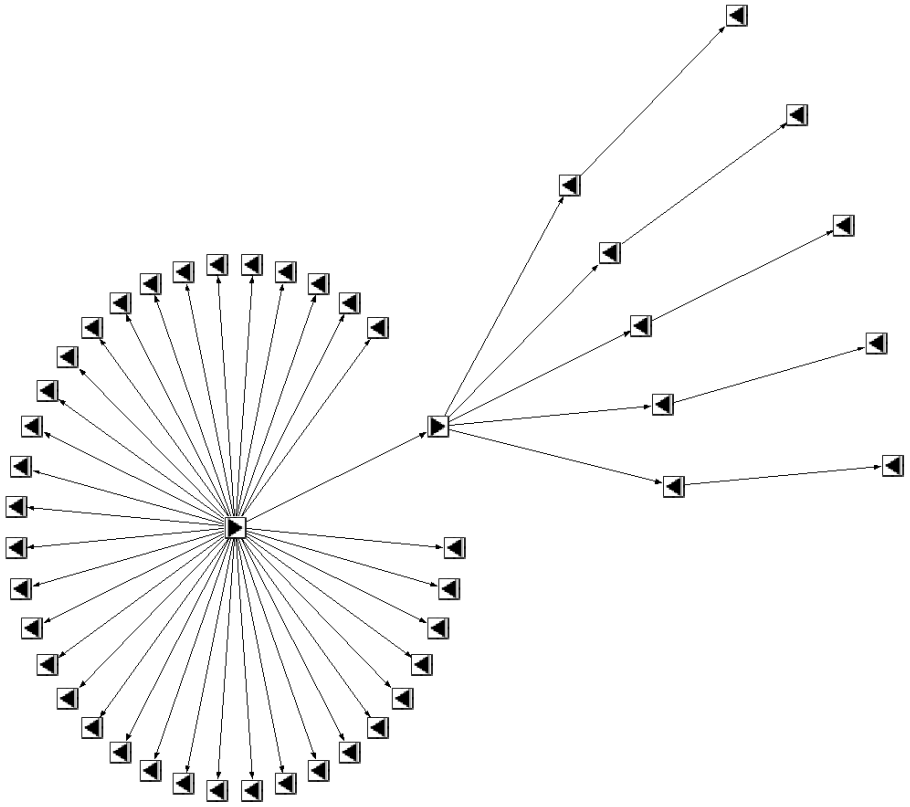

AUTOSAR Application Visualization

Degree Project, BSc in Computer Engineering

MIKAEL STOLPE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

AUTOSAR Application Visualization

Mikael Stolpe

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Tel: +46-(0)31-772 1000
Fax: +46-(0)31-772 3663

[Graph generated using the visualization tool. Shows how a connection moves through each level of the application parsed in this scenario]

Department of Computer Science and Engineering
Gothenburg, 2014

Foreword

This report is a Bachelor's thesis which was performed during the spring of 2014 as a part of the Bachelor program Computer Engineering at the Department of Computer Science and Engineering, Chalmers University of Technology.

The thesis and the resulting creation would not have been made possible if not for a couple of persons which I would like to acknowledge. First and foremost, I would thank Mr. Tobias Johansson at ArcCore for his advice, support and everlasting patience during the design and development of the visualization tool. Furthermore, I would like to thank Mr. Johan Ekberg, also at ArcCore, whom without I would not have performed this thesis at all. Mr. Erland Holmström, lecturer at the Computing Science division, Department of Computer Science and Engineering, my supervisor at Chalmers University of Technology, for helping and advising me throughout the thesis and especially for all the advice during the writing process. And finally, Mr. Sakib Sistek research engineer at the division of Networks and Systems, Department of Computer Science and Engineering, Chalmers University of Technology, whose support, advice and never-ending enthusiasm helped me tremendously at the initial stages of the thesis.

TABLE OF CONTENTS

Summary

AUTOSAR is a very large development partnership and its standards are used in the automotive business. More specifically, AUTOSAR holds a Software Component Template which describe rules that govern application[1] design in the AUTOSAR domain. Applications created using the Software Component Template can become enormous and difficult to interpret by only looking at the code. Therefore, alternative methods to look at the application are desired and one such method is creating a visual representation of the design. The report describes one way of implementing such a representation by using Zest, a graph-based visualization toolkit for Eclipse, as a framework. The tool was developed at the request of ArcCore and is therefore integrated into their software the Arctic Studio platform and developed as an Eclipse plug-in. It is realized using the MVC pattern and an agile work approach was used. The visualization tool uses a parsing method which can handle larger applications with ease. Furthermore, the tool focuses on using navigational abilities as a preferred method of simplifying the understanding of the application rather than creating advanced layout algorithms. The tool presents the application in a manner which allows the user to navigate through the application in several ways. It introduces the concept of following a connection through the application which could be used to locate different connection points and confirm correctness or locate errors in the application design.

Keywords: Visualization, AUTOSAR Software Component Template, Zest, Eclipse

Sammanfattning

AUTOSAR är ett väldigt stort samarbete inom bilindustrin som sätter standard för hur man utvecklar mjukvara till bildelar. AUTOSAR tillhandahåller ett dokument vid namn ”Software Component Template” som innehåller regler vilka bestämmer hur en applikation[2] ska se ut i det domän AUTOSAR behandlar. Applikationer skapade utifrån denna beskrivningen kan bli enorma och väldigt svåra att förstå när man bara ser koden. Således önskas alternativa metoder att tolka applikationen på. En sådan metod är att skapa en visuell representation av applikationsdesignen. Rapporten beskriver en metod för hur man implementerar en sådan representation genom att använda Zest, ett grafbaserat visualiseringsverktyg för Eclipse, som ramverk. Verktyget utvecklades på begäran av ArcCore och är på så vis integrerat i deras mjukvara Arctic Studio och utvecklat som ett Eclipse plug-in. Det realiseras med hjälp av MVC-mönstret och agile användes som arbetssätt. Visualiseringsverktyget använder en algoritm för att bygga upp modellen som kan hantera även större applikationer snabbt. Vidare fokuserar verktyget på att tillhandahålla användaren med navigationsmöjligheter för att få en enklare förståelse för applikationen istället för att skapa avancerade layoutalgoritmer. Verktyget presenterar applikationen på ett vis som tillåter användaren att navigera igenom applikationen på flera sätt. Det introduceras ett koncept som tillåter användaren att följa en koppling igenom hela applikationen. Detta kan användas för att hitta olika anslutningspunkter och bekräfta riktighet eller lokalisera fel i applikationens design.

Nyckelord: Visualisering, AUTOSAR Software Component Template, Zest, Eclipse

[1] Application refers to inter-connected components in AUTOSAR.

[2] En applikation i AUTOSAR är sammankopplade komponenter.

Designations

- Java - An object oriented programming language.
- API – Application Programming Interface
- Eclipse – An Integrated Development Platform.
- Plug-In – Software component which adds a specific feature to a software application.
- SWT – The Standard Widget Toolkit. A developing toolkit for Eclipse
- GEF – Graphical Editing Framework.
- ViewPort – Class within the GEF which represents what the user is viewing.
- Observer Design Pattern – A design pattern used in Java when objects needs to be informed about changes in another objects state.
- Zest – Graph based visualization toolkit for Eclipse, part of GEF.
- Graphviz – Open Source Visualization Software
- Draw2d – Visualization toolkit for Eclipse, part of GEF.
- UML – Unified Modeling Language, standardized way to visualize a system.
- ECU – Electronic Control Unit, generic term for embedded system in automotive electronics
- Automotive Software – Refers to software related to the automotive industry

Classes are through the report written as `Class` and methods with camel case, i.e. `longMethodName()`.

1. INTRODUCTION

BACKGROUND

ArcCore is a company which develops products used in automotive software development. Automotive software development refers to software development for the automotive business. The company are one of the vendors which provides AUTOSAR products to the automotive market. AUTOSAR is a standardized and open-source automotive software architecture [1]. The products, which ArcCore develops, use a model based approach to software development and relies on configuration files and code generation. A configuration file is basically a design of how an application should function. During development of configurations it can be difficult for the users to get an overview of the current state of the configuration. A logical step in how to solve this problem would be to go from a pure code state to a more visual view of the configuration. Therefore, ArcCore believes a graphical representation could help the user review and understand the current state of the configuration. Conceivably, this could alleviate some of the frustration and visualization the user is forced to do while working with the configuration and consequently speed up the work process.

PURPOSE

The first objective of the report is to investigate different technologies which could help solve the following tasks:

- Create a graphical representation of a configuration.
- Integrating the graphical representation into the ArcCore module.

The main objective is to create a working tool which implements these technologies and can generate a visual model of an AUTOSAR configuration. The purpose of the report can be further specified into two main goals and one secondary goals and they are as follows.

MAIN GOALS

- Create a tool that reads an AUTOSAR configuration and outputs a graphical representation of it in the form of a block diagram.
- The tool should be integrated into the ArcCore AUTOSAR tool suite Arctic Studio with two important requirements:
    1. Visualization generation should be possible to trigger from the GUI.
    2. The resulting visual output should be displayed in the GUI.

SECONDARY GOALS

- The tool should be realized in such a way that it easily can be extendable to include other parts of AUTOSAR.

RESTRICTIONS

There are three different restrictions to the purposes of this report. Firstly, the generated model will be viewed only, there is no plan on making it editable once it has been generated. Secondly, the input of the graphical representation will be restricted to the AUTOSAR Software Component Template. Finally, this tool will be developed for the Arctic Studio platform and will therefore be integrated into that system and not created as a standalone software.

DISPOSITION

This report is divided into three major sections, the model, view and user-input handlers[3]. Each section hosts an introduction with architectural design and is followed with details about that section. There is some advantage to read it from top to bottom, but not a requirement. If there is information sharing between the different sections it is cross-referenced to increase readability. A recommendation is to read Chapter 3 “Introduction to AUTOSAR Software Component Template” beforehand to get a background understanding about AUTOSAR.

Due to main focus being the development of a plugin for Eclipse it would be beneficial of having certain knowledge and understanding in software development and more precisely Eclipse plugin development. However, this is not a requirement but if the reader feels the need to gain more knowledge in the subject a good source would be the book Practical Eclipse Rich Client Platform Projects [3].

[3] A handler in this case is referred to a class which hosts as a controller of several Listeners.

2. METHOD

The tool which has been developed was created for the Arctic Studio platform. The Arctic Studio platform is an Eclipse based plug-in software created and developed by ArcCore. Subsequently, the tool is an Eclipse plug-in and developed using the Eclipse plug-in development environment. However, this does not limit the creation of the tool to be limited to the Arctic Studio platform. The only requirement to achieve the same results described in this report is that it is developed for the Eclipse IDE as a plug-in.

The project has been realized using agile software development [2] which means dividing the project into different features which will be implemented throughout the project. The idea is to work on one feature at a time and design, develop and test this before moving on to the next one. Using this method has the advantage of quickly finding if there is a problem with the general design of the tool whilst also opening up for changes or redesign if needed. Furthermore, the tool is developed using the software architectural MVC-pattern as a general design to work towards. The MVC-pattern is a well-established pattern in software development which helps keep the code divided into three distinct parts: the model, view and controller. Using this pattern keeps the model and view separate and opens up for possibilities of creating several separate views using the same data. It gives the design of the tool a very natural separation of the different responsibilities of the tool. The model, view and controller for this tool were decided to be designed as followed:

- Generating the model from an AUTOSAR configuration would not be optimal. An AUTOSAR configuration could potentially become enormous. Therefore, creating a graphical representation of such a configuration whilst keeping all the information intact would most likely slow down the program greatly. Consequently, the first step in the project is to create a module which will reparse and interpret the AUTOSAR configuration and create a model which is slimmer and more adapted to perform the task required. As a result it will be a faster and more optimized tool.
- The view part will be a window or a view integrated into the Arctic Studio platform and it will be responsible for presenting the data of the model in a graphical way. The chosen framework for presenting the data is Zest and will be used to render a diagram similar to a graph. The purpose of the view is simply to present the data and there will be no input from the graphical presentation to the model.
- Finally, the controller will register the view to the model and the view will be updated as the model is changed. Focusing on optimization and still being user friendly the controller will inform the view when the user wishes to reparse the model.

With the combination of MVC and agile it gives freedom to the development of the tool. The further the development of the tool is progressed the more features will most likely be discovered. As a consequence the model, view and controller will have to be adapted to fit these features. However, since each parts logic is separated it simplifies the process considerably.

3. INTRODUCTION TO AUTOSAR SOFTWARE COMPONENT TEMPLATE

It is important to have a rudimentary understanding about AUTOSAR to be able to grasp some of the development decisions made for this tool, this chapter will attempt to describe the important concepts and aspects necessary for this. The official description of AUTOSAR can be found on the official website and reads as follows: “AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of car manufacturers, suppliers and other companies from the electronics, semiconductor and software industry” [1]. AUTOSAR holds a development library used by a wide variety of field experts when developing cars and components for cars. This report and the tool created focuses only on a small part of AUTOSAR, specifically the AUTOSAR Software Component Template.

To give the reader a bit more understanding Figure 3.1 shows a very simple configuration using the AUTOSAR Software Component Template. The configuration is simplest described by making comparison to an object-oriented language such as Java. The larger boxes with the type as suffix could be looked upon as a class, the type defines how a prototype should look when created, what ports it should hold and other properties. The black boxes with prototype as suffix could then be seen as an object or an instantiation of a type. Furthermore, how the prototypes are connected is specified by the connectors (the grey boxes) and there is no rule saying that two prototypes instantiated from the same type needs to have the same connectors in any way. Consequently, a user could instantiate a prototype with several ports but have it completely unconnected. Fairly unusable, but the template has no restriction here.

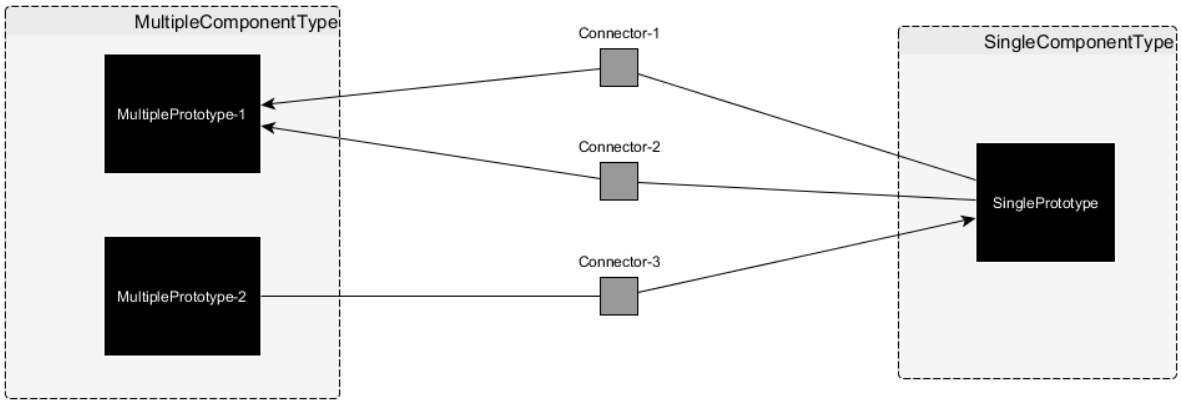

FIGURE 3.1, SIMPLE AUTOSAR SOFTWARE COMPONENT CONFIGURATION

These relevant parts for the plug-in could be looked upon in a tremendously simplified way, which is components, compositions, ports and connections.

- A component is the combination of a type and a prototype. The type defines which ports the component should have and the prototype could be seen as the instantiation of a type, giving it a unique reference.
- Connections are created to exist between two ports.
- Ports are by definition only created with a component. They are defined by the type but only exist if instantiated as a part of a prototype.
- A composition is a component and also acts as a container for other components.

The multiplicity of all these parts are not limited in any way and as a result an application can grow without limitations[4]. It is even more intricate than this, each instantiation of a part in the application holds a type and each type can hold its own definition and rules[5] making understanding a configured application possibly complex. Using the AUTOSAR Software Component Template engineers can design how the embedded software in an Electronic Control Unit[6] (ECU) should function, an example of this could be the process of raising the side window. These applications can become additionally more complex and it is even possible to describe the architectural design of a car and how the ECU's are interconnected.

Each port has an interface which specifies it as either a receiver or provider which means a port either sends a signal or receives one. Therefore, a signal is restricted to moving directly from one component to another within that composition, Figure 3.2 illustrates a situation which is not possible.

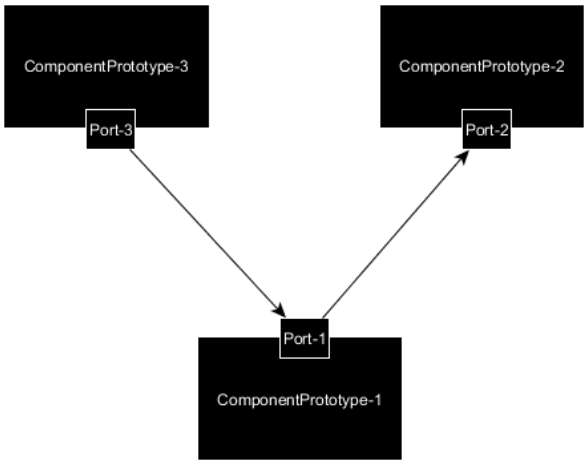

FIGURE 3.2, ILLUSTRATION OF A SITUATION NOT ALLOWED IN AUTOSAR

However, if the component is a composition those ports could also be connected to other components from the inside of itself, in effect propagating the signal further down the application. Figure 3.3 illustrates this situation further, it shows how a signal starts at ComponentPrototype-1 moves to CompositionPrototype-2 and insides that composition, to end at the ComponentPrototype-2. Once again there is nothing which prevents the signal from moving further, should the last component have been a composition instead this would have been possible.

[4] Limitations could of course be seen in computational power and memory storage. But not from the AUTOSAR Software Component Template.

[5] To explain the whole template is far from the scope of this report, the whole definition features above 700 pages and can be found on the AUTOSAR homepage: autosar.org.

[6] Electronic Control Unit, generic term for embedded system in automotive electronics.

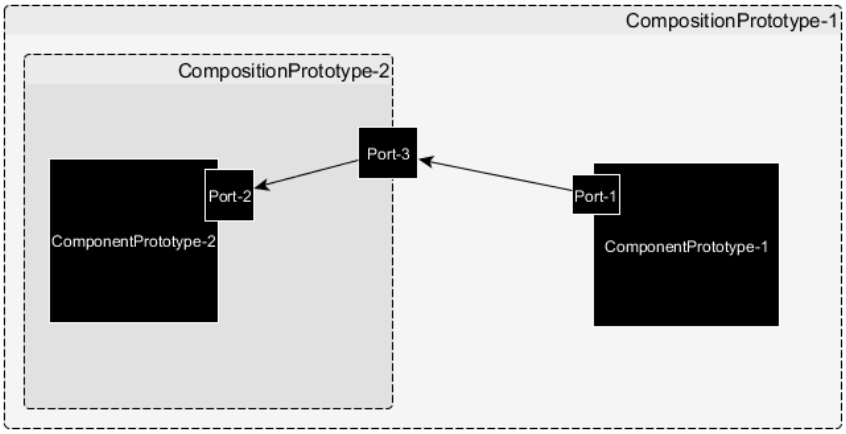

FIGURE 3.3, ILLUSTRATION OF HOW A CONNECTION PROPAGATES INSIDE A COMPOSITION

There is a concept of internal behavior of components, using this it could be possible to send a signal into the component and have it emerge from another port. However, internal behavior is not a part of the visualization and has no relevant consequence to the tool developed or the visualization of connections. If such a situation exists in the application the connection would simply appear to start at the port it emerges from.

4. REALISATION

4.1. CHOICE OF FRAMEWORK

Initially, there were several requirements on the choice of API for the visualization part of the tool. The first requirement was that the tool needed to be developed for the Eclipse platform in order to be integrated into the Arctic Studio platform. As a consequence, the framework needs to be able to be integrated into Eclipse in a preferably seamless way. Secondly, due to the possibly limitless scalability of an AUTOSAR Software configuration, it needed to be very lightweight and fast when generating the visual parts. Currently, there are only a few tools which are capable of satisfying all the needs that the tool requires and the ones found and considered were the following:

- Graphviz [3] is a powerful open source graph visualization software. It reads graph definitions from files using a text language called ".dot" and using this description it generates the graph in another more visual format. It hosts many powerful layout algorithms which would definitely be beneficial to take advantage of. It would be possible to use this API for the graph generation and create all the visual parts using Eclipse Rich Client Platform (RCP) development [4].
- Zest [5] is an Eclipse-based visualization toolkit. It has several visualization components which have been seamlessly integrated into the Eclipse development framework. One of its most powerful features is its graph drawing tools which can be incorporated into Eclipse views easily and be used to draw graphs quickly and efficiently. Zest also holds several powerful layout algorithms which could be used advantageously.
- Draw2D [6] is also a toolkit for Eclipse and is actually the framework which Zest is built upon. It is a more general graphical toolkit which is not centered on graphs in the same way that Zest is. It is seamlessly built upon The Standard Widget Toolkit for Eclipse and has powerful support for rendering graphics in an Eclipse View.
- Graphiti [7] is another Eclipse-based graphics framework. It has the added advantage of being adapted for the Eclipse Modeling Framework-domain[7] and can use those models in a natural way. This framework was considered in the early stages but due to the fact that it is in the incubation phase it was not chosen. Incubation phase is a phase in the Eclipse development process with the purpose of establishing a fully-functioning open-source project [8]. Due to the fact that the tool developed was to be integrated into the Arctic Studio platform the framework chosen needed to be open-source.

Zest is the framework chosen for developing the plug-in, with a couple main motivations. Firstly, the Graphviz file format ".dot" is compatible with Zest to some degree and thus it could be possible to take advantage of the Graphviz library in the future by possibly using a ".dot" file to generate a graph. Secondly, the concept of connection and nodes could be translated quite well to the AUTOSAR Software Component Template, as can be understood from chapter 3, the relation between ports and connections are of similar construction. Therefore, the fact that Zest would give those concepts for free gives it a clear advantage over Draw2D, especially considering the developing time is quite limited and implementing those concepts could take considerable time. Thirdly, as mentioned about Graphiti it is currently in the incubation phase and could thus not be integrated into the Arctic Studio platform. If this tool or a similar tool would be developed in the future Graphiti should definitely be considered as a potential visualization framework to use.

[7] EMF is a framework for building tools which use models and code generation. By having a model in XML one could generate Java classes with EMF.

4.2. MODEL DESIGN AND PARSING OF AN AUTOSAR APPLICATION

Having the view use all the information from the original configuration would be an impractical implementation. It would cause too much overhead due to all the information, which is unnecessary for visualizing the application, contained within the configuration. Therefore, the designed model contains only the information necessary for creating an accurate representation of the original application. However, certain lenience was allowed to this rule in order for the visualization part of the tool not to become too inefficient. There needed to be a certain balance where too little information would require too much runtime computation whilst too much might cause the tool to require unnecessary memory usage.

Throughout the development process, the parsing of the application is the concept which have been the most reworked and refactored of all the parts in the tool. The more knowledge acquired about AUTOSAR and the Software Component Template and how the tool would be used, the more features and parts were added. Consequently, the model had to be redone several times, in order to incorporate all the additional functions. At the beginning the model was very simple and only incorporated one level of a composition and the components, ports and connections within it. This initial model held a vital part in finding additional features to implement. It was an important part in the development process and helped make way for further improvements and new ideas. The second model had the concept of having compositions within compositions introduced into it. This enabled the user to move into the deeper levels of an application and to navigate up from a composition if desired. The second model added further requirements to the view to handle these functionalities. Finally, it culminated in a model design which held a tree-structure of all compositions and the relations between these compositions. It did this to enable the user to be able to get an overview of the structure of an application before creating the visualization and further enabling the user to choose exactly what part of the application to start viewing. The final step was an important step in making the visualization tool usable when working with larger applications.

DESIGN DESCRIPTION

Figure 4.1 shows the UML – diagram over the final model which holds all the information the view use to create its visualization. The model is centered on the concepts `Component`, `Composition`, `Port` and `Connector`. A `Component` holds references to all the `Ports` which are connected to the `Component` in order for them to be drawn as one entity by the view. A `Composition` is directly translated from the Composite Design Pattern [9] and refers to a `Composition` being a `Component` itself whilst also possibly having `Components` inside itself. This is a solution to deal with the concept of composition [10] from the AUTOSAR Software Component Template which follows the same principle.

As can be seen in the Figure 4.1 the `Port` is tightly coupled to both the `Connector` and `Component` in the sense that both pair of classes has references to each other. Tight coupling is usually discouraged in object oriented programming [11], however in this case it is a natural consequence of the fact that a `Connector` has no reason to exist without a `Port` and in the same way a `Port` without a `Component`. Furthermore, it adds optimization for navigating through an application[8].

The `Port` is a central component for navigating through the application and as a consequence holds a lot of information. Firstly, the `Port` holds a reference to `Connectors` which could be described as incoming and outgoing `Connectors` respectively. This information gives the `Port` access to two

[8] This is further described in chapter 4.9.

important methods, firstly it can locate which `Connector` is associated between itself and another `Port` and secondly it can find all the `Ports` which can be described as outgoing or incoming `Ports`. Both aspects are vital pieces for navigation and the generation of a `SubGraph`, which is explained in the chapter 4.9.

An important concept, which is worth mentioning in more detail, is that each of these classes hold a reference to a parent and the meaning of parent changes depending on the holder of the reference. A `Port` has a `Component` as a parent and a `Component` has another `Component` as a parent and as a consequence a `Composition` also has `Component` as a parent. This creates a hierarchical structure which further simplifies navigation, from any `Component` the root could be found by recursively calling its parent until reaching the root.

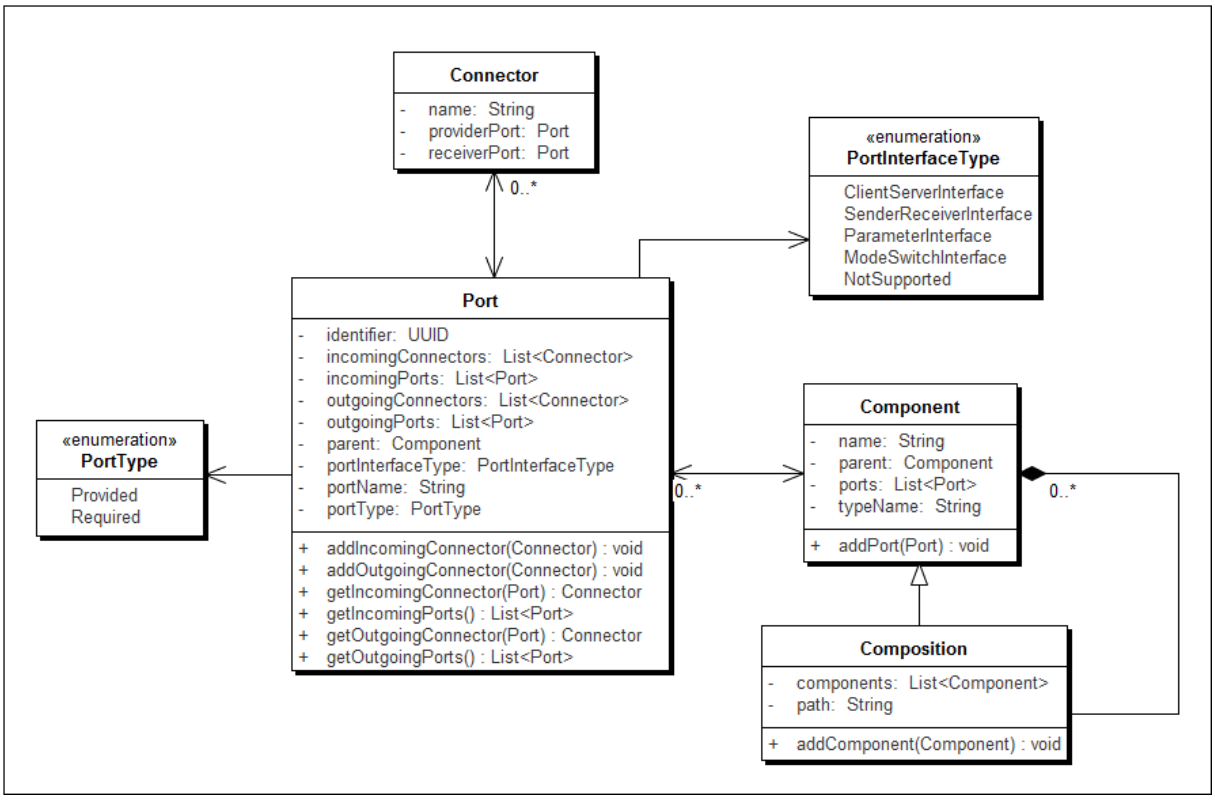

FIGURE 4.1, UML – DIAGRAM OF THE MODEL FOR THE VISUALIZATION TOOL

PARSING

Parsing the application is a rather straight forward process and is most simply described using pseudo code.

```
for each content at start level() {
    if (root) {
        parse();
    }
}

parse() {
    createComposition();
}

createComposition() {
```

```
    for each children() {
        if (composition) {
            createComposition();
            for each port() {
                createPort();
            }
        } else {
            createComponent();
        }
    }
    parseConnections on this level() {
        validate if connection is valid();
        createConnection();
    }
}

createComponent() {
    for each port() {
        createPort();
    }
}
```

Figure 4.2 shows how one could look at the algorithm as it parses the application. The slanted figures illustrates compositions with its connected components within them. Initially, the algorithm finds the top-level, the root, and begins moving through all the components found in that composition. Should a component also be a composition it recursively creates another composition. When all the ports and components for a composition have been created, the connections are parsed and created. After the last level is completed the algorithm moves upwards in the hierarchy again, finishing each composition as it moves through it.

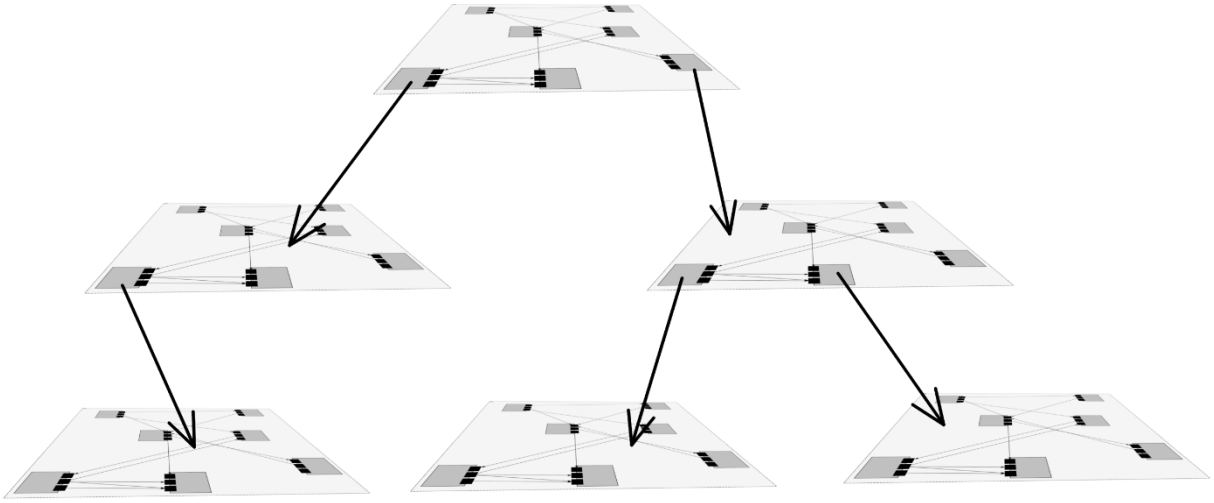

FIGURE 4.2, ILLUSTRATION OF HOW THE PARSING ALGORITHM TRAVERSE THE APPLICATION.

There are of course a lot of details omitted but the basic structure is as described above. During the parsing phase the application is also checked for simple errors which can occur when designing a configuration, such as ports connected which exist in different compositions. This information is further stored and used to inform the user about possible design flaws. Using this method and the model in Figure 4.1, fair results were achieved. When parsing an application which contains around

550000 objects (including all connections and ports) the time it took to create all relevant parts and validate their correctness was 1.4 seconds[9].

4.3. THE VIEW – MAINGRAPH AND SUBGRAPH

At the initial stages of development there were only the concept of one graph. The main focus had been one important part, to visualize the whole application and only creating one graphical representation of that application. Therefore, one graph was created to show the application, which hereinafter will be called `MainGraph`. The `MainGraph` generated one `Composition` with its `Components`, `Ports` and `Connectors`. However, as the project progressed and input was received from some of the potential users of this tool a decision was made to introduce the concept of a second graph, which hereinafter will be called `SubGraph`. The `SubGraph` would be a simpler graph than the `MainGraph`, showing only `Ports` connected to each other. More specifically, it shows one connection and how it traverses the application as it moves between `Ports`. The architectural design of this is shown in the UML – diagram in Figure 4.3. The `MainGraph` and `SubGraph` are instantiated and initialized by the `GraphController` which both serves as a controller and delegator of all the actions the user performs on the graphs. It is registered as an observer [12] of both graphs and reacts to the user interactions and informs each graph of any change which is needed to make to their respective states.

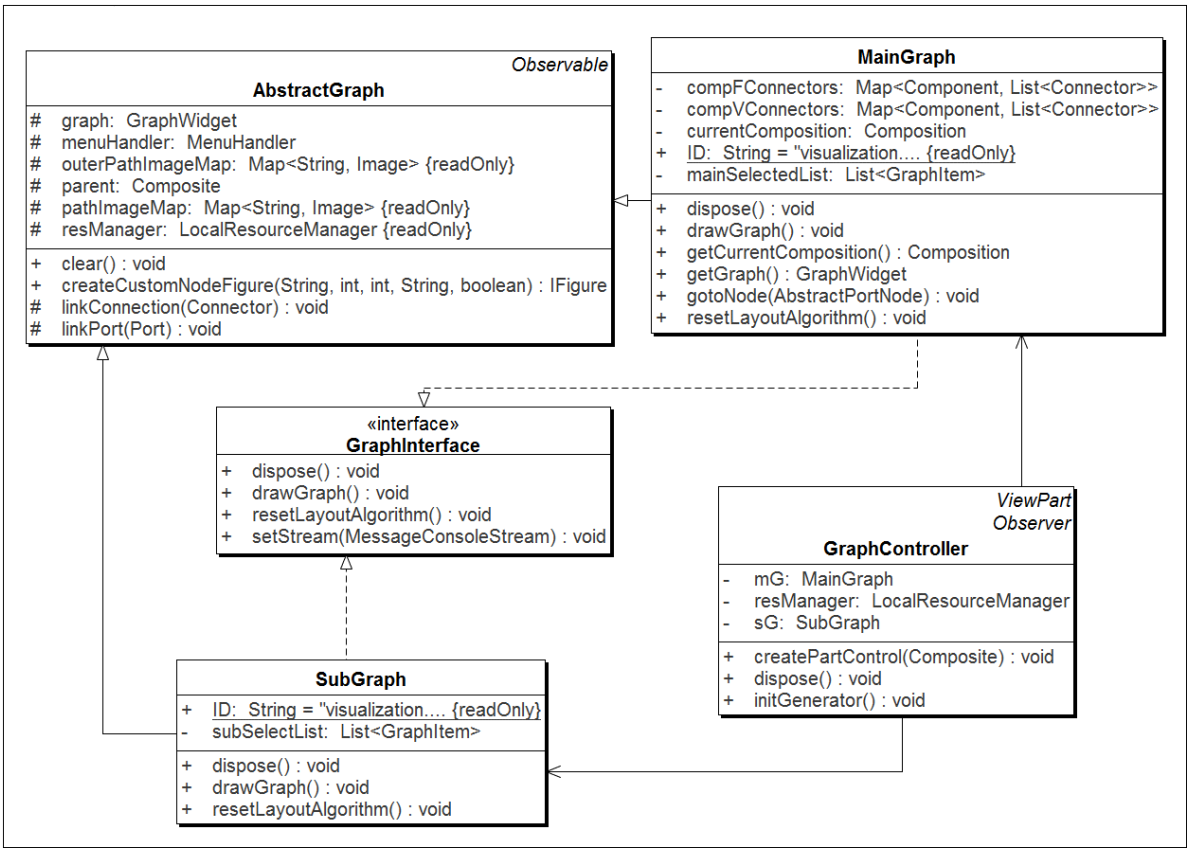

FIGURE 4.3, UML – DIAGRAM OF THE ARCHITECTURAL DESIGN OF THE VIEW

When designing the view for the plug-in there were two possible ways of implementing it, either make it reactive towards the configuration of the application or have it change when triggered by the user. The first option, making it reactive, is a very attractive concept. It would make the view change automatically when a change is done on the application. This would make for a seemingly seamless

[9] Test was run on a laptop with 10GB ram and a i7-3517 CPU @ 1.9GHz.

user experience where all the user would have to do is to look at the view to see how the change has affected the application design. When designing the tool the decision was made to not make the view reactive, it would only update when prompted by the user. The reason for this was twofold, making it reactive could possibly put an unreasonable load on the Arctic Studio platform and make the software become slow. The AUTOSAR Software Component Template puts no limitation in how big an application can become. As a consequence the visualization of such an application could potentially become time consuming. By putting the decision in the hands on the user when the application should be parsed, the user would be prepared for the potential wait and frustration could be minimized. Secondly, building a reactive view would be time consuming in itself and this would take development time from other more important features in the plug-in. However, it is entirely possible by making additions to the current tool to make the view reactive. Looking at the two reasons why it is currently not reactive it can be understood that it is possible with some limitations. The primary limitation being that the application it is used on is not too large lest it risk cluttering the performance of the Arctic Studio platform. Lastly, the second limitation is only a matter of time being available to implement it.

4.4. USER INTERFACE – DESIGN

When desingning the user interface several factors were taken into account, it needed to be uncluttered and simple to use. Furthermore, it should be kept in line with other design standards already in place by Eclipse. This chapter describes the resulting design of the UI and the functionality which comes with it. First and foremost an overview of the UI can be seen in Figure 4.4. To the left the `MainGraph`[10] can be seen, which shows the current `Composition` the user is viewing. The right section displays the current signal the user has chosen to generate in the `SubGraph`. Number 1 shows a `Component` which is not a `Composition`. A `Composition` is visually differentiated from other `Components` and can be entered into by double – clicking it. Number 2 shows a `Connector` and number 3 shows a `Port` which recieves a `Connector`. In this instance a receiving `Port` has been marked, depending on the type of port parsed the graphics it takes will change. Number 4 shows all the options the user has available which are general settings for the `MainGraph`, these options are also made available by right – clicking the white area of the `MainGraph` as shown by number 5. Depending on what area or part the user right – clicks on, a different menu with options are shown, one for the whole graph, one for a `Component` and a different one for `Port`. As a consequence of the fact that the details is difficult to disern in the overview these will be demonstrated in Figure 4.5, Figure 4.6, and Figure 4.7 respectively.

[10] Due to the model parsed being classified to a certain degree all the names for the component, port and connections have been filtered out.

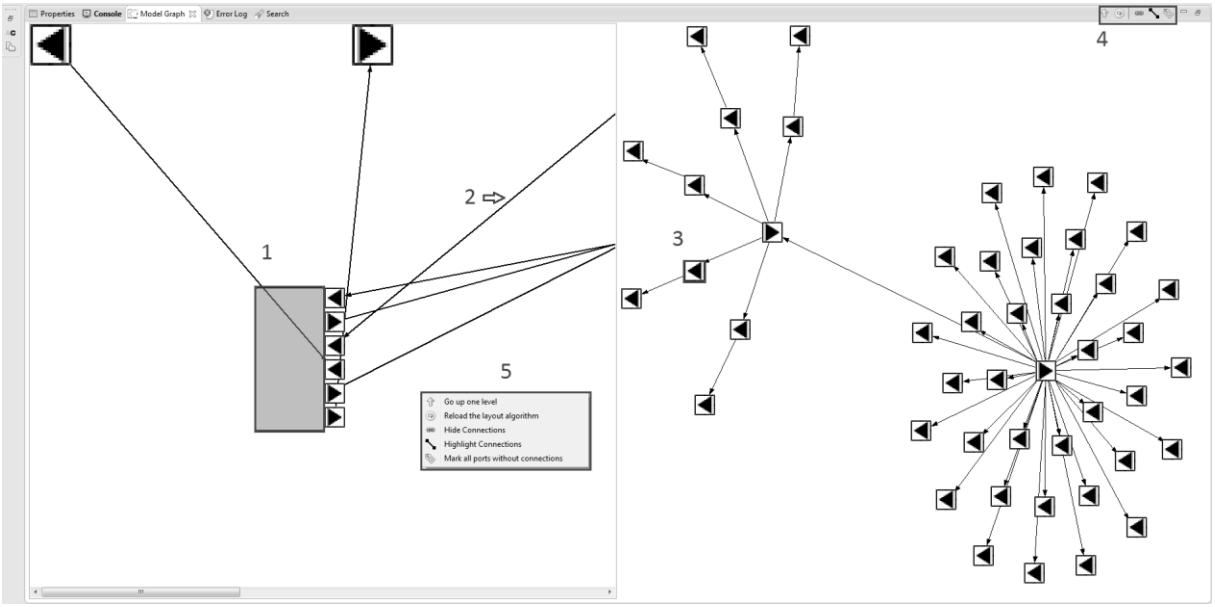

FIGURE 4.4, OVERVIEW OVER THE DESIGN OF THE USER INTERFACE

Figure 4.5 shows an enlarged version of the `MainGraph`s option menu (number 5 from Figure 4.4). "Go up one level" generates a new graph of the `Composition` which the currently displayed `Composition` resides within, if such a `Composition` is available. "Reload the layout algorithm", "hiding connections" and "highlighting connections" does exactly as they are named. "Mark all ports without connections" simply marks all `Ports` which has no connections connected to it. It is an option which could be used when looking for errors in an application design where it could be interesting to discern which ports are lacking connections.

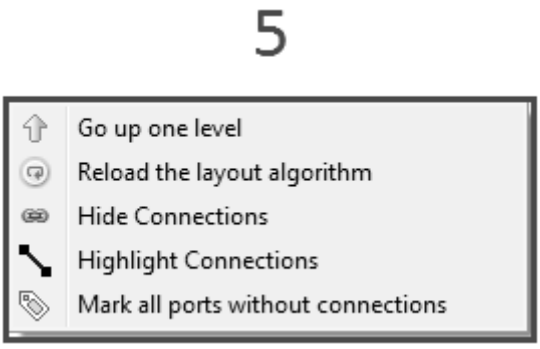

FIGURE 4.5, THE MAINGRAPHS MAIN OPTION MENU

Figure 4.6 shows the menu for a `Component`, it gives the user the additional option of "hid[ing] all connections except this component". Furthermore, the option "Jump to connected ports" are available, this is further explained in the chapter 4.9. All the settings on the `Component` are meant as a general setting for the `Component` and as a consequence are also applied to all the `Ports` connected to that `Component`.

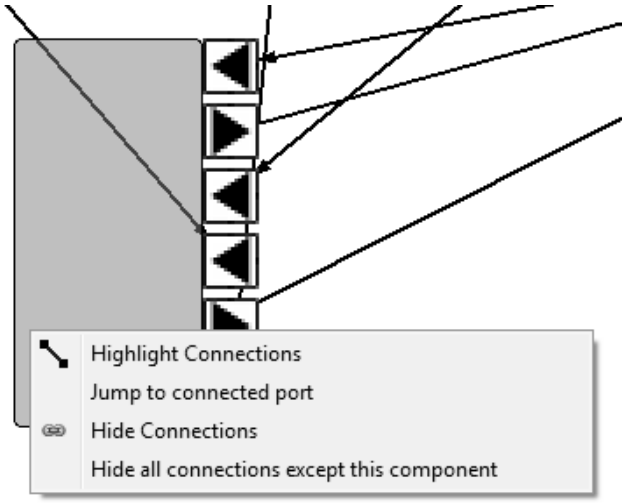

FIGURE 4.6, OPTIONS AVAILABLE TO THE USER WHEN RIGHT – CLICKING AN COMPONENT

Figure 4.7 shows the menu for a `Port`, it has the same options as a `Component` with the further addition of generating the `SubGraph`, either of outgoing connections or incoming connections. These are further explained in the chapter 4.9.

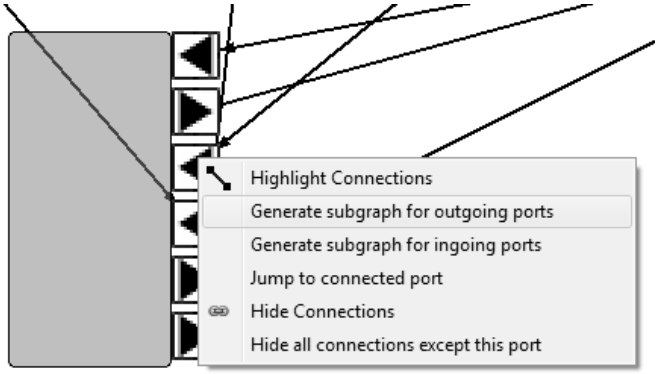

FIGURE 4.7, OPTIONS AVAILABLE TO THE USER WHEN RIGHT – CLICKING AN COMPONENT

4.5. USER INPUT – THE HANDLERS AND LISTENERS

There were a large amount of features and functionalities to add to both the `MainGraph` and the `SubGraph` in order to make the tool usable. As a consequence, the tool needed to be able to handle all relevant user input. This was done by designing Listeners [4] for all the keyboard commands and mouse maneuvers the user should be able to do. Furthermore, since the tool has a fair amount of these Listeners, handlers where designed to administer over them. Figure 4.8 shows the architectural design of all the Actions [4], Listeners and Handlers in the tool.

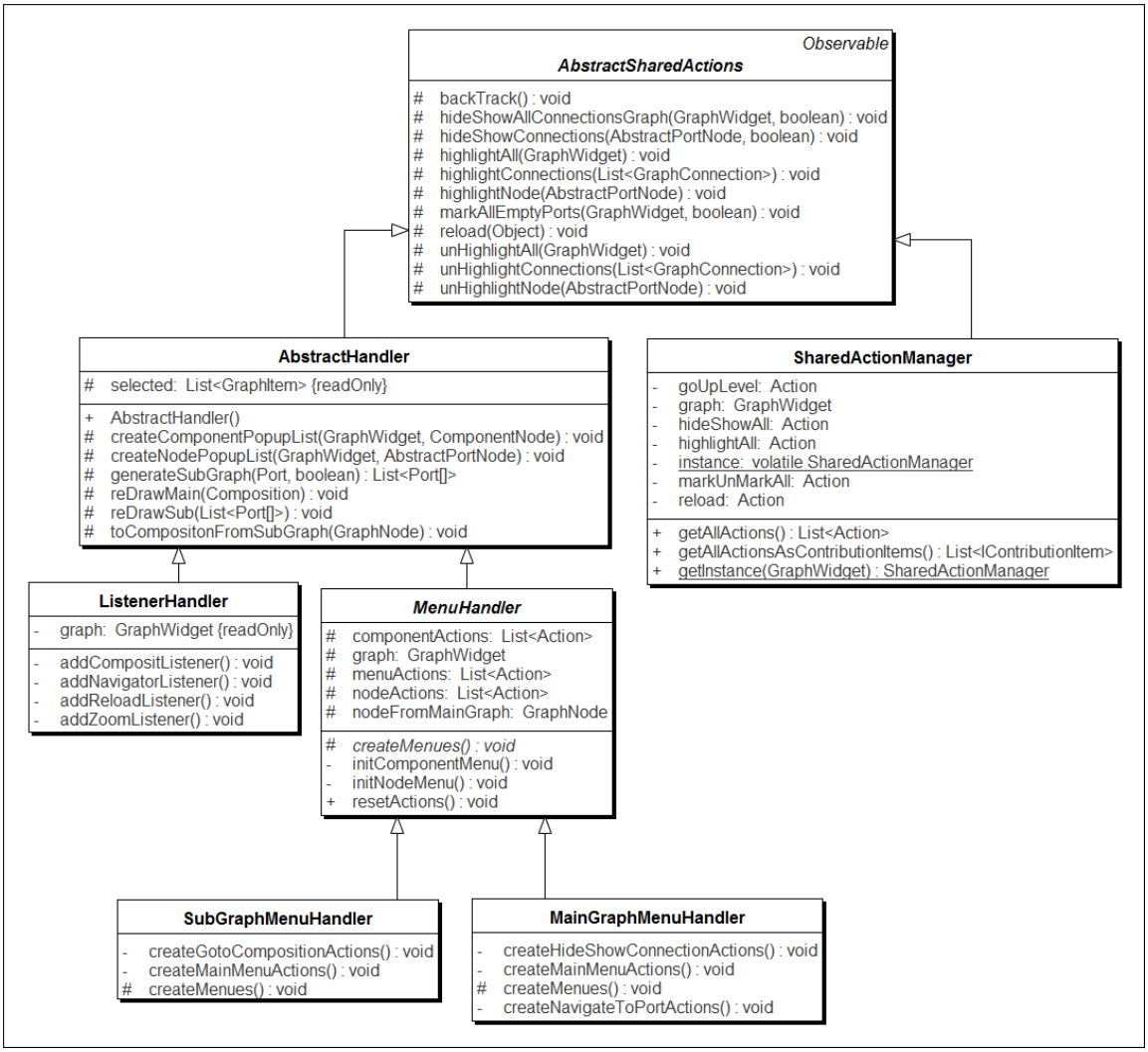

FIGURE 4.8, UML – DIAGRAM OF THE ARCHITECTURAL DESIGN OF THE ACTIONS AND MENUHANDLERS

4.6. SELECTING ITEMS – REINVENTING THE WHEEL

In the Zest framework the concept of moving one node, moving groups of nodes, selecting nodes and zooming is already handled. Initially, it seemed that letting the framework handle it with the methods already implemented was more than fine, even advantageous. The only down-side to this was that Zest has some additional key-commands implemented which is not necessary for this tool, but this was a small problem at the time. However, the larger the tool grew it became more and more apparent that with all the extra commands and the menu-system added the user interface became cluttered and it became necessary to remove the Listener which handled all the user input for Zest. A side effect of this was losing everything which was needed for moving nodes, zooming and selecting nodes and a new Listener to take care of that became a requirement. Subsequently, the `SelectItemListener` were designed. It holds several responsibilities including keeping track of what is selected, moving selected items and separating the types of what is selected[11]. The user can also pin a `Component` which has the effect of keeping it frozen in its position when the layout algorithm is run. Figure 4.9 shows how the architecture around the Listener is built up. The `DragGroupListener` is explained in detail in the next chapter.

[11] To keep the user from being able to select a connection at the same time as a port is selected, which is necessary for the correct menu to be shown when right – clicking.

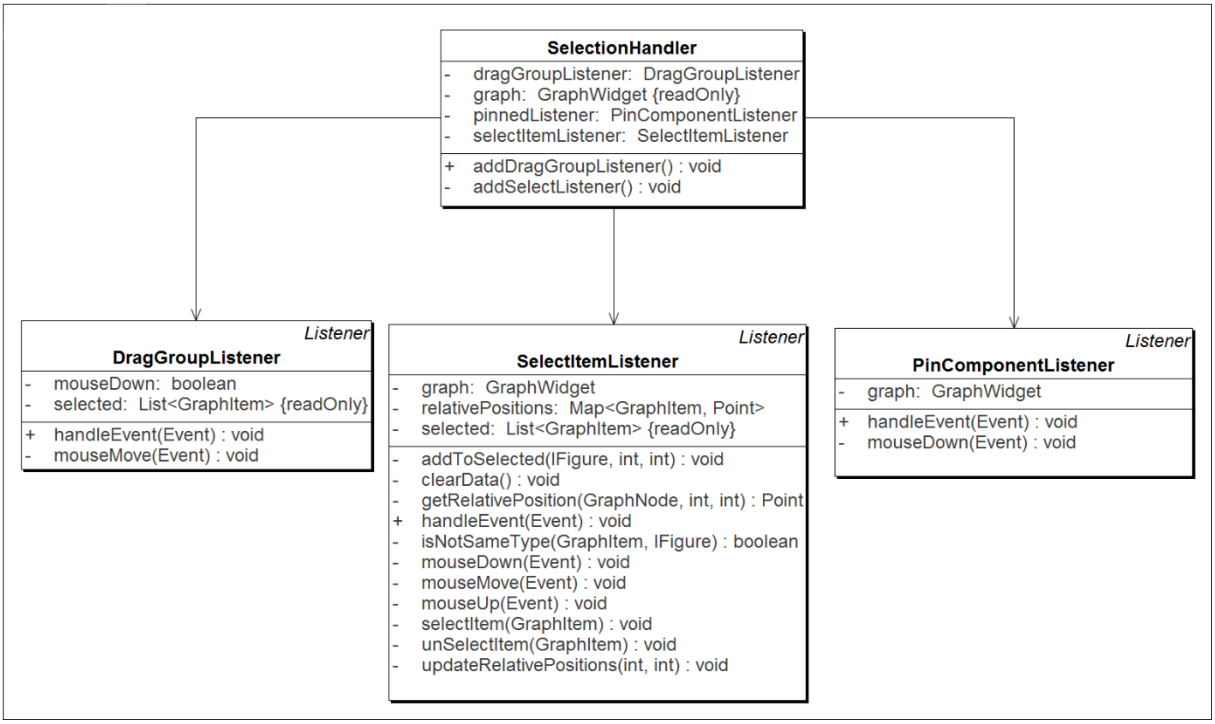

FIGURE 4.9, UML – DIAGRAM OF THE ARCHITECTURAL DESIGN OF THE SELECTION HANDLER AND LISTENERS

4.7. INTRODUCING THE CONCEPT OF PORTS TO ZEST

The AUTOSAR Software Component Template is centered on the concept of a component having ports, which can have a variety of functions. It can hold several different kinds of ports and these can be connected in the application in any manner of complexity. Consequently, it holds a very important part when looking at the application. Zest, on the other hand, has no concept of ports connected to components, it is a visualization tool which centers on graph drawing algorithms to perform its task. It has nodes and connections and there is one type of node and one type connection. Nodes are not attached to each other in any other way than by connections. Thus, there was a need to bridge this gap between the two concepts in order to make this tool valid for the AUTOSAR Software Component Template. In order to draw connections to each `Port` the `Ports` still have to be nodes in the Zest environment. Therefore, when adding a `Port` to the graph in Zest it would still be a node and handled as such. As a result, it would be rendered separately from the `Component` and logically handled as an individual part of the graph. For example, if a user would attempt to move a `Port` it would be moved away from the `Component` it is supposedly attached to. As explained above, this is not the desired result. In order to handle this problem, the concept from AUTOSAR where ports are a part of a component, was kept intact when parsing the application and a `Component` was manually grouped to its `Port`[12]. Furthermore, each `Port` would have to remember its relative position to the `Component` in order to be positioned correctly by whatever layout algorithm is being applied to the graph. This implementation puts a requirement on the tool to take into account that each `Component` possibly has `Ports` and handle the `Ports` in an appropriate way.

In addition to redesigning the Listener, which moves nodes, the concept of `Ports` being attached to a `Component` needed to be taken into account. A Mouse Listener, the `DragGroupListener` was designed which handles the new concept of `Ports`. Whenever a user would attempt to move any node in the graph the Listener will figure out which group the node is a part of and as a user moves

[12] As explained in the "Design description" – section of chapter 4.2.

that node the Listener will take care of every node in the group and make them move in unison as one. If a user tries to move a `Port` the Mouse Listener would look at its parent, which would be a `Component` and iteratively move all its `Ports`. If a user tries to move a `Component` the Listener would simply move all `Ports` iteratively.

4.8. LAYOUT ALGORITHM

In Zest, there are several powerful layout algorithms already implemented. These can greatly help the user get an overview of the graph and improve the understanding of it. However, since all the layout algorithms implemented in Zest does not take into account the newly introduced concept of `Ports`, they were all rendered useless. Layout algorithms in Zest does not group nodes together as one entity, which is a requirement of this visualization tool due to the AUTOSAR Software Component Template. Zest is a visualization tool which uses graphs, a graph is the concept of nodes and edges connected in any manner of way and a `Port` which is a part of a `Component` is a foreign concept to a graph. If any of the layout algorithms supplied by Zest were to be used the result would be undesirable. Figure 4.10 illustrates the consequence of using a radial layout algorithm integrated into Zest. As can be seen, the result is unusable.

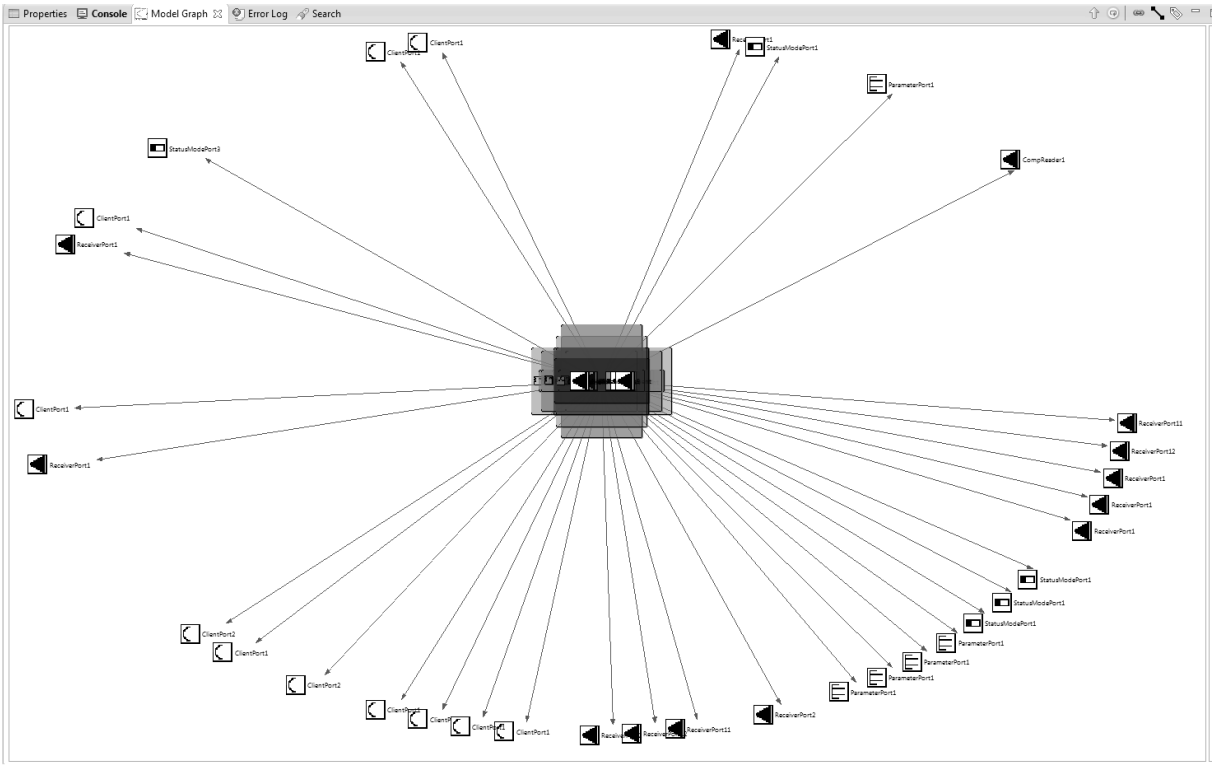

FIGURE 4.10, RESULTING LAYOUT WHEN USING ZEST'S RADIAL LAYOUT ALGORITHM

During the development of the tool several attempts were made to circumvent this problem and make use of the algorithms available but without much success. The first attempt was to take advantage of the concept of `GraphContainer`, which is available in Zest. It is a container which stores a graph or sub graph[13] inside itself and Zest treats it as one node. This quickly gave promising results but there were two downsides. The support for `GraphContainer` is limited in Zest and the behavior of the tool could become quite unreliable. Figure 4.11 shows an illustration of the result achieved when using `GraphContainers`. Firstly, the `Ports` is not visually stuck to the

[13] Not the same as `SubGraph` introduced to the tool but rather the natural concept of sub graph in graph theory.

Component but rather connected as nodes, this did not give the desired sense that Ports are actually a part of a Component. Secondly, the GraphContainers visual representation could not be customized, there was no ability to attach an icon or image to discern one type of a Component from another. Finally, number 1 in the figure demonstrates one of the unreliable behaviors, connections would at times be drawn under the container when entering it and then reemerge when reaching its destination.

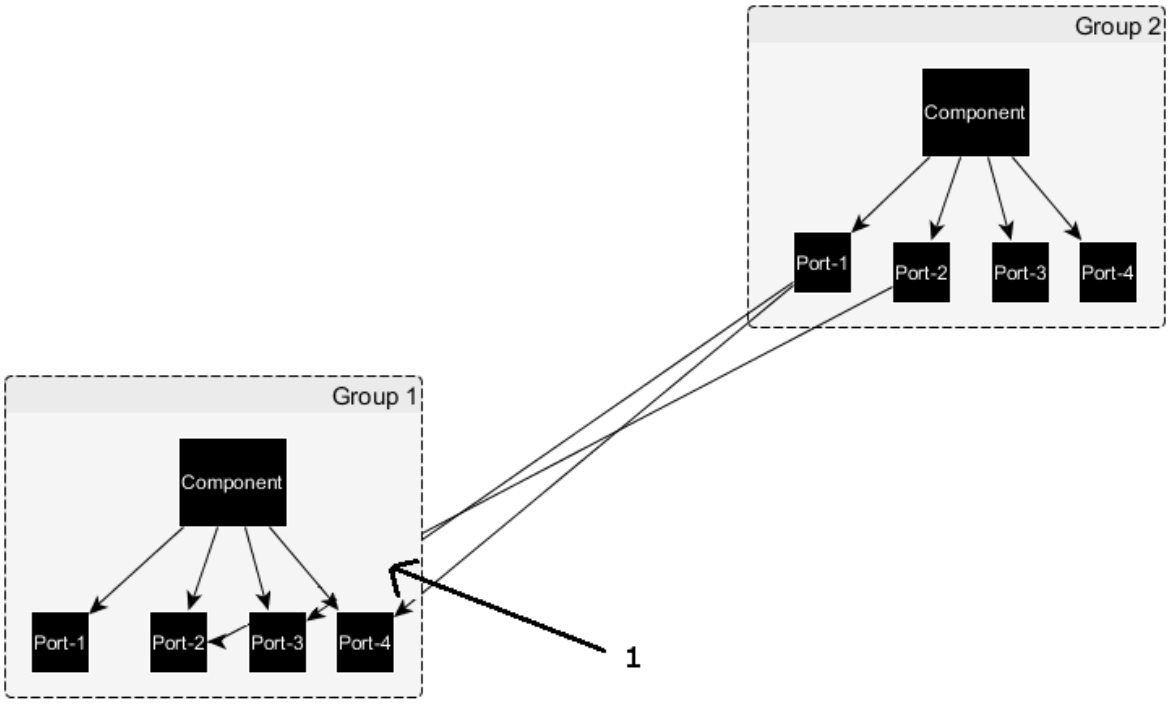

FIGURE 4.11, EXAMPLE OF HOW AN IMPLEMENTATION COULD LOOK USING GRAPHCONTAINERS IN ZEST

A second approach which was considered was to remove the concept of Ports entirely and simply create the Components and visualize the connections between them. This would certainly make it much simpler to design and visualize. Furthermore, the algorithms from Zest would be made available. However, this could very easily create an even more cluttered representation and there would be no way to discern one connection from another in larger application with components which have many incoming connections. Figure 4.12 attempts to demonstrate this problem, if each connection would instead have its own Port it would be much easier to discern where it is coming from. An even more obvious downside would be to lose the information of which Port the connection is coming from and going to, respectively.

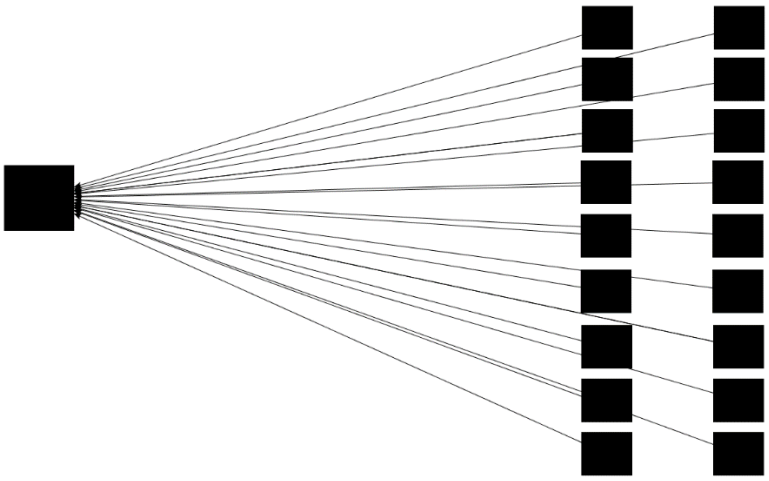

FIGURE 4.12, DEMONSTRATION OF USING NO PORTS WHEN ILLUSTRATING CONNECTIONS

Finally, it was decided to not use the algorithms from Zest in order to have the concept of ports intact from the AUTOSAR Software Component Template. However, without any type of layout algorithm the `Components` and `Ports` would simply be placed on top of each other as they are created without any structure at all, rendering the generated view useless. Consequently, there was a need to develop a new algorithm which encompasses the new `Port` design. A layout algorithm can be varying degrees of advanced, from simply placing the nodes onto the layout to creating a more advanced one, using for example a force directed algorithm [13][14]. The former was chosen for this visualization tool, due the focus when developing the plug-in and the complexity of implementing the latter.

A simple grid based layout algorithm was implemented for the visualization tool. It is a two-step algorithm which first picks out all the `Components` from all the nodes connected to the graph (so as not to move any `Ports` inadvertently) and marks them as nodes to move. During this first iteration, the algorithm also looks to see if there are any `Components` which either have no `Ports` or lack `Connectors` to the `Ports` on the `Component` and marks these to be placed last, to make the general overview of the graph a bit simpler. In the second iteration, the algorithm takes the information from the first iteration and, using a matrix array, it creates a virtual grid of the view. The algorithm will then place all the nodes into the grid. As it works its way through the application it places a node in the next available position. Finally, depending on how many `Ports` a `Component` has its size will vary. The algorithm compensates for this difference in size by checking the size and location of the node already placed straight above it, if there is one, and places the next node accordingly. The result from using this layout algorithm is shown in Figure 4.13.

[14] There are numerous other ways of creating a layout algorithms. However, this is not covered in this report.

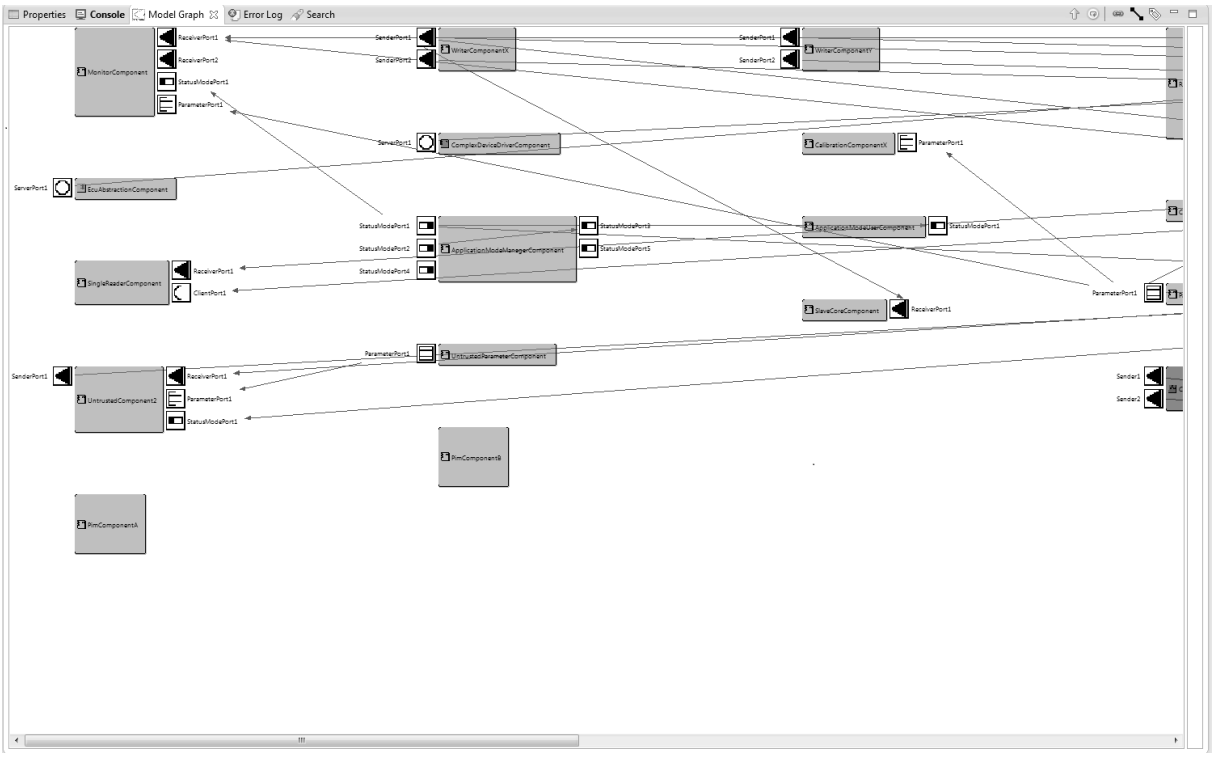

FIGURE 4.13, RESULTING LAYOUT WHEN USING THE CREATED GRID LAYOUT ALGORITHM

4.9. NAVIGATION

As mentioned in chapter 4.8, a simpler approach were taken for the layout algorithm. This decision was made mainly due to the focus and goal when developing the plug-in. The main purpose was to create a visualization tool and the ambition was to have the usability be as high as possible. The complexity of an application created using AUTOSAR is only limited by resources and imagination. To cope with this complexity there are two approaches available. The first is to create a highly advanced layout algorithm to give the user a good overview of the application design. However, due to the fact that an application could very well take any shape possible the algorithm would have to take this into account. Undoubtedly, the complexity of such an algorithm is enormous. The second approach would be to give the user navigational abilities in order to efficiently be able to move through the application. This would give the user the power to find the information in the application in a manner suitable for their specific case. This has the advantage of being more dynamic in view of the fact that it also puts a larger focus on how the user chooses to use the navigational features. Therefore, a greater focus has been put on creating navigational abilities for the visualization tool. To combat the possibility of having an enormous application three concepts have been introduced: choosing which `Composition` to initially enter, moving to connected ports and generating a `SubGraph` of outgoing or ingoing ports.

Due the concept of compositions an application can contain levels upon levels of compositions and easily grow in complexity. As a user it would be beneficial to be able to enter into the desired composition without having to navigate to it from the root. When parsing an application the algorithm looks at the whole application design and structures it as a tree-structure to enable navigating through the application after parsing it. As a consequence, a reference is held to every composition in the application and this further enables the first navigation concept of entering into a

Composition. This is made available by a dialog, which presents the user with all the available Compositions as can be seen in Figure 4.14 directly after the parsing is finished.

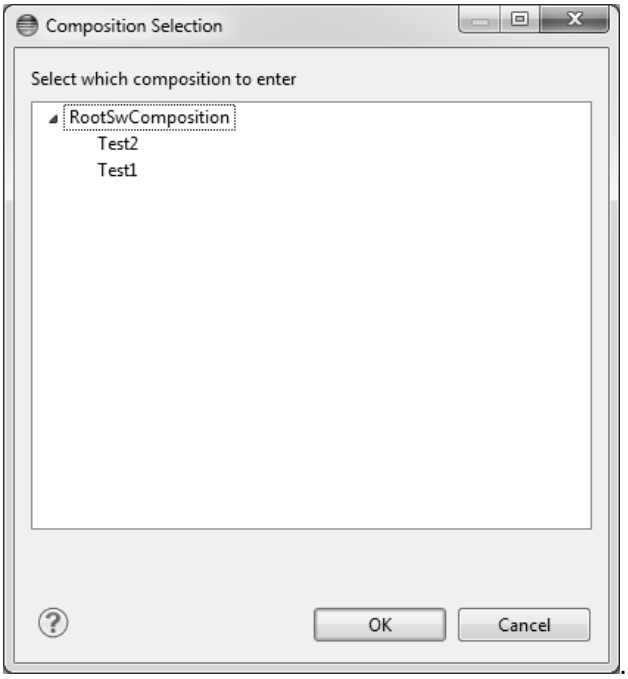

FIGURE 4.14, TREE-STRUCTURED DIALOG OF WHICH COMPOSITION THE USER CAN CHOOSE FROM

After a choice has been made the tool generates the graph which represents the Composition the user wishes to view. As mentioned in chapter 4.2, each Composition knows which Composition its parent is, as a consequence there is no requirement of the user to have navigated down a level to be able to go up a level. When prompted to "go up a level" the tool simple finds the parent and generates a new graph representing the parent.

If a graph is large with many connections present, seeing the correlation between Ports could prove difficult. Conceivably, being able to move to a connected Port would certainly relieve some of the frustration which is associated with trying to find a connected Port. In order to efficiently be able to find all the connected Ports, each Ports holds a reference to all the ingoing and outgoing connections. By looking at each connection and finding each Port connected to itself and using the location data stored in each of those Ports, one can move the user to that position. When a user shows intent to move to a connected Port a popup-list is created of all the connected Ports and when the user selects a Port the viewport is moved to the relevant position.

OUTGOING AND INGOING CONNECTIONS - SUBGRAPH

Using the same concept of ingoing and outgoing Ports and recursively traversing the underlying path which connects these Ports, it is possible to find how each connection moves through the application. One connection is only between two Ports, but from each Port another connection can branch further. It is rather a way of looking at the connections as signals which moves through the components. From this principle the tool has the ability to create a SubGraph with a couple of new concepts. This SubGraph not only gives the user the ability to see how a connection moves in a Composition, but also how it moves through the whole application. The SubGraph overrides the limitation of only viewing one Composition at a time and creates a graph which can exist on multiple levels of the application.

An additional functionality is added to the `SubGraph` in order to give the user further navigational abilities. Each node in the `SubGraph` holds a reference to the `Composition` it exists within, because of this the user can simply use the `SubGraph` to navigate by generating the desired `Composition` in the `MainGraph`. Imagine a large application with hundreds of compositions, thousands of ports and almost as many connections. Trying to visualize such an application in one's head is impossible, but even with a tool such as this it can be difficult to understand how a connection moves through the application and actually finding where the connection ends even more difficult. The `SubGraph` can to some degree help with this, Figure 4.15 and Figure 4.16 illustrates in what way. In the right window in Figure 4.15 a connection is shown, the first layer is the `Port` which the `SubGraph` is generated from. The second layer is the connected `Ports` in this `Composition`, if the signal moves through these `Ports`, as it does in this case, it means the connection is moving to another `Composition`. Hence, the `Port` which is currently right-clicked exists in another `Composition`, by using the menu option "Go to the composition this node resides in" the `MainGraph` is re-generated. After re-generation it will instead the desired `Composition` as illustrated in Figure 4.16. If there are no more `Ports` it means the signal ends there.

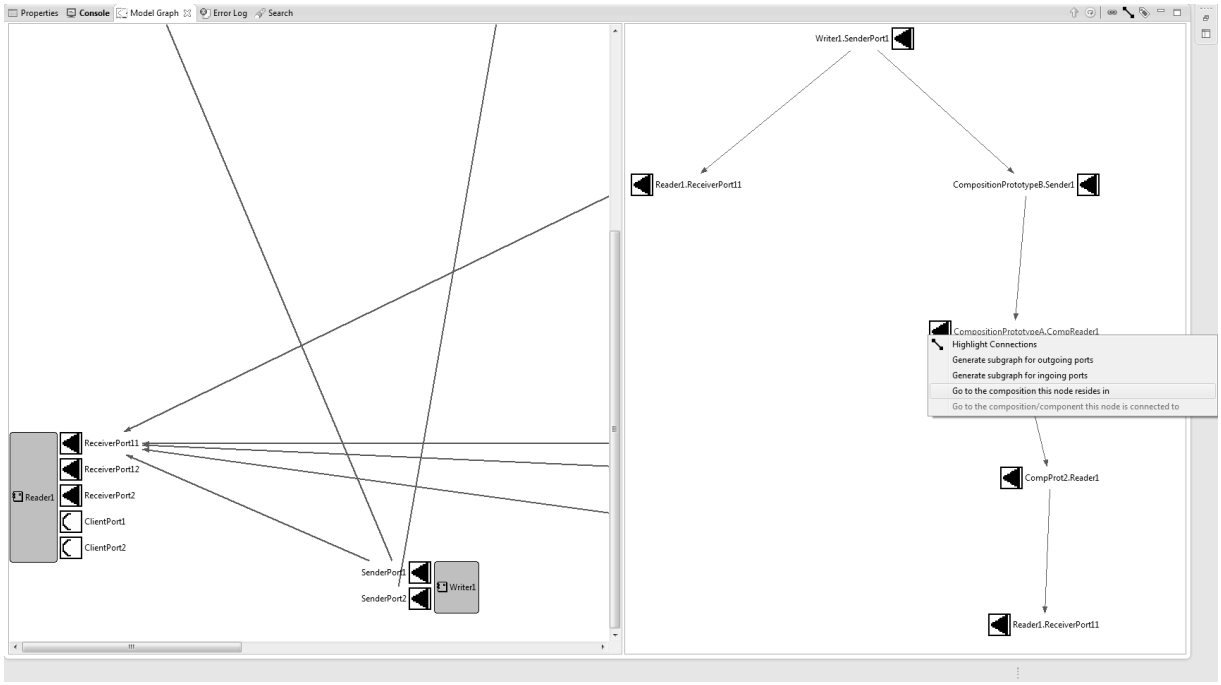

FIGURE 4.15, BEFORE MOVING INTO ANOTHER COMPOSITION USING THE SUBGRAPH

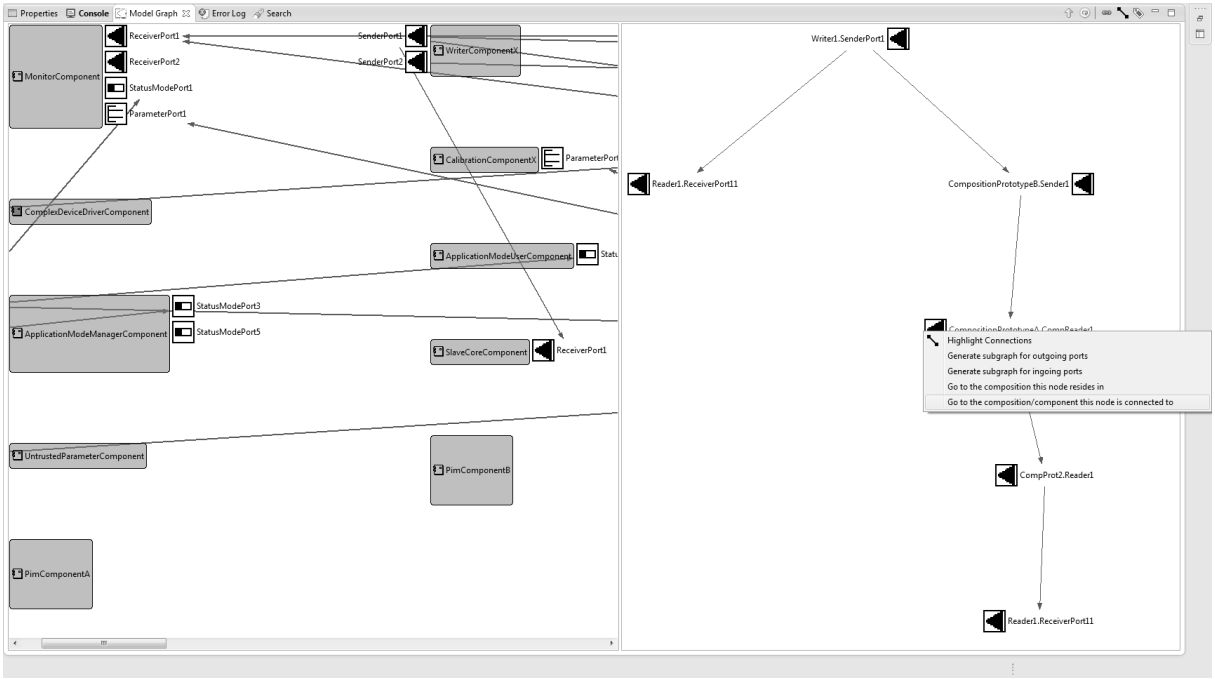

FIGURE 4.16, RESULT AFTER MOVING TO ANOTHER COMPOSITION, A NEW MAINGRAPH HAS BEEN GENERATED

Furthermore, when the user has navigated to the wanted `Composition` they can use the `SubGraph` to position the viewport [14] to show the desired `Port`. It gives the user a quick way to find a `Port` and position the `MainGraph` to show that `Port` in its `Composition`.

USING ZEST'S LAYOUT ALGORITHMS

Due to the fact that the `SubGraph` has no need to hold the concept of `Components` alive the `SubGraph` can render the `Ports` as single nodes with connections to them. This opens up the possibility to use the powerful layout algorithms which can be found in the Zest framework. The `SubGraph` takes advantage of these algorithms by giving the user the possibility to choose which layout to use at this time. Figure 4.17 demonstrates the four different algorithms currently available. The top left shows the "Grid Layout Algorithm", the top right the "Spring Layout Algorithm". The bottom left shows the "Radial Layout Algorithm" and finally the bottom right shows the "Tree Layout Algorithm". Depending on the input into the `SubGraph` and the information the user wish to obtain the choice of algorithm will vary.

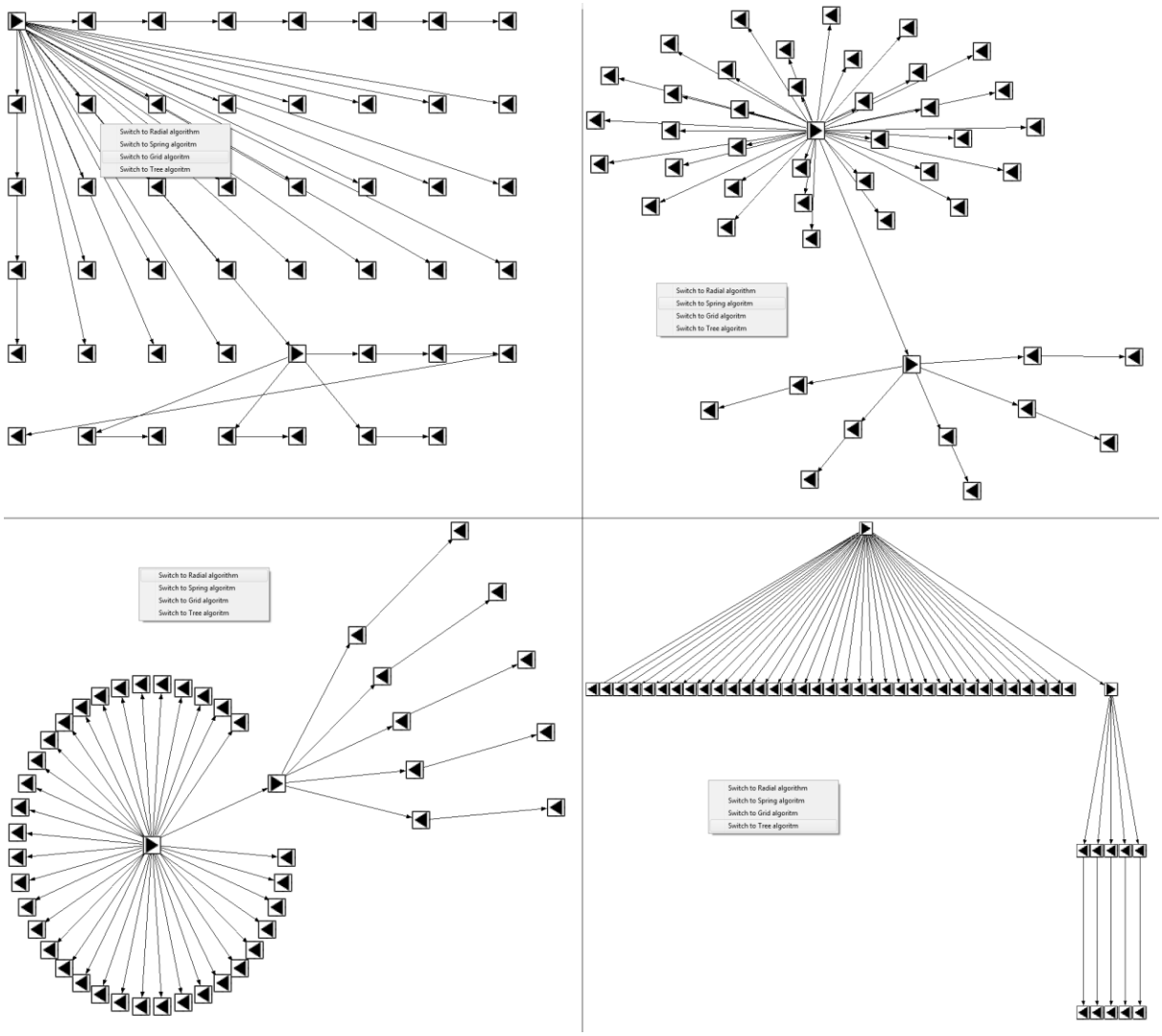

FIGURE 4.17, DEMONSTRATION OF THE LAYOUT ALGORITHMS USED IN THE SUBGRAPH.

5. ETHICAL ASPECT, HUMANITIES

This report centers on the development of a tool which will be directly used by engineers and software developers in their everyday work. Subsequently, it is not a stretch to say it could affect their everyday work aspects. In what way this tool will affect will only be speculations and only the actual use of it will tell if these speculations are correct or not.

The ambition of this tool is to be of use to these developers, it is meant to unload their everyday work and during the whole development process, from design to implementation, their opinions on how it could be used when they work with the Arctic Studio platform were taken into account. It is the belief that it could help speed up the process when looking for errors in the design of the application, such as ports which are not connected or possibly connected in a faulty way. Furthermore, it gives the user a good way to quickly get an overview of how the application is connected. If it can be successfully used in this way it is possible that it would help speed up the work and make it more efficient. As a result, it could reduce the cost for anyone working with the Arctic Studio platform.

6. RESULT

The main purpose was to create a visualization tool which could parse a configuration from the AUTOSAR Software Component Template and present it visually in an Eclipse view, furthermore it should be integrated into the Arctic Studio platform. Without a doubt, both these goals have been achieved, the created plug-in can more than accomplish these tasks. Using a configuration as a trigger point the user can ask the GUI in the Arctic Studio platform to create a visualization for them. The parsing is done and accomplished with good results, it is quick and possible underlying problems with the application design is retained. To increase usability the user is presented with options of choosing which composition to start viewing. Furthermore, the different components and ports are visually differentiated to further increase usability and understanding of the application which have been visualized. Furthermore, the user has several possibilities of how to navigate through the visualized application. The user can either focus on one component and its connections or move through the composition by following a ports connection. Finally, by generating a graph of a signal as it propagates through the application, the user can be aided when trying to follow a signal.

A secondary goal was to realize the tool in such a way that it could easily be extendable to include other parts of AUTOSAR. Due to the concept of MVC being used this has also been accomplished, when parsing the application and creating the model the whole application has to be parsed. However, the model only uses a small part of the parsed application and during the parsing phase it is possible to retain more parts of the application. Consequently, it would be fairly simple to create new models or expand the current model. With some adaptations to the view it should then be possible to display these. The complexity is dependent on what extra features is necessary to encompass for the changes to the model and how the new parts should be presented graphically.

CONCLUSIONS AND DISCUSSION

There are certainly things which could have been done differently. Indubitably, it would have been advantageous if the plug-in had been served with a more advanced layout algorithm. This would give the user a good overview of applications from the first moment when looking at the generated visualization of an application. Currently, the algorithm is very primitive and this leaves it up to the user to find its way to the relevant information, which could be difficult with a bigger application. If a way could be found to circumvent the issue with the `MainGraph` of not being able to use the layout algorithms from Zest, the problem would solve itself. Some consideration for the future could be made to attempt to incorporate the concept of ports into Zest in a fluid way. However, to minimize the problem with this limitation the plug-in hosts several navigational abilities which helps the user move around in the application with relative ease. The navigational abilities is certainly a feature which could help get a good understanding of how the underlying application is connected and possibly even find errors in it. In particular, the `SubGraph` is a great addition to the visualization tool. To be able to follow a connection as it moves through the application, viewing it independently of which level resides, gives the user the ability to quickly find a relevant connection point.

The Zest framework has been a great help when creating this tool. There were several features which initially enhanced the development process remarkably, helping to get an early version of the visualization tool working. Working with that early version as a starting point many features and functionalities could be figured out and developed. Without Zest there is no doubt the development process would initially have taken longer. However, there were also several complications due to using Zest. In particular, the overhead to the user controls caused problems later on, introducing a need to remove the mouse listener and create a new one with the needed functionality. In retrospect, it might have been preferable to use Zest as a startup framework and when a

comfortable early version had been created, remove the Zest dependability's and perhaps use Draw2D instead and create the concept of nodes and connections there. Doing so might have opened up further possibilities to create another framework where the concept of components with ports connected to them could be integrated in a more natural way. Certainly, such a functionality would simplify adding any feature involving manipulating or handling the components.

Bibliography

[1] AUTOSAR, "AUTomotive Open System ARchitecture," 2013. [Online]. Available: http://autosar.org/. [Accessed 27 03 2013].

[2] T. Stober and H. Uwe, Agile Software Development: Best Practices for Large Software Development Projects, Manhattan: Springer, 2010.

[3] "Graphviz - Graph Visualization Software," Open Source, [Online]. Available: http://www.graphviz.org/. [Accessed 15 04 2014].

[4] V. Silva, Practical Eclipse Rich Client Platform Projects, New York: Apress, 2009.

[5] The Eclipse Foundation, "Eclipse - Zest," The Eclipse Foundation, 2014. [Online]. Available: http://www.eclipse.org/gef/zest/index.php. [Accessed 06 04 2014].

[6] The Eclipse Foundation, "Eclipse - Draw2D," The Eclipse Foundation, 2014. [Online]. Available: http://www.eclipse.org/gef/draw2d/. [Accessed 10 05 2014].

[7] The Eclipse Foundation, "Eclipse - Graphiti," The Eclipse Foundation, 2014. [Online]. Available: http://www.eclipse.org/graphiti/. [Accessed 29 05 2014].

[8] The Eclipse Foundation, "Development Resources/HOWTO/Incubation Phase," The Eclipse Foundation, 15 05 2013. [Online]. Available: http://wiki.eclipse.org/Development_Resources/HOWTO/Incubation_Phase. [Accessed 29 05 2014].

[9] E. Gamma, Design patterns : elements of reusable object-oriented software, Boston: Addison-Wesley, 1995.

[10] AUTOSAR, "AUTOSAR," 31 03 2014. [Online]. Available: http://autosar.org/download/R4.1/AUTOSAR_TPS_SoftwareComponentTemplate.pdf. [Accessed 24 04 2014].

[11] D. Barnes and M. Kölling, Objects First with Java, London: Pearson, 2012.

[12] S. J. Metsker, Design patterns Java workbook, Boston: Addison Wesley, 2002.

[13] S. G. Kobourov, "Spring Embedders and Force Directed Graph," University of Arizona, Arizona, 2012.

[14] B. Moore, D. Dean, A. Gerber, G. Wagenknecht and P. Vanderheyden, Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, Durham, North Carolina: IBM, 2004.