department currently describes its software system using text files from which they are able to generate component diagrams automatically. Unfortunately as their software system is large and complex, the generated diagrams are cluttered and unreadable.
The aim of this thesis was to find a solution generating automatically readable diagrams modeling DURA software architecture. This thesis is a design study resulting in a prototype generating readable diagrams for their software. Results were evaluated both by objective criteria and surveys.

2. Motivations

Ericsson DURA department describes its software architecture using textual models written in a language they developed themselves called AMZ (YAML based [1]). From this model, they are able to generate C code which is the complete software's implementation.

As their software system can be quiet big and complex, the team also wants to be able to automatically generate UML diagrams from AMZ files. So far the team has succeeded to generate activity diagrams by using PlantUML [3]. However, the generated component diagrams especially the one representing the full software architecture is cluttered and unreadable (Figure 1).

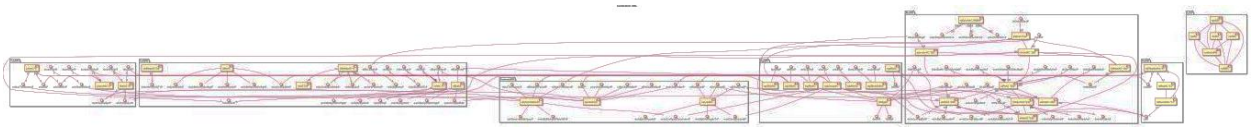


Figure 1: Original component diagram of the full software

DURA's software is composed of modules each containing a set of components and interfaces. Figure 2 shows a zoom of Figure 1 on one of its module.

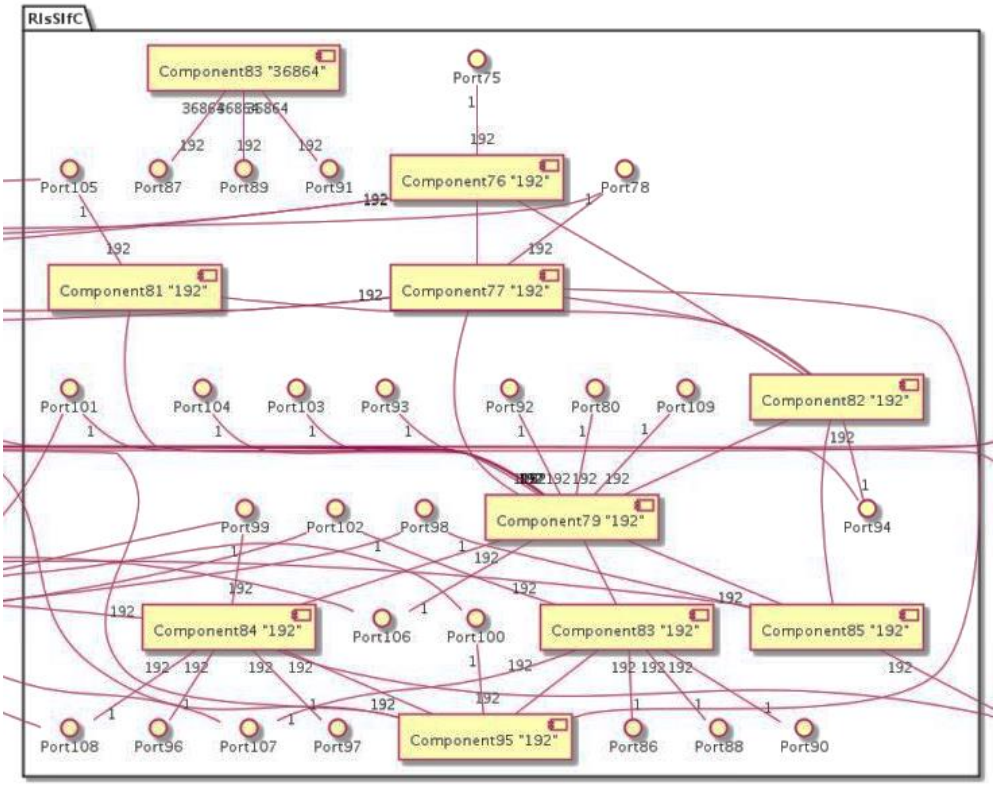


Figure 2: Original component diagram of a module (zoom of Figure 1)

As you can see in Figure 2, the resulting diagram is unreadable. For example, it is impossible to see if the interface "Port 103" is connected to the component "Component 79" or not.

As a result these diagrams are completely unusable by the team. The department researched solutions and tools generating readable component diagrams of their software without success.
The result of this thesis is a proof of concept in the form of a program generating readable diagrams from Ericsson's software textual description and representing its software architecture.

3. Aim of the thesis

This thesis work is an effort to make progress in the field of automated information visualization by applying and evaluating theoretical and empirical research findings to guide the design of our prototype.
This leads us to the following research questions:
RQ1 Is visualization better than textual description in some cases?
RQ2 How can we improve readability of the given component diagrams?
RQ3 How to evaluate the quality of the representation in term of readability?

4. Overview

This thesis report is organized as follows. The second section introduces the foundations of the work followed by related work section. In the fourth section methods used for the thesis work are described. In the next section, the design study is explained whereas further research and projects are presented in the sixth section. Then the report ends with concluding discussion.

II. Foundations

In order to clarify this work for the reader, it is needed to give the foundations of the concepts which this thesis work is based on. First a general view of software visualization is given followed by a description of relevant UML diagram standards. Then, layout algorithms, aesthetics rules and abstraction strategy are introduced. Finally, audience analysis approaches are described.

1. Software visualization

Sharing ideas and concepts through graphical visualization is an idea as old as when humans could draw (e.g. Egyptians hieroglyphs, cavemen drawings). The creation of computer and computer graphics allow us to create powerful visualization tools. For example, it is now possible to dynamically visualize the helix structure of a DNA sample.
However, generating readable software architecture visualization is challenging as their structure is usually complex and dynamic.

Automation of software visualization is also key for visualization of dynamical systems such as software. It provides **consistency** in term of synchronization with the actual state of the software but also consistency in the standards and output format. Additionally, it is **time and cost saving** since manually drawing diagrams is time consuming especially for complex system.

2. UML diagrams

Unified Modeling Language (UML) is a set of diagrams for information visualization. UML is the standard choice for visualization of software systems. It is taught in universities and understood by most software developers. This is why UML diagrams have been chosen for this thesis software visualization.

UML defines several kinds of diagrams which can be grouped into two categories [4], [5]; structural diagrams and behavioral diagrams. Structural diagrams model the static structure of a system by representing its parts as well as dependencies. Behavioral diagrams represent the system behaviors. Behavioral diagrams are especially useful to model system action flow.
Since this thesis aims to produce readable software architecture representations, only relevant UML standards to this work are introduced: class diagrams and component diagrams.

2.1 Class diagrams

Class diagrams show the relationships between classes, interfaces, packages and class instances. It is traditionally used to represent object oriented systems.

Figure 3 shows a class diagram example modeling a car owned by a person, driven by a person and having four wheels. The car has two characteristics (attributes): a name and a creation year. It also has two functionalities (methods): to go and to stop. All objects are represented as classes (rectangle box). Labels along the relationship lines are used for clarification on the nature of the

relationship. Car attributes are listed in the upper part of the class element and methods in the lower part.

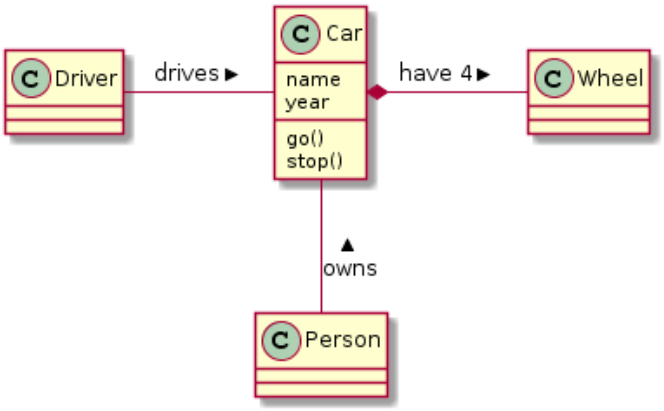


Figure 3: Class diagram example

2.2 Component diagrams

Component diagrams show components, interfaces and their dependencies. They are quite useful to model software architectures.

A component is represented by a yellow rectangle and interfaces are represented by circles (or lollipops). Connections are modelled by lines. Figure 4 shows a component diagram example. The component Data has three interfaces (port1, port2, port3) and a connection with the component Folder.

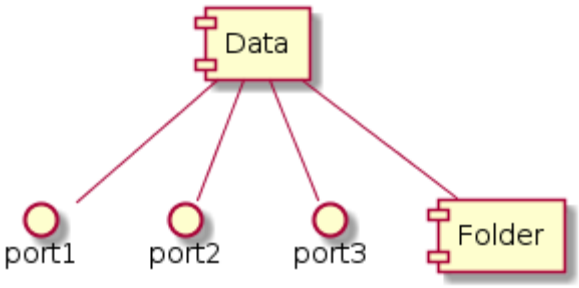


Figure 4: Component diagram example

3. Layout algorithms

Layout algorithms are strategies used for component placement in a diagram. This section introduces the two layout algorithms relevant for component/class diagrams: force-directed layout algorithms and layered layout algorithms. Other layout algorithms didn't appear to be suited for the type of system we are dealing with.

3.1 Force-directed layout algorithms

This algorithm is based on the following principle: two connected nodes attract each other whereas independent nodes are pushed apart. It assigns forces to the nodes and edges based on their positions and balanced them to reach an equilibrium state. As a result, edges tend to have almost the same length and the nodes that are not connected tend to be placed far apart from each other (Figure 5).

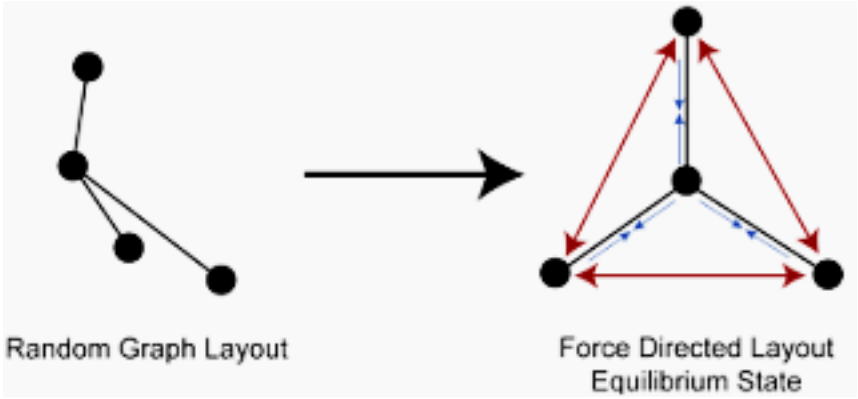


Figure 5: Force directed layout [12]

Several forced-based layout algorithms such as Kamada and Kawai [18] and Fruchterman and Reingold [19] exist. They all follow this same ground strategy and the difference between them is the available parameters to fine-tune the output such that attraction and repulsion factors.

Graph drawing tools like Graphviz [2] (neato), Gephi [13] and Tulip [14] can be used to test several force-directed algorithms on graphs and evaluate their effects on diagram readability.

3.2 Layered (Hierarchical) layout algorithms

Layered layout algorithms place elements in a hierarchical manner. Nodes are assigned a rank value and positioned into layers. For directed graphs, edges tend to be straight and directed downwards.

One of the most renowned layered algorithms is designed by Kozo Sugiyama et al. called Sugiyama algorithm [17]. The strategy consists of four main steps:

1. Removing all cycles from given graph.
2. Assigning each node to a layer. It also adds dummy nodes and edges if generated graph has long span edges.

3. Moving nodes within layers in order to minimize edge crossings (Figure 6).
4. Assigning node positions horizontally to remove dummy nodes (Figure 6).

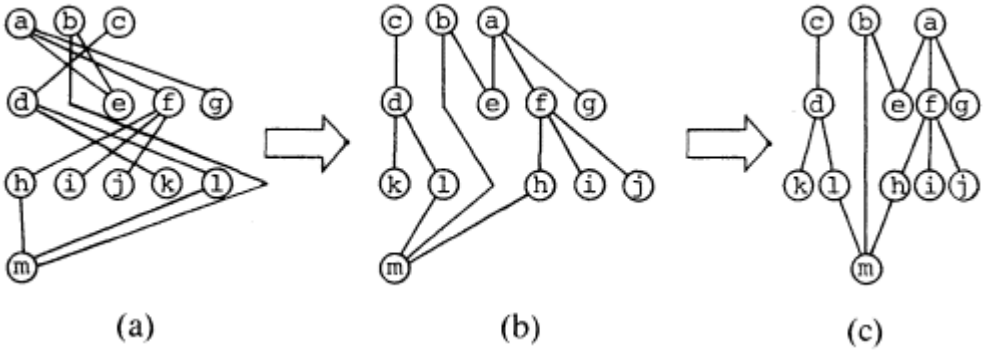


Figure 6: A progress example of Sugiyama algorithm: (a) -> (b) shows Step 3 and (b) -> (c) shows Step 4 [17]

Similarly to force-directed layout algorithms, tools such as Graphviz (dot), Gephi and Tulip can be used to test layered layout algorithms and their effects on readability.

4. Aesthetic rules

To find an adapted layout algorithm becomes a difficult problem while generating diagrams for a complex system [7]. Resulting diagrams are usually cluttered and hard to read. Research has been conducted to identify sets of generic criteria to evaluate diagram readability. Such criteria are called “aesthetic rules”.

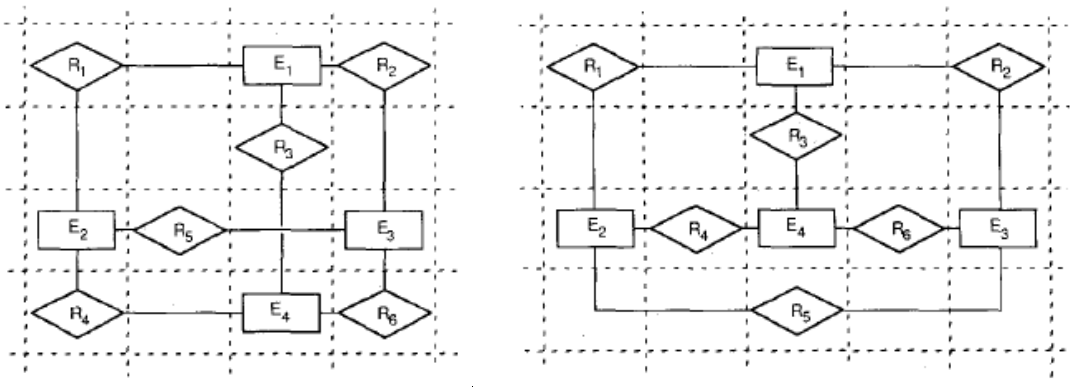
Aesthetic rules can be seen as guidelines making diagrams more easily understood and processed by the human brain. It has its root in cognitive science. Dabo Sun and Kenny Wong took those results a step further and proposed a list of UML class diagram specific aesthetic rules [24]. This criteria list is shown in Table 1.

Criteria	Description
Joining inheritance edges	Edges pointing to the same component should be merged.
Association representation	Edge labels should be alongside the edge line – not in separated label box.
Selectivity	Abstraction – showing only relevant elements to convey key notions.
Similarity – using colors	Using color to group elements together – visual grouping
Minimizing crossings	Minimize edge crossings to make them easier to follow.
Minimizing bends	Minimize edge bends to make them easier to follow.
Grouping related elements	Related/Connected components should be place as close as possible.

Proximity - Reducing length of edges	Adapting edge length to optimize the grouping of connected components and differentiate independent groups.
Avoiding overlapping	Components should not overlap
Symmetry	Component placement symmetry should be maximized.
Orientation	Option of drawing the diagram vertical or horizontally.
Orthogonality	Edges should be drawn orthogonally.
Horizontal labels	Labels should be written horizontally.

Table 1: Aesthetic criteria list [24]

However it can be hard if not impossible to satisfy all the rules at the same time as some rules are conflicting with each other. For example two equal diagrams are displayed in Figure 7. First, the aesthetic rule of minimizing number of bends is applied to the diagram which results with a crossing edge (a). Then, applying the rule of minimizing number of edge crosses to the diagram ends up with two edge bends (b) [9].



(a) **(b)**

Figure 7: Conflicts among aesthetics [9]

Besides, some rules such as minimizing the number of crossing edges are known NP-hard problems [10]. Finding the optimized layout having the least number of crossing edges cannot be solve in polynomial time, in other words it will take extremely long time for the program to optimize the layout according to this criteria for a complex diagram and is therefore unwise to try to optimize several criteria at the same time.

In summary, it is not possible to design a program optimizing the diagram layout according to all these aesthetics rules. Instead research has been conducted to identify which aesthetic rules are the most powerful ones i.e. improves readability the most or as Helen C. Purchase wrote “maximize human performance” [35] and which rules have a lesser effect.

Purchase et al. took theoretical research results and conducted experiments to identify aesthetic rules improving readability the most [29]. According to their experiments for UML class diagram

readability, minimizing edge crosses was the most important rule and reducing the number of bends was the second important one. The experiments also showed that orthogonal structure and directional arrows had lesser impacts for readability. Table 2 summarizes their findings for UML class diagrams.

Aesthetic criteria	Ranking
Minimizing edge crosses	Most important
Minimizing number of edge bends	Second
Maximizing orthogonal structure	Little impact
Directional arrows	Last

Table 2: Priority list – top 2 and last 2 rules [29]

These findings were used to design our experiments and evaluate our diagrams readability.

5. Abstraction strategy

When it comes to complex diagrams having a lot of elements, there is only so much you can do with component placement. The result will always be cluttered and more or less unreadable. Therefore it is important to have identified key concepts that should be understood when looking at the diagram in order to show only relevant elements to the viewer and have an adapted level of information.

Research has started on automatic UML class diagram generator using machine learning algorithms to predict the appropriate level of abstraction [25]. Briefly, the machine-learning engine has to be trained by an experienced technician in the software. The technician points out which classes are the most important to understand the software and which are irrelevant. The machine-learning engine will then rank every class according to criteria such as name resemblance to important classes, number of connections, and number of methods in order to create different possible abstraction level. Users will then be able to adjust which abstraction level they want to see via the tool graphical interface.
One of the abstraction strategy proposed in this thesis is based on the same principle.

6. Audience analysis

Readability is tightly related to the targeted audience. The level of information presented has to be adapted to them and this is why identifying the targeted audience characteristics is key to design quality diagrams.

However analyzing an audience is not an easy task. It requires finding relevant similar traits in several individuals to define a group profile and deciding when to split an audience group into subgroups to make profiles more consistent with reality. Research in this field yielded to several audience analysis approaches and we present two strategies that we followed for this work.

6.1 Multidimensional audience analysis

Multidimensional audience analysis is an approach developed by the researcher Michael Albers [31]. He proposes a different approach for the creation of documents dynamically adapting their

content to the audience. It is particularly suited to our thesis work as the resulting visualization has to be adapted to different kind of audience.

This approach articulates the analysis around independent criteria groups such as the level of system knowledge, key concepts for the specific audience/subset and their cognitive abilities.

By analyzing the targeted audience (or subgroups) using this strategy, one can define each sub-group characteristics in order to guide the selection of an appropriate abstraction level and parameters.

6.2 Bottom-up approach

Another approach is the “bottom-up” approach proposed by Sarah van der Land [33]. In this approach, the audience plays an active part in guiding design choices. By using survey groups (or target groups), they were able to generate higher impact graphics than the ones created by a more classical approach where the targeted audience isn’t involved during the project.

This approach consists of using target groups and submitting them to surveys during the complete project design phase in order to identify which finished product suits them the best. The difficult tasks of this approach are to identify target groups representative of the full audience as well as designing the surveys. If done correctly, one can guarantee a successful product and therefore move with confidence from the design phase to the development phase.

We applied this approach when designing the prototype and evaluating parameter effects on readability for the targeted audience subgroups.

III. Related Work

From our background research, there are no tools available on the market able to generate automatically component diagrams and adapt the abstraction level of their diagrams to the audience. On one hand there are available tools able to generate automatically UML diagrams such as PlantUML and TextUML but with no functionality to adapt the level of information displayed to the viewer whereas on the other hand research is conducted to develop machine-learning tool [25] able to predict which information should be shown to the user but without having been tested in the industry and most importantly not market ready.

This thesis combines research results from information visualization, layout algorithms, diagram readability and audience analysis approaches in a prototype, putting those results under experimentation in a company context. To our knowledge this is the first time it has been done and therefore can be **valuable**.

Several layout algorithms might influence readability as they act on component placement. Kamada and Kawai [18] and Fruchterman and Reingold [19] proposed different kinds of force-directed layout algorithms whereas Kozo Sugiyama et al.[17] designed a layered layout algorithm for hierarchical system structures. Force-directed and layered layout algorithms place elements according to different strategies and therefore might affect readability. These two algorithms were used in this thesis work to analyze their impact for readability.

Evaluation of UML diagrams quality has been done using aesthetic criteria. For instance, Dabo Sun and Kenny Wong proposed a list of aesthetic rules such as similarity, symmetry and orthogonality for diagram readability evaluation [24]. Another valuable research has been done by Helen C. Purchase et al. where they refined that list for class diagram [29]. Both research findings have been used as ground to evaluate diagram readability and the prototype quality in this thesis work.

In addition, audience analysis approaches had also an important role in this study. Micheal Albers proposed multi-dimensional audience analysis approach for the creation of documents dynamically adapting their content to the audience. He analyzed the audiences according to different criteria e.g. knowledge and abilities [31]. Another audience analysis approach named “bottom-up” was used by Sarah van der Land in her study about HIV/Aids education for young people in South Africa [33]. In this approach, audience plays an active role in guidance of the work. The two approaches were conducted during this work to understand the audience characteristics and to evaluate the results.

IV. Methods

1. Methodology

For this thesis, a design research methodology was followed to design and implement a prototype generating readable diagrams for DURA's software. Figure 8 below shows the process we followed to achieve this goal.

Figure 8: Design research process

First a comparison study was conducted to find a tool able to generate UML diagram graphics fitting our scope and limitations.
Then, several design ideas were tested such as trying out different layout algorithms and abstraction strategies.
The resulting diagrams were then evaluated using the aesthetic criteria defined by Purchase et al. [29] as well as by conducting tests with audience target groups and surveys.
Finally the results were evaluated to design the prototype architecture and implement it.

2. Subjects

The targeted audience was composed of two subgroups: Ericsson engineers (A1) and other computer scientists having no prior knowledge of the system (A2). The subjects were picked among people we know and among engineers working in the DURA's team. We could gather approximately 10 persons for each group [10 for A1 and 12 for A2]. The A2 group was composed of both students and computer scientists working at other companies.

3. Evaluation

3.1 Aesthetic criteria

In order to assess diagram's readability improvements, a grid of aesthetic criteria based on Sun-Wong and Purchase et al. [29] research results were used. Data were gathered for each software models at our disposable. Table 3 shows the grid used for each diagram evaluation.

Aesthetic criteria	Value
Edge crosses count	
Edge bends count	
Number of elements	

Table 3: Aesthetic criteria evaluation

The goal was to determine which representation type produces diagrams with the lowest aesthetic criteria values as it would mean improved readability.

3.2 Audience tests

Evaluation by the audience in the form of tests was conducted to determine which software representation gives the subjects the best overview of the system. A first set of tests was performed to establish if visualization was better than textual description to start with. Then another set of tests was used to determine which representations were the best suited for each target groups.

Interviewees were asked to solve a set of tasks. For each task, the correctness of the answer was noted as well as the time it took for the person to solve the task. These results were used to assess both performance and accuracy.

3.2.1 Tasks

The goal achieved by the system representation should be that the viewer gets a global understanding of the software structure. The tasks were chosen in such way that they measure the overview that the viewer has of the system. We identified the two tasks listed in Table 4.

Task	Answer correctness (yes/no)	Time (min)
Counting the number of elements		
Finding the most connected component		

Table 4: Survey task grid

3.2. 2 Test data

In total, we had six software modules available to us which we all used as test data. For each of these modules we generated a textual representation, an original component diagram representation and our customized representations.

3.2. 3 Test settings and randomness

To not introduce bias in the results, randomness was an important factor. Each participant was assigned data sets randomly and shouldn't have performed other tasks on the same data models. This criterion is meant to avoid any previous learnings of the model making the subject suddenly more efficient on a task not because of its representation but because he has seen the model before and remembers its structure.

3.3 Surveys

Surveys were conducted to get qualitative data about the system representations. The interviews were semi-structured. First each software representation was presented to the subject. Then they were asked which one (diagrams or text files) they felt was the easier to understand and to work with.

Ericsson engineers were also asked to choose which representation they would show to new engineers starting working in the team. The idea was to see if the choice of the experienced engineers for the beginners would match the representation(s) chosen by the subject group A2. Survey questions are given in Appendix B of this thesis report.

Comments on each representation were gathered and considered during the prototype architecture design phase.

V. Design study

1. Textual UML tools comparison

Textual UML tools provide a way to generate UML diagrams from textual models. Given the time constraints and to avoid creating a UML generation tool from scratch we explored available options. In order to be integrated in our final solution, a suitable UML generation library should fit the following requirements:

- **Environment:** It should work on a Linux operating system.
- **UML functionalities:** The tool should have component and class diagram functionalities.
- **Open-source**: As from our background research there are no available tools solving the issue treated in this thesis, the libraries used should be open-source in order for us to evaluate the possibilities to use them and extend them.
- **Stability:** As the tool is to be used in a company setting the tool should be stable i.e having active development (releases) and user support.
- **Command-line**: The aim being to automate diagram generation, the tool should be runnable by command-line.

Table 5 shows the comparison of the four most suited tools available: PlantUML, Umple, UMLGraph and TextUML. Other tools neither fulfilled the environment requirements nor had relevant UML functionalities [3], [26], [27], [28].

UML Tools	PlantUML	Umple	UMLGraph	TextUML
Open-source	Yes	Yes	Yes	Yes
Command-line(jar file)	Yes	Yes	Yes	No
UML support	Activity, component, class, sequence, use case, object	Class, state diagrams	Class, sequence	Class
OS/Environment	Windows, Linux and OS X	Mac-OS, Windows and Linux	Java	Java
Dependencies	Java, Graphviz	JVM	Graphviz	Java 8, EclipseGraphviz
Source Language	Java	Umple	Java	Java
Input format	PlantUML(.puml)	Java, C++, PHP, Ruby with embedded Umple code	Java	TextUML(.tuml)
Output format	png, svg, LaTeX format	Java, C++, PHP, Ruby, Umlet, Yuml, Textuml, Json, Papyrus XMI, svg, SQL	PostScript,gif, png, svg, jpeg, fig	png, jpg
Last Update/Release	2015	2014	2012	2015

Table 5: UML tool comparison

According to the comparison study of the four most suited tools, PlantUML [3] was fullfiled all requirements. It also was the most active project with a new version deployed this year. Additionally, PlantUML has an active user community which is important for later support if needed. All those factors led our decision to choose PlantUML to build our prototype.

2. Visibility comparison

This section describes the different transformations tried on the software models provided to us. The goal was to find out if some of those transformation ideas improved readability. First we tried out two different layout algorithms using the same graph layout library used by PlantUML. Then we applied several abstraction strategies that we created. Each diagram was then evaluated.

2.1 Layout algorithm comparison

Component placement influences aesthetic rules i.e. diagram readability. Therefore it was interesting for this research to try out different layout algorithms and see how they perform on our models.

The library used is called Graphviz. Graphviz [2] is an open source project for graph visualization. It has several modules for different diagram rendering. Among them, two are of interest to this project:

- Dot layered layout algorithm based (Sugiyama algorithm [17]).
- Neato force-directed algorithm based (Force Atlas).

PlantUML uses the Dot Graphviz module as its graph visualization library. Therefore, we decided to conduct experiments comparing how Dot performed compared to Neato. The aim of this study was to see whether or not it was worth the effort to change PlantUML source code to use the Neato module instead of Dot.

In order to analyze difference of Dot and Neato outputs on our models, it is needed to have a textual representation of each model written in Graphviz (.dot) format, since PlantUML uses its own format (.puml). We implemented Python scripts to transform all our models to Graphviz format and then ran both tools to generate the diagrams.

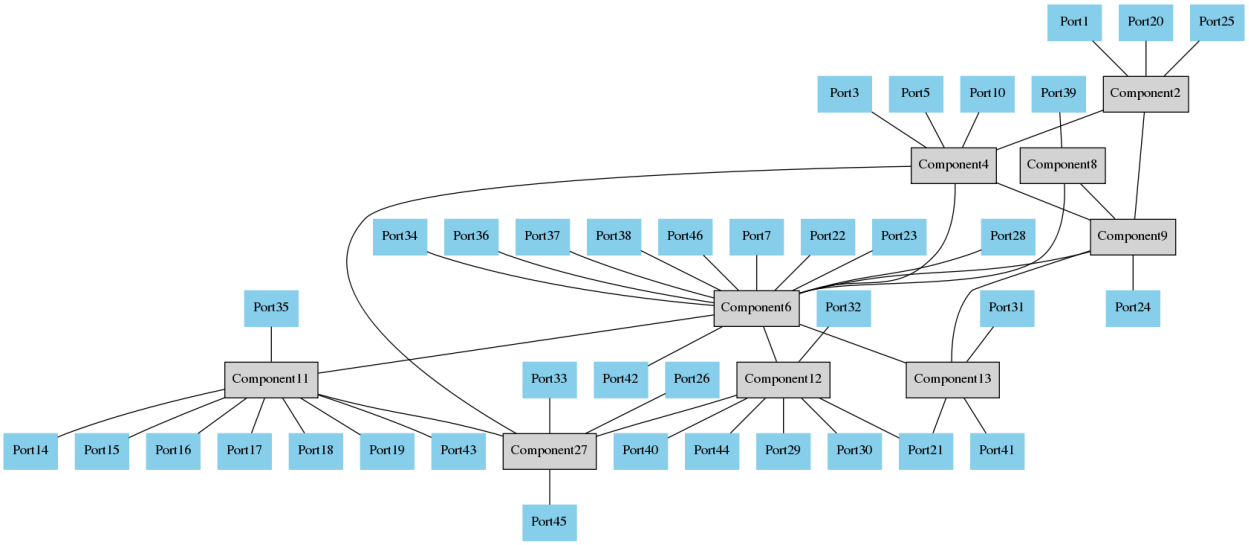


Figure 9: Dot applied on module RLS

Figure 9 shows the output of Dot applied on a module called RLS. On this figure, one can see the layered structure clearly. For example Component 4 and Component 8 are on the same layer whereas component 11, 12 and 13 are on another one.

Figure 10: Neato output on module RLS

Figure 10 shows the output of Neato applied on the same module (RLS). As we can see on this figure Neato spreads components more evenly on the view and makes clusters more distinguishable.

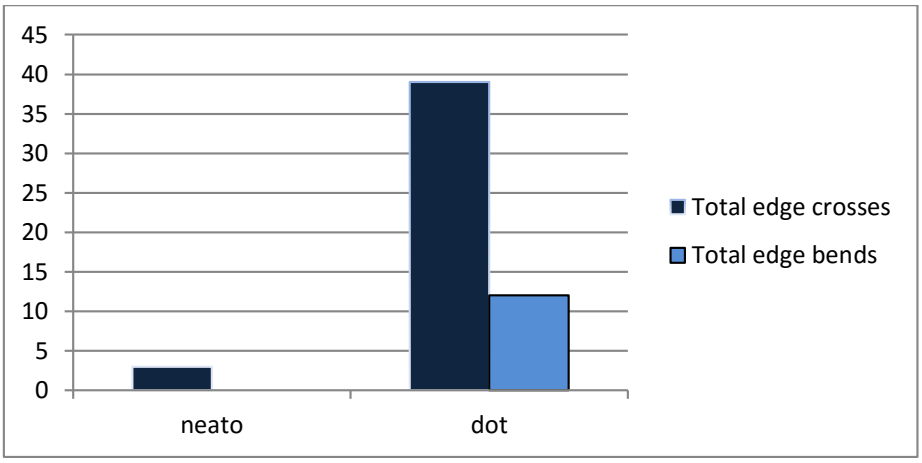


Figure 11: Total edge crosses and bends obtained from all modules

Figure 11 presents the results of the aesthetic criteria evaluation of all system modules. From this figure, one can clearly see that Neato performed better than Dot according to the aesthetic criteria. In average, Neato generated 92% less edge crosses and removes all bends compared to the diagrams generated by Dot for all modules.

In conclusion, using Neato instead of Dot for component placement seems to improve readability according the aesthetic criteria. However, the diagrams of the most complex modules were still unreadable as there were still too many elements on the view.

2.2 Abstraction strategies

Although applying a force-directed layout algorithm seems to improve readability, it is not effective enough on diagrams where the numbers of elements to display is too large. This section describes other approaches investigated to lighten complex diagrams by either decreasing the amount of information on it or modifying the way it is presented.

Figure 12 shows the complete system overview that we will be working with. As it is quite unreadable Figure 13 shows an enlarged view of one module from full software.

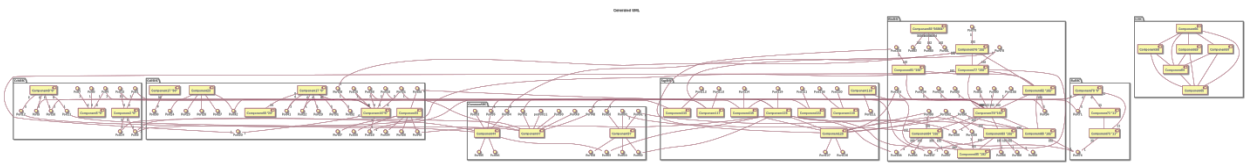


Figure 12: Original component diagram of the full software (L1SC)

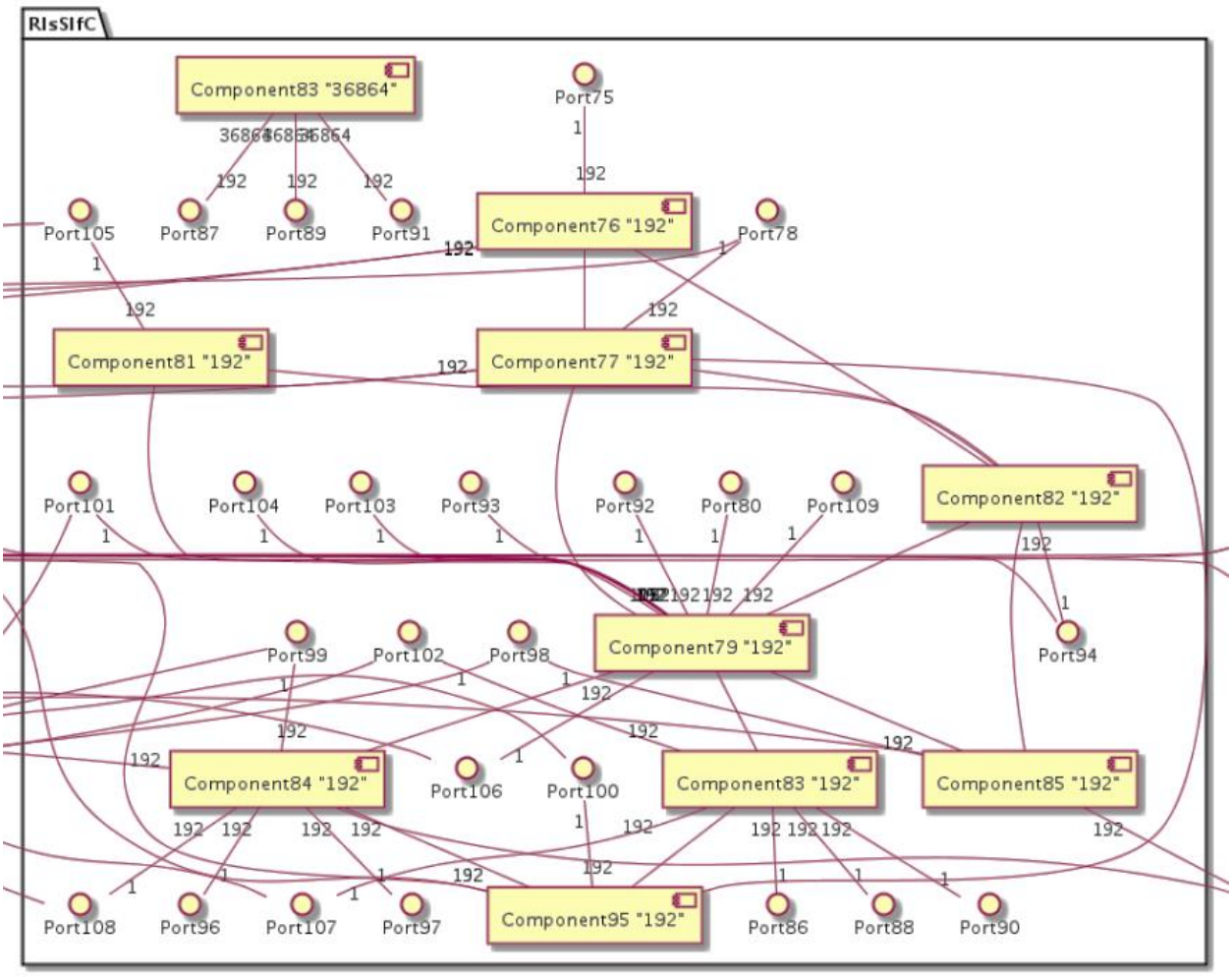


Figure 13: Enlarged view of one module from Figure 12 (zoomed)

As you can see, there are too many lines and components in Figure 13. Some lines are indistinguishable from each other making the diagram confusing. For example, it is impossible to see if the interface “Port 103” is connected to the component “Component 79” or not.

2.2.1 Formalism

In order to create more compact diagrams, we transformed the component diagram representation to a class diagram representation. To do so, one can represent components as classes and list its interfaces as attributes [34].

Figure 14 shows a conversion of the component diagram to a class diagram following the rules described above. The Data and Folder components are now classes. All Data’s interfaces are listed in the attribute section on the Data class.

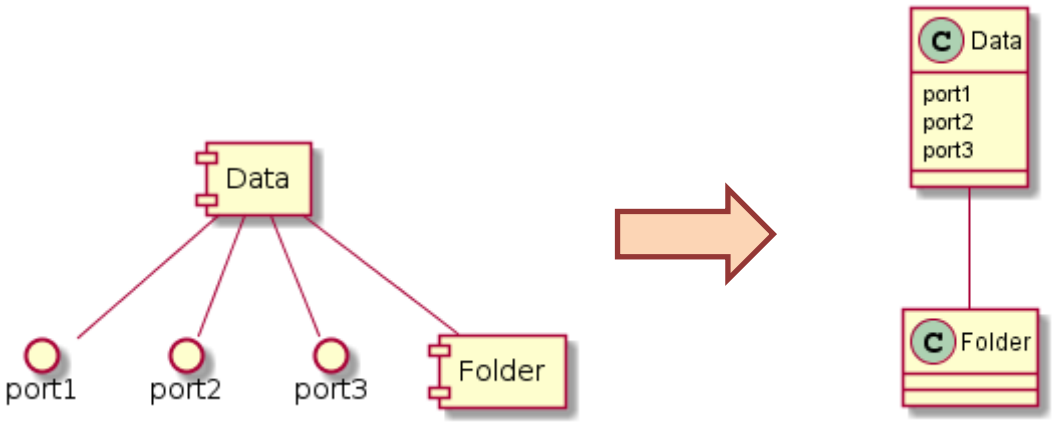


Figure 14: Component diagram conversion to class diagram

We differentiate components from ports by using the letter “C” for component objects and the letter “I” for ports (interface) objects.

2.2.2 Compact class representation

2.2.2.1 Listing interfaces

The first thing one can notice by observing Figure 12 and Figure 13 is that many ports are not connected to anything else than their own ‘parent component’. They are just lined up around their parent component taking space unnecessarily. A first transformation idea was to list unconnected ports under their parent component to free some space while keeping the original information intact. Ports having one or more connections other than their parent component’s connection are detached and have their own “I” object created.

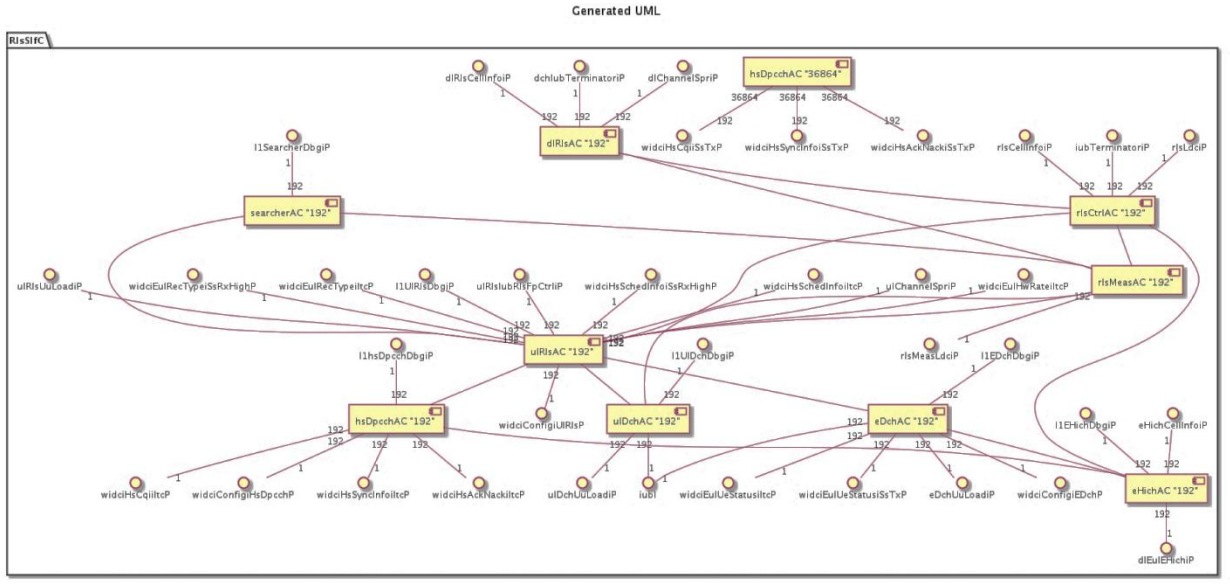


Figure 15: Original RLS module

Figure 15 displays one sub-module of the given system called RLS and Figure 16 shows this modification on the module shown on Figure 15.

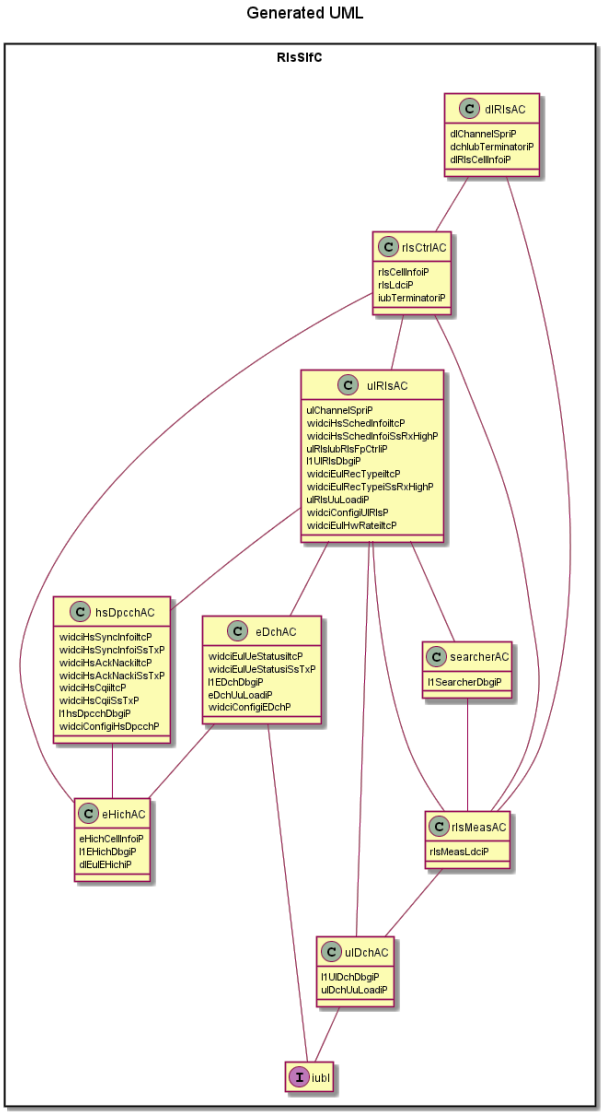


Figure 16: Compact class representation of RLS module (Figure 15)

Drastic improvements are noticeable on Figure 16 which is the most complex module of the whole system. It is easier to see how the components are connected together and the diagram is less cluttered. Figure 17 presents this strategy applied on the full system architecture (Figure 12).

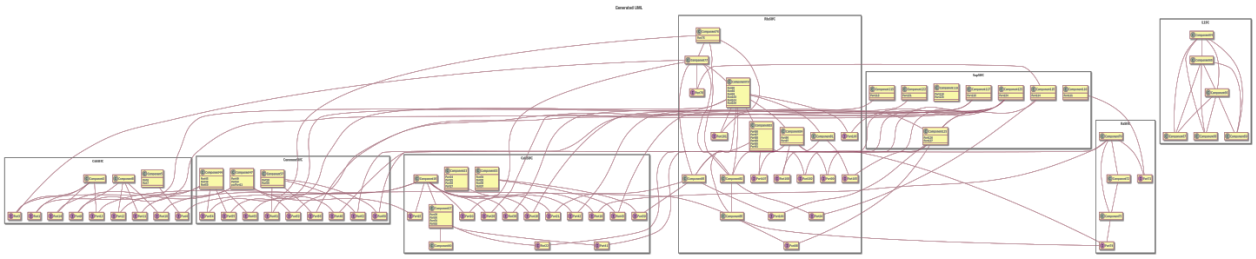


Figure 17: Compact class representation of complete system overview (Figure 12)

However the system overview, although slightly clearer, is still hard to read. There are still too many lines on the view and the ones crossing from one package to another are especially hard to follow. To address this issue, another strategy simplifying the inter-package connections was designed. It is described in the following section.

2.2.2.2 Simplifying inter-package connections

In order to de-clutter further the software architecture overview diagram, a strategy simplifying inter-packages connection was tested. The rules were the following:

- Show one line between two modules (packages) if at least one of their inside components/ports are connected together.
- Connections from one module's element to another module's element are listed in the method field of the respective elements. For example: if element1 in module1 is connected to element2 in module2, "module2.element2()" will be shown in the method list of element1 in module1 and vice versa. (Figure 18)

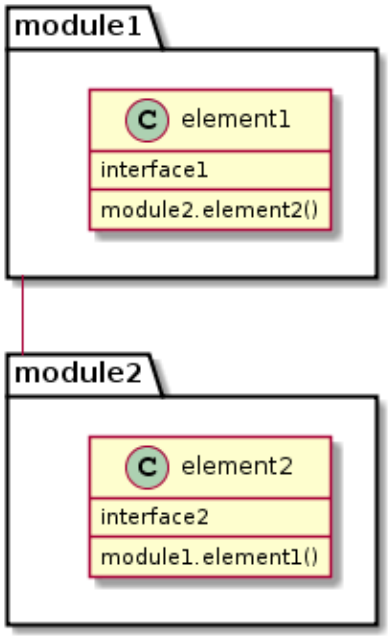


Figure 18: Connection definition inside modules

Figure 19 shows the result of this strategy on the full system architecture (L1SC).

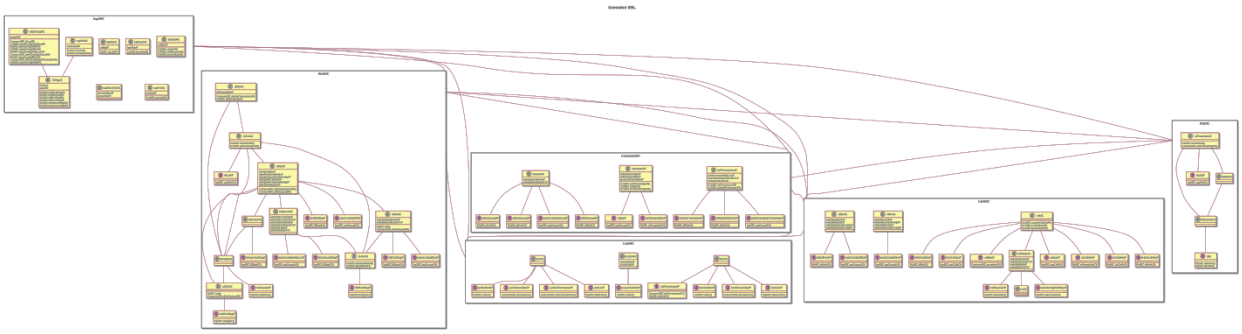


Figure 19: Listing inter-package connections (L1SC)

According to the aesthetic criteria analysis for the system overview (L1SC), this approach reduces by an average of 90% the number of edge crosses and by 78% the number of edge bends. The system overview is clearly less cluttered. It is possible to follow inter-package connections and to find out how components are linked together by reading their connection list.

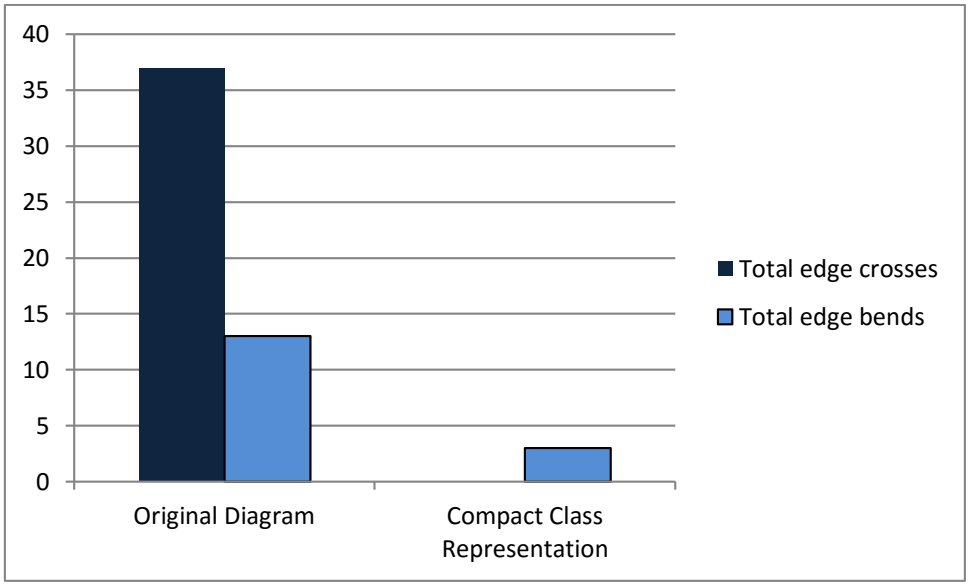


Figure 20: Total number of edge crosses and bends obtained from original diagrams and compact class representation of all system modules

This strategy gave approximately 76% declines for number of bends and removed all crosses for all system modules (Figure 20). These very promising results made this approach the core strategy of the chosen prototype.

2.2.3 Component Ranking

This approach aims to show only relevant information to the viewer by hiding “unnecessary” elements. With the help of an experience Ericsson engineer, elements were ranked according to their importance for understanding the system with a value from 1 to 10. The higher the value, the higher the importance of this element is. The prototype was then extended to take in an abstraction parameter (value: 1 to 10). The abstraction parameter and the importance of the elements displayed are closely related. A high abstraction parameter will only display elements of high importance. Elements having a ranking score below the given parameter are removed from the diagram. For example if level 6 is chosen, only components with ranking level 6 and over will be displayed. Table 6 shows main level descriptions of the strategy.

Level	Description
1	Full detail
4	Technician level(no debug components)
8	Outside engineer (no measurement components)
10	Core components

Table 6: Component ranking level descriptions

Figure 21 shows the full software overview using a technician abstraction level (level 4) where all debug components got hidden. The number of elements is decreased by approximately 22% compared to the compact class representation (Figure 19) and by 52% compared to the original component diagram (Figure 12).

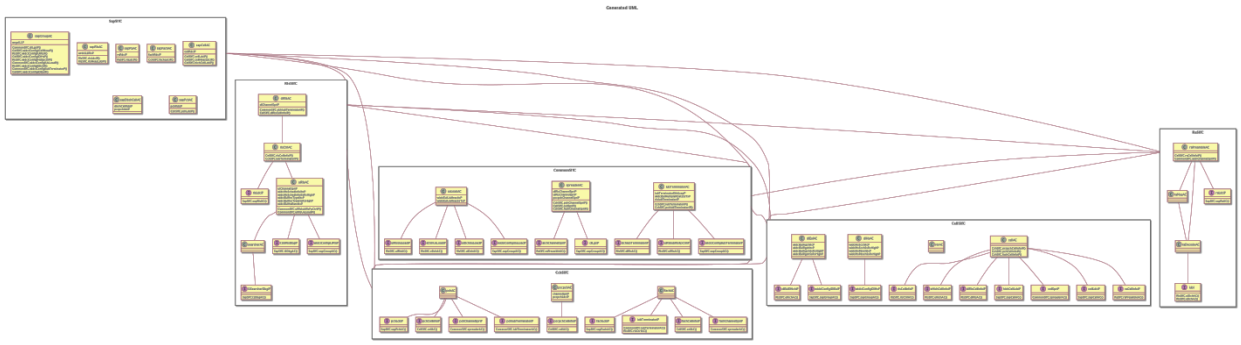


Figure 21: Ranking strategy applied to the full software architecture (Figure 19) (Level: 4)

Figure 22 shows the same diagram at a higher abstraction level (level 8). Using this level, the number of elements reduced by 42% compared to the original compact class diagram (level 1) (Figure 19) and by 64% compared to the original component diagram (Figure 12).

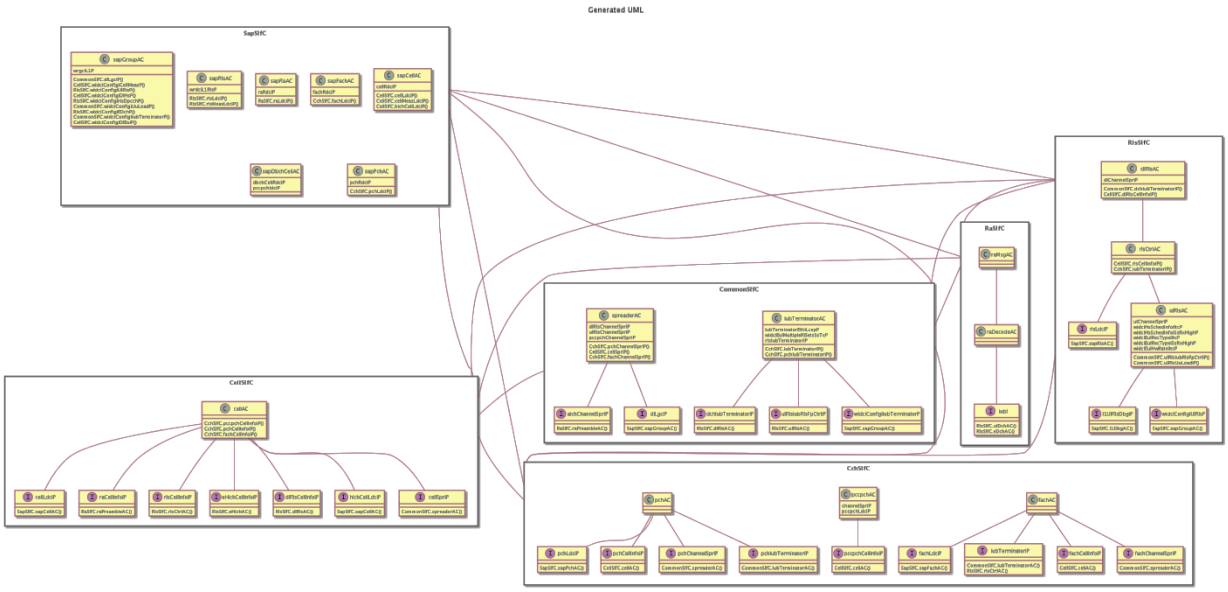


Figure 22: Ranking strategy applied to the full software architecture (Figure 18) (Level: 8)

Finally, Figure 23 shows the highest level of abstraction (level 10) where only core components are displayed. This reduces the number of elements on the view by 44% compared to the most detailed compact class diagram (level 1) and by 66% compared the original component diagram (Figure 12).

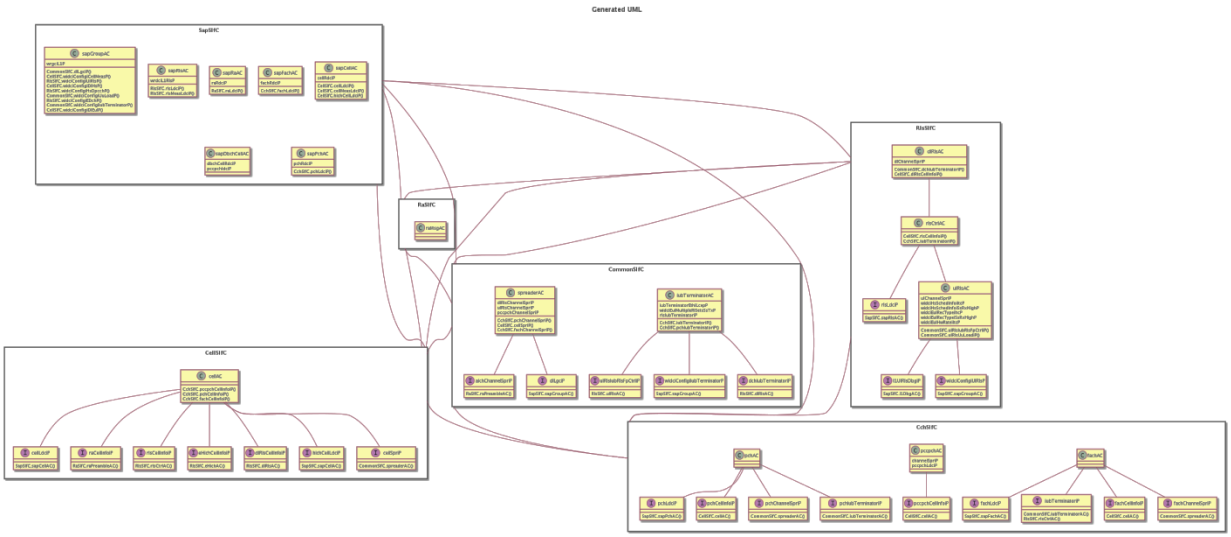


Figure 23: Ranking strategy applied to the full software architecture (Figure 19) (Level: 10)

Aesthetic criteria	Original	Level1 (compact class)	Level 4	Level 8	Level 10
Edge crosses count	>200	21	14	12	11
Edge bends count	>50	11	10	9	10
Number of elements	121	74	58	43	41

Table 7: Aesthetic criteria evaluation for the full software architecture view using component ranking

The results of the aesthetic criteria analysis are presented in Table 7. The biggest aesthetic criteria improvements are achieved when converting the original component diagram to a compact class diagram (89,5% less edge crosses, 22% less edge bends). Higher abstraction levels don't seem to reduce edge crosses and bends drastically compared to abstraction level 1 (original compact class diagram). For example, the highest level of abstraction (level 10) reduces edge crosses and edge bends by 48% resp. 10%.
The main strength of this strategy is to reduce the amount of components on the view and show only relevant information to the viewer. For example, level 1 reduces by 38% the number of elements compared to the original diagram whereas the abstraction level 10 reduces it down to 66%.
Selecting the appropriate abstraction level was determined by conducting audience surveys.

2.2.4 Grouping interfaces / components

This strategy is based on the fact that several interfaces and components have similar functionalities and therefore can be merged (or grouped) together. With the help of an experienced engineer in the team, item groups were identified and the logic was saved in a configuration file. To perform the experiments, the prototype was extended with a "grouping" option (yes/no).
Figure 24 shows the results of this strategy applied on the full software architecture. The grouping strategy reduces the number of elements in the view as it collects several elements under the same group name. However, while the number of elements decreases, the relations between grouped elements remain same which may cause edge crossings and/or bends on the view. For example, Figure 25(a) shows an example module whereas Figure 25(b) displays the output of the grouping strategy applied on Figure 25(a). Port1 and Port4 are grouped under the element Group1 but this leads to an edge cross in Figure 25(b).

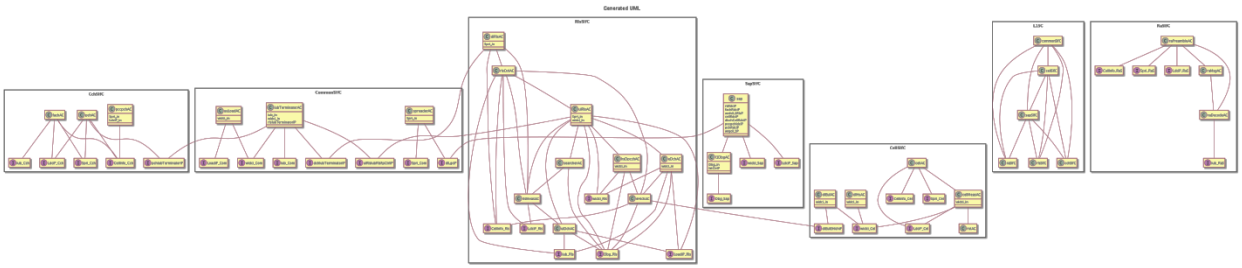
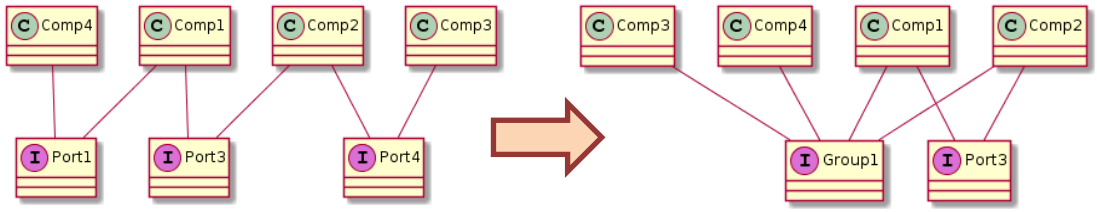


Figure 24: Grouped representation of the full software architecture (org. Figure 19)



(a) (b)

Figure 25: Changes when grouping strategy applied on an example module

Aesthetic criteria	Original	Compact class	Grouping
Edge crosses count	>200	21	67
Edge bends count	>50	11	14
Number of elements	121	74	60

Table 8: Aesthetic criteria evaluation for the full software architecture with the grouping interfaces/components strategy

Although the grouping strategy shows improvements compared to the original diagram, Table 8 shows that it performs worse than the compact class representation strategy: more edge crosses and bends. However, the grouping strategy results in having fewer elements on the view (18% less than the compact class diagram) as well as elements with shorter/custom names which might improve readability for the audience. Surveys have been conducted to verify this intuition.

2.2.5 Splitting views

Taking advantage of the compact class representation strategy, this approach's idea was to split the full system overview into several diagrams preserving the same level of information but reducing the amount of information to be processed at a time. It is based on a deductive tactic: first an overview of the modules as black boxes is generated then one view per module is created showing its internal structure. The same set of rules as the compact class diagram strategy was followed.

Figure 26 displays the software architecture overview where all connections between sub-modules are presented without showing each sub-modules' inner structure and Figure 27 is an output of an example sub-view of the full system selected with a rectangle in Figure 26.

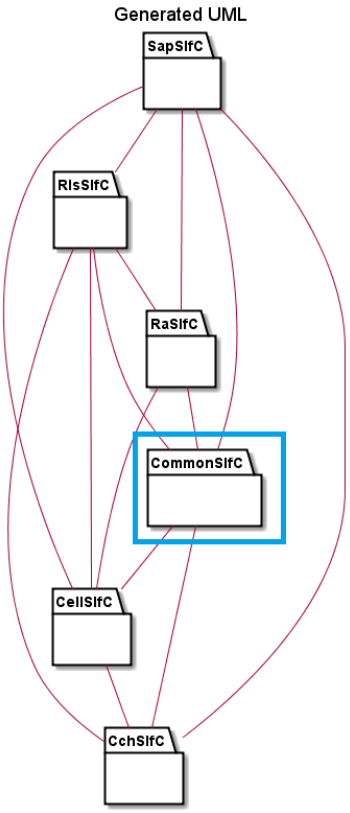


Figure 26: System architecture overview

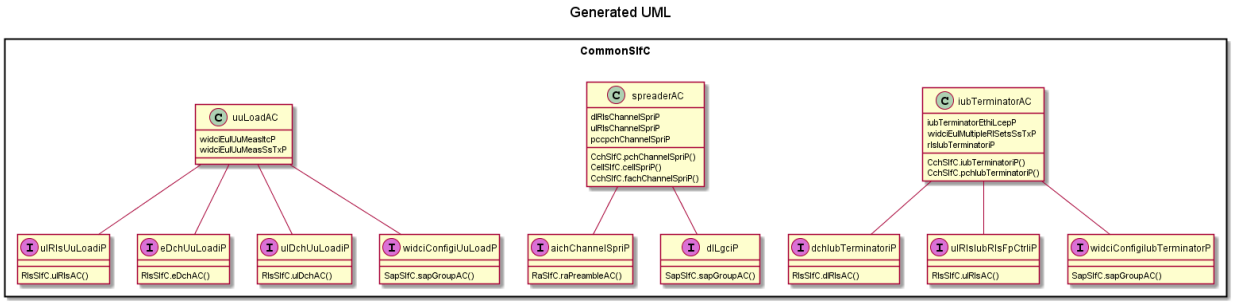


Figure 27: A package view obtained from splitting (selected with a rectangle in Figure 26)

Aesthetic criteria evaluation results of this approach are the same with the compact class representation strategy ones since it performs the same rules on the given module and produces the same diagram structure by splitting the diagram into an overview and sub-modules. Therefore its effect on readability was evaluated using surveys.

In addition, there is also a splitting function in PlantUML tool; however it only divides large diagrams into pages not sub-modules as we did in our splitting strategy.

2.2.6 Describing elements

As element names might not be easily understood by computer scientists who are not familiar with the system, this approach is meant to make elements more self-explanatory. Elements' descriptions were listed in a configuration file and shown either instead of the element name (e.g. package name) or in the upper corner of the component object.

Figure 28 shows the description strategy applied to the RLS module (Figure 16).

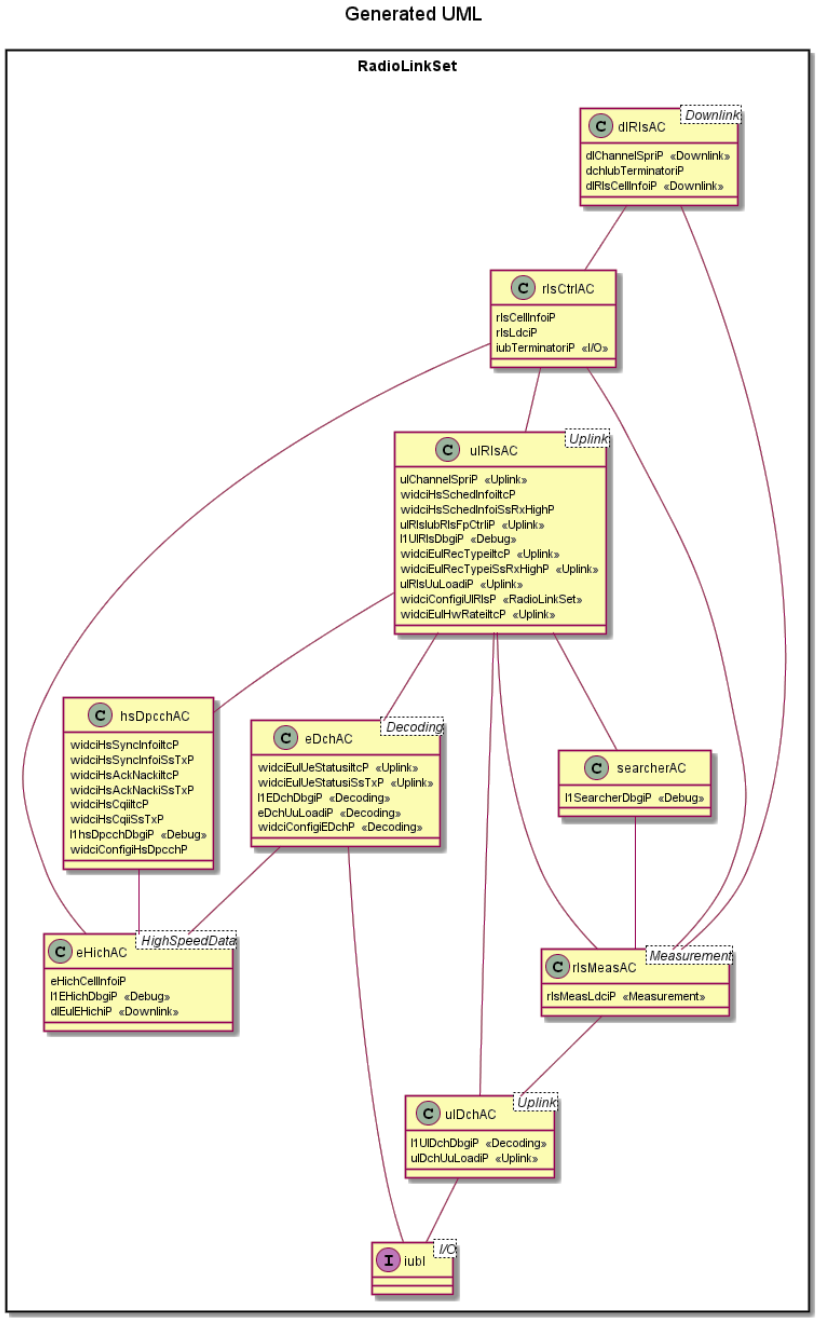


Figure 28: RLS package view (Figure 16) with added descriptions

For example, the package name has been changed from RlsSIfC to RadioLinkSet and some components such as eDchAC have a descriptive label (e.g. “Decoding”) which is more informative (Figure 28).

Since the aesthetic criteria results are similar to the compact class representation strategy, the effects of this strategy on readability were evaluated using surveys especially targeted on the “computer scientists having no prior system knowledge” audience subgroup.

3. Audience tests

Following the methodology, each subject was randomly given a task and a module representation to work with. We used three different representations for our tests. First one is textual representation which stands for YAML based text file where the software architecture is described. Second representation is original component diagram which is generated from PlantUML by using these YAML based files and the last one is compact class diagram where we applied our compact class strategy to the original component diagrams.
Each task was timed and the accuracy of the answer was noted. We present below the results obtained for the two target groups.
Note that we considered only sub-modules of the system to perform audience tests efficiently in a short time period. It was also due to the fact that original component diagram of the full system architecture was too cluttered and impossible to read for audiences to answer test questions.

3.1 Experienced engineers

The surveys were conducted with engineers working at Ericsson (T1). The results were the following:

Task 1: Finding the number of elements

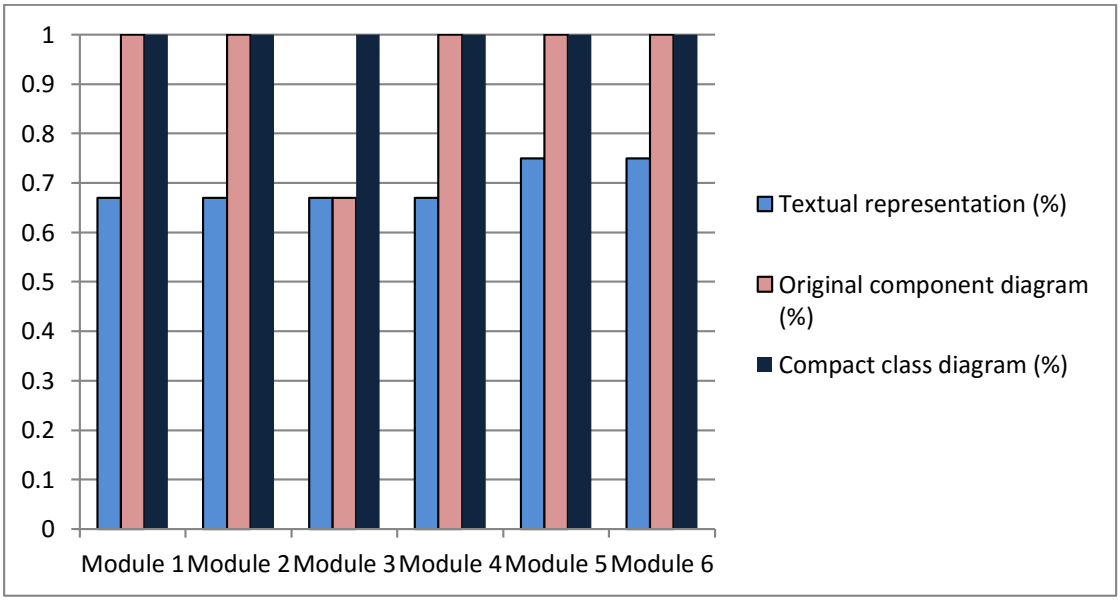


Figure 29: Accuracy chart for different module representations (Task 1) (T1)

According to Figure 29, the correctness of the answers clearly improved when using diagrams rather than textual representation (the accuracy worsen by an average of 31% when using textual models). Additionally component diagrams and compact class diagrams seem to both give almost perfect accuracy in answers.

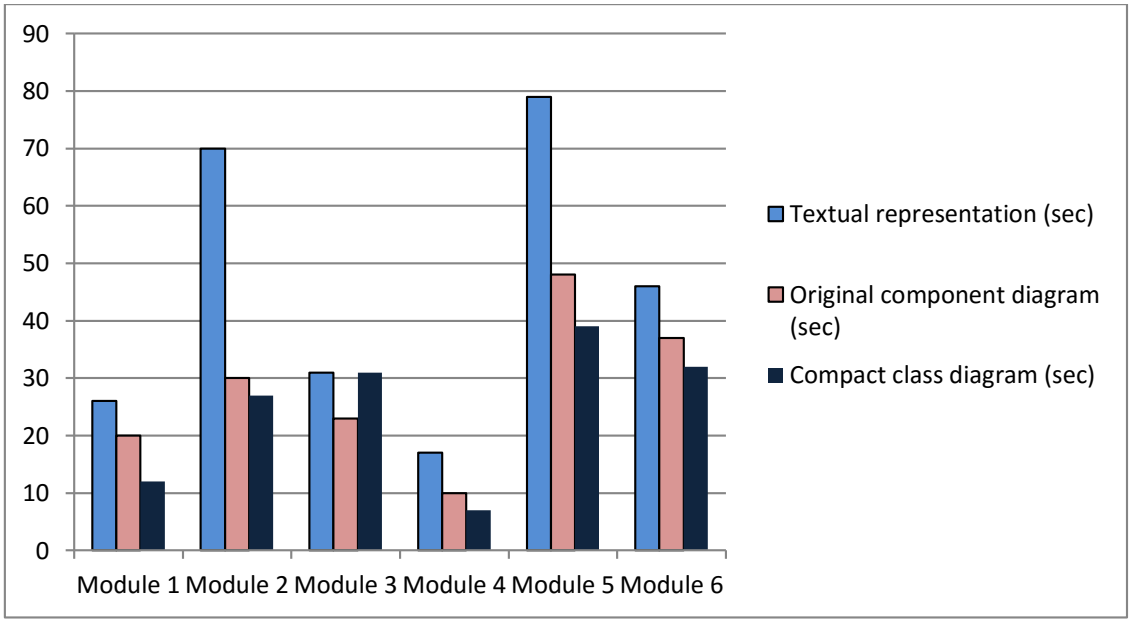


Figure 30: Time chart for different module representations (Task 1) (T1)

As you can see from Figure 30, textual representation gave the worst time performances (average of 41% worse than the diagrams). We can also observe slightly faster answers (13%) when using compact class diagrams compared to using original component diagrams.

Task 2: Finding the most connected component(s)

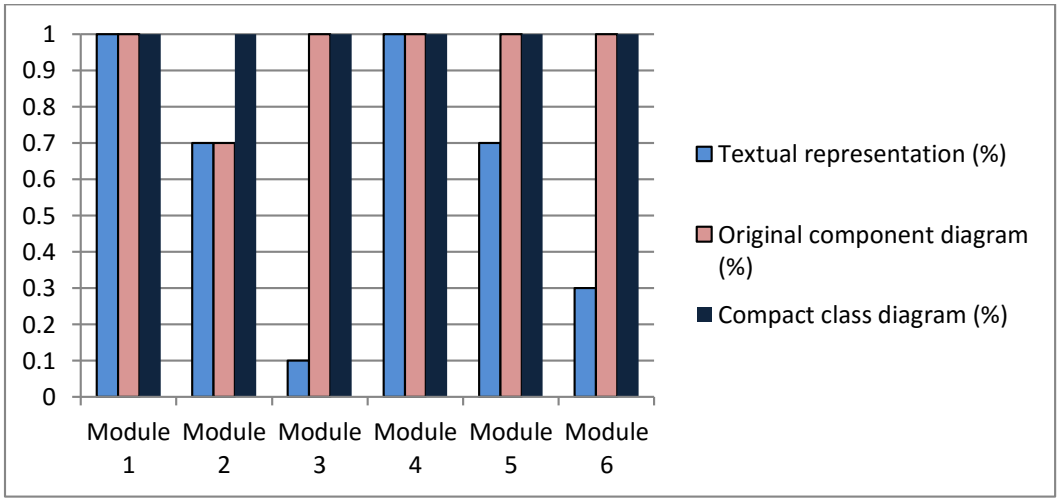


Figure 31: Accuracy chart for different module representations (Task 2) (T1)

Similarly to the first task results, as shown in Figure 31, both original component diagrams and compact class diagrams gave almost perfect answer accuracy which is not the case when using textual representations (only an average 63% of in accuracy).

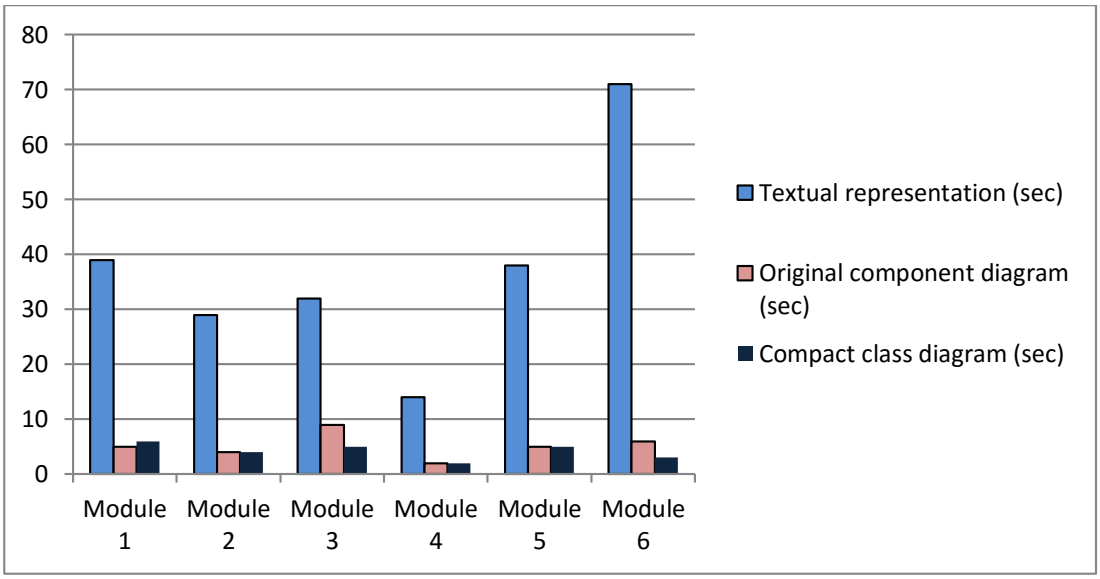


Figure 32: Time chart for different module representations (Task 2) (T1)

Finally, with respect to Figure 32, better performances were also observed during this task when using graphical representations rather than textual representation (in average 88% better).

3.2 Other computer scientists

Similar tests to the ones conducted with Ericsson engineers were used for the “computer scientists having no prior knowledge of the system” group (T2).
The results were the following:

Task 1: Finding the number of elements

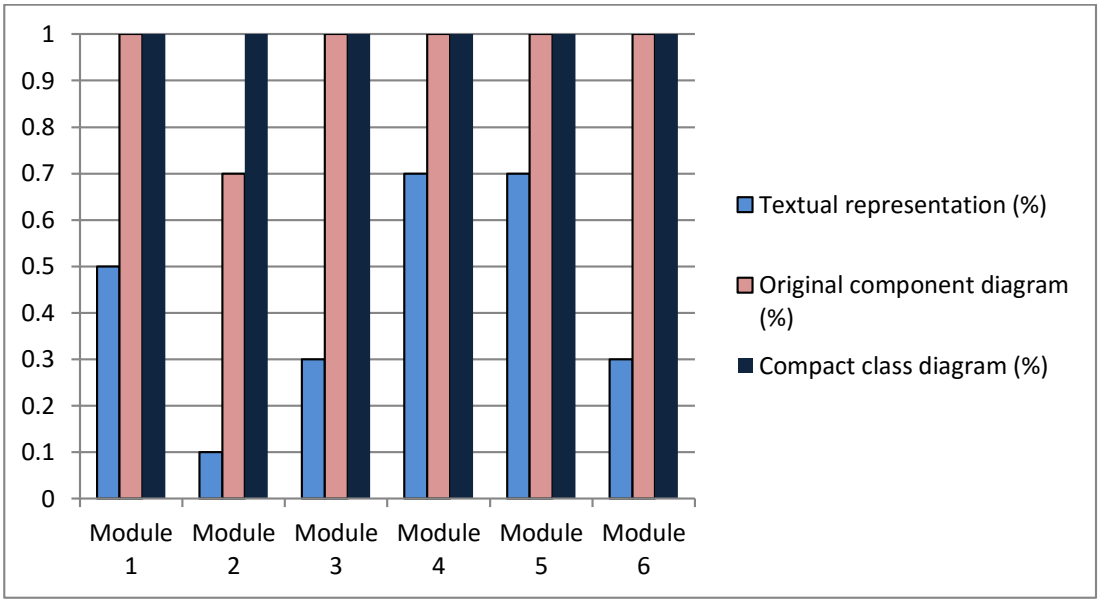


Figure 33: Accuracy chart for different module representations (Task 1) (T2)

As you can see in Figure 33, using textual representation only gives an average of 43% accuracy whereas using either graphical representation gives almost perfect accuracy as well for this target group.

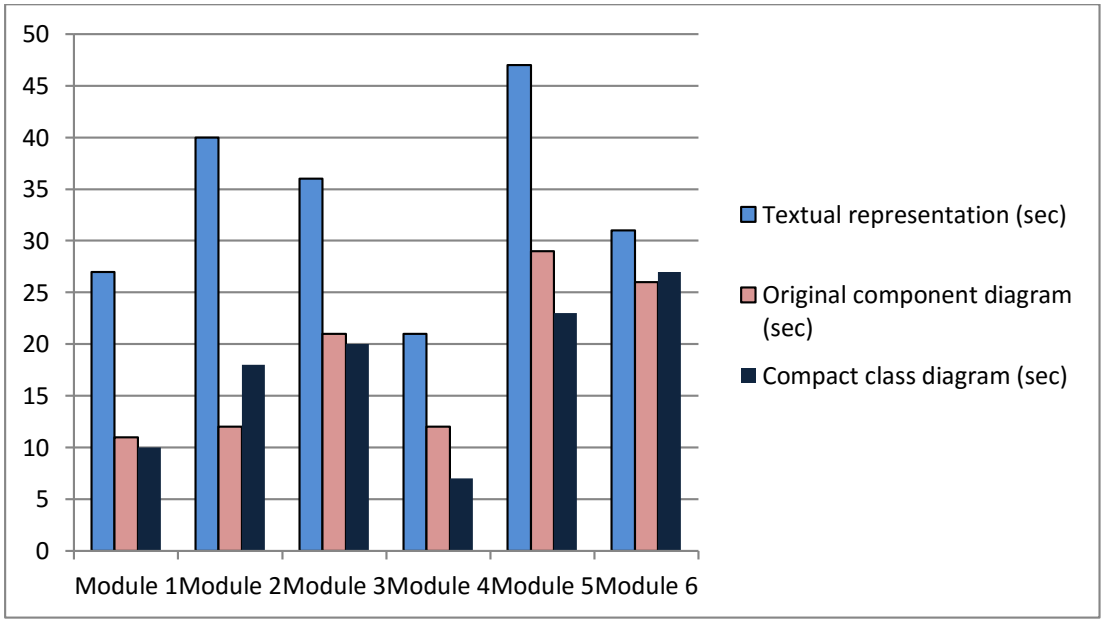


Figure 34: Time chart for different module representations (Task 1) (T2)

If we sum up all the values of Figure 34, solving this task for all modules using textual representation would have taken in average 202 seconds against 113 seconds using the original component diagrams and 106 seconds using the compact class diagram. In summary, using

graphical diagrams gave 45% better results than textual representation. However, there is no significant difference in performances when using the compact class diagrams compared to using the original component diagrams.

Task 2: Finding the most connected component(s)

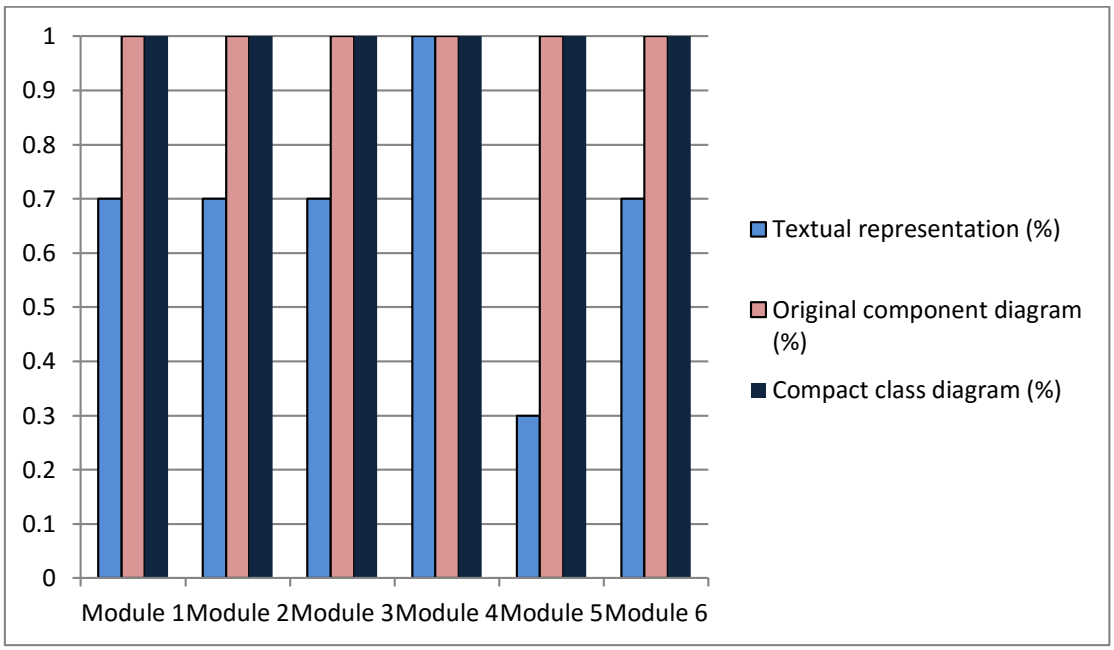


Figure 35: Accuracy chart for different module representations (Task 2) (T2)

Figure 35 shows that textual representation yields to an average of only 68% of answer correctness against perfect accuracy for both graphical representations.

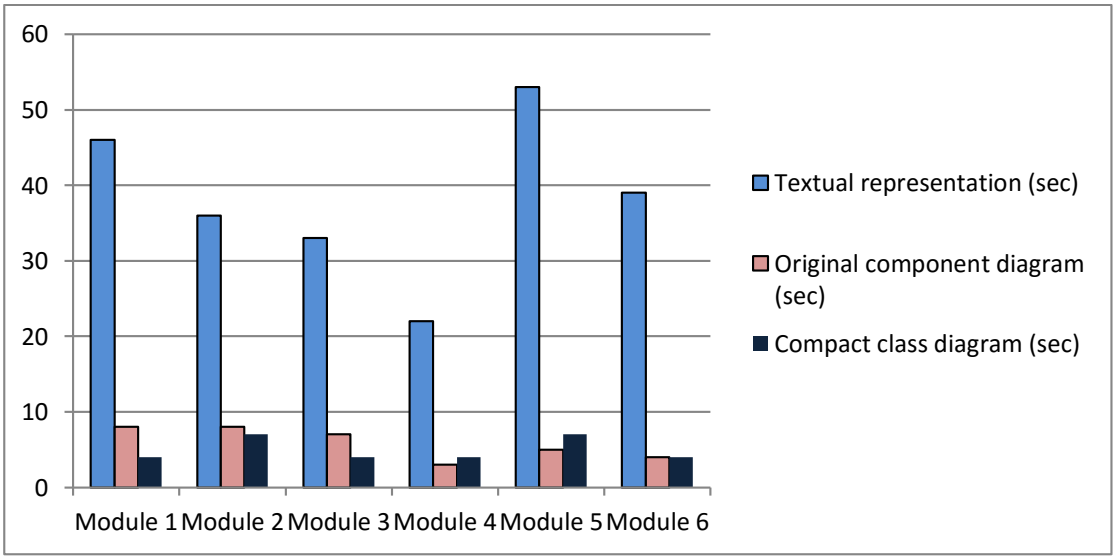


Figure 36: Time chart for different module representations (Task 2) (T2)

In Figure 36, you can see that using diagrams improves drastically the performances (86% better in average). Additionally, one can conclude that there is no significant difference when using the compact class diagrams compared to the original diagrams (9% better in average).

4. Surveys

Both Ericsson engineers (T1) and computer scientists having no prior knowledge of the system (T2) were asked to rank the system representations according to their preferences. We showed them outputs of all modules with all given representations and gave them time to clarify their preferences.

Besides pointing out which representation they would prefer to work with, experienced engineers were also asked which representation they would show to a new engineer working in the team and corresponding to the profile of a target group T2 person. Survey questions are provided in Appendix B as well as an example of textual representation is given in Appendix A of this report. Additionally, all subjects were asked to comment on why they preferred or did not prefer a representation. The results are shown in Table 9 and 10:

Representation	Comments by T1
Textual representation	- Chosen by 0% - Confusing to read
Original component representation	- Chosen by 0% - Too cluttered even for simple module
Compact class representation – level 1 (detailed)	- Chosen by 90% for module representation with the condition of interface (port) number listed inside a component is less than 10 and the total number of components is less than 15. Otherwise chosen next abstraction level to reach those criteria. - Chosen by 0% for complete software architecture overview.
Compact class representation – level 2-9 (intermediate)	- Chosen by 90% for module representation if the above conditions are not fulfilled. - Chosen by 0% for complete software architecture overview.
Compact class representation – level 10 (overview)	- Chosen by 90% for module representation if the above conditions are not fulfilled.
Grouped elements representation	- Chosen by 0% - Lack of naming standards to make it sustainable in the future - Hide valuable information

Add description	- Chosen by 10%. - 90% said that it is not necessary for experienced engineers.
Split view option	- 90% said that it is useful. It is good to have when they want a detailed view (abstraction level 1) but the overview is too crowded.

Table 9: Representation for 'experienced engineers'

Representation	Comments by T1	Comments by T2
Textual representation	- Chosen by 0% - Confusing to read	- Chosen by 0% - Impossible to understand
Original component representation	- Chosen by 0% - Too cluttered even for simple module.	- Chosen by 0% - Too cluttered.
Compact class representation – level 1 (detailed)	- Chosen by 0% - Too complex for beginners.	- Chosen by 0%
Compact class representation – level 2-9 (intermediate)	- Chosen by 0% - Diagrams still need simplification	- Chosen by 0%
Compact class representation – level 10 (overview)	- Chosen by 90%. - Only essential information shown.	- Chosen by 90% - Clearer
Grouped elements representation	- Chosen by 10% - Hide complexity by grouping related elements.	- Chosen by 10% - Group names can be more explicit.
Add description	- 90% said that it is good to have for beginners.	- 75% said that it makes the diagram easier to understand. - 16% said that the name of the components should be replaced by a more explicit name instead. - 9% said that it is not necessary.

Split view option	- 90% said that it is useful. It is good to have when they want a detailed view (abstraction level 1) but the overview is too crowded.	- 100% said that it is good to have the possibility to break the overview into smaller chunks.

Table 10: Representation for 'beginners' (engineers having no prior knowledge of the system)

From the results presented in Table 9 and 10, one can conclude that compact class diagram representation was the favorite one. The Ericsson engineers group preferred a more detailed abstraction level whereas the other subject group preferred the most abstracted version. It was also clear from the interviews that textual representation was the least appreciated representation. Grouping was chosen by some subjects but several Ericsson engineers made a very relevant comment when pointing out that it was not practical in reality as they don't have name standards in place to build a sustainable grouping logic.

5. Validation

5.1 Construct validity

The same transformations have been applied to all modules and full system overview to ensure to have enough data to draw relevant conclusions. The criteria used to gather quantitative data are the aesthetic rules which are relevant to evaluate class diagrams readability, identified by Helen C Purchase et al. [29]. Data was collected on both original diagrams and transformations.
Subject groups were both composed of approximately 10 persons with various backgrounds, nationalities, age and gender. This diversity brings stability in the statistical results.
Semi-structured interviews were conducted with a special care about randomness of the task/data the subjects had to solve/use. This was to prevent any previous learning from the model making the subject suddenly more efficient on a task not because of the representation but because he has seen the model before and remembers its structure.
Questions (content and order) were prepared in advance so that each subject had the same interview experience.

5.2 Threats to validity

Two kinds of threats to validity were identified.

Internal threats

It is not easy to identify tasks estimating the overview one has of a software representation. If time allowed it, it would have been interesting to add other tasks to the interviews such as “finding a specific element”.
The transformations were performed on the models we had at our disposal which means that the results might not be the same for other software models.

External threats

During the interviews, we were always present in the room to give the questions and time the tasks. This could have stressed the subjects in trying to solve the tasks as fast as possible impacting therefore both answer correctness and performances.

6. Results summary

First, it was clear from the evaluation results that textual representation was the least effective one as it yielded to both worse task answer correctness and performances.

Additionally, since the aesthetic criteria evaluation gave better results (e.g. less edge crosses and bends) for the compact class representation, we could conclude that layout algorithms itself had a lesser effect on readability than the compact class diagram representation. However combination of layout algorithm study and abstraction strategy might improve readability better.

Then, from the interview comments, it appeared that grouping strategy is not sustainable for the future since the DURA department doesn't follow any common rule to point element's functionality clearly in their names. However the grouping strategy is based on element names which are supposed to give information about their functionalities by using some abbreviations.

The conclusion of this research is that the compact class diagram representation is the one improving readability the most among those studied. Interestingly, interviews demonstrated us that both target groups had different abstraction level choices. On one hand, Ericsson engineer's favorite representation was the most detailed abstraction level as long as the view was not too cluttered otherwise they would prefer using a higher abstraction level to reduce the number of elements on the view. On the other hand, the other target group (computer scientists without prior knowledge of the system) chose the highest abstraction level showing only the software's core elements (level 10).

Finally, splitting the full software view into overview and sub-views as well as the descriptive labels options were appreciated by most subjects especially by the computer scientists of the second target group.

7. Prototype

This section discusses the conclusions leading to the prototype design and parameters. Based on time and requirements, two design decisions were made.

First, a decision has been made to prioritize the abstraction strategies implementation over implementing a force-directed layout algorithm in the solution. This is based on schedule concerns since both couldn't be achieved in the thesis time we had and as abstraction strategies improve diagram readability more than force-directed layout algorithms (according to aesthetic criteria evaluations), what to prioritize became obvious.

Then, another design decision was made to implement all abstraction strategies in the prototype. The reason for this is that the evaluation showed that different abstraction parameters suited different audiences and as the prototype will be used by other teams as well, it had to be flexible enough to suit different audience profiles.

These decisions led to the prototype design presented by Figure 37. The process is the following: first a raw file is processed by the PlantUML library, then the result is run through the abstractions strategies chosen by the user and finally the output diagrams are created.

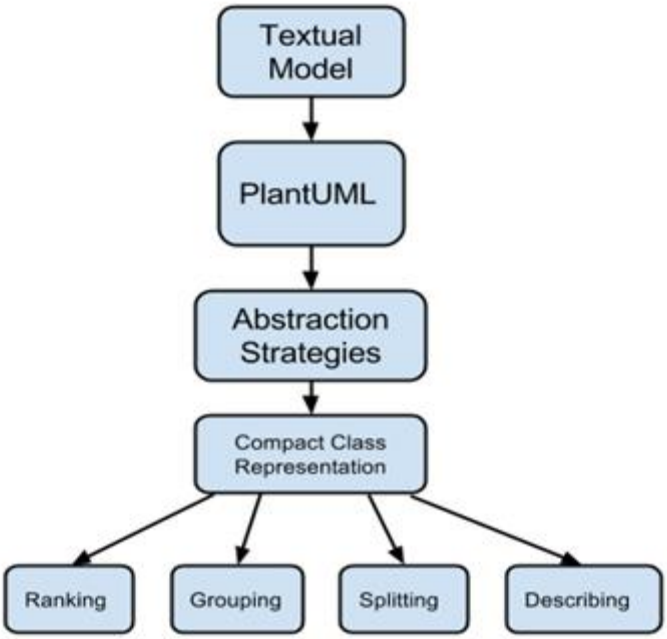


Figure 37: Prototype design

The different abstraction strategies are activated by parameters. Table 11 shows the parameters built-in the prototype and which abstraction strategy they activate. The prototype was developed using Java and compiled into a jar file.

Option	Description	Value
-group	Activate the grouping abstraction strategy	-
-filter=<value>	Input parameter for component ranking filter. Default value = 1 (show all)	Integer : 1 to 10
-split	Activate the split view functionality	-
-describe	Activate the description functionality	-

Table 11: Prototype parameters

Data about component ranking, grouping logic and component description is stored into configuration files making it easy for the users to modify them without having to rebuild the prototype. Relevant source code is provided in Appendix D as well as configuration files are given in Appendix C of this report.

VI. Further research and projects

This section is meant to give further research and projects ideas to continue the work started during this thesis.

1. Layout algorithm

For this thesis work, the experiment results suggest that switching the prototype layout algorithm from a layered layout algorithm to a force-directed layout algorithm might improve readability even further. Due to limited amount of time, it was not possible to modify PlantUML source code to use Neato instead of Dot. Another thesis work could be to extend the prototype with force-directed layout algorithm capabilities.

2. Machine-learning based tool

In the future, machine-learning based tools might be created to predict autonomously and automatically the right software architecture representation for a specific targeted audience. An interesting project could be to research the possibilities of adapting a machine learning prototype such as the one designed by M. H. Osman [25] et al. for component diagrams generation.

VII. Concluding discussion

This thesis work has confirmed that generating readable diagrams for complex software systems is feasible for the studied software by using some abstraction strategies regarding targeted audiences as well as a set of aesthetic rules.

Analyzing the targeted audience has proven to be crucial when working on diagram readability. By applying audience analysis techniques, we were able to identify audience subgroups having each different characteristics and needs for the software visualization.

Using aesthetic criteria, audience tests and surveys, we were able to answer the research questions during the thesis work as follows:

RQ1: Is visualization better than textual description in some cases?
Survey results have shown that diagrams were definitely better than textual models to give an overview of the software architecture in the studied case.

RQ2: How can we improve readability of the given component diagrams?
Experiment showed that applying different layout algorithms and abstraction strategies improved readability for the studied software architecture. A force based layout algorithm generated diagrams with better readability than a layered layout algorithm. However when dealing with a complex software like the one of this study, component placement only was not powerful enough to make the full software overview diagram completely readable. There was simply too much on one view and it didn't matter how it was placed. To improve further diagram readability, it appeared crucial to reduce the amount of elements shown on the same view. In order to achieve that, several strategies were designed and evaluated: ranking components, grouping elements and splitting views into sub-views.

RQ3: How to evaluate the quality of the representation in term of readability?
Aesthetic criteria and surveys were used to evaluate readability improvements. They concluded that generated different abstraction level base on component ranking was the one improving readability the most. Interviews conducted with two target groups indicated that different abstraction level were suited for each audience subgroups. Experienced Ericsson engineers preferred a more detailed abstraction level whereas the computer scientists without prior knowledge of the system chose the most abstracted representation containing only core software components.

A prototype was built implementing all abstraction strategies allowing each user to choose their preferred software representations. However we would advise in using the abstraction levels identified with the surveys, if the targeted audience is similar to one of the subgroup of this thesis work. Additionally, a functionality adding descriptive labels on components was built-in to generate more self-explanatory diagrams. This extra functionality was appreciated by most interviewed subjects.

In the future, tools might be able to predict autonomously the best level of abstraction and produce readable diagrams no matter the audience type or the system complexity. Promising research has begun to design a machine-learning algorithm tool for class diagram generation [25]. Some further research activities linked to this thesis work could be to investigate and extend the prototype to use different layouts algorithms and research the possibilities to adapt the machine-learning based prototype to this case.

VIII. References

1. YAML: YAML Ain't Markup Language, http://yaml.org/ , accessed 2015/03/12
2. GraphViz – Graph Visualization Software, http://www.graphviz.org/ , accessed 2015/04/24
3. PlantUML in a nutshell, http://plantuml.sourceforge.net/ , accessed 2015/04/24
4. The Unified Modeling Language, http://www.uml-diagrams.org/, accessed 2015/06/04
5. Tutorials Point, "UML – Standard Diagrams" http://www.tutorialspoint.com/uml/uml_standard_diagrams.htm , accessed 2015/06/04
6. Protsko, L. B., Sorenson, P. G., Tremblay, J. P., and Schaefer, D. A., 1991, *Towards the Automatic Generation of Software Diagrams*, IEEE Transactions on Software Engineering 17, 1 (January), 10–21
7. Holger Eichelberger, 2005, *Aesthetics and Automatic Layout of UML Class Diagrams*, Bayerischen Julius-Maximilians-Universität Würzburg
8. Eiglsperger, M. 2003, *Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach*, Ph.D. thesis, Fakultät für Informations- und Kognitionswissenschaften(Wilhelm-Schickard Institut für Informatik), Tübingen University, Germany
9. Tamassia, R., Di Battista, G., and Batini, C., 1988, *Automatic Graph Drawing and Readability of Diagrams*, IEEE Transactions on Systems, Man and Cybernetics 18, 1 (Jan./ Feb.), 61–79
10. Kozo Sugiyama, Kazuo Misue, 1991, *Visualization of Structural Information: Automatic Drawing of Compound Digraphs*, IEEE Transactions on Systems, Man and Cybernetics Vol.21 No.4
11. Helen C. Purchase, Jo-Anne Allder, and David Carrington, 2001, *User Preferences of Graph Layout Aesthetics: A UML Study*, School of Computer Science and Electrical Engineering The University of Queensland
12. VL/HCC Tutorial 2009 : Automated Diagram Drawing, http://www.eulerdiagrams.com/tutorial/AutomatedDiagramDrawing.html , accessed: 2015/05/07
13. Gephi, The Open Graph Viz Platform, https://gephi.github.io/ , accessed 2015/05/07
14. Tulip, Data Visualization Software, http://tulip.labri.fr/TulipDrupal/ , accessed 2015/05/08
15. Kieler, Welcome to the KIELER Project, www.rtsys.informatik.uni-kiel.de/en/research/kieler , accessed 2015/08/17
16. M. Petre, *Why looking isn't always seeing: Readership skills and graphical programming*, Communications of the ACM, vol. 38, no. 6, pp. 33–44, Jun. 1995.
17. Sugiyama, Kozo; Tagawa, Shōjirō; Toda, Mitsuhiko,1981, *Methods for visual understanding of hierarchical system structures*, IEEE Transactions on Systems, Man, and Cybernetics, SMC-11 (2): 109–125
18. Kamada, Tomihisa; Kawai, Satoru, 1989, *An algorithm for drawing general undirected graphs* , Information Processing Letters (Elsevier) 31 (1): 7–15
19. T. Fruchterman and E. Reingold, 1991, *Graph drawing by force-directed placement*, Softw. – Pract. Exp. 21(11):1129–1164.
20. Ron Davidson and David Harel, 1996, *Drawing graphs nicely using simulated annealing*, ACM Transactions on Graphics, 15(4):301–331.

21. On the Contribution of UML Diagrams to Software System Comprehension, http://www.jot.fm/issues/issue_2004_01/article3/ , accessed 2015/06/15
22. Real Time UML, http://werner.yellowcouch.org/Papers/rtuml/ , accessed 2015/06/15
23. Stuart K. Card, Thomas P. Moran, Allen Newell, 1983, *The Psychology of Human-Computer Interaction*, L. Erlbaum Associates Inc. Hillsdale, NJ, USA
24. Dabo Sun and Kenny Wong, 2006, *On Evaluating the Layout of UML Class Diagrams for Program Comprehension*, Department of Computing Science, University of Alberta, Canada
25. Mohd Hafeez Osman, Michel R.V. Chaudron, Peter van der Putten, Truong Ho-Quang, 2014, *Condensing Reverse Engineered Class Diagrams through Class Name Based Abstraction*, Leiden Inst. of Adv. Comput. Sci., Leiden Univ., Leiden, Netherlands.
26. Umple, http://cruise.site.uottawa.ca/umple/ , accessed 2015-07-28
27. UMLGraph Automated Drawing of UML Diagrams, http://www.umlgraph.org/index.html , accessed 2015-07-28
28. TextUML Toolkit, http://abstratt.github.io/textuml/readme.html , accessed 2015-07-28
29. Helen C. Purchase, David Carrington and Jo-Anne Allder, 2002, *Empirical evaluation of aesthetic-based graph layout* ,School of Computer Science and Electrical Engineering, The University of Queensland, Brisbane, Australia
30. Modeling Languages, UML tools, http://modeling-languages.com/uml-tools/ , accessed 2015/07/16
31. Micheal J. Albers, 2003, *Multidimensional Analysis for Custom Content for Multiple Audiences*, Department of English, University of Memphis
32. Patrick Moore, Chad Fitz ,1993*, Using Gestalt Theory to Teach Document Design and Graphics* , The University of Arkansas at Little Rock
33. Sarah van der Land, *Trust, Lust and Latex*, 2006, Faculty of Arts, Department of Business Communication, Radboud University Nijmegen, The Netherlands
34. IBM Developer Works, UML basics: The component diagram https://www.ibm.com/developerworks/rational/library/dec04/bell/ , accessed 2015/08/14
35. Per Runeson, Martin Höst, 2009, *Guidelines for conducting and reporting case study research in software engineering*, Empirical Software Engineering
36. Helen C. Purchase, 1998, *The Effects of Graph Layout*, The Department of Computer Science and Electrical Engineering, The University of Queensland, Queensland, Australia

VIV. Table of figures

X. List of tables

Appendix A

Textual Representation Example

RlsSC.amz

```
include: /WRAT_L1/swus/RlsCtrlAC/mdl/RlsCtrlAC.amz
include: /WRAT_L1/swus/DlRlsAC/mdl/DlRlsAC.amz
include: /WRAT_L1/swus/UlRlsAC/mdl/UlRlsAC.amz
include: /WRAT_L1/swus/SearcherAC/mdl/SearcherAC.amz
include: /WRAT_L1/swus/UlDchAC/mdl/UlDchAC.amz
include: /WRAT_L1/swus/HsDpcchAC/mdl/HsDpcchAC.amz
include: /WRAT_L1/swus/EDchAC/mdl/EDchAC.amz
include: /WRAT_L1/swus/RlsMeasAC/mdl/RlsMeasAC.amz
include: /WRAT_L1/swus/EHichAC/mdl/EHichAC.amz


--- # YAML: http://en.wikipedia.org/wiki/YAML
structure:
  name: RlsSIfC
  root: false
  ports:
    - {name: iubI, protocol: IubI, conjugate: false}
    - {name: rlsLdciP, protocol: RlsLdci, conjugate: true}
    - {name: rlsCellInfoiP, protocol: CellInfoi, conjugate: false}
    - {name: dlChannelSpriP, protocol: ChannelSpri, conjugate: false}
    - {name: ulChannelSpriP, protocol: ChannelSpri, conjugate: false}
    - {name: iubTerminatoriP, protocol: IubTerminatori, conjugate: false}
    - {name: widciHsSyncInfoiItcP, protocol: WidciHsSyncInfo, conjugate: false}
    - {name: widciHsSyncInfoiSsTxP, protocol: WidciHsSyncInfo, conjugate: false, cardinality:
192}
    - {name: widciHsAckNackiItcP, protocol: WidciHsAckNack, conjugate: false}
    - {name: widciHsAckNackiSsTxP, protocol: WidciHsAckNack, conjugate: false, cardinality:
192}
    - {name: widciHsCqiiItcP, protocol: WidciHsCqi, conjugate: false}
    - {name: widciHsCqiiSsTxP, protocol: WidciHsCqi, conjugate: false, cardinality: 192}
    - {name: dchIubTerminatoriP, protocol: IubTerminatori, conjugate: false}
    - {name: widciHsSchedInfoiItcP, protocol: WidciHsSchedInfo, conjugate: true}
    - {name: widciHsSchedInfoiSsRxHighP, protocol: WidciHsSchedInfo, conjugate: true}
    - {name: rlsMeasLdciP, protocol: RlsLdci, conjugate: true}
    - {name: dlRlsCellInfoiP, protocol: CellInfoi, conjugate: false}
    - {name: eHichCellInfoiP, protocol: CellInfoi, conjugate: false}
    - {name: ulRlsIubRlsFpCtrliP, protocol: IubRlsFpCtrli, conjugate: false}
    - {name: widciEulUeStatusiItcP, protocol: WidciEulUeStatus, conjugate: false}
    - {name: widciEulUeStatusiSsTxP, protocol: WidciEulUeStatus, conjugate: false}
    - {name: l1SearcherDbgiP, protocol: L1Dbgi, conjugate: true}
    - {name: l1UlDchDbgiP, protocol: L1Dbgi, conjugate: true}
    - {name: l1UlRlsDbgiP, protocol: L1Dbgi, conjugate: true}
    - {name: l1EDchDbgiP, protocol: L1Dbgi, conjugate: true}
    - {name: l1EHichDbgiP, protocol: L1Dbgi, conjugate: true}
    - {name: l1hsDpcchDbgiP, protocol: L1Dbgi, conjugate: true}
    - {name: widciEulRecTypeiItcP, protocol: WidciEulRecType, conjugate: true}
```

```
    - {name: widciEulRecTypeiSsRxHighP, protocol: WidciEulRecType, conjugate: true}
    - {name: ulRlsUuLoadiP, protocol: UlRlsUuLoadi, conjugate: false}
    - {name: eDchUuLoadiP, protocol: EDchUuLoadi, conjugate: false}
    - {name: ulDchUuLoadiP, protocol: UlDchUuLoadi, conjugate: false}
    - {name: widciConfigiUlRlsP, protocol: WidciConfigi, conjugate: true}
    - {name: widciConfigiHsDpcchP, protocol: WidciConfigi, conjugate: true}
    - {name: widciConfigiEDchP, protocol: WidciConfigi, conjugate: true}
    - {name: dlEulEHichiP, protocol: EHichi, conjugate: false}
    - {name: widciEulHwRateiItcP, protocol: WidciEulHwRate, conjugate: true}
  instances:
    - {class: SearcherAC, name: searcherAC, cardinality: 192}
    - {class: DlRlsAC, name: dlRlsAC, cardinality: 192}
    - {class: RlsCtrlAC, name: rlsCtrlAC, cardinality: 192}
    - {class: UlRlsAC, name: ulRlsAC, cardinality: 192}
    - {class: HsDpcchAC, name: hsDpcchAC, cardinality: 192}
    - {class: EDchAC, name: eDchAC, cardinality: 192}
    - {class: UlDchAC, name: ulDchAC, cardinality: 192}
    - {class: RlsMeasAC, name: rlsMeasAC, cardinality: 192}
    - {class: EHichAC, name: eHichAC, cardinality: 192}
  connections:
    - {port1: dlChannelSpriP, port2: dlRlsAC.channelSpriP, router: true}
    - {port1: rlsCellInfoiP, port2: rlsCtrlAC.rlsCellInfoiP, router: true}
    - {port1: rlsLdciP, port2: rlsCtrlAC.rlsLdciP, router: true}
    - {port1: dlRlsAC.ctrlDlRlsiP, port2: rlsCtrlAC.ctrlDlRlsiP}
    - {port1: rlsCtrlAC.ctrlUlRlsiP, port2: ulRlsAC.ctrlUlRlsiP}
    - {port1: ulChannelSpriP, port2: ulRlsAC.channelSpriP, router: true}
    - {port1: searcherAC.searcheriP, port2: ulRlsAC.searcheriP}
    - {port1: searcherAC.searcherMeasiP, port2: rlsMeasAC.searcherMeasiP}
    - {port1: iubTerminatoriP, port2: rlsCtrlAC.iubTerminatoriP, router: true}
    - {port1: ulRlsAC.hsDpcchiP, port2: hsDpcchAC.hsDpcchiP}
    - {port1: ulRlsAC.eDchiP, port2: eDchAC.eDchiP}
    - {port1: ulRlsAC.ulDchiP, port2: ulDchAC.ulDchiP}
    - {port1: ulDchAC.ulDchUlRlsiP, port2: ulRlsAC.ulDchUlRlsiP}
    - {port1: hsDpcchAC.widciHsSyncInfoiItcP, port2: widciHsSyncInfoiItcP, router: true}
    - {port1: hsDpcchAC.widciHsSyncInfoiSsTxP, port2: widciHsSyncInfoiSsTxP, router: true}
    - {port1: hsDpcchAC.widciHsAckNackiItcP, port2: widciHsAckNackiItcP, router: true}
    - {port1: hsDpcchAC.widciHsAckNackiSsTxP, port2: widciHsAckNackiSsTxP, router: true}
    - {port1: hsDpcchAC.widciHsCqiiItcP, port2: widciHsCqiiItcP, router: true}
    - {port1: hsDpcchAC.widciHsCqiiSsTxP, port2: widciHsCqiiSsTxP, router: true}
    - {port1: dchIubTerminatoriP, port2: dlRlsAC.dchIubTerminatoriP, router: true}
    - {port1: rlsCtrlAC.ctrlRlsMeasiP, port2: rlsMeasAC.ctrlRlsMeasiP}
    - {port1: dlRlsAC.dlRlsMeasiP, port2: rlsMeasAC.dlRlsMeasiP}
    - {port1: rlsMeasAC.ulRlsMeasiP, port2: ulRlsAC.ulRlsMeasiP}
    - {port1: rlsMeasAC.ulDchMeasiP, port2: ulDchAC.ulDchMeasiP}
    - {port1: ulDchAC.iubiP, port2: iubI, router: true}
    - {port1: widciHsSchedInfoiItcP, port2: ulRlsAC.widciHsSchedInfoiItcP, router: true}
    - {port1: widciHsSchedInfoiSsRxHighP, port2: ulRlsAC.widciHsSchedInfoiSsRxHighP,
router: true}
    - {port1: rlsMeasAC.rlsLdciP, port2: rlsMeasLdciP, router: true}
    - {port1: dlRlsCellInfoiP, port2: dlRlsAC.cellInfoiP, router: true}
    - {port1: eHichCellInfoiP, port2: eHichAC.cellInfoiP, router: true}
```

```
    - {port1: ulRlsIubRlsFpCtrliP, port2: ulRlsAC.ulRlsIubRlsFpCtrliP, router: true}
    - {port1: eDchAC.widciEulUeStatusiItcP, port2: widciEulUeStatusiItcP, router: true}
    - {port1: eDchAC.widciEulUeStatusiSsTxP, port2: widciEulUeStatusiSsTxP, router: true}
    - {port1: eDchAC.iubiP, port2: iubI, router: true}
    - {port1: l1UlDchDbgiP, port2: ulDchAC.ulDchACDbgiP, router: true}
    - {port1: l1EDchDbgiP, port2: eDchAC.eDchACDbgiP, router: true}
    - {port1: l1EHichDbgiP, port2: eHichAC.eHichACDbgiP, router: true}
    - {port1: l1UlRlsDbgiP, port2: ulRlsAC.ulRlsACDbgiP, router: true}
    - {port1: l1hsDpcchDbgiP, port2: hsDpcchAC.hsDpcchDbgiP, router: true}
    - {port1: widciEulRecTypeiItcP, port2: ulRlsAC.widciRecTypeItcP, router: true}
    - {port1: widciEulRecTypeiSsRxHighP, port2: ulRlsAC.widciRecTypeSsRxHighP, router:
true}
    - {port1: ulRlsUuLoadiP, port2: ulRlsAC.ulRlsUuLoadiP, router: true}
    - {port1: eDchAC.eDchEHichiP, port2: eHichAC.eDchEHichiP, router: true}
    - {port1: hsDpcchAC.hsDpcchEHichiP, port2: eHichAC.hsDpcchEHichiP, router: true}
    - {port1: rlsCtrlAC.ctrlEHichRlsiP, port2: eHichAC.ctrlEHichRlsiP, router: true}
    - {port1: l1SearcherDbgiP, port2: searcherAC.searcherACDbgiP, router: true}
    - {port1: eDchAC.eDchUuLoadiP, port2: eDchUuLoadiP, router: true}
    - {port1: ulDchAC.ulDchUuLoadiP, port2: ulDchUuLoadiP, router: true}
    - {port1: ulRlsAC.widciConfigiP, port2: widciConfigiUlRlsP, router: true}
    - {port1: hsDpcchAC.widciConfigiP, port2: widciConfigiHsDpcchP, router: true}
    - {port1: eDchAC.widciConfigiP, port2: widciConfigiEDchP, router: true}
    - {port1: eHichAC.dlEulEHichiP, port2: dlEulEHichiP, router: true}
    - {port1: widciEulHwRateiItcP, port2: ulRlsAC.widciHwRateItcP, router: true}
...
```

Appendix B

Survey Questions

Question 1: Which representation(s) of the system (that are provided to you) do you prefer among all these alternatives?
Give the reason for choosing or not choosing of each representation.

Question 2: Which representation of this system do you prefer for the beginners among all these alternatives? Give the reason for choosing or not choosing of each representation. (This question was asked only Ericsson engineers)

Representation Type	Reason
Textual representation	
Original component representation	
Compact class representation (level 1)	
Compact class representation (between level 2-9)	
Compact class representation (level 10)	
Grouped elements representation	
Descriptive representation	
Split view option	

Appendix C

Configuration Files

1. Configuration File for Description (config_desc.txt)

```
Cch          CommonChannels
iub          I/O
Lcdi         Interface
cellAC       Controller
uuLoadAC     MeasurementScheduler
Sap          SingleAccesPoint
rlsCtrlAc    EntryPoint
dl           Downlink
ul           Uplink
Meas         Measurement
eHichAC      HighSpeedData
Dch          Decoding
Dbg          Debug
Ra           RandomAccess
Rls          RadioLinkSet
```

2. Configuration File for Grouping (config_group_port.txt)

```
iub
LdciP
Spri
widci
CellInfo
LoadiP
Dbg
```

3. Configuration File for Ranking Components (rank.csv)

```
sapRaAC;10
sapFachAC;10
sapRlsAC;10
sapCellAC;10
sapDbchCellAC;10
sapPchAC;10
sapGroupAC;10
l1DbgAC;1
dlRlsAC;10
rlsCtrlAC;10
ulRlsAC;10
searcherAC;6
```

```
rlsMeasAC;1
hsDpcchAC;2
eDchAC;3
ulDchAC;3
eHichAC;1
raPreambleAC;7
raMsgAC;10
raDecodeAC;8
iubTerminatorAC;10
spreaderAC;10
uuLoadAC;7
cellAC;10
cellMeasAC;1
dlHsAC;7
dlEulAC;7
rotAC;4
fachAC;10
pccpchAC;10
pchAC;10
```

Appendix D

Source Code

AbstractDiag.java

```
public class AbstractDiag {

   /*
    * Command line examples
    * AbstractDiag -group -split filename.puml
    * AbstractDiag -filter=3 -split filename.puml
    */
   public static void main(String[] args) {
      //Parse option
      Option option = new Option(args);
      String file = option.getFile();
      String current = System.getProperty("user.dir");

      if (file.length() > 0){
         Diagram diagram = Converter.parseDiagram(file);
         String result ="";

         //Filter out components
         if(option.getFilter() > 1){
            diagram = Converter.parseDiagramFilterHide(diagram,option.getFilter());
         }
         //describe option
         if(option.getDescribe()){
            result = Converter.addDescription(diagram);
         }

         //Grouping option - output string from diagram
         else if(option.getGroup()){
            result = Converter.createGroups(diagram);
         }
         // No-port - without any interface
         else if (option.getNoport()){
            result= Converter.createDiagramNoPorts(diagram);
```

```
         }

         // default abstraction diagram
         else {
            result = Converter.createPackageAbstraction(diagram);
         }

         //Split option - If only one package no need to create the overview
         if(option.getSplit()){
            if(diagram.getClusters().size() > 1){
               String overview = Converter.createPackageOverview(diagram);
               Converter.save(overview, current, "overview.puml");
            }

            Converter.splitDiagram(result, current);
         }

         else{ //Save as normal
            Converter.save(result, current, diagram.getName().replace(" ", "") + ".puml");
         }

      }
   }
}
```

Cluster.java

```
import java.util.ArrayList;


public class Cluster{
   private String name;
   private Integer ID;
   public Integer getID() {
      return ID;
   }

   public void setID(Integer iD) {
      ID = iD;
   }

   private ArrayList<Item> items;
```

```
   public Cluster(String name, int ID){
      this.name = name;
      this.ID = ID;
      this.items = new ArrayList<Item>();
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public ArrayList<Item> getItems() {
      return items;
   }

   public void setItems(ArrayList<Item> items) {
      this.items = items;
   }


}
```

Converter.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Converter {


   private static String startUML = "@startuml \n title Generated UML\n";
```

```
private static String endUML = "\n@enduml";


/**
 * Create plantUML data file
 * @param data
 * @param fileName
 */
public static void save(String data, String folder, String fileName){
   try {
      PrintWriter out = new PrintWriter(folder + "\\" + fileName);
      out.write(data);
      out.close();
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   }
}

/**
 * Split a plantUML diagram into subview-files
 * One view per package
 * @param s
 */

public static void splitDiagram(String s, String folder){
   String[] lines = s.split("\n");
   String buffer = "";
   String packageName = "";
   for(int i = 0; i< lines.length; i++){
      if(lines[i].contains("package")){
         if(buffer.length() > 0 && packageName.length() > 0){
            save(startUML + buffer + endUML, folder, packageName + ".puml");
         }
         buffer = "";
         String[] temp = lines[i].split(" ");
         packageName = temp[1];
      }
      buffer += lines[i] + "\n";
   }
   if(buffer.length()>0 && packageName.length()>0)
      save(startUML + buffer + endUML,folder, packageName + ".puml");
}
```

```
/**
 * Parse plantUML file into diagram object
 * @param file
 * @return
 */

public static Diagram parseDiagram (String file){
   Diagram diagram = new Diagram();
   Cluster cluster = null;
   String name = null;
   HashMap<String, Integer[]> catalog = new HashMap<String,Integer[]>();
   ArrayList<Item> items = null;
   Integer ID = 0;
   Integer offset = 0;
   try {
      BufferedReader br = new BufferedReader(new FileReader(file));
      String line;
      while ((line = br.readLine()) != null) {
         line = line.trim();

         if(line.startsWith("title")){
            diagram.setName(line.substring(6));
         }
         else if(line.startsWith("package")){ //start of package
            name = line.substring(9, line.length() - 3);
            cluster = new Cluster(name,offset);
            items = new ArrayList<Item>();
            ID = 0;
         }
         else if(line.startsWith("}")){ //end of package
            if(!(cluster == null)){
               cluster.setItems(items);
               diagram.addCluster(cluster);
               offset++;
            }
         }
         else if(line.contains("--")){ //edge definition
            String[] s = line.split("--");
            Item[] i = new Item[2];
            String itemName = null;
            Pattern pattern = Pattern.compile("(\\w)*[a-zA-Z](\\w)*");
```

```
            for(int j = 0; j<2; j++){
               Matcher matcher = pattern.matcher(s[j]);
               if(matcher.find()){
                  itemName = matcher.group(0);
                  if(!catalog.containsKey(itemName)){
                     catalog.put(itemName, new Integer[]{offset,ID});
                     Item.Type type = s[j].contains("[" + itemName)? Item.Type.COMPONENT: Item.Type.INTERFACE;
                     items.add(new Item(itemName,offset,name,ID, type));
                     i[j] = items.get(ID);
                     ID++;

                  }
                  else{
                     Integer[] temp = catalog.get(itemName);
                     //Same cluster
                     if(temp[0] == offset)
                        i[j] = items.get(temp[1]);
                     else
                        i[j] = diagram.getClusters().get(temp[0]).getItems().get(temp[1]);
                  }

               }
            }
            //Build connection
            i[0].addConnection(i[1]);
            i[1].addConnection(i[0]);
         }
      }
      br.close();
   }
   catch(Exception e){
      e.printStackTrace();
   }
   return diagram;
}


/**
 * Package abstraction : remove all connections going from inside a
 * package to another package. Instead just show inter-package connections
 * and inner package structure.
 * @param diagram
```

```
    * @return
    */
   public static String createPackageAbstraction(Diagram diagram){
      String classFile =  "";
      String clusterTemp = "";
      //classFile += "title " + diagram.getName() + "\n";
      for(int i = 0; i < diagram.getClusters().size(); i++){
         Cluster c = diagram.getClusters().get(i);
         classFile += "package " + c.getName() + " <<Rect>> {\n";
         for(int j = 0; j < c.getItems().size(); j++){
            Item item = c.getItems().get(j);
            String tempEdges = "";
            if(item.getType().equals(Item.Type.COMPONENT)){
               classFile += "\tclass " + item.getName() + "{\n";
               for(int k = 0; k < item.getConnection().size(); k++){
                  Item con = item.getConnection().get(k);
                  Boolean isInSame = con.getCluster() == c.getID();
                  if(!isInSame){
                     String temp = c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName() +
"\n";
                     if(!clusterTemp.contains(temp))
                        clusterTemp += c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName()
+ "\n";
                  }
                  if(con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() <= 1){
                     classFile += con.getName() + "\n";
                  }
                  else if (con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() > 1){
                     String temp = "\tclass " + con.getName() + "<<I,orchid>>{\n" + con.getConList() + "\n}\n";
                     if(!classFile.contains(temp))
                        tempEdges += temp;
                     if(isInSame)
                        tempEdges += "\t" + item.getName() + " -- " + con.getName() + "\n";
                  }
                  else if (con.getType().equals(Item.Type.COMPONENT)){
                     String temp = "\t" +con.getName() + " -- " +  item.getName() + "\n";
                     if(isInSame && !classFile.contains(temp))
                        tempEdges += "\t" +item.getName() + " -- " + con.getName() + "\n";;
                  }
               }
               classFile += item.getConList() + "\t}\n" + tempEdges + "\n";
            }
```

```
            else if (item.getType().equals(Item.Type.INTERFACE)){
               if(item.getConnection().size() > 1){
                  for(int k = 0; k < item.getConnection().size(); k++){
                     Item con = item.getConnection().get(k);
                     if(con.getType().equals(Item.Type.INTERFACE)){
                        classFile += "\t" +item.getName() + " -- " + con.getName() + "\n";
                     }
                  }
               }
            }
         }
         classFile +=diagram.getHideString(c.getName())+"}\n";
      }
      classFile= startUML + clusterTemp + classFile + endUML;
      return classFile;
   }
   /**
    * No interfaces/ports option
    * @param diagram
    * @return
    */
   public static String createDiagramNoPorts(Diagram diagram){
      String classFile = "";
      String clusterTemp = "";
      for(int i = 0; i < diagram.getClusters().size(); i++){
         Cluster c = diagram.getClusters().get(i);
         classFile += "package " + c.getName() + " <<Rect>> {\n";
         for(int j = 0; j < c.getItems().size(); j++){
            Item item = c.getItems().get(j);
            String tempEdges = "";
            if(item.getType().equals(Item.Type.COMPONENT)){
               classFile += "\tclass " + item.getName() + "{\n";
               for(int p=0;p<item.getPackageLinks().size();p++){
                  classFile += item.getPackageLinks().get(p) + "()\n";
               }
               for(int k = 0; k < item.getConnection().size(); k++){
                  Item con = item.getConnection().get(k);
                  Boolean isInSame = con.getCluster() == c.getID();
                  if(!isInSame){ //Package links
                     String temp = c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName() +
"\n";
                     if(!clusterTemp.contains(temp))
```

```
                        clusterTemp += c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName()
+ "\n";
                  }
                  if(isInSame && con.getType().equals(Item.Type.INTERFACE)
                        && con.getConnection().size() <= 1){ //Simple lollipop port
                     //classFile += con.getName() + "\n";
                     for(int h = 0; h < con.getPackageLinks().size(); h++){
                        Boolean b = item.addPackageConnection(con.getPackageLinks().get(h));
                        if(b)
                           classFile += con.getPackageLinks().get(h) + "()\n";
                     }

                  }
                  else if(isInSame && con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() > 1){
                     //classFile += con.getName() + "\n";
                     for(int h = 0; h < con.getPackageLinks().size(); h++){
                        Boolean b = item.addPackageConnection(con.getPackageLinks().get(h));
                        if(b)
                           classFile += con.getPackageLinks().get(h) + "()\n";
                     }
                  }
                  else if (con.getType().equals(Item.Type.COMPONENT)){
                     String temp = "\t" +con.getName() + " -- " +  item.getName() + "\n";
                     if(!classFile.contains(temp))
                        tempEdges += "\t" +item.getName() + " -- " + con.getName() + "\n";
                  }
               }
               //item.getConList() +
               classFile +=  "\t}\n" + tempEdges + "\n";
            }

         }
         classFile += "}\n";
      }
      classFile = startUML + clusterTemp + classFile + endUML;;
      return classFile;
   }


   /**
    * Create diagram overview
    * Packages as black boxes
```

```
    * @param diagram
    * @return
    */
   public static String createPackageOverview(Diagram diagram){
      String classFile = startUML;
      String clusterTemp = "";
      for(int i = 0; i < diagram.getClusters().size(); i++){
         Cluster c = diagram.getClusters().get(i);
         classFile += "package " + c.getName() + " <<Rect>> {\n}\n";
         for(int j = 0; j < c.getItems().size(); j++){
            Item item = c.getItems().get(j);
            if(item.getType().equals(Item.Type.COMPONENT)){
               for(int k = 0; k < item.getConnection().size(); k++){
                  Item con = item.getConnection().get(k);
                  Boolean isInSame = con.getCluster() == c.getID();
                  if(!isInSame){
                     String temp = c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName() +
"\n";
                     if(!clusterTemp.contains(temp))
                        clusterTemp += c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName()
+ "\n";
                  }
               }
            }
         }

      }
      classFile += clusterTemp + endUML;
      //System.out.println(classFile);
      return classFile;
   }



   /**
    * Filter logic using the plantuml hide attribute
    * @param diagram
    * @param limit
    * @return
    */
   public static Diagram parseDiagramFilterHide (Diagram diagram, int limit){
      //Parse ranking
```

```
   ArrayList<String> ranks = new ArrayList<String>();
   try (BufferedReader br = new BufferedReader(new FileReader("config\\rank.csv"))) {
      String line;
      while ((line = br.readLine()) != null) {
         String[] s = line.split(";");
         if(!ranks.contains(s[0]) && Integer.parseInt(s[1]) >= limit)
            ranks.add(s[0]);
      }
   } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
   } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
   }
   //Remove elements
   for(int i = 0; i < diagram.getClusters().size(); i++){
      Cluster c = diagram.getClusters().get(i);
      for(int j = 0; j < c.getItems().size(); j++){
         Item item = c.getItems().get(j);
         if(item.getType().equals(Item.Type.COMPONENT) && !ranks.contains(item.getName())){
            diagram.addHideItem(item.getClusterName(), item.getName());
            for(int k = 0; k < item.getConnection().size(); k++){
               Item con = item.getConnection().get(k);
               Boolean isInSame = item.getClusterName().equals(con.getClusterName());
               if(isInSame && con.getType().equals(Item.Type.INTERFACE)
                     && (con.getConnection().size() > 1 || con.hasOutsideConnection())){
                  int cnt = con.getHidenCount() + 1;
                  con.setHidenCount(cnt);
                  int conNb = con.getInternalConnectionCount();
                  if(conNb == cnt)
                     diagram.addHideItem(con.getClusterName(), con.getName());
               }
            }
         }
      }
   }

   return diagram;
}
```

```
/**
 * Create groups for ports and components
 * @param diagram
 * @return
 */

public static String createGroups (Diagram diagram){

   String classFile = "@startuml\n";
   classFile += "title " + diagram.getName() + "\n";

   for(int i = 0; i < diagram.getClusters().size(); i++){ //each package
      Cluster c = diagram.getClusters().get(i);
      classFile += "package " + c.getName() + " <<Rect>> {\n";
      String tempEdgesComp = ""; // for grouping components to avoid to have multiple line connections
      ArrayList <String> hiddenList= new ArrayList<String>();

      if(!diagram.getHideDict().isEmpty() && diagram.getHideDict().containsKey(c.getName()))
         hiddenList = diagram.getHideDict().get(c.getName());


      for(int j = 0; j < c.getItems().size(); j++){ // each item (component or port)

         Item item = c.getItems().get(j);
         String tempEdges = ""; // for grouping ports to avoid to have multiple line connections
         HashMap<String, String> groupList= new HashMap<String,String>(); // to avoid multiple grouping

         if(!hiddenList.contains(item.getName())){
            if(item.getType().equals(Item.Type.COMPONENT)){

               String tempComp= getGroupName(item.getName(),item.getType());

               // (!L1SC) not to include L1 overview for grouping components
               if((tempComp!=null)&& !item.getClusterName().contains("L1SC")){

                  //if it is not already registered
                  if(!(groupList.containsKey( tempComp + item.getClusterName()))){
                     groupList.put(tempComp+ item.getClusterName(),item.getName());
                     classFile += "\tclass " + tempComp + "{\n";
                  }

               }
```

```
else{ // have no component group
   tempComp= item.getName();
   classFile += "\tclass " + tempComp + "{\n";
}



for(int k = 0; k < item.getConnection().size(); k++){ //each connections of component
   Item con = item.getConnection().get(k);

   if(!hiddenList.contains(con.getName())){

      // interface(1) connection with component
      if(con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() <= 1 ){
         String tempGroup= getGroupName(con.getName(),con.getType());
         if((tempGroup!= null) ){

            //if it is not already registered
            if(!(groupList.containsKey( tempGroup + con.getClusterName()+"_in"))){
               groupList.put(tempGroup+con.getClusterName()+"_in",con.getName());

               //to avoid multiple same port groups in grouped components
               if(tempComp.equals(item.getName())||!classFile.contains(tempGroup +"_in" + "\n")){

                  classFile +=  tempGroup +"_in" + "\n";
               }
            }
         }
         else{ // have no port group
            classFile += con.getName() + "\n";
         }

      }// end port(1) connection  with component

      // port(>1) connection with component
      else if (con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() > 1  ){
         String tempGroup = getGroupName(con.getName(),con.getType());
         if((tempGroup!= null)){
            //if it is not already registered
            if(!(groupList.containsKey( tempGroup+con.getClusterName()+"_out"))){
               groupList.put(tempGroup+con.getClusterName()+"_out",con.getName());
               //if it is not already registered
```

```
            String newName= tempGroup+"_"+
            con.getClusterName().substring(0,3);//item.getClusterName().substring(0,3);
            // creating port class for showing outer connections
            String testEdgesComp1= "\tclass " + newName+ "<<I,orchid>>{\n}\n";
            String testEdgesComp2= "\t" + tempComp + " -- " + newName + "\n";
            //to not have more lines while grouping components
            if (!(tempEdgesComp.contains(testEdgesComp2))){
               tempEdges+= testEdgesComp1;
               tempEdges+= testEdgesComp2;
               tempEdgesComp+= testEdgesComp1; // add edges for further checks
               tempEdgesComp+= testEdgesComp2;
            }
         }
      }
      else{ // have no port group
         tempEdges += "\tclass " + con.getName()+ "<<I,orchid>>{\n}\n";
         String testEdges = "\t" + tempComp + " -- " + con.getName() + "\n";

         if (!tempEdges.contains(testEdges)){ // if connection already written
            tempEdges+= testEdges;
         }
      }
   }// end port(>1) connection  with component
   else if (con.getType().equals(Item.Type.COMPONENT)){ // component connection with component
      String test1="";
      String test2="";
      String tempComp2= getGroupName(con.getName(),con.getType());
      if((tempComp2!=null)&& !item.getClusterName().contains("L1SC")){
         if(!(groupList.containsKey( tempComp2 + item.getClusterName()))){
            groupList.put(tempComp2+ item.getClusterName(),con.getName());
            test1 = "\t" +tempComp2 + " -- " +  tempComp + "\n";
            test2 = "\t" + tempComp + " -- " + tempComp2 + "\n";
            // check if edges are defined already
            if(!classFile.contains(test1)&& !classFile.contains(test2))
               tempEdges += "\t" + tempComp + " -- " + tempComp2 + "\n";
         }
      }
      else  { // have no component group
         test1 = "\t" +con.getName() + " -- " +  tempComp + "\n";
         test2 = "\t" + tempComp + " -- " + con.getName() + "\n";
```

```
                        // check if edges are defined already
                        if(!classFile.contains(test1)&& !classFile.contains(test2))
                           tempEdges += "\t" + tempComp + " -- " + con.getName() + "\n";
                     }
                  } // end component connection with component
               } // check hiddenList of component connections
            }// end for loop (each  connections of component(item))
            classFile += "\t}\n" + tempEdges + "\n";
         } // end if item is component

         else if (item.getType().equals(Item.Type.INTERFACE)){ // if item is port
            if(item.getConnection().size() > 1){ // port more than 1 connection
               for(int k = 0; k < item.getConnection().size(); k++){ //each connections of port item
                  Item con = item.getConnection().get(k);
                  if(con.getType().equals(Item.Type.INTERFACE)){ //int- int connections
                     String tempGroup = getGroupName(con.getName(),con.getType());
                     if((tempGroup!= null)){
                        if(!(groupList.containsKey( tempGroup+con.getClusterName()))){
                           groupList.put(tempGroup+con.getClusterName(),con.getName());
                           classFile += "\t" +item.getName() + " -- " + tempGroup+ "_"+con.getClusterName() +
"\n";
                        }
                     }
                     else{ //have no port group
                        classFile += "\t" +item.getName() + " -- " + con.getName() + "\n";
                     }
                  }
               }
            }
         }// end if item is port
      }// check if item is in hiddenList
   }// end for loop checking each item in the package
   classFile += "}\n";
}// end for loop checking each package

   classFile += "hide empty members \n@enduml";
   //System.out.println(classFile);
   return classFile;
}
```

```
private static String getGroupName(String name,Item.Type type){
   String group= null;
   BufferedReader br = null;
   try{
      String line;
      if (type.equals(Item.Type.INTERFACE)) // to distinguish same names for ports and components
         br = new BufferedReader(new FileReader("config\\config_group_port.txt"));
      else
         br= new BufferedReader(new FileReader("config\\config_group_comp.txt"));

      while ((line = br.readLine())!= null){
         if (name.contains(line.trim())){ //check if it is in config file
            group= line.trim();
            break;
         }
      }
      br.close();
   }
   catch(Exception e){
      System.out.println("error" );
      e.printStackTrace();
   }
   return group;

}

/**
 * Add description for ports and components
 * @param diagram
 * @return
 */
public static String addDescription(Diagram diagram){
   String classFile =  "";
   String clusterTemp = "";
   //classFile += "title " + diagram.getName() + "\n";
   String descPack= null;
   String descComp= null;
   String descPort= null;
   HashMap<String, ArrayList<String>> hiddenList= new HashMap<String, ArrayList<String>>();

   for(int i = 0; i < diagram.getClusters().size(); i++){
      Cluster c = diagram.getClusters().get(i);
```

```
descPack= getDescription(c.getName());
ArrayList<String> list= new ArrayList<String>();

if(diagram.getHideDict().containsKey(c.getName())){
   list= diagram.getHideDict().get(c.getName());
}

if (descPack!=null){
   hiddenList.put(descPack, new ArrayList<String>());
   hiddenList.replace(descPack, list);
   c.setName(descPack);
}
else{

   hiddenList.put(c.getName(),new ArrayList<String>());
   hiddenList.replace(c.getName() ,list);

}

classFile += "package " + c.getName() + " <<Rect>> {\n";


for(int j = 0; j < c.getItems().size(); j++){
   Item item = c.getItems().get(j);
   String tempEdges = "";

   if(item.getType().equals(Item.Type.COMPONENT)){
      descComp= getDescription(item.getName());
      if (descComp!=null&& !item.getClusterName().contains("L1SC")){
         classFile += "\tclass " + item.getName() +"< " +descComp + "> {\n";
      }
      else{
         classFile += "\tclass " + item.getName() + "{\n";
      }

      for(int k = 0; k < item.getConnection().size(); k++){
         Item con = item.getConnection().get(k);

         Boolean isInSame = con.getCluster() == c.getID();
         descPort= getDescription(con.getName());
         if(!isInSame){
```

```
                String temp = c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName() +
"\n";
                if(!clusterTemp.contains(temp)){
                   clusterTemp += c.getName() + " -- " + diagram.getClusters().get(con.getCluster()).getName()
+ "\n";
                }
             }

             if(con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() <= 1){
                if (descPort!=null){
                   classFile += con.getName() +"  <<"+descPort+ ">>\n";
                }
                else{
                   classFile += con.getName() + "\n";
                }
             }
             else if (con.getType().equals(Item.Type.INTERFACE) && con.getConnection().size() > 1){
                String temp="";
                if (descPort!=null){
                   temp= "\tclass " + con.getName() +"< " +descPort +
                   "><<I,orchid>>{\n"+con.getConList()+"\n}\n";
                }
                else{
                   temp= "\tclass " + con.getName() + "<<I,orchid>>{\n"+con.getConList()+"\n}\n";
                }
                if(!classFile.contains(temp)){
                   tempEdges += temp;
                }
                if(isInSame)
                   tempEdges += "\t" + item.getName() + " -- " + con.getName() + "\n";

             }
             else if (con.getType().equals(Item.Type.COMPONENT)){
                String temp = "\t" +con.getName() + " -- " +  item.getName() + "\n";
                if(isInSame && !classFile.contains(temp))
                   tempEdges += "\t" +item.getName() + " -- " + con.getName() + "\n";;
             }
          }
          classFile += item.getConList() +"\t}\n" + tempEdges + "\n";
       }
       else if (item.getType().equals(Item.Type.INTERFACE)){
          if(item.getConnection().size() > 1){
```

```
               for(int k = 0; k < item.getConnection().size(); k++){
                  Item con = item.getConnection().get(k);
                  if(con.getType().equals(Item.Type.INTERFACE)){
                     classFile += "\t" +item.getName() + " -- " + con.getName() + "\n";

                  }
               }
            }

         }
      }

      String temp= "";
      if(list != null && !list.isEmpty()){
         temp += getHideString(hiddenList.get(c.getName()));
      }

      classFile += temp +"}\n"; //

   }
   classFile= startUML + clusterTemp+ classFile + endUML;
   return classFile;
}


private static String getHideString(ArrayList<String> hiddenList){
   String result = "";
   for(int i=0;i<hiddenList.size();i++){
      result += "hide " + hiddenList.get(i) + "\n";
   }
   return result;
}

private static String getDescription(String name){

   String description= null;
   BufferedReader br = null;
   try{
      String line;
      String[] tokens;
      br= new BufferedReader(new FileReader("config\\config_desc.txt"));
```

```
         while ((line = br.readLine())!= null){
            tokens  = line.split(" ");
            if (name.contains(tokens[0])){
               description= tokens[1];
               break;
            }
         }
         br.close();
      }
      catch(Exception e){
         System.out.println("error" );
         e.printStackTrace();
      }
      return description;
   }



}
```

Diagram.java

```
import java.util.ArrayList;
import java.util.HashMap;


public class Diagram {
   private String name;
   private ArrayList<Cluster> clusters;
   private HashMap<String, ArrayList<String>> hideDict;
   public Diagram(String name){
      this.name = name;
      this.clusters = new ArrayList<Cluster>();
   }

   public Diagram() {
      this.name = "";
```

```
   this.clusters = new ArrayList<Cluster>();
   hideDict = new HashMap<String, ArrayList<String>>();
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public ArrayList<Cluster> getClusters() {
   return clusters;
}

public void setClusters(ArrayList<Cluster> clusters) {
   this.clusters = clusters;
}

public HashMap<String, ArrayList<String>> getHideDict() {
   return hideDict;
}

public void setHideDict(HashMap<String, ArrayList<String>> hideDict) {
   this.hideDict = hideDict;
}

public void addCluster(Cluster cluster){
   clusters.add(cluster);
}

public void fixPackagesConnection(){
   for(int i = 0; i < this.getClusters().size(); i++){
      Cluster c = this.getClusters().get(i);
      for(int j = 0; j < c.getItems().size(); j++){
         Item item = c.getItems().get(j);
         for(int k = 0; k < item.getConnection().size(); k++){
            Item con =  item.getConnection().get(k);
            if(!con.getClusterName().equals(c.getName()))
               item.addPackageConnection(con.getClusterName());
         }
```

```
         }
      }
   }

   public void addHideItem(String cluster, String itemName){
      if(!hideDict.containsKey(cluster))
         hideDict.put(cluster, new ArrayList<String>());
      ArrayList<String> a = hideDict.get(cluster);
      if(!a.contains(itemName)){
         a.add(itemName);
         hideDict.replace(cluster, a);
      }
   }
   public String getHideString(String cluster){
      String result = "";
      if(hideDict.containsKey(cluster)){
         ArrayList<String> a = hideDict.get(cluster);

         for(int i=0;i<a.size();i++){
            result += "hide " + a.get(i) + "\n";
         }
      }
      return result;
   }
}
```

Item.java

```
import java.util.ArrayList;


public class Item {
   public enum Type {
      COMPONENT, INTERFACE
   }

   private Boolean isConnected;
   private String   name;
   private Type type;
   private int ID;
   private ArrayList<Item> connection;
   private Integer cluster;
```

```
private String clusterName;
private ArrayList<String> packageLinks;
private Integer hidenCount;

public ArrayList<String> getPackageLinks() {
   return packageLinks;
}

public void setPackageLinks(ArrayList<String> packageLinks) {
   this.packageLinks = packageLinks;
}

public String getClusterName() {
   return clusterName;
}

public void setClusterName(String clusterName) {
   this.clusterName = clusterName;
}

public Integer getCluster() {
   return cluster;
}

public void setCluster(Integer cluster) {
   this.cluster = cluster;
}

public Boolean getIsConnected() {
   return isConnected;
}

public void setIsConnected(Boolean isConnected) {
   this.isConnected = isConnected;
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
```

```
}

public Type getType() {
   return type;
}

public void setType(Type type) {
   this.type = type;
}

public int getID() {
   return ID;
}

public void setID(int iD) {
   ID = iD;
}

public ArrayList<Item> getConnection() {
   return connection;
}

public void setConnection(ArrayList<Item> connection) {
   this.connection = connection;
}

public Integer getHidenCount() {
   return hidenCount;
}

public void setHidenCount(Integer hidenCount) {
   this.hidenCount = hidenCount;
}

public Item(){
   this.isConnected = false;
   this.name = "";
   this.cluster = -1;
   this.ID = -1;
   this.connection = new ArrayList<Item>();
   this.hidenCount = 0;
}
```

```
public Item(String name, int cluster, String clusterName, int ID, Type type){
   this.isConnected = false;
   this.name = name;
   this.cluster = cluster;
   this.clusterName = clusterName;
   this.type = type;
   this.ID = ID;
   this.connection = new ArrayList<Item>();
   this.packageLinks = new ArrayList<String>();
   this.hidenCount = 0;
}

public void addConnection(Item item){
   if(!this.connection.contains(item))
      this.connection.add(item);
   this.isConnected = (this.connection.size() > 1);
}
public String getConList(){
   String result = "";
   for(int i = 0; i < this.connection.size(); i++){
      if(this.cluster != this.connection.get(i).getCluster()){
         result += this.connection.get(i).getClusterName() + "." + this.connection.get(i).getName() + "()\n";
      }
   }
   return result;
}
public void removeLink(String clusterName, String itemName){
   int ref = -1;
   for(int i = 0; i<this.connection.size();i++){
      if(this.connection.get(i).getName().equals(itemName)
            && this.connection.get(i).getClusterName().equals(clusterName)){
         ref = i;
         break;
      }
   }
   if(ref != -1){
      this.connection.remove(ref);
   }
}

public Boolean addPackageConnection(String packageName){
```

```
      Boolean res = true;
      for(int i=0; i<this.packageLinks.size(); i++){
         if(this.packageLinks.get(i).equals(packageName)){
            res = false;
         }
      }
      if(res)
         this.packageLinks.add(packageName);
      return res;
   }

   public Boolean hasOutsideConnection(){
      Boolean result = false;
      for(int i = 0; i<this.connection.size();i++){
         if(this.connection.get(i).getCluster() != this.getCluster())
         {
            result = true;
            break;
         }
      }
      return result;
   }
   public int getInternalConnectionCount(){
      int ref = 0;
      for(int i = 0; i<this.connection.size();i++){
         if(this.connection.get(i).getClusterName().equals(this.getClusterName())){
            ref++;
         }
      }
      return ref;
   }

}
```

Option.java

```
public class Option {
   /*
    * Options:
    * -group: group elements together based on config file
    * -filter=7: filter elements with lower scores than for example 7 (default is 1 - show all)
    * -split: generate an overview of packages interconnection and splits views
    * -noport: show only components
    * -describe : show description of elements
    * File name is last
    */
   private String file;
   private Boolean group;
   private int filter;
   private Boolean split;
   private Boolean noport;
   private Boolean describe;


   public Option(String... args){
      group = false;
      filter = 1;
      split = false;
      noport = false;
      file = "";
      describe= false;

      for(int i = 0; i < args.length; i++){
         String temp = args[i].toLowerCase();
         if(temp.equals("-group"))
            group = true;
         else if(temp.equals("-split"))
            split = true;
         else if(temp.equals("-noport"))
            noport = true;
         else if(temp.contains("-filter")){
            int l = temp.indexOf('=') + 1;
            filter = Integer.parseInt(temp.substring(l, temp.length()));
         }
         else if(temp.equals("-describe"))
            describe = true;
         else{
```

```
            file = args[i];
        }

    }
}

public String  getFile() {
    return file;
}

public void setFile(String  file) {
    this.file = file;
}

public Boolean getGroup() {
    return group;
}

public void setGroup(Boolean group) {
    this.group = group;
}

public int getFilter() {
    return filter;
}

public void setFilter(int filter) {
    this.filter = filter;
}

public Boolean getSplit() {
    return split;
}

public void setSplit(Boolean split) {
    this.split = split;
}

public Boolean getNoport() {
    return noport;
}
```

```
   public void setNoport(Boolean noport) {
      this.noport = noport;
   }

   public Boolean getDescribe() {
      return describe;
   }

   public void setDescribe(Boolean description) {
      this.describe = description;
   }



}
```