Visualization of a Task and Message Allocation Tool

MSc in Software Engineering

Supervisor: Mr. Paul Emberson

Mohammad Amin Kuhail

September 2006

Number of words: 27514.
This includes References and Abstract and Excludes Table of Contents and Appendix. Calculated with Microsoft Word-Word Count tool.
To my parents,  
MAMK,  
and the people of Palestine
Abstract

Many engineering and science software applications rely on the visualization of information as a key support tool. These kinds of tools help people comprehend and analyze information and summarize facts.

This thesis studies and presents a software engineering approach for a visualization of relational information. This information is associated with a tool which is used for real-time systems. The tool deals with four sets of objects. These objects are processors, tasks, messages, and networks. As an input to the tool, processors are related to networks, and tasks are related to messages. As an output, this tool relates tasks to processors and messages to networks. Input relations remain the same. Output relations, however change with time.

The key challenge of the project is to provide a static and dynamic visualization solution for these relations in a readable, structured, and extensible manner taking into account a large number of objects and relations among them.

Through the spiral software model, this thesis gives a detailed design of the visual output. It first studies and examines the relevant visualization techniques and principles, then presents an initial solution and iterates it based on the feedback of the stakeholders.
Acknowledgments

I’d like to thank my supervisor Mr. Paul Emberson for his insightful ideas and great supervision and support during the work of this project.

I’d also like to thank Hani Qaddumi Scholarship Foundation which funded my Masters degree here and gave me a great opportunity to study at one of the best universities in the UK.

Finally I’d also like to thank my parents, my brothers, and my only sister for their great encouragement and support.
4.2.2 Test strategy: ................................................................. 89
4.2.3 Test Environment Requirements: ................................. 89
4.2.4 Module Testing: ............................................................. 90
4.2.6 Non-functional Requirements Testing: ......................... 94
5 Evaluation ................................................................................. 95
5.1 Software Model Evaluation .................................................. 95
5.1.1 The choice of the model .................................................. 95
5.1.2 The following of the model ............................................. 95
5.2 Evaluation of the fulfilment of the requirements ............... 95
5.2.1 Evaluation of the fulfilment of the functional requirements: 96
5.2.2 Evaluation of fulfilment of the non-functional requirements: 98
5.3 Summary ............................................................................... 100
6 Conclusion and further work ................................................... 102
6.1 Conclusion ............................................................................ 102
6.1.1 Information Visualization Concepts ............................... 102
6.1.2 Major issues and decisions ............................................. 103
6.2 Further work ......................................................................... 103
7 References .............................................................................. 104
8 Appendix ................................................................................. 106
8.1 FileProcessingController.java ........................................... 106
8.2 XMLParser.java ................................................................. 107
8.3 AllocationController.java .................................................. 113
8.4 AllocationStore.java .......................................................... 127
8.5 Message.java ....................................................................... 128
8.6 MessagePriorityAllocator.java .......................................... 129
8.7 Network.java ....................................................................... 130
8.8 NetworkMessageMapper.java ............................................ 131
8.9 Processor.java ..................................................................... 131
8.10 ProcessorTaskMapper.java ................................................ 132
8.11 Task.java .......................................................................... 132
8.12 TaskPriorityAllocator.java ................................................ 133
8.13 Drawer.java ....................................................................... 134
8.14 Main.java .......................................................................... 145
8.15 NodeGroup.java ............................................................... 145
8.16 Point.java .......................................................................... 146
8.17 TaskInformation.java ......................................................... 147
1 Introduction

Information visualization is a method of presenting data or information in interactive graphical forms through using 2-D or 3-D colour graphics and animation. This visualization aims at providing the viewer with an understanding of the information and contents [1] [2].

Information visualization is a growing as a research field and as an industry. Currently, there is a lot of literature and research which targets this vast field. However, it still is a difficult task to adapt and utilize what has been done for a particular visualization problem. This is due to the fact of the diversity of the sciences and considerations this field combine. Not only is it a field of computer graphics, but also information systems, geography, human computer interaction, and information science. Knowledge in visualization criteria, data structures and algorithms, domain analysis, and usability issues may all be important for finding a good solution for a visualization problem [3].

1.1 Aims

The main aim of this project is to visualize the information an algorithm, used for real-time systems, takes and produces. This algorithm takes a set of objects that are related to each other, and then produces allocation of some objects to others. These sets of objects are tasks, messages, networks, and processors. As an input to the algorithm, processors are connected to networks, and messages are sent from tasks to other tasks. The algorithm then allocates tasks to processors and messages to networks while keeping the input relations as they are. When a processor is connected to a network, it means that the tasks allocated to this processor can send messages to other tasks allocated to another processor which is connected to the same network. This is not to mean that tasks which are allocated to the same processors can’t send messages to each other. In fact, they of course can do.

The main aim can be summarized into the following sub aims.

- Visualize the task allocation to processors, the message allocation to networks, the processors connections with networks, and the messages relations to tasks.
- Visualize large number of objects\(^1\) (around 50 processors and a few hundred messages and tasks) in a usable way.
- Visualize the dynamic change of task and message allocation over time.
- Show information about each object.
- Help researchers in detecting errors of the algorithm, and evaluating the quality of their solutions and the correctness of their requirements.

\(^1\) The term object is used in the thesis to denote message, processor, network, or task.
1.2 Background

This section provides basic background information about real-time systems and their importance, then it explains why information visualization, generally and particularly in real-time systems, is important. Finally, it gives a summary of the key challenges of information visualization.

- Real-time systems

As mentioned earlier, this thesis is concerned with visualizing a real-time systems algorithm. This algorithm allocates tasks to processors that reside on different networks. It also allocates messages on different networks. A message is sent from one task to another.

A real time system is a software and hardware system which is subject to a real-time constraint. A real-time constraint is typically an operational deadline from event to system response [4]. Real-time systems can be hard or soft. Hard real-time systems are the ones that have tasks with deadlines that can’t be missed. Normally a major failure or damage will occur if a deadline is missed.

Soft real-time systems’ deadlines, on the other hand, can be missed. Fast response is a requirement but a delay of few seconds will not cause damage. An example of a soft real-time system is a video real player. This system tolerates a small delay of a frame.

A task in real-time systems is a single instance of a real-time computation that must complete by a certain time [5]. This certain time is its deadline. Each task has a priority which is a numerical value indicates which task can pre-empt the other or which task should run before the other. Tasks need to communicate with each other because sometime a task that is managing a resource another task with requires. For the tasks to communicate, they send each other messages [6].

Task allocation in a multi-processor system is the process of assigning the order of tasks to run based on some information associated with the tasks like their deadline and criticality. It also assigns which tasks should run on which processors. Allocation of tasks changes with time as a better solution is found.

Message allocation assigns messages to different network. These messages are sent from one task to another so they can communicate with each other. Like task allocation, Allocation of messages changes over time as the algorithm finds a better solution.

- Why visualizing real-time systems allocation?

Currently, data visualization finds many applications in science and engineering. Software engineering applications are an example. These include data-flow diagrams, program nesting trees, and entity relationship diagrams [7]. Another example would be free body diagrams in physics.
Below, a summary of the key points, why information visualizing, in science and engineering generally and real-time systems allocation particularly, is of a great benefit, is provided:

- Data visualization makes data easier to understand. As a matter of fact, humans have a great capability to store and retrieve information from visualizing the world that we live in. About 50% of the human brain has neurons which assist with vision input and processing. Consequently, this makes sense that visualization helps us to gain insight by using our vast [8].
- It helps researchers and average people too to analyze data efficiently. This means visualization assists in accessing data faster and easier, spotting significant and interesting patterns, and monitoring data more accurately. This particularly comes in handy in our allocation problem as we expect to deal with large number of bits of data that are difficult to follow without the assistance of visual diagrams. Also, with these features in mind, researchers can evaluate the quality of their solutions much faster if their solutions are visualized. In our project, an example of quality is that the messages should be sent faster. Messages can be slowed down if they are allocated to a busy network. With a visual diagram of the solution, the researcher should be able to evaluate this sort of quality quickly and easily.
- It helps researchers find errors in their simulations and experiments [9]. I was informed by the designer of the allocation algorithm that the algorithm could mistakenly allocate a message to a particular network, and then this message could be sent to a task that doesn’t reside on the same network. Therefore, the message will not be received. Obviously, it would be of a great benefit to spot such a bug through visual diagrams.
- It serves as an efficient and easy way to convey information among collaborators [9].
- It saves a great deal of time. Obviously, one would spend a lot of time on analyzing massive and complex sets of data without the support of a good visualization tool.

1.3 Structure of the report

The thesis is structured as follows:

- Chapter 2 provides an overview of information visualization principles and issues. It then discusses some available technologies in visualizing graphs. Finally, it discusses two case studies in brief.
- Chapter 3 states and analyzes the functional and non-functional requirements of the thesis project.
- Chapter 4 gives a software engineering approach for designing the requirements specified in chapter 3. Furthermore, it provides a testing for the functional requirements at the end of the chapter.
- Chapter 5 systematically evaluates the success of the project based on the requirements, the literature review, and the feedback of the users.
- Chapter 6 provides conclusions of the knowledge that has been gain throughout the phases of this project.
2 Literature Review

This chapter aims to provide the reader with the literature relevant to this project.

First section is about the tasks and messages allocation. This section gives a brief overview of the tool. Then it presents its functionality, and sums up the nature of the information of this tool that we aim to visualize in this project.

Second section provides the basic relevant knowledge of information visualization. As each visualization problem has its own criteria, the section will not go into deep details of some of the algorithms or the techniques that are available; it will rather discuss them and present different points of views and brief criticism as they are needed. This section first provides an introduction of the information visualization. Then, it explains the basic visualization process, and finally the issues and challenges of information visualization including the usability of visualizing large quantities of information.

Third section presents the reader into the visualization of graphs. It gives the basic information on it including history, background, and relevant types of graphs. Then, it discusses some graph drawing techniques and provides a brief analysis about each one. Finally, it gives a brief idea about graph colouring.

Last section concludes the knowledge has been gained in this chapter.

2.1 Tasks and Messages allocation Tool

2.1.1 Overview and Motivation

Real time systems, including complex embedded systems, are liable to change due to maintenance, enhancements, and upgrades. Developing new systems is not a good idea because of the high cost of these systems. Therefore, modular systems came as a solution as they allow individual hardware and software modules to be replaced or modified without affecting the rest of the system [10] [11].

It is crucial for a modular system to allocate software tasks and messages onto hardware modules while meeting a number of constraints such as timing and memory ones. It is also important to optimize the design for the allocation to provide flexibility and avoid the high cost that may result from a late change in the design. Therefore, the goal of the Tasks and Messages Allocation Tool is to map the software tasks and messages onto a hardware platform while meeting these constraints. The tool will also give the mappings properties which reduce the impact of future change in the project [10].
2.1.2 Functionality

Figure 1 shows the input and output for the Tasks and Messages Allocation Tool. As an input, it takes two XML files and produces one XML file as an output. The first input to the tool is a system XML file. This file contains the attributes of hardware modules\(^2\), networks, messages, and tasks. It also contains information of mapping the messages into tasks as well as mapping processors into networks. The mapping included in the system file is constant. This means that on the completion of the allocation process, the mapping will remain the same.

![Figure 1: Tasks and Messages Allocation Tool Input/Output](image)

Another input to the tool is an initial configuration XML file. If this input is not provided, the tool will generate a random one. This file along with the system file is necessary for the tool to start producing scheduling information for the system \([10]\). These bits of information are then used to assist carrying out a search to find the optimum configuration for the system.

Figure 2 shows an example of an XML system file. In this example, two processors with ids P0 and P1 are first listed. Each one is included between hardware-module tags. The attributes associated with each processor are ID, name, criticality, memory, precision, and clock speed. Three networks N0, N1, and N2 are then listed right after the processors. Each network has a name, ID, and bandwidth. Each network is wrapped within tags as with processors. These tags are network tags. The interface tag then maps processors and networks to each other. In our example, P0 is mapped to N0 and N2, and P1 is mapped to N1 and N2. Therefore, the relation between processors and networks is a many to many relation. In other words, zero or many processors can be associated with zero or many networks and vice versa.

As we follow figure 2, two tasks T0 and T1 are listed. The attributes of each task are name, ID, period, deadline, wcet, bcet, max_jitter, and criticality. A message M0 is then listed right after these two tasks. This message is sent from task T0 to task T1. This message has name, ID, deadline, and size as attributes. One can conclude easily that one message is associated with two tasks. On the other hand, though it is not shown in this example, zero or more messages can be sent from or to one task.

\(^{2}\) The term “hardware module” and “processor” will be used interchangeably in this thesis.
<?xml version='1.0'?>
<!DOCTYPE tat:system SYSTEM "system.dtd">
<tat:system xmlns:tat='http://www.cs.york.ac.uk/atu/tat'>
<tat:hardware-module id="P0" name="Processor 0">
<tat:scheduling-details>
<tat:attribute name="criticality" value="1"/>
</tat:scheduling-details>
<tat:object-details>
<tat:attribute name="memory" value="41943040"/>
<tat:attribute name="precision" value="32"/>
<tat:attribute name="clock_speed" value="25"/>
</tat:object-details>
</tat:hardware-module>
<tat:hardware-module id="P1" name="Processor 1">
<tat:object-details>
<tat:attribute name="memory" value="41943040"/>
<tat:attribute name="precision" value="32"/>
<tat:attribute name="clock_speed" value="25"/>
</tat:object-details>
</tat:hardware-module>
<tat:network id="N0" name="P0 comms">
<tat:object-details>
<tat:attribute name="bandwidth" value="1073741824"/>
</tat:object-details>
</tat:network>
<tat:network id="N1" name="P1 comms">
<tat:object-details>
<tat:attribute name="bandwidth" value="1073741824"/>
</tat:object-details>
</tat:network>
<tat:network id="N2" name="P0 - P1 comms">
<tat:object-details>
<tat:attribute name="bandwidth" value="51200"/>
</tat:object-details>
</tat:network>
<tat:interface module-id="P0" network-id="N0"/>
<tat:interface module-id="P1" network-id="N1"/>
<tat:interface module-id="P0" network-id="N2"/>
<tat:interface module-id="P1" network-id="N2"/>
<tat:task id="T0" name="Task 0">
<tat:scheduling-details>
<tat:attribute name="period" value="50"/>
<tat:attribute name="deadline" value="50"/>
<tat:attribute name="wcet" value="15"/>
<tat:attribute name="bcet" value="0"/>
<tat:attribute name="max_jitter" value="4"/>
<tat:attribute name="criticality" value="1"/>
</tat:scheduling-details>
</tat:task>
</tat:hardware-module>
</tat:system>
As mentioned earlier, the output of the allocation process is a configuration XML file. In structure, it is the same as the initial configuration file which is taken as an input. Figure 3 shows a simple example of an XML configuration file. In this example, the first allocation is for a task with ID T0, allocation_id P1, and priority 1. This means that a Task with ID T0 is allocated with a processor P1 and given a priority 1. The allocation as a whole is included between two object-configuration tags. Similarly with task T1 and T2 are allocated to the same processor but are given the priority 2. The code of message allocation is totally similar to the task allocation with the difference that the ID of the object it is allocated to is a network ID.

The relation between processors and tasks is a one-to-many relation. This means that zero or more tasks are allocated to one processor. Similarly with messages and networks, zero or more messages are allocated to one network.

Figure 3: An example of an XML configuration file, adapted from [10]
To sum up, the tasks and messages allocation tool takes four sets of objects, processors, networks, tasks, and messages with zero or many processors are associated to zero or many networks and each two tasks are associated to zero or many messages on permanent basis. The tool then allocates zero or many tasks to a processor and zero to many messages to a network, and so on for the rest of processors and networks. This allocation changes over time. Clearly, the type of data we have to visualize is relational. Table 1 summarizes all the relations between these objects along with the nature of these relations.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Nature of the relation</th>
<th>Static/Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors and Networks</td>
<td>Many to many relation</td>
<td>Static</td>
</tr>
<tr>
<td>Messages and Tasks</td>
<td>Many to Two</td>
<td>Static</td>
</tr>
<tr>
<td>Processors and Tasks</td>
<td>One to Many</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Networks and Messages</td>
<td>One to Many</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

**Table 1:** Relations of the tool's sets of objects
2.2 Information Visualization

This section discusses some of the most important general issues of information visualization that are applicable to visualizing information regardless of their structure. However, though these techniques are general, the section will discuss these techniques and relate them to the specific problem that we are trying to solve in this project.

2.2.1 Brief Introduction

A picture is worth a thousand words! Pharaohs seemed to believe in this saying ages ago to convey information and facts about their great history through their famous beautiful drawings. Paintings on cave walls such as those at Lascaux, France, or petroglyphs scratched on a natural rock surfaces throughout the world were also attempts to convey information visually [12]. Nowadays, the essence of this saying still applies. However, the information to be visualized is much larger and more complex.

Dealing with so much information has increasingly become a part of the life of not only scientific researchers but also many other ordinary people. Examples of information systems we deal with daily are e-mails, e-news, e-commerce websites, e-learning facilities, and transportation systems. The more people need to access information and the larger and more complex this information is, the more the need arises for showing this information in a usable way. The field of information visualization has become as an approach to make this information intelligible to the user.

Visualization is an interface between two powerful information processing systems, the human mind and the modern computer [13]. Through information visualizing, we can transform information and knowledge into visual form making use of humans’ natural visual capabilities. Thus, a method is required is to generate the interactive visual representation of the information that exploits the perceptual capabilities of the human visual system and the interactive capabilities of the cognitive problem solving loop [15].

2.2.2 Visualization Process

Figure 4 shows the steps of the visualization process. Usually when users require visualizing some information, they may not give this information in a structured way that is ready to use for the designer. Therefore, the first step of the visualizing process is to transform this raw information into a well-organized data format. The result of this phase step is a dataset containing of a set of data entities each of which has associated data attribute values [15].
The second step transforms the organized dataset information into a visual form. The third step takes the visual form of the second step and produces views. These views display the visual form on the screen and provide user-usability features such as navigation. Finally, this view is presented to the user through the human visual system. Users interpret the underlying information. As the user interacts with the information, he/she edits the resulting information, and makes further interpretations [15]. The user interaction can also be understood as a part of the system that iterates and enhances it, that is, the designer of the visualization uses the user feedback to make further changes to the design to enhance the usability of the graph.

**Visual Mapping**

As mentioned earlier, visual mapping is the second step of the visual processing. It is the heart of the visualization as it communicates information from computer to human. The dataset is computationally mapped into visual form by some function \( F \), which takes the dataset as input and generates the visual representation as output, and then the visual representation is communicated to the users. These users must cognitively reverse the visual mapping by inverting function \( F \) to extract the information from the visual representation [14].

The visual mapping is done by two important steps; first, each data set is mapped to a visual symbol. These symbols include icons, lines, dots, etc. Second, attributes of each symbol are mapped to these visual entities. Common visual attributes are size, colour, transparency... etc. As these attributes increase, we have a problem of visualizing them. Therefore, there is a need for prioritizing these attributes to show the more important ones. Another way for coping with this problem is through interaction. This way as the user interacts with a data entity, he/she can get all its attributes [15].

As visual mapping is involved with transformation information into a visual farm, identifying the structure of this information is necessary to adapt the already-made visualization techniques [15].

Our thesis is concerned with visualizing data coming from a Task and Message Allocation Tool. As we recall from section 2.1, we have processors that are related to networks, networks that are related to messages, messages that are related to tasks, and tasks that are related to processors (see table 1). Therefore, the nature of the information we have is relational. An example of relational information is software engineering diagrams. This includes UML classes, and state-transitions diagrams. Graphs are said to suit relational information (See section 2.3 for more details)
Visualizing other type of data like tabular or quantitative data is out of the scope of this project.

2.2.3 Visualization Issues and Challenges

2.2.3.1 Insight

Human vision contains millions of photoreceptors and is capable of rapid processing and pattern recognition. This ability not only enables humans to comprehend a visual form but also go beyond that, that is, discovering trends and insights. Strength of visualization is the capacity for discovery and the recognition of new insights unexpected by users and potentially unforeseen by the visualization designers. Therefore, identifying the types of insights the user wants is so crucial in the requirements phase. There are two types of insights, simple and complex insights. The simple ones include easy-to-do computational features such as search, minimum, average, and percentages. Complex insight, on the other hand, is not straightforward to provide. Depending on the visualization problem, designers should provide as much complex insight as they can. Complex insight includes patterns. Patterns can be thought of trends, distributions, trends, or structures. Other complex insights are the detection of exceptions, identifying the relationships. Furthermore, some visualization applications which target large amount of data provide clustering, which is grouping the similarities to make advantage of the available space [15].

2.2.3.2 The need for interactivity

The two most challenging characteristics of information to be visualized are:

*Complexity*: Support abstract information that may have multiple interrelated data types and structures. In this project, we have two complex relationships; the relationship between processors and networks, and the relationship between tasks and messages.

*Scalability*: Supporting very large quantities of information. In this project, we expect to visualize around fifty processors and several hundreds of messages. In fact, that is not considered to be highly large. Due to these two characteristics, visual representation on its own is not enough. Interactive techniques must be designed to support the visual output [17].

2.2.3.3 Usability issues for visualizing large quantities of information

This thesis is concerned with visualizing a moderate large quantity of information. Therefore this section studies only the usability of visualizing large quantity of information.

Usability is the quality of a system that makes it easy to learn, easy to use and encourages the user to regard the system as a positive help in getting the job done
It is a key requirement for visualization. After all, we want to visualize data to easily comprehend it. So what’s the use if this visual output is not easy to read? Visualizing massive amounts of information is considered one of the key problems in the visualization research. This is due to the fact that the more quantity of information we have, the more difficult they are to visualize and pack on the available screen space [15].

Shneiderman suggested “overview first, zoom and filter, then details on demand” technique [36]. This technique provides an overview of the full information space and relaxes information details. The user can get detailed information by interacting with the information space. This can be done by zoom + pan, overview + detail, and focus + context. Advantages of this technique are: it supports formation of mental models of the information space, it reveals what information is present or not, it reveals relationships between the parts of the information providing broader insight, it enables direct access and navigation to parts of the information simply by selecting it from the overview, and finally this technique encourages user exploration and thus increases his/her visualization skills [15]. However, some researches argued that this technique may not always produce an elegant solution. Some disadvantages will be discussed in more details in the navigation sub-section.

One of the techniques of reducing the amount of space required to visualize information is clustering and scaling. The goal of cluster analysis is to divide a large data set into a number of sub-sets, called clusters or aggregates, based on some given similarity measures. Figure 5 explains that the original data set contains 11 balls of different colours. These balls can be grouped into three clusters of balls so that each cluster only contains balls of the same colour [3]. Clustering has helped users in many applications to find their required information so easily. Example of this, the application Xerox which helps users to deal with a large information space by repeatedly clustering documents at various levels [3].

After forming these new groups, the next decision is determining the new attribute values of these groups. Basically, these attributes should represent the contained member entities. Statistical summaries such as mean, minimum, maximum, and count are commonly used. The visual representation of the aggregates should also reveal some hints of their contents. Example of this is the aggregate towers which are shown in figure 6. The aggregates are shown as towers whose height represents the number of balls within each cluster.
of entities in the aggregate. Zooming out causes more aggregation as needed and zooming in reveals true entities further and takes aggregation out more [15].

Figure 6: Aggregate Towers, adapted from [15]

Navigation

Navigation is a wide topic in usability. We are only going to discuss some important issues of it, especially those which are concerned with visualizing rather large quantities of information.

1. Zoom + pan

The most essential requirement for zooming is that the user interface must allow the user to zoom in and out freely and easily, across various levels of details [3]. Zooming in allows users to go in depth to see what the details of the visual information are. Zooming out, however, gets the user again back to the overview [15]. Panning enables the user to move across the visual information.

Criticism for this technique is that the user gets lost when he zooms in the information space since the overview is lost [15]. Solution for this is by providing a small overview window along with the main window of the information space (See figure 7). Alternatively, provide the structure of the information along with the information space. This is especially applicable on hierarchal information structures.

Figure 7: Zoom In+ Overview
2. Focus + Context Views

As figure 8 shows, Focus+ Context view is a technique used to expand a portion of the information space while keeping the context in space. This expanded portion reveals detailed information about it but unfortunately distorts the surrounding important context. Another alternative of focus + Context is to use a magnifying glass. Though this one doesn’t distort the surrounding context, it actually hides a lot of it. Another problem with the focus+ Context view technique is the limited scalability of it; which is typically under 10:1 zoom factor [15].

![Figure 8: Focus + Context Example, adapted from [18]](image)

Interaction

Ideally, we would like to visualize every bit of information we have in the dataset. Unfortunately, for massive and complex information, this is impossible. Therefore, we need to think of a way through which we extract the hidden information from the visual information space [15]. Here is where interaction comes in handy. There are many interaction techniques; we discuss two of them here.

1. Selecting

Selecting enables users to select one or more entities. This selection gives user many benefits. These benefits include: Users can reveal detailed information about entities, highlighting important entities or those which are overlooked in a crowded information space, grouping a set of related entities, and extracting these entities for future use [15].

Selecting can be done either directly or indirectly. Obviously, direct selection is done through a direct contact with entities, such as clicking or pointing at them [15]. On the other hand, indirect selection is done by the help of other facilities such as the information structure of the visual information space.
2. Filtering

Filtering enables users to dynamically reduce the information displayed on the screen and show only these ones of interest [15]. Search engines are one of the best techniques for this purpose. Filtering combines searching with visualization. This is of course useful for the user because of these two reasons: (1) when the result of the search is too large for the user to comprehend, filtering allows him to browse through the result and even filter it more and more to extract what he is interested in, and (2) when the result is too small, filtering allows the user to show other related information that may help [19].

Graphs can be one of the most complicated data structures (See section 2.3), filtering can be used to form simpler structures for the visualization. Botafogo presented a technique which uses structural analysis to make hierarchies from hypermedia network structure [38].
2.3 Visualization of Graphs:

Last section discussed the general issues of information visualization. In the visualization process, it was mentioned that identifying the structure of the information that we need to visualize is a major concern which influences the visualization design. From table 1 in section 2.1, we identified the nature of this information, which is, relational. This section discusses in brief the issues of the visualization of graphs, which are used to visualize relational information.

2.3.1 Brief History

Mathematicians for centuries investigated geometric representation of graphs for information visualization [7]. In the 17th century, analytic geometry, theories of errors of measurement saw a great development. The probability theory was born at the same century as well. Over the 18th and 19th centuries, the interest towards economic and social statistics increased rapidly. Such interest was accompanied by a rise in visual thinking [20]. Mathematical proofs as well as functions were visualized by diagrams and calculations were aided by monograms. Statistics helped in the development of various graphics forms. These forms made the properties of empirical numbers, their trends, and distributions easier to understand [20].

In the 1960s, computer scientists started using graph drawings as diagrams to help them with the understanding of software. Knuth was one of the very computer scientists to draw flowcharts to visualize his algorithms [7].

Nowadays graphs are massively used in many sciences and statistics applications. Mathematicians use them for geometry drawing. Computer scientists use them for drawing flowcharts, UML diagrams, Entity relationship diagrams, state charts, and real-time systems diagrams. Engineers use them for electric circuits drawing, civil engineering drawings, and cartography [21]. Graphs can also be found in further applications such as sociology (social networks), biology (evolutionary trees), and chemistry (molecular drawing) [7].

2.3.2 Background

Graphs can be thought of as plot graphs; the ones that mathematicians use to visualize mathematical functions. Pie chart graphs as well as bar graphs are also called graphs. These are mostly used for statistical purposes. The graphs which we are interested in in this thesis are not any of these types. Our main focus is on the graphs in computer science. Therefore, in the rest of this thesis, the term “graph” is used only for the computer science graphs.

Graphs are used to visualize information that can be modelled as objects and relations amongst these objects [7]. Another definition for a graph is defined as a set of objects
called nodes or vertices\textsuperscript{3} connected by links called lines or edges\textsuperscript{4} [22]. Figure 9 shows a very basic graph. The three circles A, B, and C are called nodes or vertices. In this graph the nodes are visualized as circles. However, they can be in many different shapes like rectangles, triangles, squares, and so on. 1, 2, and 3 are the edges of this graph. Each individual edge links two nodes with each other. These edges serve as relations between the objects\textsuperscript{5}; for example, edge 1 relates object A and object C.

![Figure 9: A basic Graph of three nodes and three edges](image)

Graphs are used extensively in computer technology as a tool to model associative and structural information [23]. To illustrate this, figure 10 shows how we can visualize the relations which table 1 shows. These are many-to-many relation, one-to-many relation, and many-to-two Relation. First part shows a many-to-many relation. In this relation, a black or a white object can have zero or many objects of the other kind. In the second part, one white object is associated with zero or many black objects. In the last part, a black object is associated with two objects and a white object is associated with zero or many messages.

![Figure 10: Representation of relations in table 1 as graphs](image)

Graphs can range from simple graphs with few numbers of nodes and edges to highly complex ones with a huge number of nodes, edges, and crossings amongst these edges. Nonetheless, the idea of graphs is still the same; nodes as well as edges that connect these nodes. (See figure 11)

\textsuperscript{3} The term “node” is used in this thesis rather than “vertex”.
\textsuperscript{4} The term “edge” is used in this thesis rather than “line”.
\textsuperscript{5} The term “object” is used to mean node in the drawing techniques sub-section.
**Basic definitions**

- A *graph* $G = (N, E)$ consists of a finite set $N$ of nodes and a finite multiset $E$ of edges, that is, unordered pairs $(u, n)$ of nodes [7].
- Two edges of a graph are called *adjacent* if they share a common node. Similarly, two vertices are called adjacent if they share a common edge, in which case the common edge is said to join the two nodes. An edge and a node on that edge are called incident [26].
- A *path* in a graph $G = (N, E)$ is a sequence $(n_1, n_2, \ldots, n_h)$ of distinct nodes of $G$, such that $(n_i, n_{i+1}) \in E$ for $1 \leq i \leq h-1$ [7]. For example, ACB is a path of the graph in figure 2. YWK is a path too in the simple graph in figure 3.
- **Topology**: The sequence of edges contouring the faces\(^6\) of the drawing. Example, graph A and B in figure 12 has the same topology.

**Types of graphs**

There are many types of graphs in graph theory. They vary according to many factors. This sub-section covers important types of graphs that are relevant to this project.

In terms of direction, graphs can be either directed or undirected.

---

\(^6\) A face is a topologically connected region bounded by edges [42].
In a *directed graph*, an edge from node 1 to node 2 is not the same as an edge from node 2 to node 1. On the other hand, in *undirected graphs*, a direction from a node to another doesn’t matter. Therefore an edge from node 1 to node 2 is the same as from node 1 to node 2. The graph in figure 2 is an example of an undirected graph. Directed graphs are necessary for us in this project particularly because we have each message which is sent from a certain task to another, so we would like to know the direction of a message. However, in the rest of the relations amongst objects, there is no significance in showing directions. Example, we would want to know whether a certain processor is connected to this network or not, but there wouldn’t be any need in showing an arrow from this processor to that network, just a plain line that connects them both would do it (See figure 13, A processor is represented as P, and a network is represented as N). So therefore, in this thesis we are interested equally in directed and undirected graphs.

In terms of planarity, graphs can either be planar or non-planar. *Planar graphs* are the ones that can be drawn without intersections or crossings amongst the edges. Figure 14 shows two drawings of one identical graph. The first one is a non-planar drawing where edge AC intersects with edge BD. The second one, however, is a planar drawing of the same graph where there are no intersections at all between edges.

Consequently, Even if a graph has intersections, it is still planar if it can be redrawn with no crossings. On the other hand, crossings can’t be avoided in *non-planar graphs*. Figure 14 shows the smallest basic non-planar graphs [27]. The first one is called $K_5$ since it has 5 nodes. In this graph edges AC, AD, BE, BD, and EC intersect with each other.

$K_{3,3}$ is another non-planar graph; it shows six nodes that are fully connected to each other. The nature of a many to many relation, which we need to deal with in this thesis, can lead to a non-planar graph (See figure 15). Ideally, we would like to have no crossings at all as they confuse the user and make it difficult for him to extract a relationship between an object and another so easily. This thesis will discuss some techniques of reducing these crossings (See 2.3.3.2, Layered digraphs).
Graphs can contain loops. A *loop* is an edge (directed or undirected) with both ends the same [22]. *Acyclic graphs* are the ones that contain no loops. *Cyclic graphs*, on the other hands, contain them. Loops can only happen in the project problem if a task sends itself a message (See task T in figure 16). Figure 16 shows graphs which have loops. The first and second ones are directed graphs, the third one is undirected. Figure 17 shows an undirected graph that looks like it has a loop. However, it doesn’t as there is no way if we visit a node we will go back to the same node if we follow the direction of the corresponding edges.
2.3.3 Graph drawing

Basics

Currently there are many graph drawing algorithms for different types of graphs. Therefore, one of the things that needs to be taken into consideration when drawing a graph is the type of the graph and how its nodes are connected to each other. For instance, Trees drawing algorithms should be considered when drawing a graph that fits into a tree (See 2.3.3.2, Trees). The way users prefer the graph to be drawn is also an important issue. For example, users often want the drawing of graph G to illustrate the combinatorial properties of G. If G is an acyclic digraph, then it may be important to draw all the edges following the same direction, to emphasize the absence of cycles [7]. As a follow up point, users in different environments may have different preferences. Therefore, the environment or the “application domain” of where the graph will be used is of great importance.

2.3.3.1 Drawing Conventions

A drawing convention is a basic rule that the drawing must satisfy to be admissible [3]. For example, in an electric circuit graph, one might think of a convention of using either vertical or horizontal lines for the edges and special symbols for the nodes (see figure 18).

![Electric circuit](image)

**Figure 18:** Electric circuit

Generally, the nodes are represented by symbols as circles or boxes, and each edge is represented by a curve or line between two nodes. Drawings conventions specify different ways of how edges can be drawn to connect these nodes.

Here are some of these conventions:

*Polyline Drawing:* Each edge is drawn as a straight line segment.
*Straight-line Drawing:* It is a special case of Polyline drawing where each edge is drawn as a straight line segment
*Orthogonal Drawing:* It is again a special case of Polyline drawing. Each edge is drawn as a polygonal chain of alternating horizontal and vertical segments.

<table>
<thead>
<tr>
<th>Polyline Graph</th>
<th>Straight-line Graph</th>
<th>Orthogonal Drawing</th>
</tr>
</thead>
</table>

**Table 2:** Drawing conventions
Aesthetics

The readability of a graph is the capability of conveying the meaning of this graph quickly and clearly [40]. Aesthetics are properties of a readable graph that we should aim for as much as possible. These properties include [7].

1. **Crossing**: Minimization of the total number of crossings between edges. Ideally, a planar graph is preferred but not all graphs are planar. Most researches emphasize on this feature as it enhances the readability of a graph.

2. **Area**: Minimization of the area of the drawing. In some applications, space utilization is important particularly the ones when scaling down is not allowed. The smallest area can be defined as the smallest rectangle with horizontal and vertical sides covering the drawing.

3. **Total Bends**: Minimization of the total number of bends along the edges. This aesthetic is especially important for polygon drawings particularly orthogonal ones.

4. **Edge Lengths**: Edge lengths should be short but not too short

5. **Angular Resolutions**: Maximization of the smallest angle between two edges incident on the same node. This aesthetic is vital for straight-line drawings.

It is difficult to meet all these properties altogether. This is due to the fact that some of them can be mutually exclusive. For example, a symmetrical graph may require a certain number of edge crossing [3]. Furthermore, some studies showed that following all of them as much as possible doesn’t necessarily lead into a readable graph [28]. Consequently, depending on their particular problem, graph software engineers decide which property of these aesthetics should be relaxed and which one should be implemented. NickWorks is a visualization tool which helps analysts to work on large networks of telephone calling data. Developers of NickWorks chose to relax some of the optimization criteria so that they speed up the process. For example, the aesthetics of evenly distributed nodes is not required for investigation traffic on a telephone network. Therefore, this criterion was considerably relaxed by including only a final polishing algorithm to separate the overlapping nodes [3].
2.3.3.2 Graph drawing techniques

This sub-section will discuss the most famous and common drawing techniques and try to provide a brief analysis for each one.

1. Straight-line drawing technique:

As mentioned earlier in 2.3.3.1, in straight line-drawing technique, straight-lines are used to connect objects with each other.

![Figure 19: Straight line drawing](image)

Brief Analysis

Straight-line technique offers relative flexibility in connecting objects at different positions. This is because a straight-line technique allows the visualization of objects at fixed positions. However, sometimes changing the positions is desirable to increase the readability of the graph. In this project, the objects we need to visualize don’t have physical positions.

For connecting an object with another, only straight lines are allowed. This means that there won’t be any bends in a path between an object and another. This of course reduces the complexity of the graph and improves its readability (See 2.3.3.1). Nonetheless, lack of readability is still one of this technique’s drawbacks. Readability is a main concern of the usability of the visualization of graphs. This particularly is important for graphs of large amount of information. It also is an important principle of information visualization. Readability is not a problem for straight-line graphs of a small number of nodes and edges. However, as these nodes increase, crossings amongst edges increase and thus decrease the readability of the graph. There is a lot of literature on improving the crossings amongst edge to improve the readability. However, it remains an issue for huge graphs of complex relations. As a follow up point, structurability is also important as a usability issue. It helps user search for a certain object that belongs to a certain group within a certain area. Another disadvantage of this technique is the lack of structurability this becomes as a result of attempting to reduce crossings by changing the position of nodes. This of course means that similar nodes may end far away from each other, and thus makes the graph unstructured.

2. Forced-based drawing:

Force-directed layout technique is usually selected for undirected graphs, this being ideal for simulating physical and chemical models [29]. In this technique,
the graph is viewed as a virtual physical system, where nodes of the graph are bodies of the system. These bodies have forces action on or between them. These forces are physics-based, and thus have a natural analogy, such as gravitational attraction or magnetic repulsion [30], and then they are applied to the nodes, pulling them closer till the system comes to an equilibrium state. At that moment the graph is drawn [31]. In other words, in every iteration, all pairs of nodes need to be visited and their mutual repulsive forces computed.

**Brief Analysis**

Adopting this technique would yield into a good looking graph and help the user interact with it. Users can watch how the graph evolves from a clutter set of nodes connected to each other into an equilibrated nice graph. Some tools also enable users to pull some nodes and watch how the graph will again go back into its equilibrium state [31]. Nevertheless, due to its interactive nature, speed of drawing is a considerable problem with this technique especially when there is a large number of nodes and edges. Another problem is that it is difficult to group specific nodes together since each node will take its position naturally. In our project, we have different four sets of nodes; processors, networks, tasks, and messages. Having each set of these overlap with the others will confuse the user and make it hard for him to read the visual output. Therefore, discussing the details the forced-based drawing is out of the scope of this system.

3. **Orthogonal drawing:**

As mentioned earlier in sub-section 2.3.3.1, orthogonal drawing is a technique for graph drawing in which each edge is drawn as a polygonal chain of alternating horizontal and vertical segments. Because they contain only horizontal and vertical lines, orthogonal drawings are known for their high clarity and readability. This is also important especially for computer screens which approximate non-horizontal and non-vertical straight lines with a staircase type of line drawing [40].

Orthogonal drawing aims for reducing the number of edge crossovers and area covered. These are of great interest in the areas of VLSI and PCB [34]. One of the algorithms that have been developed to provide orthogonal drawings is Topology-Shape-Metrics Model (TSM-Model). The aim of this algorithm is to minimize edge crossing, utilize space, and minimize the number of bends. This is done through three steps:

- **Planarization:** In this step, the edges crossings of the graph are minimized [35]. In this step, the graph is represented as a topology (See figure 20).
- **Orthogonalization:** This step takes the planar topology T from the output of the first step, and generates an orthogonal shape H which guarantees minimum bend count. H includes the edge order, the description of bends on the edges, and their angles. Since we are talking about an orthogonal shape, the angles are multiples of 90° [35]. Many algorithms try to minimize bends of orthogonal graphs. These algorithms are out of the
scope of the project. However, figure 21 shows some common bends that could be minimized by the help of this example.

- *Compaction:* In this step, the dimensions of the shape H are calculated; nodes are mapped on two dimensional points (coordinates), and edges on line-segments. This planar embedding should need as little area as possible [35].

![Figure 20: Planarization](image)

![Figure 21: Bend-stretching transformations, adapted from [7]](image)

![Figure 22: Compaction, adapted from [7]](image)
**Brief Analysis:**

Orthogonal drawing achieves readability as it maximizes the minimum angle between the adjacent edges. This is one of the graph aesthetics. Figure 22 is an example of how easy to follow the relations between the black nodes and the white nodes. Orthogonal drawings are also easily structured, since there are only two classes of line segments; horizontal and vertical. Figure 23 shows how easy it is to form structured groups of objects or nodes and draw relations between these objects. However, this nature creates the problem that a node can’t be of more than a four degree, that is, a node can’t have relations with more than four other nodes. Many techniques were suggested in the literature to solve this problem too [36]. Another main advantage of this technique is performance, because orthogonal drawings contain only horizontal and vertical lines; they are faster to draw for a computer screen. This is because non-vertical and non-horizontal straight lines are in the end approximated as non-vertical as staircase type of line drawing [40].

![Figure 23: Structurability of orthogonal drawings](image)

Unfortunately because orthogonal drawings only contain horizontal and vertical segments, these bends come as a result of trying to connect nodes which don’t share either vertical or horizontal segments with other nodes. As mentioned earlier in this section, there are a lot of algorithms suggested in literature to cope with this problem. These kinds of algorithms also aim for making the graph more readable as well as minimizing the area of drawing. The techniques of minimizing the bends of orthogonal graphs are out of the scope of this project. This is due to the fact that these techniques are applied on general orthogonal graphs, that is, the graphs which can have any sort of relations and nodes. However, in our particular visualization problem, we have fixed relations; one-to-many, many-to-many, and two-to-many relations (Refer to 2.1.2 table 1). As we can’t have any other relations, the visual output can be controlled; so the bends can also be controlled too. As mentioned in the previous disadvantage, orthogonal drawings suffer from the bends problem. Bends minimization algorithm tries to minimize these bends by changing the position of the nodes. However, in some sort of graphs, particularly the geographical ones, we would like to keep some nodes at certain positions so they represent this geographical location. Example of this is the graph shown in figure 23. The nodes here are meant to be suited at these positions. If we attempt to connect these nodes together using orthogonal drawing, we will end up with a lot of unrecoverable bends.
4. Trees:

Terminologies for Trees:

A tree is a connected acyclic graph. A rooted tree consists of a tree T and a distinguished node r of T from which the tree originates. A binary tree is a rooted tree where each node has at most two children [7]. Trees fit a one-to-many relation between objects, which are one of the relations that we are interested in in this thesis (Refer to 2.1.2 table 1).

If n is a node of T, then the sub-tree rooted at n consists of sub-graph induced by all nodes on paths originating from n; and of course it has root n. The depth of a node n of T is the number of edges of the path of T between n and the root. The height of T is the maximum depth of a vertex of T [7].

Layering:

Layering is a powerful approach in structuring trees and thus making them clean and easily readable. It is simply done by placing each node with depth i into layer Li. In particular, the root of T is placed into layer L0 (see Figure 22). Layered-tree-Draw is a powerful algorithm for drawing binary trees which has been widely used in visualization application. The basic idea behind this algorithm is to divide and conquer a tree (see figure 22), this means that no matter how massive a tree is, in the end it still consists of small sub-trees. The base case for the algorithm is that if T consists of a single node, its drawing is trivially defined, then we start the divide approach by recursively applying the algorithm to draw the left and right sub-trees of T, and then each sub-tree is dealt with as a separate tree. Afterwards, all drawing of the sub-trees are moved towards each other till their horizontal distance becomes 2. Finally, the root r of T is placed vertically one unit above and horizontally half way between its children. If r has only one sub-tree, say left one, then r is placed at a horizontal distance 1 to the right of its left child [7].

Figure 24: Geographical graph, adapted from [37]
Figure 25: Layered drawing of a binary tree $T$: (a) A tree is visualized in a layered way (b) Conquer step of layered-tree draw. (c) Co-ordinates assigned by the algorithm, adapted from [7]

Brief Analysis:

This algorithm generally leads into a structured, readable, and eye-pleasing graph. However, it still has some disadvantages. One of these disadvantages is that the algorithm may not always yield a grid drawing, since $x$-coordinates are in general rational numbers, this, however, can be overcome by placing the sub-trees at horizontal distance 2, 3, 4, ..., etc, such that the distance between the roots of the sub-tree is even. Another disadvantage with this algorithm is that it may not always achieve optimal width of area. Studies suggested that this problem can be solved by means of linear programming [38].

Some slight modifications to the algorithm, particularly in the part that mentions the distance between the sub-trees, can accommodate the algorithm to fit into drawing trees in general.

Another disadvantage with trees is that they don’t adapt some kinds of relations. For example, Many-to-many relations, which are one of the relations we are interested in, can’t be visualized by trees.

5. Layered drawing of Digraphs:

Layered digraphs, also named as hierarchical graphs, are used in many applications. Examples range from social networks to computer networks, websites structure to UML diagrams. Most of these layered digraphs adopt a hierarchical approach for creating polyline drawing of digraphs with nodes arranged in horizontal layers.

The hierarchical approach consists of three steps: Layer assignment; which assigns nodes to horizontal layers and determines their $y$-coordinates, Crossing assignment; which orders the nodes within each layer to reduce the number of the edge crossings,
and *Horizontal coordinate assignment*; which determines an x-coordinate for each node. Layering assignment involves assigning the nodes of an acyclic graph into subsets L1, L2, …, Lh, such that if (u,v) \( \in E \), where \( u \in L_i \) and \( v \in L_j \), then \( i > j \). An acyclic digraph with a layering is a layered digraph. The number of layers is the graph height, h. The width of a digraph is the number of nodes in the largest layer \([7]\).

Normally, in layered digraphs, nodes are drawn on horizontal lines. Particularly, nodes in layer \( L_i \) are drawn on the horizontal line \( y=i \) (see figure 26).

![Figure 26: Example of a layered digraph](image)

In order to get a readable layered digraph, three conditions have to apply:

1. *The layered digraph should be compact, which means, the height and the width of the digraph should be as small as possible, and the distance between layers is a constant.*

The longest path layering technique achieves the minimization of height. It does so by placing all sinks\(^7\) in layer \( L_1 \), and then each remaining node is the next layer and so on. Though this technique minimizes height in linear time, it has the disadvantage that it may yield digraphs of large widths.

One of the algorithms which tries to minimize the width of a graph to a positive integer W is the Coffman-Graham-Layering algorithm. It achieves this and tries to keep the height as small as possible. This, however, doesn’t solve the problem where we want to keep a digraph of a specific width W and height H. This problem unfortunately occurs in a problem which is similar to the problem we are trying to solve in this thesis. Particularly, the precedence-constrained multiprocessor scheduling problem, which is to assign each task to one of W processors, so that all tasks are completed in time H. This can be done if and only if G has a layering of width W and height H. Unfortunately, this problem of finding a layering with minimum width subject to having minimum height is an \( \text{NP}^8 \)-complete problem and is out of the scope of this thesis. However, as

\(^7\) A sink is a node that leads to the longest path in a digraph, basically it is like a root in a tree, which means, it doesn’t have any ingoing edges, and all edges are coming out of it.

\(^8\) Non-Deterministic Polynomial Complete is a term used in complexity theory to identify a particular class of problem. In an \( \text{NP} \)-complete problem, the relationship between the number of input parameters to the problem and the problem complexity is exponential \([39]\).

35
visualizing the scheduled tasks dynamically is a part of this project, this problem will be tackled later on in chapter 4.

2. *The layering should be proper, which means, the crossing of edges should be minimized, and the edges should be as readable as possible. To handle these two requirements, dummy nodes are needed to break up long edges in a layered graph* (see figure 24).

![Layered Digraph with Dummy Nodes](image)

**Figure 27:** A layered digraph that contains dummy nodes to break up long edges

Minimizing the crossings among the edges of a layered digraph can be done by modifying the order of the nodes in each layer. Though this definition simplifies the process of reducing the crossings, it is still not simple at all; it is actually an NP complete problem. This problem comes from the fact that nodes of some layers, if not all, are related to each other. Therefore, if we are to modify the order of the nodes in layer 1 and 2 for example, it may minimize the crossings between these two layers but add more crossings amongst the edges of the proceeding two layers, layer 3 and 4. In fact, the problem remains even if there are only two layers.

A variety of solutions for this crossings problem were suggested in the literature. The basic one is the layer-by-layer sweep. In this technique, a node ordering of layer $L_1$ is chosen. Then, for $i=2,3,\ldots,h$, the node ordering of layer $L_{i-1}$ is held fixed while reordering the nodes in layer $L_i$, to reduce the crossings between edges whose endpoints are in layer $L_{i-1}$, and layer $L_i$. Another variation of this algorithm is to hold the node orderings of $L_{i-1}$ and $L_{i+1}$ fixed, while reordering the nodes of layer $L_i$ to reduce crossings between layers $L_{i-1}$, $L_i$, and $L_{i+1}$.

One of the common used algorithms to reduce the crossings amongst two layers $L_1$ and $L_2$ is the adjacent-exchange algorithm. In this algorithm, we simply choose an initial order for $L_2$, then we scan the nodes of $L_2$ from left to right, exchanging an adjacent pair $u,n$ of nodes, whenever $c_{un} > c_{nu}$, and then we repeat this process until the number of crossings does not reduce anymore [3]. This algorithm unfortunately has the disadvantage of being $O(|L_2|^2)$. A slightly modified algorithm called split. In this algorithm, we choose a pivot node $p \in L_2$, and place each node $u \neq p \in L_2$ to the left of $p$ if $cup < cpu$, and to the right of $p$ otherwise. Then the algorithm is applied recursively to the sets of nodes to the left and right of $p$. This algorithm has worst case time complexity $O(|L_2|^2)$ but, in practice, it runs in $O(|L_2| \log |L_2|)$ time. There are also some that run in linear time but they are not covered here due to their complexity and unimportance to this thesis (see [7] for more details).
3. *The number of dummy nodes should also be as small as possible. This is due to the fact that its easier for the eye to read short edges, bends occur at dummy nodes position and the more bends we have, the less readable a graph is.*

An algorithm has been mentioned in literature to tackle this problem. It suggests moving a node downward. This, however, is conditional. For a node to be moved downward, the number of out edges should be smaller than the number of inner edges [7].

**Brief Analysis**

This thesis is concerned with drawing the way tasks and messages are allocated to processors and networks dynamically. Layered drawing of Digraphs is an excellent approach to simulate this dynamic allocation since it gives us ability to simulate time using layers (e.g. time T1 means L1 and so on), and tasks or messages changes through the relations between tasks of each layer (e.g. message M1 at time T1 was at position P2, at time T2 it became at position P5). However, as we discussed, care should be taken to provide the three important characteristics. If they can’t be achieved to a high extent altogether, then they should be prioritized (*Refer to 4.1.2.3 for more discussion*).
2.3.4 Graph colouring:

In the previous sections we discussed basic information about graphs. Then, we went on techniques for drawing these graphs. We never, however, mentioned anything about the colouring of these graphs. This section provides brief information about it.

Graph colouring is the assignment of colours to certain objects of a graph (see figure 25). These objects can be nodes, edges, faces, or mixture of those [40]. The assignment of these colours is used to make a graph readable in such a way that no adjacent nodes, vertices, or faces will have the same colour. Colouring nodes is the basic problem of graph colouring since colouring edges or faces can be also thought of as node colouring problem. Finding the minimum number of colours to colour a graph is unfortunately an NP-complete problem [40].

![A coloured graph with 3 colours](image)

Figure 28: A coloured graph with 3 colours, adapted from [41]

**Brief Analysis**

As figure 29 shows, the colouring of the nodes of this graph doesn’t take into account that some nodes are different or similar to each other. The only issue is considered here is to show the graph in such a way that no adjacent nodes will have the same colour. In this project, however, we are concerned with a graph which has different types of entities, or objects; processors, networks, tasks, and messages. If we deal with all these objects similarly, then we will end up in a confusing graph. Figure 29 shows a graph that contains three networks N1,N2, and N3, and two processors P1, and P2, and three messages M1,M2, and M3 and six tasks T1,T2,T3,T4,T5, and T6. The graph also shows the relations between these objects according to table 1. One way of colouring this graph would yield into messages and processors having the same colour, and tasks and networks having the same colour too. Another way might give different colours to networks themselves. Therefore, Colouring a graph in a general way, overlooking the structure of the graph nodes, may result in a confusing visual output.
Figure 29: A coloured graph which contains networks, tasks, messages, and processors
2.4 Case studies

Unfortunately, we couldn’t find case studies which are highly related to this project. However, there are examples which visualize relational information. We will discuss two of them in this section.

2.4.1 Circuit diagram

Figure 30 shows an example of these examples. It is a visualization of a circuit diagram by aiSee graph visualization company [42].

![Circuit Diagram]

**Figure 30:** aiSee circuit diagram case study, adapted from [42]

**Brief Analysis**

The graph uses orthogonal drawing technique to visualize related objects. Different sets of objects are related together in separate groups. Objects related together inside groups use greenish light blue edges. However, when they interact with objects out of their groups they use red edges. Two interesting notes can be made about this graph.
Firstly, grouping related objects in yellow boxes gives structurability to the diagram (See 2.3.3.2, orthogonal drawing). Secondly, the minimization of bends in the graph has not been optimized since there are a lot of them especially in the red edges that could have been easily avoided (See 2.3.3.2, orthogonal drawing).

2.4.2 History of computer programming languages graph:

Figure 31 shows another example of a graph that shows the history of computer. The graph also was generated by aiSee.

![Graph of computer programming languages history](image)

**Figure 31**: aiSee programming languages case study, adapted from [43]

**Brief Analysis**

The graph uses straight-line drawing (See 2.3.3.2, straight-line drawing) as well as curve drawing technique to visualize related objects. Interesting notes can be made about this diagram. Firstly, different objects had different shapes to visualize them (See 2.2.2, Visual Mapping). Secondly, objects which belong to one family were coloured by a single colour. Thirdly, the use of straight-line drawings as well as curves may help with utilizing the area of the graph. However, it is reducing the readability of the graph. Finally, the use of different colours for edges conveys different meanings.
2.5 Conclusion

Chapter 2 gave an important relevant knowledge needed to work out the solution of our visualization problem. It discussed some issues, challenges, and techniques of visualizing information, especially visualizing large amount of information. The chapter then went specifically in visualizing graphs since they are used to visualize relational information. Issues of graph drawing were discusses. Brief analysis was carried out as comments on each graph drawing techniques which were mentioned. Finally, two case studies were briefly analyzed.

To conclude, the issues of information visualization can be considered as a general guidance. The graph drawing techniques are also helpful. However, not any of them can be directly implemented to solve our particular visualization problem. This is due to the fact that most of them are applied for graphs that have only one set of nodes. In other words, there is no difference between a node and another in the sense of these techniques. In our project, however, we are dealing with four different groups nodes, and it would more readable if we deal with each node based on the group it belongs to. Therefore, in chapter 4, the design chapter, we will try to benefit from the techniques and issues that have been discussed to provide the solution for our visualization problem based on the requirements of chapter 3.
3 Requirements

This chapter describes the requirements specification, system specification, functional and non-functional requirements of the project. The chapter is intended to be a guideline for designing and implementing the visualization system. The conformance of the system will be examined at the evaluation phase.

3.1 Introduction

The requirements provided in this chapter are elicited from the client Mr. Paul Emberson, the designer of the Task and Messages allocation tool. Some of these requirements evolved a bit with time as the project progressed. This is due to the feedback of the client along with other stakeholders. These requirements are intended to provide a framework and scope to guide the project in successive phases, and to provide an unambiguous system interpretation communication between the client and the developer. These requirements will act as a test of fulfilment on the completion of the project. The requirements are traceable and easy to follow.

3.2 Scope

The main purpose of the software program which we are trying to design is to relations amongst tasks, messages, networks, and processors. The software program will provide this visualization through an output graph file. The design of a graph reader is out of the scope of this project. Furthermore, the software excludes the design of any database. The system consists solely of the software implementation of the system – hardware, OS platform and JRE provision are outside the scope of the project.

3.3 Definitions, Acronyms and Abbreviations

- User: The person/s who will use the software once it is completed.
- TMAV: Task and Message Allocation Visualization System.
- FR: Functional Requirement.
- NFR: Non-functional Requirement.
- System: TMAV System.
- Major Basic Functionality: Refer to 3.5 Product Functions.
- The Client: Mr. Paul Emberson, the designer of the allocation tool.
3.4 **Overview**

The rest of the chapter provides an overall description of the product in terms of its prospective, functions, user characteristics, constraints and assumptions. Also, it lists the functional and the non-functional requirements of the system as well as the other requirements. Dependencies amongst requirements as well as rational behinds some requirements are provided. Furthermore, priorities to each requirement are specified based on feedback from the client. Finally, the document also includes supporting information for further reference.

3.5 **Product Characteristics**

3.5.1 **Product Perspective:**

The product is dependant on the output and the input of the Tasks and Messages Allocation Tool. The input of the tool is the XML system file which contains all objects of the system along with their attributes, it also contains the permanent associations with objects, which are, the mapping of messages to tasks as well as the mapping of processors to networks.

3.5.2 **Product Functions:**

The main functionality of the system is the provision to visualize the relations of the tasks, messages, processors, and networks with each other (*See Table 1, 1.1.2*).

3.6 **User Characteristics**

1. The typical user will have used a computer for basic tasks before and be capable of operating a standard keyboard and mouse.
2. The typical user will be between 16 and 60 years of age.
3. The typical user will have no significant physical disability that precludes him from using standard desktop computer components, including visual impairments etc.
4. No specific technical or IT skills are required to use the system.

3.7 **Constraints**

1. The software shall be able to run on a standard desktop operating system (*circa year 2000+*) that supports a standard Java run-time environment (*Java 1.5*).
3.8 Assumptions

1. It is assumed that the host computer will have a Java run-time environment available to hosted applications (Java 1.5+).
2. It is assumed that the user is familiar with general software.
3. It is assumed that the user will be able to provide a graph language software reader\textsuperscript{10}.
4. It is assumed that the input of the TMAV system is free from bugs (i.e. the xml format is correct, the ids are checked against duplication, etc).

3.9 Stakeholders

1. Client
2. Real-time specialists
3. Computer graphs specialists\textsuperscript{*}
4. Installation Engineer
5. System Developer
6. Authority\textsuperscript{*}
7. Printing System

3.10 Requirements Specification

3.10.1 Functional Requirements

- **FR1**: The system shall visualize processors to networks mapping.
  - *Elicited from*: Client
  - *Dependencies*: None.
  - *Priority*: High.

- **FR2**: The system shall visualize messages to tasks mapping.
  - *Elicited from*: Client.
  - *Dependencies*: None.
  - *Priority*: High.

- **FR3**: The system shall visualize the tasks allocated to processors.
  - *Elicited from*: Client.
  - *Dependencies*: FR1, FR2.
  - *Priority*: High.

- **FR4**: The system shall visualize the messages allocated to networks.
  - *Elicited from*: Client.
  - *Dependencies*: FR1, FR2.

\textsuperscript{10} Examples include Yworks, and Graphlet.
\textsuperscript{*} Unfortunately, these stakeholders couldn’t be provided for this project during the lack of time and the difficulty of providing them for this project
• **Priority**: High.

- **FR5**: The system shall visualize the change of task allocation to processors over time.
  - **Elicited from**: Client.
  - **Dependencies**: None.
  - **Priority**: Low.
  - **Rational**: It has been agreed with the client that the fulfilment of this requirement is unimportant. Fulfilling is considered to be a bonus.
  - **Fit Criteria**: The system shall refresh the visualization each 5 seconds.

- **FR6**: The system shall visualize the change of message allocation to networks over time.
  - **Elicited from**: Client.
  - **Dependencies**: None.
  - **Priority**: Low.
  - **Rational**: It has been agreed with the client that the fulfilment of this requirement is unimportant. Fulfilling is considered to be a bonus.
  - **Fit Criteria**: The system shall refresh the visualization each 5 seconds.

- **FR7**: The system shall show the attributes associated with each object.
  - **Elicited from**: Client, Users.
  - **Dependencies**: FR1, FR2, FR3, FR4.
  - **Priority**: High
  - **Rational**: This requirement is elicited after discussing with the client the usability issues of large graphs (See 2.2.3.3).

- **FR8**: The system shall allow the user to search easily for an object.
  - **Elicited from**: Client, Developer.
  - **Dependencies**: FR1, FR2, FR3, FR4.
  - **Priority**: High.
  - **Rational**: This requirement is elicited after discussing with the client the usability issues of large graphs (See 2.2.3.3).

- **FR9**: The system shall visualize the tasks sorted according to their priorities.
  - **Elicited from**: Client, Developer.
  - **Dependencies**: FR1, FR3.
  - **Priority**: Low.
  - **Rational**: This requirement is elicited after discussing with the client some insight issues (See 2.2.3.1)

- **RF10**: The system shall read the data to be visualized from a two xml files, a system xml file and a configuration xml file.
  - **Elicited from**: Client.
  - **Dependencies**: FR1, FR3.
  - **Priority**: High.

- **RF11**: The system shall present the output as a graph file.
  - **Elicited from**: Client.
  - **Dependencies**: FR1, FR3.
- **Priority**: High.
- **Rational**: It was agreed with the client that the system will provide a graph file as a visual output (See Providing a GML graph format file, 4.1.7.2).

- **RF12**: Each set of objects should be visualized in a clear separate group.
  - **Elicited from**: Real-time specialists.
  - **Dependencies**: FR1, FR2, FR3, FR4.
  - **Priority**: Intermediate.
  - **Rational**: Researchers pointed out that structurability helps in reading the visual output easier and quicker (See 2.33.2, Orthogonal drawing).

### 3.10.2 Non-Functional Requirements

#### 3.10.2.1 Performance Requirements

- **NFR1**: The system shall process events at a reasonable speed.
  - **Elicited from**: Client.
  - **Priority**: Low.
  - **Fit Criterion**: The average processing time of running the software will not take more than 2 seconds.

#### 3.10.2.2 Usability Requirements

- **NFR3**: The output of the system should be readable by users.
  - **Elicited from**: Client, Real-time specialists.
  - **Priority**: High.
  - **Fit criteria**: Most of the users should be able to understand easily the output graph by quickly reading the manual. They should be able to quickly find a relation they are interested in. This means: Users should easily find the networks that are connected to a particular processor and vice versa. They also shall be able to find the tasks allocated to a particular processors and vice versa. The system shall help in finding the allocated messages to a particular network and vice versa. Finally, the system shall help them to find the messages associated with a certain task and vice versa.

- **NFR4**: Most users shall like to use the system.
  - **Elicited from**: Client, Developer.
  - **Priority**: High
  - **Fit Criterion**: 90% of users extensively acquainted with the functionality of the system shall rate the system as a positive experience during use.

- **NFR5**: The system shall visualize around 50 tasks and several hundreds of messages in a readable manner.
  - **Elicited from**: Client, Developer
  - **Priority**: Intermediate.
  - **Fit Criterion**: 90% of users should be able through the usability implemented techniques read the graph as specified in NFR1 fit criteria (See 2.2.3.3).
• **NFR6**: Through the system, a relevant researcher can identify interesting patterns.
  - *Elicited from*: Client, Developer.
  - *Fit Criterion*: 90% of relevant researchers should be able to quickly identify which networks have the largest or the smallest number of messages, and which processor have the largest or the smallest number of tasks *(See 2.2.3.1)*.

• **NFR7**: Through the system, a relevant researcher can identify the errors of the tool.
  - *Elicited from*: Client, Developer.
  - *Priority*: High.
  - *Fit Criterion*: Most of relevant researchers should be able to quickly identify what message will not be received because it is sent to a task which doesn’t reside on the same network the message is sent from.

• **NFR8**: The system shall report an exception to the user.
  - *Elicited from*: Client, Developer.
  - *Priority*: Low.
  - *Fit Criterion*: If an input file is not specified or its content is not proper, or if the time of the dynamic solution is negative or infinite, then the system shall report the bug and show it to the user. If any other unexpected exception happens, the system shall report it to the user as well.

3.10.2.3 Maintainability Requirements

• **NFR9**: The software should be written in a way that is moderately extensible.
  - *Elicited from*: Developer.
  - *Priority*: High.
  - *Fit Criterion*: The system shall be modular and with clearly defined functionality and interfaces so that later extensions may be made.
4 Implementation

This chapter describes is structured into two sections; design and testing.

The design section discusses briefly the chosen software model as well as alternatives. Then, the evolution of the design prototype of the software is discusses with insightful analysis based on the requirements, the feedback of the users, and the literature review. As the section advances, the design details go in depth. First, high level designs are provided such as use case diagrams and conceptual design. Then, low level design such as class diagrams, interface description, collaboration diagrams etc are shown. Finally, brief discussions and analysis are made to cover major significant decisions, and issues.

Second section provides a testing for the system based on the functional requirements. Several test cases are provided with brief descriptions.

4.1 Design:

This section gives a concrete description of all the design phases. It states the design decisions that have been taken and the reasons behind these decisions. It also gives a brief explanation of the available alternatives and provides critical analysis of each of them. In this section, it will be shown clearly how the requirements, the relevant literature, and the input of the users are reflected on the design decisions.

4.1.1 Software model

4.1.1.1 Spiral software model:

The spiral software model was decided to be the model of this software project. In simple words, the spiral model is defined as a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts [45]. As figure 32 shows, the phases of the model as any other model; requirements analysis, primarily design, details design, coding, and testing. Each phase of this model starts with the objectives of the design and end with the client.
The three basic reasons why this model has been chosen are:

1. Takes the feedback of the user into a great account. And this, of course, is one of the principles of the visualization of information (see 2.2.2) and particularly the visualization of graphs (see 2.3.3 a).
2. Suits developers who are not highly experienced. Considering this project is a research project, I of course didn’t have a great experience of information visualization before I took onto this project.
3. As the project is a visualization project, the customers can not always precisely describe what they want till they see it. This model provides them prototypes through which they can better judge what they like and dislike in the design.

4.1.1.2 Alternatives:

One of the alternatives for the spiral software model is the waterfall model. This model is easy to follow, enforces discipline approach, and emphasizes the documentation. However, this model doesn’t have a great flexibility for the requirements to change. Furthermore, it doesn’t engage the user into all the phases of the project.

The incremental software model was another alternative for this project. It is similar to the waterfall model with the exception that it allows the phases of the model to overlap with each other. Therefore, like the spiral model, it does not require a complete set of requirements at once. The model also offers the advantage of providing early functionality. However, it still doesn’t provide the prototyping feature the evolutionary model does.
4.1.2 Prototype design:

This section first discusses why orthogonal drawing technique was chosen as a technique for developing the visualization design. Then it gives a brief description of how the visualization prototype design evolved according to the feedback of the users as well as the feedback of the users.

4.1.2.1 Options of drawing techniques:

Table 3 makes a brief comparison amongst the graph drawing techniques that have been discussed in literature in section 2.3.3

<table>
<thead>
<tr>
<th>Technique name</th>
<th>No Bends (Refer Aesthetics)</th>
<th>Maximization of minimum angle between edges (Refer Aesthetics)</th>
<th>Readability (Refer Usability requirements)</th>
<th>Structurability (Refer Usability requirements)</th>
<th>Suitability for all relations of the project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line drawing</td>
<td>No bends</td>
<td>Algorithms try to achieve it but never 100%</td>
<td>Readability decreases as graph grows</td>
<td>Difficult to structure as graph grows</td>
<td>Suitable</td>
</tr>
<tr>
<td>(Refer to analysis)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orthogonal drawing</td>
<td>Suffers from bends</td>
<td>Achieves it 100%</td>
<td>Achieves high readability</td>
<td>Easily structured</td>
<td>Suitable</td>
</tr>
<tr>
<td>(Refer to analysis)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trees drawing</td>
<td>No bends</td>
<td>Not a concern as readability is achieved</td>
<td>Achieves best readability</td>
<td>Easily structured</td>
<td>Only suitable for one-to-many relation</td>
</tr>
<tr>
<td>(Refer to analysis)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forced-based drawing</td>
<td>-</td>
<td>Not a concern</td>
<td>Achieves reasonable readability</td>
<td>Difficult to structure as graph grows</td>
<td>Suitable</td>
</tr>
<tr>
<td>(Refer to analysis)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Drawing techniques comparison

Looking at table 3, we can conclude that orthogonal drawings are the most suitable for this project. However, two issues need to be considered here. Firstly, bends must be minimized as much as possible. Secondly, the orthogonal drawing techniques can’t be directly implemented for this project. This, as mentioned in the literature review conclusion, is due to the fact that graph drawing techniques are developed for graphs that have one set of similar nodes. In this project, however, we have four sets of objects. Therefore, we need to adopt a technique which achieves the advantages of orthogonal drawing techniques, the requirements of the project, and gives us ability to visualize different sets of objects and relations amongst them. The next section will discuss the evolution of our visualization problem.
4.1.2.2 Prototype design Evolution for static solution:

First Design Attempt:

Figure 33: First Attempt for visualization

Figure 31 shows the first attempt to visualize the relations in table 1 as an orthogonal drawing. Processors are visualized as coloured big circles that are drawn on vertical lines. Any spot on these vertical lines means a connection with a network. Networks are represented by horizontal lines. A connection between a processors and a network is represented by a cross. Tasks are visualized as vertical branches from processors. These vertical lines come on the side of the horizontal network lines. This gives messages a chance to be associated with their networks and the tasks as well.

In this figure, we have three processors, seven networks, three messages, and fifteen tasks. The first processor P1 is connected to the networks N1, N4, N6, and N7. The second processor P2 is connected to networks N2, N4, N5, and N7. The third processor P3 is connected to networks N3, N5, N6, and N7. Tasks T1...T8 are allocated to processor P1. Tasks T9...T12 are allocated to processor P2. The rest are allocated to processor P3. Message M1 is sent from T1 to T3 from N3, M2 is sent from T2 to T4 from N3, and M2 is sent from T4 to T8 from N6. Distances between processors, tasks, and networks can be flexibly changed according to the design.

Analysis

This graph fulfils the functional requirements of the project. In addition to that, it is easy to identify four separate sets of objects. Therefore, this graph achieves structurability (See 3.10.1, FR12). However, this is not absolutely true as there might be confusion in a situation like message M2. This message is drawn right above M1 which is sent from N3. As M2 is close to the horizontal line of N4, it might be confused especially if we have a lot of messages in the graph. Another concern with
this representation is the vertical grey lines which come out of each single message to map them with the associated task. These lines intercept the horizontal lines of networks and make the graph sort of clutter.

When shown to the client of the software, he commented that for a small amount of objects, the relations between objects can be detected. However, the following of relations becomes more difficult as the number of objects increases. This comment is true as the grey message vertical lines increase as more messages we have. This becomes a crucial problem when we have hundreds of messages to visualize (Refer to 3.10.2.2, NFR5).

Another problem another stakeholder identified with this graph is the poor choice of colours. He says it confused him and made it difficult for him to follow the relations. What creates such a problem is the use of many colours, and particularly, those which are close to each other, like red and orange.

Same stakeholder suggested that the graph should not emphasize the processors networks relations but rather, the tasks messages relations as well as the allocations.

**Second Design Attempt**

![Second attempt of visualization](image)

**Figure 34:** Second attempt of visualization
Figure 32 shows the second attempt to visualize the relations in table 1 as an orthogonal drawing. Processors are visualized on the bottom most left side as light blue circles. These blue circles allow horizontal lines coming from its right side so that it enables them with the networks. The blue circles also have vertical lines which connects the tasks. The tasks, which are represented as orange circles, have horizontal lines from which they can be connected to messages, the pink circles. Again as with the previous design, distances between objects are variable. The design allows changing them.

Analysis

Like the first graph above, this graph fulfils the functional requirements of the project. According to the same stakeholders again, the enhancements which have been made to this graph include: processors networks relations have the same importance as any other relations, the use of colours is better as each set of objects have a unique colour, there is no problem if two messages come from one network (*See MSG4 and MSG3*), the relations between tasks and messages is neater and easier to understand. Another important advantage this design provides is the extensibility (*See Structure of packages, Allocation Process, 4.1.7.2*).

One of the stakeholders again raises the issue of the difficulty of following the lines if we have to deal with a lot of objects in the graph. The flexibility of using horizontal and vertical lines enables us to add additional labels to the graph to make things clearer. For example, in figure 32, MSG4 label has been added on the message more readable. Final note on the graph, from an orthogonal drawing perspective, tasks and messages impose bends on the graph. Figure 33 shows these bends. The presence of these bends increase the area of the graph, and also adds more complexity for the graph and decreases its readability.

![Figure 35: Bends of second attempt of visualization](image)
Third Design Attempt:

The third attempt is slightly different from the second one. Enhancements are discussed in the analysis.

![Diagram](image)

**Figure 36:** Third attempt of visualization

Analysis:

Figure 36 shows the third attempt to visualize the relations in table 1 as an orthogonal drawing. It certainly doesn’t differ a lot from the second attempt. The only difference is the minimization of the task bends in the second attempt. It also contains red lines which detects a bug of an algorithm (See 3.10.2.2, NFR7). These red lines detects if a message is allocated wrongly. The remaining criticism for this algorithm is that the nodes don’t have a proper visual mapping. In other words, these coloured circles don’t convey the meaning of these objects. Another disadvantage is that as messages increases, a lot of area is occupied. This is due to the fact that each message requires a single horizontal line. No two messages can be visualized in the same horizontal segment. Final note a stakeholder made is that the relation between processors and networks still seem a bit difficult to comprehend. This may be the case due to the fact that we have some unwanted crossings.
Fourth Design Attempt:

The Fourth design attempt aims at including all the enhancement suggestions of the clients, the features of the previous design attempts, and the functional and non-functional requirements of the project.

Analysis:

As figure 37 shows, this attempt offers some enhancements. These include: the relation between the tasks and messages is easier to understand, the drawing of the messages is clearer as there is only one object to represent the message rather than two objects in the previous designs. Another important notice is that messages are grouped clearly in boxes so that they can be easily associated with their networks. Users liked this graph much better but they still had few comments about it. Firstly, tasks of processor P1 came after tasks of processor P2, and this causes 4 crossings and made it less readable. Secondly, the more processors we have, the further processors are from networks and the further tasks are from messages. Though this is true, the representation of processors and tasks this way reduces readability. The alternative for this is drawn in figure 38 where all processors and tasks are drawn on the same vertical segment. Though this gives them the advantage of being close to messages and networks, from an orthogonal perspective, it is more difficult to follow the relations between tasks and processors. Figure 36 visually compares the two different alternatives. It also shows how the distance problem is solved in alternative B. the
flexibility of the horizontal lines makes it easy for us to add extra labels for the far tasks and processors. In addition to that, according to the client, usually there are not many processors to visualize.

Figure 38: Two alternatives of visualizing tasks and processors

Last problem which users identified in this example is that the areas of messages are not fully utilized. For example, in the most left box of messages, only the top of it is utilized. The second most left box has its bottom part utilized. As mentioned in the literature review, the minimization of area is one of the graph aesthetics. In fact, the drawing of messages also utilizes more area than the previous designs. Example, MSG1 and MSG2 can be drawn on the same vertical segment. In spite of that, we still have some empty areas in the graph. The client of the software emphasized that these areas should be left since they gave him an insight. The insight, he was talking about,
is that through these areas he can know where more messages are sent and what networks have more messages. This helps him evaluate and criticize the quality of the solution. As we’ve discussed in the literature view, insight, is a very important thing in information visualization.

Moving from users to literature, graph theory suggests another problem with this visualization design. This problem is, that since the relation between processors and networks is a many-to-many relation, crossings may happen. Minimizing the crossings is one of the most important aesthetics in graph theory. Consequently, the design should take this into account. Another issue with this graph is how well it can handle a large number of objects. The usability of visualizing large amount of information was discussed in chapter 2 (Refer to 2.2.3.3). In short words, techniques like search, clustering, selecting objects, and zooming in and out helps in coping with this problem. Therefore, the design should allow users to search (Refer to 2.2.3.3), zoom in and out, filter, and select objects. The solution this design provided takes some of these aspects in account. Firstly, it shows the objects as entities which have their names on it. It doesn’t show any additional attributes with these objects. Secondly, the visualization of messages takes as less area as possible but taking into account that they must be structured in such a way that they will be clearly readable to the user (See Message sorting, 4.1.7.2). Thirdly, the design clearly takes search into account. However, issues of zooming in and out, as well as interacting are left to software graph readers. As agreed with the client of this project, TMAV system will provide a file which has this design written in a graph language. This is due to the fact that these software programs have been tested and offer a great deal advantages for reading graphs, especially for visualizing large graphs. Yworks is an example of these software programs (See figure 38). Filtering, however, is an issue that these softwares don’t provide. Therefore, the design should take this into consideration and allow the user to show the relation he is interested in. If for any reason, the user thinks the presence of two many messages makes the graph clutter, then perhaps showing only tasks, messages, and processors makes the graph much easier to comprehend.

Table 4 summarizes the advantages the prototype has and the issues need to be considered in the design of this prototype:

<table>
<thead>
<tr>
<th>Advantages provided by software readers</th>
<th>Advantages</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Zoom + Pan. (Refer to )</td>
<td>1. Fulfiling the functional requirements (Refer to FR)</td>
<td>1. Minimization of crossings in the processors networks</td>
</tr>
<tr>
<td>2. Interaction. (Refer to )</td>
<td>2. Readable task message relation. (Refer to NF)</td>
<td></td>
</tr>
<tr>
<td>3. Focus + Context (Refer to )</td>
<td>3. Structurability (Refer to )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Minimization of bends (Refer to )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Flexibility of adding labels</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. A message is visualized as only one entity.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. More than a message can be drawn on the same vertical segment (Refer to)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8. Visualize only object IDs, and attributes can be retrieved by selecting.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9. Visualize messages which are allocated wrongly (Refer to)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10. Insight (Refer to )</td>
<td></td>
</tr>
</tbody>
</table>
problem. (Refer to).
2. Minimization of area which is occupied by messages while preserving the structurability of these messages. (Refer to)
3. Clustering.

Table 4: Summary of advantages and issues of the fourth prototype

4.1.2.3 Prototype design Evolution for dynamic solution:

The previous section discussed the evolution for the prototype of the static solution, which is, a high requirement. This section tries to carry out similar analysis to the dynamic solution. However, it should be noted that this requirement is secondary.

First Prototype Attempt:

Figure 39 shows the first attempt of a prototype that visualizes the dynamic solution of the visualization problem. This solution takes an advantage of what has been discussed about layered graphs in the literature review (Refer to). It in a way it suits this problem as the layers we need are the time layers and because in the layers, arrows indicate the change that each task goes through from a layer to another. Each task that changes is given a colour, and then the arrow leads to its new position. Tasks that don’t change are given a white colour. Processors are visualized as rectangles which group all what is under them. So for example, T1, T2, and T3 are associated with P1 at t1. T2, T9, and T4 are associated with the same processor at t2. This solution can be utilized to visualize dynamic allocation of messages to networks as well; messages are just needed to be substituted with tasks, and networks with processors.

![Figure 39: First attempt to visualize the dynamic solution](image)

Analysis
Unfortunately there wasn’t a sufficient time within this project to get the feedback of the clients about this prototype. However, from a graph theory perspective, The use of straight lines to visualize changes make the graph clutter as the number of tasks and changes increases (See 2.3.3.2), and also as the distance between the old task’s position and its new one increases. We can, however, cope with this problem by reducing the crossings as much as we can by implementing one of the techniques that have been discussed in the literature (see 2.3.3.2, digraphs layouts). However, this is not realistic as these problems assume that we can change the position of these tasks as we please. In this solution, actually we have restrictions. For example, we know T1, T2, and T3 are associated with P1 from their position, so we can’t shift T1 under P2. However, we can change the position of these tasks as long as they stay under their processors. This of course may help reduce the crossings but not good enough though.

The use of colours for the changing tasks is good to identify the new position of these tasks, and we can even get rid of arrows. However, as more changing tasks we have, as it’s difficult for us to have distinct colours to represent these changing tasks. In this project, we expect to have a large number of tasks and messages; therefore, the use of colours is not the ideal solution here.

**Second Prototype Attempt**

Figure 40 shows the second attempt of a prototype that visualizes the dynamic solution of the visualization problem. This solution translates the solution provided in the first design attempt into an orthogonal graph. Time lines now are vertical segments. Processors are horizontal rectangles. Tasks changes are represented by an arrow which indicates the task change to the other position. Only two colours are used for tasks. White is for the unchanging ones, and light blue for the changing ones.
**Figure 40:** Second Dynamic prototype

**Analysis:**

As we mentioned earlier when we chose orthogonal drawings to be the technique for the static solution, orthogonal drawings are known for their high clarity especially if there are no bends. Also, since the vertical arrows can’t be drawn over each other. They avoid the problem that the previous example had which is having straight lines intercept each others and causing crossings. However, we still get crossings here too as we need to connect the old task’s position with its new one. With all this in mind, the relations are still easier than the previous example’s to follow.

The use of two colours also reduces the confusion of the reader and makes the graph cleaner and easier to comprehend.

Unfortunately, because of the lack time, this solution will be taken into consideration in the design in the early phases but will not be implemented.
4.1.3 Use cases design:

In this section, the design of the use cases is shown. The use cases shown were designed after the last prototype design was developed. However, their design is not highly dependant on the prototype design as they are more related with the necessary functionality to visualize the system. This section provides the reader with the top and high level use case diagrams of the project. For both types of use cases, it will be shown clearly how the requirements are mapped to these use cases.

The main four primary actors that the use cases have are:

- **USER**: This actor represents the user of the system.
- **DATA-STORE**: This actor represents the raw information that is required to be visualized.
- **ALLOCATION-STORE**: This actor represents the structured information that is ready for the visualization process. In other words, it is the basic data structure that the system deals with for retrieving information.
- **VISUALIZATION-SYSTEM**: This actor is the system which takes the structured information and restructures it in such a way that is ready to be presented as a visual output.
- **FILE-PROCESSING**: This is the system that is responsible for opening, writing, and refreshing files which the system deals with.

4.1.3.1 Top Level Use case Diagram

![System Overview Diagram](image)

**Figure 41**: Use case 1, run

**Use Case 1**

**Name**: run

**Context**: In this use case, the user runs the whole system. The user provides the file names of the xml input files as well as the type of the visualization he wants, whether it is static or dynamic. This running will trigger some of the rest of the other use cases and the actors in the system so they perform the visualization of the system. In brief,
when the user runs the system, the file processing actor store will performs its use case to get the raw data and store it, then this data is visualized by the visualization-system, and finally the visual output is written as a file that is ready to be read.

**Primary Actor:** User.

**Trigger:** The user intends to run the system.

---

**Use Case 2**

**Name:** Parse-data.

**Context:** The input of the TMAV system is two xml files; the system file and the configuration file (*Refer to 3.10.1, RF10*). These two files are parsed through the file processing system and then saved in a structural way by the allocation-store. The allocation-store keeps this data so that it can later on be retrieved for the visualization process.

**Primary Actor:** file-processing.

**Trigger:** When the user runs the system, second step, the file-processing reads the information to be visualized from the data-store.

---

**Use Case 3**

**Name:** static-visualize
Context: In this use case, the visualization system asks the allocation-store to provide it with the information that is necessary to visualize. Once it gets it, it visualizes the objects of the system in a static manner.

Primary Actor: visualization-system.

Trigger: When the user runs the system and chooses a static visualization, this use case comes as a third step right after the parse-data use case.

Use Case 4

Name: dynamic-visualize

Context: In this use case, the visualization system asks the allocation-store to provide it with the information that is necessary to visualize. Once it gets it, it visualizes the first phase of the dynamic visual output. Then, it keeps asking the file-processing to read the data from the data-store every 5 seconds (Refer to 3.10.1, FR5, FR6). During these 5 seconds, the visualization system visualizes the objects.

Primary Actor: visualization-system.

Trigger: When the user runs the system and chooses a dynamic visualization, this use case comes as a third step right after the parse-data use case.
Use Case 5

**Name:** Present-data  
**Context:** In this use case, the visualization system provides the file-processing the visual output information. Once the file-processing gets it, it writes it as a file (Refer to FR11, 3.10.1).  
**Primary Actor:** File-processing.  
**Trigger:** When the user runs the system and chooses a dynamic or static visualization, this use case comes as a last step right after the static-visualize or dynamic-visualize use case.

4.1.3.2 Sub Level Use Case Diagram

- Parse-data

**Use Case 6**
Name: parse-sys-file
**Context:** In this use case, the allocation-store reads the system XML file (sys file) from the data store and saves it to the allocation store.

**Primary Actor:** file-processing

**Supporting Actor:** allocation-store and data-store.

**Trigger:** When the user runs the system, second step, the file-processing reads the information to be visualized from the data-store.

**Main Success Scenario:**
1. File-processing reads the sys-file and creates an allocation store.
2. allocation-store saves tasks, messages, processor, and networks as entities.
3. allocation-store saves the mapping between processors and networks, and messages and tasks.

**Exceptions:**
1. If the sys file doesn’t exist, an error message will be shown to the user, and the system will exit *(Refer to NFR8, 3.10.2).*

2. If the file exists, but its format is wrong, an error message will be shown to the user, and the system will exit *(Refer to NFR8, 3.10.2).*

Use Case 7

Name: parse-conf-file

**Context:** In this use case, the allocation-store reads the configuration XML file (conf file) from the data store and saves it to the allocation store.

**Primary Actor:** file-processing

**Supporting Actor:** allocation-store and data-store.

**Trigger:** When the user runs the system, second step, the file-processing reads the information to be visualized from the data-store.

**Main Success Scenario:**
1. File-processing reads the conf-file.
2. allocation-store saves the allocations of tasks to processors.
3. allocation-store saves the allocation of messages to networks.

**Exceptions:**
1. If the conf file doesn’t exist, an error message will be shown to the user, and the system will exit *(Refer to NFR8, 3.10.2).*

2. If the conf exists, but its format is wrong, an error message will be shown to the user, and the system will exit *(Refer to NFR8, 3.10.2).*

Static-visualize
Use Case 8

**Name:** visualize network-processor mapping

**Context:** In this use case, the visualization system asks the allocation-store to provide it with the structured information that is ready for the visualization process.

**Supporting Actor:** visualization system.

**Trigger:** When the user runs the system and chooses a static visualization, this use case comes as a third step right after the parse-data use case.

**Main Success Scenario:**
1. Visualization system retrieves the processors and networks.
2. Visualization system allocates locations for the processors and mapping.
3. Visualization system saves the visual information.

**Exceptions:**
2. If a processor doesn’t have a corresponding network or vice versa, it is not visualized, and is shown as a bug to the user (*Refer to NFR8, 3.10.2*).

Use Case 9

**Name:** visualize tasks

**Context:** In this use case, the visualization system asks the allocation-store to provide it with the sorted tasks.

**Supporting Actor:** visualization system.

**Trigger:** When the user runs the system and chooses a static visualization, this use case comes as a third step right after the parse-data use case.

**Main Success Scenario:**
1. Visualization system retrieves the tasks.
2. Visualization system allocates locations for each task, and links it with its corresponding processor.
3. Visualization system saves the visual information.

**Exceptions:**
2. If a task doesn’t have a corresponding processor, it is not visualized and is shown as a bug to the user (Refer to NFR8, 3.10.2).

Use Case 10

Name: visualize messages
Context: In this use case, the visualization system asks the allocation-store to provide it with the sorted messages.
Supporting Actor: visualization system.
Trigger: When the user runs the system and chooses a static visualization, this use case comes as a third step right after the parse-data use case.

Main Success Scenario:
1. Visualization system retrieves the messages.
2. Visualization system allocates locations for each message, and links it with its corresponding network, and task.
3. Visualization system saves the visual information.

Exceptions:
2. If a message doesn’t have a corresponding network or a task, it is not visualized and is shown as a bug to the user (Refer to NFR8, 3.10.2).

Dynamic-visualize

![Dynamic-visualize diagram](image)

**Figure 48:** The use cases of the Dynamic-visualize high level use case

Use Case 11

Name: static-relations-visualize
Context: In this use case, the visualization system asks the allocation-store to provide it with the static relations. These are the processors mapping to networks and the messages mapping to tasks.
Supporting Actor: visualization system.
Trigger: When the user runs the system and chooses a dynamic visualization, this use case comes as a third step right after the parse-data use case.

Main Success Scenario:
1. Visualization system gets static relations.
2. Visualization system visualizes processors connected to networks (Refer to FR3, 3.10.1).
3. Visualization system visualizes tasks mapped to messages (Refer to FR2, 3.10.2).
3. Visualization system saves the visual information.

Exceptions:
2. If a processor doesn’t have a corresponding network or vice versa, it is not visualized and is shown as a bug to the user (Refer to NFR8, 3.10.2).
3. If a message doesn’t have a corresponding task, it is not visualized and is shown as a bug to the user (Refer to NFR8, 3.10.2).

Use Case 12

Name: task processor allocator visualize
Context: In this use case, the visualization system asks the allocation-store to provide it with the current tasks allocated to all processors; it keeps refreshing every 5 seconds.
Supporting Actor: visualization system.
Trigger: When the user runs the system and chooses a dynamic visualization, this use case comes as a third step right after the parse-data use case.
Main Success Scenario:
1. Visualization system retrieves the allocated tasks
2. Visualization system visualizes the allocated tasks to the corresponding processors and displays its corresponding time.
3. Visualization system saves the visual information.
4. Every 5 seconds, refreshes the parse-conf use case (Refer to FR5, FR6, 3.10.1).
5. Repeat 1-4.
6. Exit after a time the user already provided.

Exceptions:
2. If a task doesn’t have a corresponding processor, it is not visualized and is shown as a bug to the user.

Use Case 13

Name: message network allocator visualize
Context: In this use case, the visualization system asks the allocation-store to provide it with the current messages allocated to all networks; it keeps refreshing every 5 seconds.
Supporting Actor: visualization system.
Trigger: When the user runs the system and chooses a dynamic visualization, this use case comes as a third step right after the parse-data use case.
Main Success Scenario:
1. Visualization system retrieves the allocated messages.
2. Visualization system visualizes the allocated messages to the corresponding networks and displays its corresponding time.
3. Visualization system saves the visual information.
4. Every 5 seconds, refreshes the parse-conf use case (Refer to FR5, FR6, 3.10.1).
5. Repeat 1-4.
6. Exit after a time the user already provided.

Exceptions:
2. If a message doesn’t have a corresponding network, it is not visualized and is shown as a bug to the user (Refer to NFR8, 3.10.2).
4.1.4 System architecture description:

4.1.4.1 Stakeholders and Concerns

The identified stakeholders who may be potential audiences of the architectural description include the followings:

**User**: The user of the system.

**Architect**: A person who is involved in the activities of designing, documenting the system architecture.

**Developer**: A person who carries out the practical activity of implementing and developing the required system in code.

**Client**: A person acting as the representative of the system customer. Mr. Paul Emberson the client for the TMAV system.

**Maintainer**: A developer who is responsible for maintaining the system and ensuring that future requirements of the system may be implemented as augmentations to the current code-base.

**Tester**: A person responsible for testing the system before it is delivered to ensure that it meets with the functional and non-functional requirements of the requirements specification. For this project, the developers of the system will also undertake this role.

The concerns of the stakeholders include:

**Purpose of the system**: The main functionality of the system the provision to visualize the relations of the tasks, messages, processors, and networks with each other as shown in Table 1 in Chapter 1. The major Quality Attributes expected by stakeholders are usability, availability, performance.

**Feasibility of the system**: The system is feasible as all hardware and software requirements are completely fulfilled.

**Associated risks**: The miscommunication or omission of user requirements, incorrect architectural analysis, inconsistent design and implementation, incomplete functionality, unsatisfied quality requirements, and unrealistic resource constraints, represent the likely risks of this system’s development and deployment.

**Maintainability, deployability and evolvability**: The TMAV system shall be easily maintainable during its lifetime. Moreover, it shall be capable of being extended to accommodate new features or increase its scalability in the future development.

4.1.4.2 View Points

**Viewpoint 1**

**Viewpoint Name**: Conceptual Structure

**Stakeholders**: Architect, Developer, Maintainer, Tester

**Stakeholder Concerns**:

--Describe the conceptual software components of the system, the interactions among components

--Indicate how the system decompose itself and achieve its functionality

**Vocabulary, Language, Modelling Techniques or Analytical Methods**:

--**Viewpoint Language**: UML diagrams
--Modelling Techniques: Components, connection, connectors, protocol
Source: Refer to Conceptual view
Rationale: The viewpoint is used to demonstrate the conceptual software components of a system, and their communications. Moreover, it can be used for performance, availability, security, functionality analysis. The viewpoint provides a blueprint of the system structure and measures whether the planned architecture fulfils the system requirements.
Additions: None

Viewpoint 2
Viewpoint Name: Module Elements
Stakeholders: Architect, Developer, Maintainer
Stakeholder Concerns:
--Show the system decomposes into subsystems, modules and layers
--Describe interfaces which provide services for other elements but hide private information
Vocabulary, Language, Modelling Techniques or Analytical Methods:
--Viewpoint Language: UML diagrams
--Modelling Techniques: Decompose, module, subsystem, interface, use, layer
Source: Refer to Module view
Rationale: The viewpoint is used to show the module decomposition and how a conceptual model can be realised in terms of various elements.
Additions: None

4.1.4.3 Views

Conceptual View:

Figure 49: Conceptual View of the system
Module View:

4.1.4.4 Consistency across views:

In fact, there is a correspondence between the two views. The functionality is realised in the conceptual view, which can be considered as component 1, 2, and 3. The user has to interact with the system through an interface display. After the data is parsed, the control is switched to component 2 which saves the raw data that has been parsed into a structured format. Once the data is saved in component 1, component 3 can visualise the system either statically or dynamically, this happens as it asks component 2 to get the structured data it parsed. Component 2 gets this data through component 1. In the module view, the layers of the system have been shown clearly as the lowest layer is the data processing layer. The upper layer is the visualization layer which takes the ready data from the data processing layer and makes it ready as a visual output. The highest layer is the interface layer that enables the user to interact with the system.

Therefore, any functional components can be categorised into different layers, meanwhile, each layer can also be traced in the conceptual view. Furthermore, the development of the system mostly focuses on the visualization and the data processing layer.
4.1.4.5 Rationale

The Visualization architecture which was deployed in the TMAV system focuses mainly on the data processing as well as the visualization of this data. This is a direct implementation of the visualization process that has been mentioned in Chapter 2, the literature review (Refer to).

As there is no back-end database requirement specified by the client, the data architecture is not concerned and is to be saved into local drives within the user computers.

4.1.5 Decomposition description:

4.1.5.1 Programs in the system:

The system is made up of one program. The program doesn’t provide a proper graphical user interface. The user, rather, is supposed to interact with it using a simple DOS-command window (Refer to). The output of the software will be a graph format file which can be open by graph reading software programs (Refer to).

In this software, we have three packages. The first one is fileprocessing. This package handles the parsing of the xml input files. It saves the raw data from these files and saves them in the allocation package. The fileprocessing package also writes the ready graph format output file.

The second package is the allocation package. This package includes the data structure of the objects that we deal with in this project. These objects are the tasks, messages, networks, and processors. Not only does this package have these data structures, but it also defines other objects to save the mappings and allocations of these objects with each other. In this package, there are algorithms for retrieving information, and getting this information ready to be visualized as graphs in the visualization package. Last but not least, this package makes it possible for storing as many allocations for the system objects as we need. This means that in a dynamic solution, as time changes, the allocation of tasks to processors change, therefore we need to have a data structure that is capable of saving these different allocations for every different time.

The final package is the visualization package. This package only interacts with the fileprocessing package to get the structured information from the allocation package. The package interacts with the fileprocessing package to write a graph format file. In case of a dynamic solution, the visualization package keeps writing the accumulated visual output and also refreshing the processing of reading the configuration file through the fileprocessing package. This dynamic process ends at a specific time specified by the user.
4.1.5.2 Significant Classes in Each Package

fileprocessing package:

Figure 51: fileprocessing package

1. XMLParser:

XMLParser is the class which parses the xml input files for TMAV system. It has got important methods that deal with the xml file format elements. When an element starts, its checked whether it is a processor, a network, a task, or a message. Then a flag of these becomes true and the rest are false. This class has an instance of the allocation package controller class so that it can save the information it parses through the data structure of the allocation package. Since this class is internal in the package, methods statElement and endElement are private as they are only used internally in the class.

2. FileProcessingController:

FileProcessingController is the main class of the package. It actually represents the package. Therefore, classes out of this package needs only to deal with this class to
get all the operation this package does. This class does all the public methods the XMLParser class do. Furthermore, it writes the graph output file.

**allocation package:**

![Figure 52: allocation package](image)

1. **Task:**

   Task is an object that represents a task. It encapsulates all task attributes that were specified in Chapter 2 (See section 2.1). These attributes include the name, Id, deadline, period, and wcet. Zero or more messages can be sent from a class. This enables a class object to save its relation with message objects. As this relation is permanent, any additional data structure is not required to save this relation. On the other hand, a task has a dynamic relation with processors. Since the relation changes with time, it can’t be saved in the task object.
2. Message:

*Message* is an object that represents a message. It encapsulates all message attributes that were specified in Chapter 2 (See section 2.1). These attributes include the name, Id, deadline, and size. Two tasks are associated with each message. This enables a class object to save its relation with message objects. As this relation is permanent, any additional data structure is not required to save this relation. On the other hand, a message has a dynamic relation with networks. Since the relation changes with time, it can’t be saved in the message object.

3. Processor:

*Processor* is an object that represents a processor. It encapsulates all processor attributes that were specified in Chapter 2 (See section 2.1). These attributes include the name, and Id. Zero or many networks can be associated with a processor. This enables a class object to save its relation with network objects. As this relation is permanent, any additional data structure is not required to save this relation. On the other hand, a processor has a dynamic relation with tasks. Since the relation changes with time, it can’t be saved in the processor object.

4. Network:

*Network* is an object that represents a network. It encapsulates all network attributes that were specified in Chapter 2 (See section 2.1). These attributes include the name, Id, bandwidth, and latency. Zero or many processors can be associated with a network. This enables a class object to save its relation with processor objects. As this relation is permanent, any additional data structure is not required to save this relation. On the other hand, a network has a dynamic relation with messages. Since the relation changes with time, it can’t be saved in the network object.

5. AllocationStore:

*AllocationStore* is an object which includes all the allocations that have been stored for visualization purposes at different times. These allocations are tasks allocations to processors as well as messages allocations to networks. Therefore, this object has zero or many ProcessorTaskMapper objects as well as zero to many NetworkMessageMapper objects.

6. ProcessorTaskMapper:

*ProcessorTaskMapper* is an object which includes an allocation of tasks to processors that have been done at a certain time T. Therefore, this object maps 1 processor object to many TaskPriorityAllocator since a task is given a priority when it’s allocated to a processor.

7. NetworkMessageMapper:

*NetworkMessageMapper* is an object which includes an allocation of messages to networks that have been done at a certain time T. Therefore, this object maps one
network object to many MessagePriorityAllocator since a message is given a priority when it’s allocated to a processor.

8. TaskPriorityAllocator:

TaskPriorityAllocator is an object which assigns a priority to a task when a task is allocated to a processor.

9. MessagePriorityAllocator:

MessagePriorityAllocator is an object which assigns a priority to a message when a task is allocated to a processor.

graph package:

![Diagram of graph package]

Figure 53: graph package

1. TaskInformation:

TaskInformation is an object that stores information about each task as it being visualized. This is useful for messages that need to be drawn later on, and which are associated with these tasks. The information this object saves include: the graph id of the task. Each object is represented as a node on the visual graph output. This node has an id. For a message to be drawn, a straight line has to be drawn from this message to this task. Therefore, it is crucial for us to know the task graph ID. In
addition to that, the task ID is saved besides its x and y co-ordinates as well as the number of networks this task has.

2. NodeGroup:
NodeGroup is an object that represents a rectangle which is supposed to siege a network along with its associated allocated messages. This object saves the x and y co-ordinates of this rectangle as well as its width and height.

3. Drawer:
Drawer is an object that does the visualization of the system. As we recall from chapter 2, a graph is a set of nodes connected to each other by edges. For us to visualize this graph, each node should have its x and y co-ordinates, width, and height. In addition to that, we should assign fixed distances between processors, tasks, messages, and networks. These values can be changed flexibly by the class user. This object also gets the networks and the processors all sorted and ready to be given their co-ordinates for visualization. The main methods of this class are the ones which provide x and y co-ordinates for the graph nodes and draw necessary edges to connect these nodes, and the ones that do the visual mapping of these nodes (see section 2.2). This class saves the visual information as text which later on will be given to the Main class which will write it on a file.

4. Main:
Main is the main object of the TMAV system. It basically has an instance of a FileProcessingController class. This instance reads the input files and then saves the data in a structured way which is ready for visualization by the help of an AllocationController object. The Main object then asks the Drawer object to provide the ready text visual output. This text is then written as a file for the user to read using a graph software reader.

4.1.5.3 Mapping from Requirements to Classes:

fileprocessing package

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Class providing Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF10</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>RF11</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>NFR5</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>MR1</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>MR2</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>DR1</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>PR1</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>PR2</td>
<td>XMLParser, FileProcessingController</td>
</tr>
<tr>
<td>PR3</td>
<td>XMLParser, FileProcessingController</td>
</tr>
</tbody>
</table>

allocation package

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Class providing Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement</td>
<td>Class providing Requirements</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>RF1</td>
<td>AllocationController, AllocationStore, Processor, Network</td>
</tr>
<tr>
<td>RF2</td>
<td>AllocationController, AllocationStore, Message, Task</td>
</tr>
<tr>
<td>RF3</td>
<td>AllocationController, AllocationStore, Processor, Task, TaskPriorityAllocator, ProcessorTaskMapper</td>
</tr>
<tr>
<td>RF4</td>
<td>AllocationController, AllocationStore, Network, Message, MessagePriorityAllocator, NetworkMessageMapper</td>
</tr>
<tr>
<td>RF5</td>
<td>AllocationController, AllocationStore, Processor, Task, TaskPriorityAllocator, ProcessorTaskMapper</td>
</tr>
<tr>
<td>RF6</td>
<td>AllocationController, AllocationStore, Network, Message, MessagePriorityAllocator, NetworkMessageMapper</td>
</tr>
<tr>
<td>RF7</td>
<td>Task, Message, Processor, Network, TaskPriorityAllocator, MessagePriorityAllocator</td>
</tr>
<tr>
<td>RF8</td>
<td>AllocationController</td>
</tr>
<tr>
<td>RF9</td>
<td>TaskPriorityAllocator, Task, AllocationController</td>
</tr>
<tr>
<td>RF10</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>RP1</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>RP2</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>RP3</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>NFR1</td>
<td>AllocationController</td>
</tr>
<tr>
<td>NFR2</td>
<td>AllocationController</td>
</tr>
<tr>
<td>NFR4</td>
<td>AllocationController</td>
</tr>
<tr>
<td>NFR5</td>
<td>AllocationController, Message</td>
</tr>
<tr>
<td>NFR6</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>MR1</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>MR2</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
<tr>
<td>DR1</td>
<td>AllocationController, AllocationStore, Message, MessagePriorityAllocator, Network, NetworkMessageMapper, Processor, ProcessorTaskMapper, Task, TaskPriorityAllocator</td>
</tr>
</tbody>
</table>

**graph package**

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Class providing Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF1</td>
<td>Drawer, NodeGroup, Point</td>
</tr>
<tr>
<td>RF2</td>
<td>Drawer, NodeGroup, Point, TaskInformation</td>
</tr>
<tr>
<td>RF3</td>
<td>Drawer, Point, TaskInformation</td>
</tr>
</tbody>
</table>
### 4.1.6 Dependencies Descriptions:

#### 4.1.6.1 Component Diagram:

![System component diagram](image)

**Figure 54:** System component diagram
As the above figure shows, the TMAV system consists of three components. The first component is allocation component. This component is independent as it doesn’t use any other component. This component provides the basic data structure through which the information of the entities and relations will be saved. It also saves the tasks, message, processors, and networks in such a way that they will be easier to visualize later on. This means that in the many to many relation between processors and networks crossings will be reduced as much as possible by adapting the order of processors and networks. The component also sorts the tasks according to their priority and the message according to their visualized length. As the figure shows, the second component fileprocessing uses the allocation one. The fileprocessing parses the input files and saves the data through the allocation data structure. The graph component visualizes information. It gets them from the allocation component but through the fileprocessing component.

4.1.6.2 Inheritance Relationships
There is no inheritance relationship among all classes.

4.1.7 Detailed Design:

4.1.7.1 Collaboration diagrams:

Visualization Process:

Figure 55: Visualization Process collaboration diagram

Figure 53 shows the conceptual visualization process collaboration diagram for the design of this project. This diagram conforms to the visualization process that was
discussed in the literature review (Ref to). Firstly, the main object creates a
FileProcessingController object. This object requests from the XMLParser object to
read the two XML input file (Ref to). The XMLParser parses the data and saves it
through the AllocationController which maps to the dataset in the visualization
process (Ref to). Once the information has been saved in a structured way, the main
object creates a drawer object, which is responsible for visualizing the information.
This object requests the information to be drawn from the FileProcessingController.
Finally, the main object writes the visual information into a file through the
FileProcessingController object (Refer to).

**Parsing Process:**

Figure 54 shows a detailed collaboration diagram for the design of the data parsing
mechanism. This diagram conforms to the use cases introduced in this chapter for
parsing data (Ref to). It also shows in more details what was shown in abbreviation in
the previous collaboration diagram. Firstly, the XMLParser object parses the two
input XML files. It basically gets all the raw information from these two files.
Secondly, through the AllocationController object, a new allocation is created once
we parse a new configuration file. If the system file is a new system file, the adding of
objects start. Once this ends, the adding of allocations start in the AllocationStore
object. This clearly separates the saving of the objects themselves and their allocation.
As mentioned before, the reason why this design separates these two processes is
because these objects may have zero or more allocations. Also, the allocation changes
with time but the objects themselves along with their attributes do not.

![Collaboration Diagram](#)

**Figure 56:** Parsing process collaboration diagram
4.1.7.2 Significant Design decisions and Algorithms:

Structure of packages:

Figure 55 shows the structure of packages. Each package has its own unique job and its own representative class. This makes the design more structured and thus make it easy for developers to understand it and perhaps extend it as well. Furthermore, when a developer only needs to deal with one class of each package, it avoids any chance of confusion (Refer to 3.10.2.3, NFR9).

Fileprocessing package is responsible for reading and writing files. When it reads a file, it extracts the information and forwards it to the allocation package. The allocation package gets this information and saves it in a structured manner so that it can be easily visualized later on. If graph package has the same instance of the AllocationController of the allocation package, it will create redundancy and confusion. However, if the access to this object is only through the Fileprocessing object, then it reduces the complexity from the graph package as it makes it only deal with one object instead of two.

![Figure 57: Structure of packages, grey colour means representative class](image)

Allocation Process:

The design took into consideration the difference between the dynamic and static relations (Refer to 3.10.1, 3.10.2). The permanent relations are saved in the objects themselves. This means a Task object has direct references to all messages that are sent from it, a message object also has a direct reference to a task from which it is sent and a task to which it is sent, a processors object has a direct references to the networks it is connected to, and a network has references to the processors which are connected to it. However, the allocation of tasks to processors and messages to networks changes with time. Therefore, in our design we save these allocations in separate objects. These objects are AllAllocationStore, ProcessorTaskMapper, and
NetworkMessageMapper. This also helps in the extensibility of the software to fit into the dynamic solution.

**Crossing reduction**

Minimizing the crossings is one of the most important aesthetics in graph theory since it increases the readability of a graph. Crossings happen a lot in many-to-many relations amongst objects. In this project, we have a many-to-many relation between processors and networks (Refer to table 1). Therefore, though minimizing these crossings is not a requirement. It indirectly is as readability is a requirement. Figure 56 shows how the algorithm works. First it picks the first processor and draws it, then it compares the rest of the processors with the draw ones, and picks the most similar one. Then, the most similar one is visualized either over or below the processors. Actually when it assigns it below the processors, it creates a new solution. The algorithm continues like that till the processors are done. The number of solution this algorithm creates is \([ 1+ 2+...+2^{n-2}]\) where \(n\) is the number of processors . Once all these solutions are generated. Their number of crossings can be calculated which is \(O(n.m)\) where \(m\) is the number of networks. The comparison this algorithm makes is \(O(n.m)\) as well. So the worst case performance of this algorithm eventually is \(O(n.m) + O(n.m)* [1+2+..+2^{n-2}]\). Last comment about this algorithm is that it is designed to be implemented in the AllocationController object. This is due to the fact that this object is responsible for getting the data ready for being visualized. The algorithm saves the solutions in a vector of multi-dimensional arrays.

![Crossings reduction algorithm](image)

**Figure 58:** Crossings reduction algorithm, the figure only shows one possible solution

**Messages sorting**
Though there is no requirement of sorting messages. Sorting them can be a great benefit for the readability for the graph. This is because the design sorts the messages according to their visual length and also visualizes messages which belong to a task right after each other. Figure 57 shows three messages which belong to task $T_{30}$. The messages are sorted according to their visual length. Furthermore, a small message was allowed to be drawn on the same vertical segment of the last most left message. This feature gives the user a flexibility to pick all the messages which are associated to a task from the first glance since they come right after each other. This of course makes users identify task message relations quicker and easier.

![Figure 57: Messages sort](image)

**Java**

Java was decided to be the programming language of implementation. This is due to the fact that I’m very familiar with it. In addition to that, Java is completely object oriented. Also, the XML parsing packages it supports are easy and flexible to use. Finally, Java is an independent platform programming language. This makes the software portable.

**Providing a GML graph format file**

It was agreed with the client that the software will produce a GML graph format. This file can be read by a graph reader. Yworks software is one of these software graph readers. Graphlet is another example. The reasons why this decision has been made are:

- Graph readers are reliable and tested softwares.
- Graph readers offer advantages which are especially important for visualizing large graphs. These include zoom + pan, focus + context, and interaction.
Building these functionalities for this software is difficult to provide for the limited time of the project.

- The output of this software can be sent to many users as a light file which can be open by plenty amounts of famous graph reader softwares.

The reasons why GML was chosen as the graph language for the output file:

- Supports attaching arbitrary information to graphs, nodes and edges, and is therefore able to emulate almost every other format [46].
- Platform independent [46].
- Easy to implement [46].
- Flexible [46].
- Has the capability to represent arbitrary data structures [46].

There are many different file formats for graphs. The capabilities of these file formats range from simple adjacency lists over adjacency lists with labels or coordinates to complex formats which can store arbitrary data. This has lead to an almost "babylonic" situation where we have a large number of different, mostly incompatible formats. Exchanging graphs between different programs is painful, and sometimes impossible [46].

**Visualizing a large graph:**

In an attempt to defeat visualizing a large graph, the design had the following features:

- Minimizing area on which messages are visualized.
- Hiding the attributes of the objects.
- Providing selection feature through graph reader softwares such as Yworks.
- Providing features such as zoom + pan, Context + Focus.
- Allowing the user to search for an object.
- Allowing the user to filter an object (but not through the graph reader though).

**The colouring of a graph:**

When colouring the graph, the design benefited from the second case study. In that case study, each object which belonged to a certain group had the same colour *(Refer to 2.4.2)*. This helps the user identify easily objects of one group and provide structurability to the visualization.

Processors and networks had visualization mapping which conveyed their meanings. A processor had a processor icon and a network had a network icon. Tasks were coloured light yellow so that their IDs can be shown clearly (because their ids have dark colour). The background was white so that it doesn’t cost the user a lot of ink when he prints the visual output. The messages were shown in a dark red colour. This colour makes a contrast with the background, thus, helps the user identifies it easily.
4.2 Testing:

Testing is designed to prescribe the scope, approach, resources, and schedule of all testing activities that will be carried out during the software development process. We will identify the items to which need to be tested, the features to be tested, the types of testing to be performed, the resources and schedule required to complete testing, and the risks associated with the plan.

4.2.1 Scope:

TMAV’s main functionality is to read information from two xml files and the visualize tasks mapping to messages, processors mapping to networks, tasks allocation to processors, and messages allocation to networks. Since the dynamic solution wasn’t implemented, it will not be tested here. The system will be able to run on Modern JRE supported operating systems or better, provided that a recent JVM is installed on the machine. Non functional requirements will be evaluated in the evaluation chapter.

4.2.1.1 fileprocessing

fileprocessing modules shall be performing the following functionalities for TMAV:
- Read the data to be visualized from a two xml files, a system xml file and a configuration xml file (Refer to FR10, 3.10.1).
- Present the output as a graph file (Refer to FR11, 3.10.1).

4.2.1.2 allocation

allocation modules shall be performing the following functionalities for TMAV:
- Provide the data ready to visualize processors to networks mapping, tasks to messages mapping, tasks allocated to processors, and the messages allocated to networks. (Refer to FR1- FR3, 3.10.1)
- Provide the data ready to visualize the change of task allocation to processors and message allocation to networks.
- Provide the data ready to show the attributes associated with each object.
- Sort the tasks according to their priority.

4.2.1.3 graph

graph modules shall be performing the following functionalities for TMAV:
- Visualize processors to networks mapping, tasks to messages mapping, tasks allocated to processors, and the messages allocated to networks. (Refer to RF1- RF3, 3.10.1)
- Visualize the change of task allocation to processors and message allocation to networks.
- Show the attributes associated with each object.
- Retrieve the data parsed from two xml input files.
- Visualize set of objects in a clear separate group

4.2.2 Test strategy:

The test strategy consists of a series of different tests that will fully exercise the TMAV system. The primary purpose of these tests is to uncover the system limitations and measure its full capabilities. A list of the various planned tests and a brief explanation follows below.

4.2.2.1 Module Test:

Module Testing will be performed by the coder, while using main method testing, which will try all of the significant behaviours of the class. (Refer to )

4.2.2.2 Performance Test:

The design took performance into account (See 4.1.7.2, crossings reduction algorithm). However, it wasn’t a high requirement. Therefore, due to the lack of time this testing will not cover it.

4.2.2.3 Usability Test:

It is hard to test usability in a short time. However, the design took it into a great account (Refer to 4.1.7.2 and 4.2.6.1). A small survey will cover some usability issues in the evaluation chapter (See 5.2.2). In addition to that, usability tease cases will be provided at the end of this chapter

4.2.3 Test Environment Requirements:

- *A DEL compatible PC.*
- *AME Athlon™XP 2400+ 1 GHz processor (minimum)*
- *512mb RAM*
- *4 GB Hard Drive*
- *Windows 2000 or higher*
- *A GML graph reader installed software(Preferably Yworks)*
4.2.4 Module Testing:

4.2.4.1 Testing the fileprocessing functionality:

In this test case, testing fileprocessing module of the TMAV system will be exercised using main method of the main class (i.e. FileProcessingController.class). All the key functionality of the fileprocessing module is exercised

<table>
<thead>
<tr>
<th>Test case No.</th>
<th>Class name</th>
<th>Method Name</th>
<th>Test Content</th>
<th>Output</th>
<th>Pass criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>FileProcessingController</td>
<td>readFile</td>
<td>Reading input files, system and configuration file. The files are of correct format. <em>(Refer to 3.10.1,FR10)</em></td>
<td>Informatio n is visualized correctly. If the file don’t exist or of a wrong format, an exception is generated and the software exists.</td>
<td>The informatio n shall be parsed correctly, therefore, the informatio n shall be visualized correctly</td>
</tr>
<tr>
<td>2.</td>
<td>FileProcessingController</td>
<td>writeFile</td>
<td>Writing a file, the file name is provided <em>(Refer to 3.10.1,FR11)</em></td>
<td>When a not faulty name was given for the file, it was saved correctly. However, when we gave a file name which starts with characters like /, an exception is caught and the software exists.</td>
<td>The file shall written and saved on the class folder with a gml extension</td>
</tr>
</tbody>
</table>
### 4.2.4.2 Testing the allocation functionality:

Again testing allocation module of the TMAV system will be exercised using main method of the main class (i.e. AllocationController.class).

<table>
<thead>
<tr>
<th>Test case No.</th>
<th>Class name</th>
<th>Method Name</th>
<th>Test Content</th>
<th>Output</th>
<th>Pass criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>AllocationController</td>
<td>crossingSolutions</td>
<td>Provide the data ready to visualize processors to networks mapping <em>(Refer to 3.10.1,FRI)</em></td>
<td>The mapping is correct. However, the crossings are not always minimized.</td>
<td>The allNetworks and allProcessors vectors should contain all processors and networks, and should be in a right positions in these vectors so they can be mapped together as a visual output. This can be checked by us looking through the xml files, and check the mapping of networks to processors. In addition to that, the minimization of crossings in the visual output shall be checked too</td>
</tr>
<tr>
<td>2.</td>
<td>AllocationController</td>
<td>-</td>
<td>Provide the data ready to visualize the change of task allocation to processors and message allocation to networks.</td>
<td>- Not provided</td>
<td>-</td>
</tr>
</tbody>
</table>
### 3. AllocationController

**print**

Provide the data ready to show the attributes associated with each object

*(Refer to 3.10.1, FR5, FR6)*

The printing is done correctly

All objects along with their attributes shall be printed out

### 4. AllocationController

**sortAllTasks**

Sort the tasks according to their priority

*(Refer to 3.10.1, FR9)*

The printing is done correctly

printTasks shall print all tasks according to their priority

### 5. AllocationController

**getMessage**

Search for an object

*(Refer to 3.10.1, FR8)*

The test passed

Look up the xml system file, find an object whether it is a message, task, network, or a processor. Provide its ID for all these methods. Then print the name or any other attribute of these objects. Match this from the xml input file. If it is matched, then the search is true

---

### 4.2.4.2 Testing the graph for functionality:

<table>
<thead>
<tr>
<th>Test case No.</th>
<th>Class name</th>
<th>Method Name</th>
<th>Test Content</th>
<th>Output</th>
<th>Pass criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Drawer</td>
<td>draw</td>
<td>Visualize processors to networks mapping,</td>
<td>The mapping is correct. The</td>
<td>Check the objects in the xml input files, the mappings, and the allocations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tasks to messages mapping, tasks allocated to processors, and the messages allocated to networks (Refer to 3.10.1, FR1-4).</td>
<td>allocation is correct. All objects are shown.</td>
<td>Check whether this match the visualized output.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Drawer</td>
<td>-</td>
<td>Visualize the change of task allocation to processors and message allocation to networks (Refer to 3.10.1, FR5-6).</td>
<td>- Not provided</td>
<td>-</td>
</tr>
<tr>
<td>3.</td>
<td>Drawer</td>
<td>draw</td>
<td>Show the attributes associated with each object (Refer to 3.10.1, FR1-7).</td>
<td>The attributed are not shown</td>
<td>Match the attributes of the objects in the xml files with those shown in the visual output</td>
</tr>
<tr>
<td>4.</td>
<td>Drawer</td>
<td>draw</td>
<td>Visualize each set of objects in a clear separate group (Refer to 3.10.1, FR12).</td>
<td>The visualizatio n is done as mentioned in the pass criteria.</td>
<td>Tasks shall be visualized on the left side. Networks on the top, Processors on the toppest left part. Messages are between networks and tasks.</td>
</tr>
</tbody>
</table>
4.2.6 Non-functional Requirements Testing:

4.2.6.1 Scalability:

For testing scalability, we generated input files which contain 50 processors and a few hundred tasks (Refer to 3.10.2, NFR5). The software didn’t crash. However, it was difficult for us to identify the relations of tasks to processors. However, Yworks helped users a bit in identifying some of these relations through interaction and zooming in and out with keeping an abstract view of the graph (See figure 60).

The inclusion of this number of processors made processors quite far from networks. This actually has been taken into consideration on the design phase (See 4.1.2.2, fourth design attempt). However, due to the lack of time, the implementation of extra labels wasn’t done.

![Figure 60: A large graph is shown as an abstract view on the left side, on the main screen the view is zoomed out on some tasks](image)

In general, visualizing large graphs was also taken into account on the design phase (See 4.1.7.2, visualizing a large graph). Users suggested that filtering could have defeated this problem. Due to the lack of time, the implementation of filtering was only done at the kernel of the software but not the visual output. In other words, filtering the relations of the graph is trivial from the data model view, and is easy to implement on the visual output.

4.2.6.2 Degradability:

For testing durability, we generated only an xml input files and didn’t generate a configuration file. Unfortunately, the system raised an exception and the software didn’t visualize the static relations in the system file. This actually wasn’t taken into consideration on the design phase because it wasn’t a requirement and the assumption was that the user will provide a system and a configuration xml files together without any sort of bug (Refer to 3.8). Modifying few methods in XMLParser class in will solve this problem.
5 Evaluation

This chapter attempts to evaluate the success of the project. It does so by first looking at the software model of the project. The choice of this model as well as how much it was followed is discussed. The chapter then discusses the fulfilment of the functional and non-functional requirements. Insightful comparisons will be made between what has been done and what was expected from the users, the requirements, and the theory.

5.1 Software Model Evaluation

5.1.1 The choice of the model

The spiral model gave a great flexibility for the development of this project. It gave a chance of communicating with the users during all the phases of the prototype design. This was so important for this project since it is a visualization project. It is related to the users’ perceptions and preferences. The comments the users gave during the phases of the design helped a lot in enhancing the quality of the output, and certainly fulfilling the requirements (Refer to 4.1.2.2). Furthermore, the model helped a lot in defeating the problem of the designer not having enough background about information visualization. The more interacting with users, the deeper analysis and better decisions were made (Refer to 4.1.2.2).

Choosing other software models such as the waterfall model or the incremental model wouldn’t have provided the flexibility of interacting with the client during all the phases of the project.

5.1.2 The following of the model

The collected requirements were iterated several times as the system was understood better. In the prototype design, using the spiral model, four iterations were made to the static part and two to the dynamic part (Refer to 4.1.2.2). The design decisions were taken according to the final iteration of the requirements and the final prototype design. No iterations were made during the implementation itself since the client did not request any change at that stage. Module testing was done as the implementation was in progress. In general, during all the phases the client was updated with the latest of the development. His input along with other users’ was taken into a great consideration during all the phases of the project.

5.2 Evaluation of the fulfilment of the requirements

This section will try to evaluate the requirements of this project. These requirements have three different degrees of priority. High means an important expected requirement. Intermediate means its achievement is not essential but it would be good if they were. Low means the requirement is not important and is not expected to be
achievement. Its achievement is considered to be a bonus. The importance of each requirement will be associated with every requirement we evaluate.

5.2.1 Evaluation of the fulfilment of the functional requirements:
For evaluating the fulfilment of the functional requirements, we exercised tens of examples for visual outputs the software produces. Figure 58 shows an example of them. This figure will help us assessing the functional requirements. In some cases, we will refer to the module testing which has been made too.

Figure 61: A visualization sample the software produced
In the following points, we will provide a brief evaluation for each functional requirement:

- **The system shall visualize processors to networks mapping [high]** (Refer to 3.10.1, FR1):
  This requirement is achieved successfully (Refer to 4.2.4) Processors are visualized on the most left top part; networks are visualized on the top. The connection between these two is visualized as blue diamonds (See figure 58).

- **The system shall visualize the messages allocated to networks [high]** (Refer to 3.10.1, FR4):
  This requirement has been achieved successfully (Refer to 4.2.)
  Each network has its allocated messages below it (See figure 58).

- **The system shall visualize messages to tasks mapping [high]** (Refer to 3.10.1, FR2):
  This requirement has been achieved successfully (Refer to 4.2.)
  Tasks are visualized below the processors; messages are visualized right below the networks, and a message is visualized as an arrow. The arrow’s beginning indicates from which task it’s sent. In addition to that, the arrow’s end indicates to which task the message is sent.

- **The system shall visualize the change of task allocation to processors as well as the message allocation to networks over time [low]** (Refer to 3.10.1, FR5, RF6):
  These two requirements are not achieved since they are very low requirements. They actually are not expected from a 3 months project. However, this thesis provided a prototype and use cases design for the dynamic change of allocation (Refer to 4.2.).

- **The system shall show the attributes associated with each object [high]** (Refer to 3.10.1, FR7):
  This requirement has been achieved in the software design as in the allocation package, the objects of Task, Message, Network, and Processors are designed to include the attributes of these objects (Refer to 4.2). However, the requirement couldn’t be achieved on the visualization side as GML doesn’t offer this feature.

- **The system shall allow the user to search easily for an object [high]** (Refer to 3.10.1, FR8):
  This requirement has been achieved in the software design as in the allocation package, the class AllocationController provides methods for search. These methods are getTask, getMessage, getProcessor, and getNetwork (Refer to 4.2).

- **The system shall visualize the tasks sorted according to their priorities [low]** (Refer to 3.10.1, FR9):
  This requirement has been achieved successfully (Refer to 4.2).
• The system shall read the data to be visualized from a two xml files, a system xml file and a configuration xml file [high] (Refer to 3.10.1, FR10,11): This requirement has been achieved successfully (refer to 4.2). However, the system doesn’t report the specific sort of error if for example; some information is missing in the input files. It was actually agreed with the client that the implementation will not take this into consideration and it will expect to get perfect input files.

• The system shall present the output as a graph file [high] (Refer to 3.10.1, FR11): This requirement has been achieved successfully (Refer to 4.2).

• Each set of objects shall be visualized in a clear separate group [high] (Refer to 3.10.1, FR12): This requirement has been achieved successfully (Refer to 4.2).

5.2.2 Evaluation of fulfilment of the non-functional requirements:

Evaluating the non-functional requirements is not easy since they are related to the quality and quality is hard to judge. This section will try, however, doing so through a survey which targeted seven real-time research associates who are experienced in the real-time systems filed. After a quick introduction to our visualization system, the researchers will allowed to visualize the graph output using yworks software. Furthermore, they were shown several examples of visual outputs.

In this section, we will also try to use what has been mentioned in the literature to evaluate the final product of the software.

In the following, we will provide a brief evaluation for each functional requirement:

• The system shall process events at a reasonable speed [intermediate] (Refer to 3.10.2, NFR1): We exercised tens of examples, including examples of visualizing large graphs. The file didn’t take more than 5 seconds to be generated. However, the performance of each single method was not tested because there was not a sufficient time and the requirement itself is a very low requirement. Nonetheless, significant algorithms such as the crossings reduction algorithm took the performance into account (Refer to).

• The output of the system should be readable by users [high] (Refer to 3.10.2, NFR3): During the phases of the design, different users were involved with the prototype design (Refer to 4.1.2.2). In the survey we carried out after implementing the solution, six research associates out of seven mentioned that the relations were visualized clearly. Only one researcher mentioned that the processors networks relations are difficult to understand. According to him, this is because the connections are visualized on horizontal and vertical segments. From the network view, the user
should only look at the vertical segments to identify the connections with the processors. For him it was confusing as he might be tempted to look at the horizontal segments as well. However, this problem can be easily defeated the more the user uses the system. Interestingly, other researchers thought that messages should be allowed to be visualized horizontally as well. Though this feature is not a requirement, the flexibility of the design allows this note to be implemented by letting the visual graph rotate easily.

From a theory point of view, the readability issues of the visual graph were discussed in the design phase (See 4.1.7.2). Through comparing the final implemented visual graph (See figure 58) with the last prototype design (See figure 35), there are only few changes the implemented visual graph added. These changes are the use of colours, grouping which includes the network, and the sorting of messages. The rational behind these choices was discussed in the design phases (Refer to the colouring the graph, 4.1.7.2).

- **The system shall visualize around 50 tasks and several hundreds of messages in a readable manner [intermediate] (Refer to 3.10.2, NFR3 and 4.2.6.1):**
  As a visual printout, the system couldn’t visualize this number of tasks and several hundreds of messages in a readable manner. This is due to the fact that the implementation focused more on implementing the visualization of relations in a readable manner. However, the design took into consideration the scalability. It did so by providing insight features such as search, and interacting. Furthermore, it was considered in the design that the user should be able to filter the relations he wants to visualize (See visualizing a large graph, 4.1.7.2). Clustering algorithms were not considered in the implementation of this project due to the lack of time. Finally, graph readers software programs provide important features for visualizing large graphs. These include Zoom + pan, Focus + Context, and interaction. Since our output file is read by these softwares, the problem of visualizing large graphs is in a way tackled as well (Refer to 2.2.3.3).

- **Through the system, a relevant researcher can identify interesting patterns [intermediate] (Refer to 3.10.2, NFR6):**
  The client along with the involved researchers could identify the following patterns in few seconds:
  - The network which has the biggest/smallest number of messages.
  - The network which is connected to the biggest/smallest number of processors.
  - The processor that has the biggest/smallest number of tasks
  - The task that has the biggest/smallest number of sent messages.

One of the insights a researcher suggested is that the message should be thicker as its size increases. This feature was not implemented as it was not a requirement. However, most researchers agreed that it helps in the analysis of the graph.

- **Through the system, a relevant researcher can identify the errors of the tool [high] (Refer to 3.10.2, NFR7):**
  In agreement with the client, the software successfully identified the messages which were allocated wrongly by the tool. This message is coloured by a distinct strong red
colour so that the researcher can see it so easily. The client didn’t specify any other bug the tool may cause.

- **The system shall report an exception to the user [intermediate] (Refer to 3.10.2, NFR8):**
  The exceptions this system took into consideration were if the wrong input files were specified or if the software was run while the file didn’t exist. These exceptions are caught in try catch statements in the fileprocessing package (Refer to). However, due to the lack of time, the system didn’t consider exceptions that may be able to be generated during an enormous number of objects to be visualized. Also, if there was a relation missing from one of the input files, the system only report an exception but doesn’t state the exact error (Refer to).

- **The software should be written in a way that is moderately extensible [high] (Refer to 3.10.2, NFR9):**
  The implementation is reasonably extensible (Refer to). Things were taken into consideration were the ability of the software to fit into the dynamic solution for future development (Refer to). Another thing is the design can easily be used to generate different types of graph outputs, rather than GML, such as GraphML and GXL (Refer to).

### 5.3 Summary

In summary, this chapter tried to evaluate the success of this project. In this evaluation, the fulfilment of requirements is classified into three categories. First set of requirements was fully achieved. Second one is partially achieved. Last one is only tackled in the design.

In numbers, table 5 shows that we have twenty requirements. Thirteen of them are high requirements. Ten of them were fully achieved and three were partially achieved. In the rest of requirements, we have four low requirements. All of them were tackled in the design. One of them was partially achieved, and the other was fully achieved. Last three requirements were intermediate ones. These were partially achieved.

<table>
<thead>
<tr>
<th>Requirement number</th>
<th>Requirement type</th>
<th>Requirement priority</th>
<th>Requirement fulfilment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1.3</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>FR4</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>FR2</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>FR5, 6</td>
<td>Functional</td>
<td>low</td>
<td>Tackled in the design, Not achieved in the implementation</td>
</tr>
<tr>
<td>FR7,3</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>FR9</td>
<td>Functional</td>
<td>Low</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>FR10</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>FR11</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>Requirement</td>
<td>Functional</td>
<td>Level</td>
<td>Status</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>-------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>FR12</td>
<td>Functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>NFR1</td>
<td>Non-functional</td>
<td>Intermediate</td>
<td>Partially achieved</td>
</tr>
<tr>
<td>NFR3,4</td>
<td>Non-functional</td>
<td>High</td>
<td>Tackled in the design, partially achieved</td>
</tr>
<tr>
<td>NFR5</td>
<td>Non-functional</td>
<td>Low</td>
<td>Tackled in the design, partially achieved</td>
</tr>
<tr>
<td>NFR6</td>
<td>Non-functional</td>
<td>Intermediate</td>
<td>Tackled in the design, partially achieved</td>
</tr>
<tr>
<td>NFR7</td>
<td>Non-functional</td>
<td>High</td>
<td>Fully achieved</td>
</tr>
<tr>
<td>NFR8</td>
<td>Non-functional</td>
<td>Intermediate</td>
<td>Partially achieved</td>
</tr>
<tr>
<td>NFR9</td>
<td>Non-functional</td>
<td>High</td>
<td>Partially achieved</td>
</tr>
</tbody>
</table>

Table 5: Summary of the fulfilment of requirements
6 Conclusion and further work

This thesis presented a software engineering solution for visualizing information associated with a Task and Message Allocation tool. The solution depended on available information visualization technologies and issues as well as the feedback of the users. This chapter gives the reader the conclusions from the work done to achieve this solution. It also states further work which should help to tackle the issues which couldn’t be implemented in this project, and enhance the quality of the solution.

6.1 Conclusion

This section provides a conclusion of the information visualization concepts that were found to be useful and insightful during the development of this project. Then, it summarizes the major issues and decisions which were taken in the design to achieve the requirements of the project. The section also links how the concepts of information visualization were reflected on the design of the tool. Throughout the conclusion, we try to give an insight where this present work might be of help in related problem areas.

6.1.1 Information Visualization Concepts

Below are the most important concepts of information visualization which helped greatly in developing this software:

- Information visualization process has three important phases: (1) Transformation of raw information into a structured one. (2) Visually mapping this structured information. (3) Presenting this visual mapping to the user.
- A good visualization provides insights for the user
- Visualizing large amount of information can be tackled by clustering, zoom+ pan, focus + context views, and interaction. Based on the requirements analysis and the domain, the designer can decide which technique he or she should go for.
- Graphs are used to visualize relational information.
- The most important aesthetic of a graph is the minimization of crossings between edges.
- The more edges in a graph we have, the less readability of straight-line graphs, including layered graphs, we get.
- Orthogonal graphs are highly readable provided their bends are minimized. However, they shouldn’t be used for graphs which have nodes that should visualize specific positions.
6.1.2 Major issues and decisions

Here are the most important issues and decisions were taken in this project. The issues depended on the above information visualization concepts besides the feedback of the user.

- The software model, which was followed to achieve the solution of this project, is the spiral model. It helped in engaging the users in the phases of the project.
- Through comparing the available options for visualizing relational information, orthogonal drawings were chosen for visualization the relations which the task and message allocation tool deals with. Four prototypes based on orthogonal drawings were developed to provide the visualization solution. Users’ feedbacks and information visualization theory helped in the evolution of these phases.
- The visualization design tried to give the user different insights. Some of which were simple such as search, and some help to assess the quality of the solution.
- The visualization design increased the readability of the graph by minimizing the crossings in the processors networks mapping. Furthermore, it minimized bends, and area especially in visualizing messages. Minimizing crossings can be of a great help for orthogonal graphs such as VLSI circuits.
- Large graphs were partially tackled in the system by allowing the user to search, zoom + pan, focus + context, and interact with the graph.
- The visualization of the dynamic solution was discussed in the prototype and the use cases design.
- The software fulfilled most of the functional as well as some of the non-functional requirements of the client.

6.2 Further work

Here are some of the bits we think should be done to enhance the quality of the solution and consider some issues into more depth:

- The dynamic solution design should be discussed and analyzed. Then, implemented.
- Clustering algorithms should be considered to visualize large graphs of the static and the dynamic solution.
- Filtering the visualization of relations should be implemented.
- Allowing the system to visualize graphs in different graph formats which are more extensible and compatible with web browsers. Examples are: GXL, SVG, and GraphML.
- The system should be able to visualize the static relations on their own in case the configuration file is not provided by the user.
7 References

http://web.cs.wpi.edu/~matt/courses/cs563/talks/datavis.html
[2] UIUC DLI Glossary
http://dl.i.grainger.uiuc.edu/glossary.htm
[4][10] Real-time, Embedded, and Specialized Systems PTF.
http://realtime.omg.org/
http://en.wikipedia.org/wiki/Real_time_operating_system
[8] Visual Universe, Data into insight
http://www.dlib.org/dlib/july01/07editorial.html
[14] Jacob Somervell, D. Scott McCrickard, Chris North, Maulik Shukla, An Evaluation of Information Visualization in Attention-Limited Environments, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106 USA
[15] Chris North, Information visualization, Center for Human-Computer Interaction, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA.
[18] aiSee: Screenshots
http://www.absint.com/aisee/shots.htm
[19] Sougata Mukherjea, Information Visualization for Hypermedia Systems, C&C Research Labs, NEC USA,110 Rio Robles, San Jose, Ca 95134, USA.
[21] Geometry in Action: Graph drawing
[24] Yworks: Simple graph
[25] Complex graph
http://www.ercim.org/publication/Ercim_News/enw44/herman2.jpg
http://en.wikipedia.org/wiki/Graph_%28mathematics%29
[27] Planar graph, Wikipedia
http://en.wikipedia.org/wiki/Planar_graph
[28] Helen C. Purchase, Matthew McGill, Linda Colpoys, and David Carrington, Graph
drawing aesthetics and the comprehension of UML class diagrams: an empirical study.
[29] Graph drawing, Wikipedia
http://en.wikipedia.org/wiki/Graph_drawing
[31] What is a Force Directed Algorithm
[32] Rodrigo A. Botafogo, Ehud Rivlin, and Ben Shneiderman. "Structural Analysis of
Hypertexts: Identifying Hierarchies and Useful Metrics" in ACM Transactions on Information
Using Vision to Think, Morgan Kaufmann.
[34] aiSee, Forced-directed layout
[35] Holger Eichelberger, Aesthetics and Automatic Layout of UML Class Diagrams
[37] Geographical graph
Inform, 1983.
[39] Glossary
www.cs.uga.edu/~potter/Complntell/gaglossary.htm
[41] Graph coloring, Wikipedia
http://en.wikipedia.org/wiki/Graph_coloring
[42] aiSee: Circuit diagram graph
http://www.aisee.com/welcome.htm
[43] aiSee: History of programming languages graph
http://www.aisee.com/welcome.htm
[44] Joseph J. LaViola Jr., Bend Stretching and Minimization in Planar Orthogonal Drawings,
Final Project - CS 252
[46] The GML graph format
http://www.infosun.fmi.uni-passau.de/Graphlet/GML/
8 Appendix

8.1 FileProcessingController.java

```java
package fileprocessing;
/*
 * @author mak500
 */
import java.util.Vector;
import allocation.*;
import java.io.*;

public class FileProcessingController {
    private XMLParser parser;
    public FileProcessingController() {
        parser=new XMLParser();
    }

    public void readFile(String file1,String file2){
        parser.parse(file1, file2);
    }

    public void writeFile(String fileName,String txt){
        FileOutputStream fout;
        try {
            // Open an output stream
            fout = new FileOutputStream (fileName);
            // Print a line of text
            new PrintStream(fout).print(txt);
            // Close our output stream
            fout.close();
        } // Catches any error conditions
        catch (IOException e) {
            System.err.println("Unable to write to file");
            System.exit(-1);
        }
    }

    public Object[][] getNetworkProcessorInformation(int solutionIndex){
        return parser.getNetworkProcessorAllocation(solutionIndex);
    }

    public Vector getProcessorMapping(){
        return parser.getProcessorMapping();
    }
}
```
8.2 XMLParser.java

package fileprocessing;

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
import java.util.*;
import allocation.*;

public class XMLParser extends DefaultHandler {

    boolean inNetwork=false;
    boolean inTask=false;
    boolean inMessage=false;
    boolean inInterface=false;
    boolean inProcessor=false;
    boolean inObjectConfiguration=false;

    String processorTempID;
    String processorTempName;
    String networkTempID;
    String networkTempName;
    int networkTempBandwidth;
}
int networkTempLatency;
String taskTempID;
String taskTempName;
int taskTempPeriod;
int taskTempWCET;
int taskTempDeadline;
String messageTempName;
String messageTempID;
String tempSentTo;
String tempSentFrom;
int messageTempDeadline;
int messageTempSize;
String tempAllocationID;
int tempPriority;
String tempID;

AllocationController aController = new AllocationController();

private CharArrayWriter contents = new CharArrayWriter();

// Override methods of the DefaultHandler class
// to gain notification of SAX Events.

public XMLParser(){
    processorTempID="";
    processorTempName="";
    networkTempID="";
    networkTempName="";
    networkTempBandwidth=0;
    networkTempLatency=0;
    taskTempID="";
    taskTempName="";
    taskTempPeriod=0;
    taskTempWCET=0;
    taskTempDeadline=0;
    messageTempName="";
    messageTempID="";
    tempSentTo="";
    tempSentFrom="";
    messageTempDeadline=0;
    messageTempSize=0;
    tempAllocationID="";
    tempPriority=0;
    tempID="";
}

public void falseAll(){
inNetwork=false;
inTask=false;
inMessage=false;
inInterface=false;
inProcessor=false;
inObjectConfiguration=false;
}

public void startElement( String namespaceURI,String
localName,String qName,Attributes attr ) throws SAXException {
contents.reset();

if ( localName.equals( "hardware-module" ) ) {
    falseAll();
    inProcessor=true;
    processorTempID= attr.getValue( "id" );
    processorTempName=attr.getValue("name");
}

if ( localName.equals( "network" ) ) {
    falseAll();
    inNetwork=true;
    networkTempID = attr.getValue( "id" );
    networkTempName=attr.getValue("name");
}

if ( localName.equals( "interface" ) ) {
    falseAll();
    inInterface=true;

    aController.addNetworkProcessorAllocation(attr.getValue("network-id"),attr.getValue("module-id") );
}

if ( localName.equals( "task" ) ) {
    falseAll();
    inTask=true;
    taskTempID = attr.getValue( "id" );
    taskTempName=attr.getValue("name");
}

if ( localName.equals( "message" ) ) {
    falseAll();
    inMessage=true;
    messageTempID = attr.getValue( "id" );
    messageTempName=attr.getValue("name");
    tempSentFrom=attr.getValue("from-task-id");
    tempSentTo=attr.getValue("to-task-id");
}

if ( localName.equals( "attribute" ) ) {
    if(inNetwork){
        if(attr.getValue( "name" ).equals("bandwidth")(networkTempBandwidth= (new Integer(attr.getValue("value"))).intValue();)
            if(attr.getValue( "name" ).equals("latency")){networkTempLatency= (new Integer(attr.getValue("value"))).intValue(); }
    }

    if(inTask){
        if(attr.getValue( "name" ).equals("period")){taskTempPeriod= (new Integer(attr.getValue("value"))).intValue();}
            if(attr.getValue( "name" ).equals("wcet")){taskTempWCET= (new Integer(attr.getValue("value"))).intValue();}
                if(attr.getValue( "name" ).equals("deadline")){taskTempDeadline= (new Integer(attr.getValue("value"))).intValue();}
if (inMessage) {
    if (attr.getValue("name").equals("deadline")) {
        messageTempDeadline = (new Integer(attr.getValue("value"))).intValue();
    }
    if (attr.getValue("name").equals("size")) {
        messageTempSize = (new Integer(attr.getValue("value"))).intValue();
    }
}

if (localName.equals("object-configuration")) {
    falseAll();
    inObjectConfiguration = true;
    tempID = attr.getValue("id");
}

if (localName.equals("config-attribute")) {
    if (inObjectConfiguration) {
        if (attr.getValue("name").equals("allocation_id")) {
            tempAllocationID = attr.getValue("value");
        }
        if (attr.getValue("name").equals("priority")) {
            tempPriority = (new Integer(attr.getValue("value"))).intValue();
        }
        if (aController.isProcessor(tempAllocationID)) {
            aController.addProcessorTaskAllocation(tempAllocationID, tempID, tempPriority, tempAllocationID);
        }
        if (aController.isNetwork(tempAllocationID)) {
            aController.addNetworkMessageAllocation(tempAllocationID, tempID, tempPriority, tempAllocationID);
        }
    }
}

public void endElement(String namespaceURI, String localName, String qName) throws SAXException {
    if (localName.equals("message")) {
        aController.addMessage(messageTempID, messageTempName, messageTempDeadline, messageTempSize);
    }
}
aController.addTaskMessageAllocation(tempSentFrom, tempSentTo, messageTempID);
}
if (localName.equals("task")) {
aController.addTask(taskTempID, taskTempName, taskTempPeriod, taskTempDeadline, taskTempWCET);
}
if (localName.equals("network")) {
aController.addNetwork(networkTempID, networkTempName, networkTempBandwidth, networkTempLatency);
}
if (localName.equals("hardware-module")) {
aController.addProcessor(processorTempID, processorTempName);
}

public void characters(char[] ch, int start, int length) throws SAXException {
    contents.write(ch, start, length);
}

public void parse(String file1Name, String file2Name) {
try {
    // Create SAX 2 parser...
    XMLReader xr = XMLReaderFactory.createXMLReader();
    // Set the ContentHandler...
    aController.createAllocation(1);
    xr.setContentHandler(this);
    // Parse the file...
    xr.parse(new InputSource(new FileReader(file1Name)));
    xr.parse(new InputSource(new FileReader(file2Name)));
} catch (Exception e) {
    e.printStackTrace();
}

    public Object[][] getNetworkProcessorAllocation(int solutionIndex){
aController.crossingSolutions(0, 0, 0,
(Vector)aController.allProcessors.clone(), new Vector(), new
Vector(), false);
return aController.getSolution(solutionIndex);
}
public Vector getProcessorMapping(){
return aController.getProcessorMapping();
}
public Vector getNetworks(){
return aController.getNetworks();
}
public void print(){
//aController.printTasks();
aController.printSolution(0);
aController.sortAllTasks();
aController.getMessagesSorted("PRC_0");
//Vector
networks=aController.getAllocatedNetworksForProcessor("CFM_1");
//System.out.println("***********************************************
*****");
//for(int i=0;i<networks.size();i++){
// System.out.println(((Network)networks.get(i)).getID());}
}
public Vector getAllocatedTasks(String pID){
return aController.getAllocatedTasksForProcessor(pID);
}
public Vector getMessages(String nID){
return aController.getAllocatedMessagesForNetwork(nID);
}
public Network getNetwork(String mID){
return aController.getAllocatedNetworkForMessage(mID);
}
public Task getTask(String tID){
return aController.getTask(tID);
}
public void sortMessages(String tID){
aController.getMessagesSorted(tID);
}
public Vector rankMessages(String tID){
return aController.rankMessages(tID);
}
public boolean checkNetwork(String nID,String tID){
return aController.checkNetwork(nID,tID);
}
8.3 AllocationController.java

package allocation;
/**
 * @author mak500
 */
import java.util.Vector;

public class AllocationController {

    private Vector allAllocationStores;
    public Vector allTasks;
    public Vector allMessages;
    public Vector allNetworks;
    public Vector allProcessors;
    private Vector processorWillBeUsed;
    private Vector processorUsed;
    private Vector networksUsed;
    private Vector CrossingSolutions;

    /** Creates a new instance of AllAllocations */
    public AllocationController(Vector allAllocationStores, Vector allTasks, Vector allMessages, Vector allNetworks, Vector allProcessors) {
        this.allAllocationStores = allAllocationStores;
        this.allTasks = allTasks;
        this.allMessages = allMessages;
        this.allNetworks = allNetworks;
        this.allProcessors = allProcessors;
    }

    public AllocationController() {
        allAllocationStores = new Vector();
        allTasks = new Vector();
        allMessages = new Vector();
        allNetworks = new Vector();
        allProcessors = new Vector();
        processorWillBeUsed = new Vector();
        processorUsed = new Vector();
        CrossingSolutions = new Vector();
    }

    public void setAllocationStores(Vector allAllocationStores) {
        this.allAllocationStores = allAllocationStores;
    }

    public Vector getAllocationStores() {
        return allAllocationStores;
    }

    public void createAllocation(int ID) {
        Vector allNetworkMessageAllocations = new Vector();
        Vector allProcessorTaskAllocations = new Vector();
        
    }
}


AllocationStore allocationStore=new AllocationStore(allNetworkMessageAllocations, allProcessorTaskAllocations, ID);
    allAllocationStores.add(allocationStore);
}

public void addTask(String ID,String name,int period,int deadline,int wcet){
    Task task=new Task(ID,name, period, deadline, wcet);
    allTasks.add(task);
}

public void addMessage(String ID,String name,int deadline,int size){
    Message message=new Message(ID,name,deadline, size);
    allMessages.add(message);
}

public void addNetwork(String ID,String name,int bandwidth,int latency){
    Network network=new Network(ID,name,bandwidth,latency);
    allNetworks.add(network);
}

public void addProcessor(String ID,String name){
    Processor processor=new Processor(ID,name);
    allProcessors.add(processor);
}

public Task getTask(String ID){
    for(int i=0;i<allTasks.size();i++){
        Task task=(Task)allTasks.get(i);
        if (task.getID().equals(ID)){return (Task)allTasks.get(i);}
    }
    return null;
}

public Message getMessage(String ID){
    for(int i=0;i<allMessages.size();i++){
        Message message=(Message)allMessages.get(i);
        if (message.getID().equals(ID)){return (Message)allMessages.get(i);}
    }
    return null;
}

public Network getNetwork(String ID){
    for(int i=0;i<allNetworks.size();i++){
        Network network=(Network)allNetworks.get(i);
        if (network.getID().equals(ID)){return (Network)allNetworks.get(i);}
    }
    return null;
}

public Processor getProcessor(String ID){
    for(int i=0;i<allProcessors.size();i++){
        Processor processor=(Processor)allProcessors.get(i);
        if (processor.getID().equals(ID)){return (Processor)allProcessors.get(i);}}
return null;
}

public int getProcessorIndex(String ID){
    for(int i=0;i<allProcessors.size();i++){
        Processor processor=(Processor)allProcessors.get(i);
        if (processor.getID().equals(ID)){return i;}
    }
    return -1;
}

public int getTaskIndex(String tID){
    for(int i=0;i<allTasks.size();i++){
        Task task=(Task)allTasks.get(i);
        if (task.getID().equals(tID)){return i;}
    }
    return -1;
}

public void addNetworkProcessorAllocation(String networkID,String processorID){
    //you should check whether the addition has been made before or not
    getNetwork(networkID).processors.add(getProcessor(processorID));
    getProcessor(processorID).networks.add(getNetwork(networkID));
}

public Vector getAllocatedNetworksForProcessor(String processorID){
    return getProcessor(processorID).networks;
}

public Vector getAllocatedProcessorsForNetwork(String networkID){
    return getNetwork(networkID).processors;
}

public void addTaskMessageAllocation(String t1ID,String t2ID,String mID){
    getMessage(mID).setSentFrom(getTask(t1ID));
    getMessage(mID).setSentTo(getTask(t2ID));
    getTask(t1ID).outMessages.add(getMessage(mID));
}

public Vector getPTA(){
    return ((AllocationStore)allAllocationStores.get(allAllocationStores.size()-1)).allProcessorTaskAllocations;
}

public Vector getNMA({
    return null;
}
public void addProcessorTaskAllocation(String pID, String tID, int priority, String allocationID) {
    // Here you should check for duplicated allocation, tasks, and processors IDs
    TaskPriorityAllocator tpAllocator = new TaskPriorityAllocator(allocationID, getTask(tID), priority);
    for (int i = 0; i < getPTA().size(); i++) {
        ProcessorTaskMapper ptm = (ProcessorTaskMapper) getPTA().get(i);
        if (ptm.getProcessor().getID().equals(pID)) {
            for (int j = 0; j < ptm.allocatedTasks.size(); j++) {
                if (priority > ((TaskPriorityAllocator) ptm.allocatedTasks.get(j)).getPriority()) {
                    ((ProcessorTaskMapper) getPTA().get(i)).allocatedTasks.add(j, tpAllocator);
                    return;
                }
            }
            ((ProcessorTaskMapper) getPTA().get(i)).allocatedTasks.add(tpAllocator);
            return;
        }
    }
    if (getProcessor(pID) != null && getTask(tID) != null) {
        Vector allocatedTasks = new Vector();
        allocatedTasks.add(tpAllocator);
        ProcessorTaskMapper newPtm = new ProcessorTaskMapper(getProcessor(pID), allocatedTasks);
        getPTA().add(newPtm);
    }
}

public Vector getAllocatedTasksForProcessor(String pID) {
    for (int i = 0; i < getPTA().size(); i++) {
        return ((AllocationStore) allAllocationStores.get(allAllocationStores.size() - 1)).allNetworkMessageAllocations;
    }
}
ProcessorTaskMapper
ptm=(ProcessorTaskMapper)getPTA().get(i);
   if(ptm.getProcessor().getID().equals(pID)){
      return
      ((ProcessorTaskMapper)getPTA().get(i)).allocatedTasks;
   }
   return null;
}

public Processor getAllocatedProcessorForTask(String tID){
   for(int i=0;i<getPTA().size();i++){
      ProcessorTaskMapper
      ptm=(ProcessorTaskMapper)getPTA().get(i);
      Vector allocatedTasks=ptm.allocatedTasks;
      for(int j=0;j<allocatedTasks.size();j++){
         Task
         task=((TaskPriorityAllocator)allocatedTasks.get(j)).getTask();
         if (task.getID().equals(tID)){
            return
            ((ProcessorTaskMapper)getPTA().get(i)).getProcessor();
         }
      }
   }
   return null;
}

public void addNetworkMessageAllocation(String nID,String mID,int priority,String allocationID){
   //Here you should check for duplicated allocation, messages, and networks IDs
   MessagePriorityAllocator mpAllocator=new MessagePriorityAllocator(allocationID,getMessage(mID), priority);
   for(int i=0;i<getNMA().size();i++){
      NetworkMessageMapper
      nmm=(NetworkMessageMapper)getNMA().get(i);
      if(nmm.getNetwork().getID().equals(nID)){
         ((NetworkMessageMapper)getNMA().get(i)).allocatedMessages.add(mpAllocator);
         return;
      }
   }
   if(getMessage(mID)!=null && getNetwork(nID)!=null){
      Vector allocatedMessages=new Vector();
      allocatedMessages.add(mpAllocator);
      NetworkMessageMapper newNmm=new NetworkMessageMapper(getNetwork(nID), allocatedMessages);
      getNetwork(nID).addNetworkMessage(newNmm);
      return;
   }
}

getNMA().add(newNmm);
}
}

public Vector getAllocatedMessagesForNetwork(String nID){
    for(int i=0;i<getNMA().size();i++){
        NetworkMessageMapper
            nmm=(NetworkMessageMapper)getNMA().get(i);
        if(nmm.getNetwork().getID().equals(nID)){
            return
((NetworkMessageMapper)getNMA().get(i)).allocatedMessages;
        }
    }
    return null;
}

public Network getAllocatedNetworkForMessage(String mID){
    for(int i=0;i<getNMA().size();i++){
        NetworkMessageMapper
            nmm=(NetworkMessageMapper)getNMA().get(i);
        Vector
            mpAllocators=nmm.allocatedMessages;
        for(int j=0;j<mpAllocators.size();j++){
            Message
                message=((MessagePriorityAllocator)mpAllocators.get(j)).getMessage();
            if (message.getID().equals(mID)){
                return
((NetworkMessageMapper)getNMA().get(i)).getNetwork();
            }
        }
    }
    return null;
}

public void sortAllTasks(){
    sortAllProcessorsForTasks();
    Vector tasks=new Vector();
    for(int i=0;i<allProcessors.size();i++){
        Processor
            p=(Processor)allProcessors.get(i);
        Vector
            tpAllocators=getAllocatedTasksForProcessor(p.getID());
        for(int j=0;j<tpAllocators.size();j++){
            TaskPriorityAllocator
                tpa=(TaskPriorityAllocator)tpAllocators.get(j);
            tasks.add( tpa.getTask() );
        }
    }
}
public int getPriority(String tID) {

    for (int i = 0; i < getPTA().size(); i++) {
        ProcessorTaskMapper ptm = (ProcessorTaskMapper) getPTA().get(i);
        Vector allocatedTasks = ptm.allocatedTasks;
        for (int j = 0; j < allocatedTasks.size(); j++) {
            Task task = ((TaskPriorityAllocator) allocatedTasks.get(j)).getTask();
            if (task.getID().equals(tID)) {
                return ((TaskPriorityAllocator) allocatedTasks.get(j)).getPriority();
            }
        }
    }

    return -1;
}

public void sortAllProcessorsForTasks() {
    allProcessors = processorUsed;
}

public void getMessagesSorted(String tID) {
    sortAllProcessorsForTasks();
    sortAllTasks();
    Vector v = rankMessages(tID);
    Task t = getTask(tID);
    if (t != null) {
        Vector messages = t.outMessages;
        for (int i = 0; i < messages.size(); i++) {
            for (int j = i; j < messages.size() - 1; j++) {
                Integer i1 = (Integer) v.get(i);
                Integer i2 = (Integer) v.get(j + 1);
                int p1 = i1.intValue();
                int p2 = i2.intValue();
                if (p1 > p2) {
                    Object temp = messages.get(i);
                    messages.set(i, messages.get(j + 1));
                    messages.set(j + 1, temp);
                }
            }
        }
    }
}
public Vector rankMessages(String tID) {
    Task task = getTask(tID);
    if (task != null) {
        Vector messages = task.outMessages;
        Vector messagesRanks = new Vector();
        int t1_index = getTaskIndex(tID);

        for (int i = 0; i < messages.size(); i++) {
            Message message = (Message) messages.get(i);
            Task task2 = message.getSentTo();
            int t2_index = getTaskIndex(task2.getID());

            int result = t1_index - t2_index;
            messagesRanks.add(new Integer(result));
        }

        return messagesRanks;
    }
    else return new Vector();
    //return sortAllTasks2(messagesRanks);
}

public void crossingSolutions(int solutionIndex, int matrixIndex, int pIndex, Vector pWBUsed, Vector pUsed, Vector networks2, boolean comingFromDown) {
    System.out.println(matrixIndex + " and "+solutionIndex);
System.out.println(allProcessors.size());
if (pUsed.size() == allProcessors.size() || pWBUsed.size() == 0)
{System.out.println("The solutions are generated");System.out.println(pWBUsed.size());processorUsed = pUsed; return;}

Object[][] solution;
Vector sprocessors;

if (matrixIndex == 0 && solutionIndex == 0)
{
    solution = new Object[allProcessors.size()][allNetworks.size()];
    CrossingSolutions.add(solution);
    Processor p = (Processor) allProcessors.get(pIndex);
    Vector networks = p.networks;
    for (int i = 0; i < networks.size(); i++)
    {
        Network n = (Network) networks.get(i);
        solution[0][i] = n;
        networks2.add(n);
    }
    CrossingSolutions.set(0, solution);
}
else
{
    solution = (Object[][]) CrossingSolutions.get(solutionIndex);
}

pUsed.add(allProcessors.get(pIndex));
for (int k = 0; k < pWBUsed.size(); k++)
{
    Processor pp = (Processor) pWBUsed.get(k);
    Processor tbr = (Processor) allProcessors.get(pIndex);
    if (pp.getID().equals(tbr.getID())) {pWBUsed.remove(k); break;}
}

if (comingFromDown) {pIndex = getProcessorIndex(((Processor) pUsed.get(pUsed.size() - 1)).getID());}

sprocessors = similarProcessors(pUsed, pWBUsed);
int sizex = 1;
if (sprocessors.size() > 0) {sizex = 1;} else {sizex = 0;}
System.out.println("We are still ok");
for (int i = 0; i < sizex; i++)
{
    boolean createSolution = true;
    Processor p = (Processor) sprocessors.get(i);
    if (i == 0) createSolution = false;
    upSolution(matrixIndex, solutionIndex, networks2, p, pWBUsed, pUsed, createSolution);
}
processorUsed = pUsed;
networksUsed = networks2;
public void printSolution(int solutionIndex) {

    Object[][] solution = (Object[][]) CrossingSolutions.get(solutionIndex);
    System.out.println("We are still ok here too print");
    for (int i = 0; i < processorUsed.size(); i++) {
        Processor p = (Processor) processorUsed.get(i);
        System.out.println(p.getID());
        for (int j = 0; j < allNetworks.size(); j++) {
            if (solution[i][j] != null) {
                Network n = (Network) solution[i][j];
                System.out.println(n.getID());
            } else {
                System.out.println("Null");
            }
        }
    }
}

public void printTasks() {
    sortAllTasks();

    for (int i = 0; i < allTasks.size(); i++) {
        Task t = (Task) allTasks.get(i);
        System.out.println(t.getID() + " +getPriority(t.getID()) + " +getAllocatedProcessorForTask(t.getID()).getID());
        System.out.println("**************************************************************");
        getMessagesSorted(t.getID());
        for (int j = 0; j < t.outMessages.size(); j++) {
            Message m = (Message) t.outMessages.get(j);
            System.out.println("Message :
" + m.getID() + " From "+ m.getSentFrom().getID() + " To "+ m.getSentTo().getID());
        }
    }
}

public void upSolution(int matrixIndex, int solutionIndex, Vector networks1, Processor p2, Vector pWBUsed, Vector pUsed, boolean createSolution) {

    Object[][] solution = (Object[][]) CrossingSolutions.get(solutionIndex);
Vector networks2=(Vector)p2.networks.clone();
Vector tempnetworks=new Vector();

int count=0;

nextpIndex=getProcessorIndex(p2.getID());

for(int m=0;m<networks1.size();m++){
    Network n1=(Network)networks1.get(m);
    for(int o=0;o<networks2.size();o++){
        Network n2=(Network)networks2.get(o);
        if(n1.getID().equals(n2.getID())){
            solution[matrixIndex+1][m]=n2;tempnetworks.add(n2);
        }
    }
}

System.out.println("Count is "+count);

for(int i=0;i<tempnetworks.size();i++){
    Network ne1=(Network)tempnetworks.get(i);
    for(int j=0;j<networks2.size();j++){
        Network ne2=(Network)networks2.get(j);
        if(ne2.getID().equals(ne1.getID())){networks2.remove(j);}
    }
}

for(int q=0;q<networks2.size();q++){
    Network t=(Network)networks2.get(q);
    solution[matrixIndex+1][networks1.size()]=t;
    networks1.add(t);
    count++;
}

if(createSolution){CrossingSolutions.add(solution);solutionIndex++;}
else{CrossingSolutions.set(solutionIndex,solution); }

matrixIndex++;
crossingSolutions(solutionIndex, matrixIndex, nextpIndex, pWBUsed, pUsed,networks1, false);
public void downSolution(int matrixIndex, int solutionIndex, Processor p2, Vector pWBUsed, Vector pUsed, Vector networks1) {

    Object[][] solution = (Object[][]) CrossingSolutions.get(solutionIndex);

    Object[][] temp = solution;
    int max = 0;
    int count = 0;
    // Vector networks1 = new Vector();
    Vector networks2 = p2.networks;
    for (int i = 0; i < matrixIndex; i++) {
        for (int j = 0; j < allNetworks.size(); j++) {
            if (temp[i][j] != null) solution[i + 1][j] = temp[i][j];
        }
    }
    temp = solution;

    max = ((Processor) processorUsed.get(0)).networks.size();
    for (int q = 1; q < processorUsed.size(); q++) {
        int result = ((Processor) processorUsed.get(q)).networks.size();
        if (max < result) max = result;
    }

    for (int m = 0; m < networks1.size(); m++) {
        Network n1 = (Network) networks1.get(m);
        for (int o = 0; o < networks2.size(); o++) {
            Network n2 = (Network) networks2.get(o);
            if (n1.getID().equals(n2.getID())) { count++;

            }
        }
    }

    for (int i = 0; i < matrixIndex; i++) {
        for (int j = 0; j < max; j++) {
            if (temp[i][j] != null) solution[i + networks2.size() - count][j] = temp[i][j];
        }
    }

    Vector vtemp = new Vector();

for(int 
s=0;s<allNetworks.size();s++){
    if(solution[0][s]!=null)networks1.add(solution[0][s]);
    for(m=0;m<networks1.size();m++){
        n1=(Network) networks1.get(m);
        for(int 
o=0;o<networks2.size();o++){
            n2=(Network)networks2.get(o);
            if(n1.getID().equals(n2.getID())){ solution[0][m]=n1;count++;
            } else{vtemp.add(n2);}
        }
    }
    for(int q=0;q<vtemp.size();q++){
        Network t=(Network)vtemp.get(q);
        solution[0][q]=t;
    }
}
CrossingSolutions.add(solution);
solutionIndex++;matrixIndex++;
crossingSolutions(solutionIndex,
matrixIndex, getProcessorIndex(p2.getID()), pWBUsed,
pUsed,networks1,true);
}

public Vector similarProcessors(Vector pUsed,Vector pWBUsed){
    double[] result=new double[allProcessors.size()];
    double max=0.0;
    Vector processors=new Vector();
    for(int m=0;m<pUsed.size();m++){
        Processor p1=(Processor)pUsed.get(m);
        for(int i=0;i<pWBUsed.size();i++){
            Processor p2=(Processor)pWBUsed.get(i);
            result[i]=result[i]+(similarNetworks(p1,p2))*(m+1);
        }
    }
}
max=result[0];

for(int j=1;j<pWBUsed.size();j++)
{
    if(max<result[j])max=result[j];
}

for(int k=0;k<pWBUsed.size();k++){
    Processor pk=(Processor)pWBUsed.get(k);
    if(result[k]==max)processors.add(pk);
}

return processors;
}

public double similarNetworks(Processor p1, Processor p2){
    Vector n1=p1.networks;
    Vector n2=p2.networks;
    double result=0.0;
    int count=0;

    for(int i=0;i<n2.size();i++){
        String n1_id=((Network)n2.get(i)).getID();
        for(int j=0;j<n1.size();j++){
            String n2_id=((Network)n1.get(j)).getID();
            if(n2_id.equals(n1_id)){result=result+1.1;count++;}
        }
    }
    result=result-(n2.size()-count);
    return result;
}

public boolean isProcessor(String ID){
    if (getProcessor(ID)!=null){return true;}
    else{return false;}
}

public boolean isNetwork(String ID){
    if (getNetwork(ID)!=null){return true;}
    else{return false;}
}

public Object[][] getSolution(int solutionIndex){
    Object[][] solution=(Object[][])CrossingSolutions.get(solutionIndex);
    return solution;
}

public Vector getProcessorMapping(){
    return processorUsed;
}
public Vector getNetworks()
{
    return networksUsed;
}

public boolean checkNetwork(String nID,String tID)
{
    Processor p=getAllocatedProcessorForTask(tID);
    Vector networks=getAllocatedNetworksForProcessor(p.getID());
    for(int i=0;i<networks.size();i++){
        Network n=(Network)networks.get(i);
        if(n.getID().equals(nID)){
            return true;
        }
    }
    return false;
}

8.4 AllocationStore.java

package allocation;

/**
 * @author mak500
 */
import java.util.Vector;

public class AllocationStore {
    public Vector allNetworkMessageAllocations;
    public Vector allProcessorTaskAllocations;
    private int ID;

    /** Creates a new instance of AllAllocations */
    public AllocationStore(Vector allNetworkMessageAllocations,Vector allProcessorTaskAllocations,int ID) {
        this.allNetworkMessageAllocations=allNetworkMessageAllocations;
        this.allProcessorTaskAllocations=allProcessorTaskAllocations;
        this.ID=ID;
    }
}
package allocation;

/**
 * @author mak500
 */
public class Message {

    private String ID;
    private String name;
    private int deadline;
    private int size;
    private Task sentTo;
    private Task sentFrom;

    /** Creates a new instance of Message */
    public Message(String ID, String name, int deadline, int size) {
        this.ID = ID;
        this.name = name;
        this.deadline = deadline;
        this.size = size;
    }

    public void setID(String ID) {
        this.ID = ID;
    }

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

    public void setSize(int size) {
        this.size = size;
    }

    public void setDeadline(int deadline) {
        this.deadline = deadline;
    }

    public void setSentTo(Task sentTo) {
        this.sentTo = sentTo;
    }

    public void setSentFrom(Task sentFrom) {
        this.sentFrom = sentFrom;
    }

    public String getName() {
        return name;
    }

    public String getID() {
        return ID;
    }

    public int getSize() {
        return size;
    }

    public int getDeadline() {

return deadline;
}
public Task getSentTo(){
    return sentTo;
}
public Task getSentFrom(){
    return sentFrom;
}

8.6 MessagePriorityAllocator.java

package allocation;

/**
 * @author mak500
 */
public class MessagePriorityAllocator {

    private String allocationID;
    private Message message;
    private int priority;

    /** Creates a new instance of MessagePriorityAllocator */
    public MessagePriorityAllocator(String allocationID, Message message, int priority) {
        this.allocationID = allocationID;
        this.message = message;
        this.priority = priority;
    }

    public void setAllocationID(String allocationID){
        this.allocationID = allocationID;
    }
    public void setMessage(Message message){
        this.message = message;
    }
    public void setPriority(int priority){
        this.priority = priority;
    }

    public String getAllocationID(){
        return allocationID;
    }
    public Message getMessage(){
        return message;
    }
    public int getPriority(){
        return priority;
    }
}
8.7 Network.java

package allocation;

/**
 * @author mak500
 */
import java.util.Vector;

public class Network {
    private String ID;
    private String name;
    private int bandwidth;
    private int latency;
    public Vector processors;
    /** Creates a new instance of Network */
    public Network(String ID, String name,int bandwidth, int latency) {
        this.ID=ID;
        this.name=name;
        this.bandwidth=bandwidth;
        this.latency=latency;
        processors=new Vector();
    }

    public void setID(String ID){
        this.ID=ID;
    }

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

    public void setBandwidth(int bandwidth){
        this.bandwidth=bandwidth;
    }

    public void setLatency(int latency){
        this.latency=latency;
    }

    public String getName(){
        return name;
    }

    public String getID(){
        return ID;
    }

    public int getBandwidth(){
        return bandwidth;
    }

    public int getLatency(){
        return latency;
    }
}
8.8 NetworkMessageMapper.java

```java
package allocation;

/**
 * @author mak500
 */
import java.util.Vector;

public class NetworkMessageMapper {
    private Network network;
    public Vector allocatedMessages;
    /** Creates a new instance of NetworkMessageMapper */
    public NetworkMessageMapper(Network network, Vector allocatedMessages) {
        this.network=network;
        this.allocatedMessages=allocatedMessages;
    }
    public void setNetwork(Network network){
        this.network=network;
    }
    public Network getNetwork(){
        return network;
    }
}
```

8.9 Processor.java

```java
package allocation;

/**
 * @author mak500
 */
import java.util.Vector;

public class Processor {
    private String ID;
    private String name;
    public Vector networks;
    /** Creates a new instance of Processor */
    public Processor(String ID,String name) {
        this.ID=ID;
        this.name=name;
        networks=new Vector();
    }
    public String getName(){
        return name;
    }
}
```
```java
public String getID()
{
    return ID;
}
}

8.10 ProcessorTaskMapper.java
package allocation;
/**
 * @author mak500
 */
import java.util.Vector;
public class ProcessorTaskMapper {
    private Processor processor;
    public Vector allocatedTasks;
    /** Creates a new instance of ProcessorTaskMapper */
    public ProcessorTaskMapper(Processor processor, Vector allocatedTasks) {
        this.processor=processor;
        this.allocatedTasks=allocatedTasks;
    }
    public void setProcessor(Processor processor){
        this.processor=processor;
    }
    public Processor getProcessor(){
        return processor;
    }
}

8.11 Task.java
package allocation;
/**
 * @author mak500
 */
import java.util.*;
public class Task {
    /** Creates a new instance of Task */
    private String ID;
    private String name;
    private int period;
    private int deadline;
    private int wcet;
    private Vector outMessages;
    public Task(){
    }
```
public Task(String ID, String name, int period, int deadline, int wcet) {
    this.ID = ID;
    this.name = name;
    this.period = period;
    this.deadline = deadline;
    this.wcet = wcet;
    outMessages = new Vector();
}

public void setID(String ID) {
    this.ID = ID;
}

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

public void setPeriod(int period) {
    this.period = period;
}

public void setDeadline(int deadline) {
    this.deadline = deadline;
}

public void setWCET(int wcet) {
    this.wcet = wcet;
}

public String getName() {
    return name;
}

public String getID() {
    return ID;
}

public int getPeriod() {
    return period;
}

public int getDeadline() {
    return deadline;
}

public int getWCET() {
    return wcet;
}

//public void sortMessages()

8.12 TaskPriorityAllocator.java
package allocation;

/**
 * @author mak500
 */
public class TaskPriorityAllocator {
    private String allocationID;
}
private Task task;
private int priority;
/** Creates a new instance of TaskPriorityAllocator */
public TaskPriorityAllocator(){
}

public TaskPriorityAllocator(String allocationID, Task task, int priority) {
    this.allocationID = allocationID;
    this.task = task;
    this.priority = priority;
}

public void setAllocationID(String allocationID) {
    this.allocationID = allocationID;
}
public void setTask(Task task) {
    this.task = task;
}
public void setPriority(int priority) {
    this.priority = priority;
}

public String getAllocationID() {
    return allocationID;
}

public Task getTask() {
    return task;
}
public int getPriority() {
    return priority;
}

}
private int count=0;
private double initially=0.0;
public Vector processorsCoordinates;
public Vector networksCoordinates;
public Vector distancesBetweenNetworks;
public Vector taskInformations;
/** Creates a new instance of ProcessorNetworkDrawer */
public Drawer(Point origin, Point distanceBetweenProcessors, Point
distanceBetweenNetworks, FileProcessingController fpc) {
    this.origin=origin;
    this.distanceBetweenProcessors=distanceBetweenProcessors;
    this.distanceBetweenNetworks=distanceBetweenNetworks;
    this.fpc=fpc;
    graphText="";
    distanceBetweenMessages=20.0;
    distanceBetweenTasksAndProcessors=distanceBetweenNetworks.getY()+25.0;
    processorsCoordinates=new Vector();
    networksCoordinates=new Vector();
    distancesBetweenNetworks=new Vector();
    nodeGroups=new Vector();
    taskInformations=new Vector();
    solution=fpc.getNetworkProcessorInformation(0);
    processorMapping=fpc.getProcessorMapping();
    networks=fpc.getNetworks();
    double x=0.0;
    for(int i=0;i<networks.size();i++){
        x=0.0;
        double y=0.0;
        networksCoordinates.add(new Point(x, y));
    }
}

public double getNetworkX(String nID){
    for(int i=0;i<networks.size();i++){
        Network n=(Network)networks.get(i);
        if(n.getID().equals(nID))return
            ((Point)networksCoordinates.get(i)).getX();
    }
    return 0;
}

public String drawNodeImage(String image,String label,int id,double x,double y,String labelPosition){
count++;
String graphText=" ";
graphText=graphText+"node"+"\n";
graphText=graphText+"["+"\n";
graphText=graphText+id +id+"\n";
graphText=graphText+label +""+label+"\n";
graphText=graphText+graphics +"\n";
graphText=graphText+[ "+"\n";
graphText=graphText+x +x+"\n";
graphText=graphText+"y "+y+"\n";

null
public String drawTask(String shape, String label, int id, double x, double y, double width, double height) {
    count++;
    String here = "";
    String graphText = "";
    graphText = graphText + "node" + "\n";
    graphText = graphText + "[" + "\n";
    graphText = graphText + "id    "+id+"\n";
    graphText = graphText + "label  " + "+label+"\n";
    graphText = graphText + "graphics    " + "\n";
    graphText = graphText + "[    " + "\n";
    graphText = graphText + "x    " + "+x+"\n";
    graphText = graphText + "y    " + "+y+"\n";
    graphText = graphText + "w    " + "+ width+"\n";
    graphText = graphText + "h    " + "+ height+"\n";
    graphText = graphText + "type  " + shape + "" + "\n";
    graphText = graphText + "fill  " + \"#FFFF99\" + \"\n";
    graphText = graphText + "outline  " + \"#000080\" + \"\n";
    if(label.equals("")) {here = "";} else {here = " text " + label + " color " + \"#000080\" + \"\n";
    graphText = graphText + "ncolor  " + \"#000080\" + \"\n";
    graphText = graphText + "nfontName  " + \"Tahoma\" + \"\n";
    graphText = graphText + "nmodel  " + \"sides\" + \"\n";
    graphText = graphText + "nanchor  " + \"c\" + \"\n";
    LabelGraphics = " + "\n";
    return graphText;
}

public String drawSimpleUndirectedEdge(int source, int target) {
    String graphText = "";
    graphText = graphText + "edge" + "\n";
    graphText = graphText + "[" + "\n";
    graphText = graphText + "source    " + +source+"\n";
    graphText = graphText + "target    " + +target+"\n";
    graphText = graphText + "graphics    " + "\n";
    graphText = graphText + "[    " + "\n";
    graphText = graphText + "fill  " + \"#000080\" + \"\n";
    return graphText;
}

public String drawMessagedEdges(int source, int target) {
    String graphText = "";
    graphText = graphText + "edge" + "\n";
    graphText = graphText + "[" + "\n";
    graphText = graphText + "source    " + +source+"\n";
    graphText = graphText + "target    " + +target+"\n";
    graphText = graphText + "graphics    " + "\n";
    graphText = graphText + "[    " + "\n";
    graphText = graphText + "fill  " + \"#000080\" + \"\n";
    graphText = graphText + \"dotted\" + \"\n";
    return graphText;
}
public String drawMessage(int source, int target, String label, String color) {
    String graphText = "";
    graphText += "edge" + "\n";
    graphText += "[" + "\n";
    graphText += "source " + source + "\n";
    graphText += "target " + target + "\n";
    graphText += "label " + label + "\n";
    graphText += "graphics "+"\n";
    graphText += "fill " + color + "\n";
    graphText += "sourceArrow " + "white_diamond"
    graphText += "targetArrow " + "delta" + "\n";
    graphText += "LabelGraphics "+"\n";
    graphText += "text " + label + "\n";
    graphText += "color " + "#000080" + "\n";
    graphText += "fontSize 8" + "\n";
    graphText += "fontName " + "Dialog" + "\n";
    graphText += "rotationAngle 270.0" + "\n";
    graphText += "model " + "six_pos" + "\n";
    graphText += "position " + "head" + "\n";
    graphText += "]" + "\n"
    return graphText;
}

public void drawNetworks() {
    int last = 0;
    for (int j = 0; j < networks.size(); j++) {
        String label = ((Network) networks.get(j)).getID() + "";
        double x = ((Point) networksCoordinates.get(j)).getX();
        double y = ((Point) processorsCoordinates.get(0)).getX() + distanceBetweenNetworks.getY();
        double groupHeight = taskInformations.size() * distanceBetweenTasks * 1.23;
        double yGroup = y - distanceBetweenMessages * 2 + groupHeight / 2;
        NodeGroup n = (NodeGroup) nodeGroups.get(j);
        double xGroup = n.getX();
        double widthGroup = n.getWidth();
        String image = "http://www.yworks.com/yed/resource/clients.png";
        graphText += this.drawNodeGroup("rectangle", "", count * 40, xGroup, yGroup, widthGroup, groupHeight);
        graphText += drawNodeImage(image, label, count, x, y, "s");
    }
    last = count - 1 - networks.size();
    for (int k = 0; k < processorMapping.size(); k++) {
        for (int l = 0; l < networks.size(); l++) {
double x=((Point)networksCoordinates.get(l)).getX();
double y=((Point)processorsCoordinates.get(k)).getY();
if(solution[k][l]!=null){
    graphText=graphText+drawNodeShape("diamond",
"",count ,x,y,15.0,15.0, "#003366");
    graphText=graphText+drawSimpleUndirectedEdge(count-1, k);
    graphText=graphText+drawSimpleUndirectedEdge(count-1, last+l+1);
}
}
}

public String draw(){
    int i=0;int j=0;int k=0;int l=0;
    for(i=0;i<processorMapping.size();i++){
        String label=(((Processor)processorMapping.get(i)).getID()+"";
        double x=(origin.getX()+(processorMapping.size()-1-i)*distanceBetweenProcessors.getX());
        double y=(origin.getY()+(processorMapping.size()-1-i)*distanceBetweenProcessors.getY());
        processorsCoordinates.add(new Point(x, y));
        graphText=graphText+drawNodeImage(image, label, count, x, y, "s");
    }
    initialy=initialy+((Point)processorsCoordinates.get(0)).getY()+60.0;
    for(k=0;k<processorMapping.size();k++){
        Processor p=((Processor)processorMapping.get(k));
        Vector allocatedTasks=fpc.getAllocatedTasks(p.getID());
        double initialx=((Point)processorsCoordinates.get(k)).getX();
        initialy=initialy+distanceBetweenTasks*1.3;
        for(l=0;l<allocatedTasks.size();l++){
            TaskPriorityAllocator tpa=(TaskPriorityAllocator)allocatedTasks.get(l);
            Task t=tpa.getTask();
            initialy=initialy+distanceBetweenTasks;
            graphText=graphText+this.drawTask("rectangle",
            t.getID(),count,initialx,initialy, 40.0, 20.0);
            taskInformations.add( new TaskInformation(t.getID(),
            count-1, initialy, initialx,networks.size()));
            if(l==0){graphText=graphText+drawSimpleUndirectedEdge(count-1,k);} 
            else{graphText=graphText+drawSimpleUndirectedEdge(count-1,count-2); }
    }
drawMessages();
return graphText;
}

public void savePositions(){
    int i=0; int j=0;
    for(i=0; i<taskInformations.size(); i++){
        TaskInformation tI1=(TaskInformation)taskInformations.get(i);
        Task t1=fpc.getTask(tI1.getTaskID());
        fpc.sortMessages(t1.getID());
        Vector messages=t1.outMessages;
        for(j=0; j<messages.size(); j++){
            int position=0;
            int position2=0;
            Message m=(Message)messages.get(j);
            Network n=(Network)fpc.getNetwork(m.getID());
            int nIndex=getNetworkIndex(n.getID());
            position=((Integer)tI1.lastPosition.get(nIndex)).intValue();
            position++;
            Task t2=m.getSentTo();
            TaskInformation tI2=getTaskInformation(t2.getID());
            int t2Index=getTaskIndex(t2.getID());
            if(t2Index>=getTaskIndex(t1.getID())){
                setPositions(i, t2Index, position, nIndex);
            }else{
                int max=maxPositions(t2Index, i, nIndex);
                setPositions(t2Index, i, max+1, nIndex);
                position=max+1;
            }
        }
    }
}

public void zeroPositions(){
    int i=0; int j=0;
    for(i=0; i<taskInformations.size(); i++){
        TaskInformation tI1=(TaskInformation)taskInformations.get(i);
        Task t1=fpc.getTask(tI1.getTaskID());
        Vector messages=t1.outMessages;
        for(j=0; j<networks.size(); j++){
            tI1.lastPosition.set(j, new Integer("0"));
        }
    }
}
public void setNetworksXAxes(){
    Point point=(Point)processorsCoordinates.get(0);
    double xPosition=point.getX();

    for(int i=0;i<networks.size();i++){
        Network n=(Network)networks.get(i);
        Vector positions=new Vector();
        for(int j=0;j<taskInformations.size();j++){
            TaskInformation t=(TaskInformation)taskInformations.get(j);
            Vector allPositions=t.lastPosition;
            positions.add(t.lastPosition.get(i));
        }

        int max=0;
        for(int k=0;k<positions.size();k++){
            int p=((Integer)positions.get(k)).intValue();
            if(p>max)max=p;
        }

        System.out.println("Network "+n.getID()+" has"+max+" positions");

        double yPoint=((Point)networksCoordinates.get(i)).getY();
        xPosition=xPosition+distanceBetweenMessages*3.5+(max)*distanceBetweenMessages;
        networksCoordinates.set(i, new Point(xPosition, yPoint));
        double width=distanceBetweenMessages*2+(max)*distanceBetweenMessages;
        NodeGroup ng=new NodeGroup(0.0,width,xPosition-distanceBetweenMessages*max-distanceBetweenMessages+(width/2),0.0 );
        nodeGroups.add(ng);
    }

    drawNetworks();
}

public void drawMessages(){
    savePositions();
    setNetworksXAxes();
    zeroPositions();
    int i=0;int j=0;
    for(i=0;i<taskInformations.size();i++){
        TaskInformation tIl=(TaskInformation)taskInformations.get(i);
        Task t1=fpc.getTask(tIl.getTaskID());
        // Additional code here...
    }
}
fpc.sortMessages(t1.getID());
Vector messages=t1.outMessages;

for(j=0;j<messages.size();j++){
    int position=0;
    int position2=0;
    Message m=(Message)messages.get(j);
    Network n=(Network)fpc.getNetwork(m.getID());
    int nIndex=getNetworkIndex(n.getID());
    position=((Integer)tI1.lastPosition.get(nIndex)).intValue();
    position++;
    
    Task t2=m.getSentTo();
    TaskInformation tI2=getTaskInformation(t2.getID());
    int t2Index=getTaskIndex(t2.getID());
    if(t2Index>getTaskIndex(t1.getID())){
        setPositions(i, t2Index, position,nIndex);
    }
    else{
        int max=maxPositions(t2Index, i,nIndex);
        setPositions(t2Index, i, max+1,nIndex);
        position=max+1;
    }
    position--;  

double x=getNetworkX(n.getID())-  
(position)*distanceBetweenMessages;
    double y1=tI1.getY();
    double y2=tI2.getY();

    graphText=graphText+drawNodeShape("ellipse", "",
    count, x, y1, 0.0, 0.0,"#003366");
    graphText=graphText+drawNodeShape("ellipse", "",
    count, x, y2, 0.0, 0.0, "#003366");
    int dif=java.lang.Math.abs(t2Index-
    getTaskIndex(t1.getID()));

    if(dif>m.getID().length()+2)
    {
        dif=m.getID().length()+2;
        //graphText=graphText+drawMessage(count-2, count-1,
        m.getID().substring(0, dif-2)+" "+position);
    }
    else{
        if(dif<3)(dif=4;)
        //graphText=graphText+drawMessage(count-2, count-1,
        m.getID().substring(0, dif-2)+"..."+ " "+position);
    }

    int graphID1=tI1.getGraphID();
    int graphID2=tI2.getGraphID();
    graphText=graphText+drawMessagedEdges(count-2,
    graphID1);
    graphText=graphText+drawMessagedEdges(count-1,
    graphID2);
public void drawMessagesAgain()
{
    String color="";
    zeroPositions();
    int i=0; int j=0;
    for(i=0; i<taskInformations.size(); i++){
        TaskInformation tI1=(TaskInformation) taskInformations.get(i);
        Task t1=fpc.getTask(tI1.getTaskID());
        fpc.sortMessages(t1.getID());
        Vector messages=t1.outMessages;
        for(j=0; j<messages.size(); j++){
            int position=0;
            int position2=0;
            Message m=(Message) messages.get(j);
            Network n=(Network)fpc.getNetwork(m.getID());
            int nIndex=getNetworkIndex(n.getID());
            position= ((Integer) tI1.lastPosition.get(nIndex)).intValue();
            position++;
            Task t2=m.getSentTo();
            TaskInformation tI2=getTaskInformation(t2.getID());
            int t2Index=getTaskIndex(t2.getID());
            if(t2Index>getTaskIndex(t1.getID())){
                setPositions(i, t2Index, position, nIndex);
            }else{
                int max=maxPositions(t2Index, i, nIndex);
                setPositions(t2Index, i, max+1, nIndex);
                position=max+1;
            }
            position--;
            double x=getNetworkX(n.getID()) -
            (position)*distanceBetweenMessages;
            double y1=tI1.getY();
            double y2=tI2.getY();
            graphText=graphText+drawNodeShape("rectangle", ",", count, x, y1, 0.0, 0.0, ",#C0C0C0");
            graphText=graphText+drawNodeShape("rectangle", ",", count, x, y2, 0.0, 0.0, ",#C0C0C0");
            int dif=java.lang.Math.abs(t2Index-
            getTaskIndex(t1.getID()));
            if(fpc.checkNetwork(n.getID(), t2.getID()))
            {color="#800000";}
            else{color="#FF0000";}
        }
    }

    drawMessagesAgain();
}
if (dif > m.getID().length() / 2)
{
    dif = m.getID().length() + 2;
    graphText = graphText + drawMessage(count - 2, count - 1, m.getID().substring(0, dif - 2), color);
}
else{
    if (dif <= 3) {dif = 4;}
    graphText = graphText + drawMessage(count - 2, count - 1, m.getID().substring(0, dif - 2) + "...", color);
}

public void setPositions(int i, int j, int lastPosition, int nIndex) {
    for (int s = i; s < j; s++) {
        ((TaskInformation) taskInformations.get(s)).lastPosition.set(nIndex, new Integer(lastPosition));
    }
}

public int maxPositions(int i, int j, int nIndex) {
    int max = 0;
    for (int s = i; s < j; s++) {
        int m = ((Integer) ((TaskInformation) taskInformations.get(s)).lastPosition.get(nIndex)).intValue();
        if (m > max) max = m;
    }
    return max;
}

public TaskInformation getTaskInformation(String tID) {
    for (int i = 0; i < taskInformations.size(); i++) {
        TaskInformation ti = (TaskInformation) taskInformations.get(i);
        if (ti.getTaskID().equals(tID)) return ti;
    }
    return null;
}

public int getTaskIndex(String tID) {
    for (int i = 0; i < taskInformations.size(); i++) {
        TaskInformation ti = (TaskInformation) taskInformations.get(i);
        if (ti.getTaskID().equals(tID)) return i;
    }
    return -1;
}
public int getNetworkIndex(String nID) {
    for (int i = 0; i < networks.size(); i++) {
        Network n = (Network) networks.get(i);
        if (n.getID().equals(nID)) return i;
    }
    return -1;
}

8.14 Main.java
package graph;

/**
   *
   * @author mak500
   */
import fileprocessing.*;
import allocation.*;
import java.util.Vector;
public class Main {

    /** Creates a new instance of Main */
    public Main() {
    }

    public static void main(String[] args) {
        FileProcessingController fpc = new FileProcessingController();
        fpc.readFile("sys.xml","conf.xml");
        Drawer drawer = new Drawer(new Point(0.0, 0.0), new Point(40.0, 40.0),
            new Point(60.0, 60.0), fpc);
        String result = drawer.draw();
        String string1 = "graph
            [
            hierarchic   1
            label ""
directed 1
            
            ]
            
            fpc.writeFile("output.gml", string1 + result + string2);
            System.out.println(string1 + result + string2);
    }
}

8.15 NodeGroup.java
package graph;

/**
   *
   * @author mak500
public class NodeGroup {

/** Creates a new instance of NodeGroup */
private double height;
private double width;
private double x;
private double y;

public NodeGroup(double height, double width, double x, double y) {
    this.height = height;
    this.width = width;
    this.x = x;
    this.y = y;
}

public double getHeight() {
    return height;
}

public double getX() {
    return x;
}

public double getY() {
    return y;
}

public double getWidth() {
    return width;
}

public void setHeight(double height) {
    this.height = height;
}

public void setWidth(double width) {
    this.width = width;
}

public void setX(double x) {
    this.x = x;
}

public void setY(double y) {
    this.y = y;
}

}

8.16 Point.java

package graph;

/**
 * @author mak500
 */
public class Point {

    private double x;
    private double y;

    /** Creates a new instance of Point */
    public Point(double x, double y) {

this.x=x;
this.y=y;
}

public void setX(double x)
{
    this.x=x;
}

public void setY(double y)
{
    this.y=y;
}

public double getX()
{
    return x;
}

public double getY()
{
    return y;
}

8.17 TaskInformation.java
package graph;

/**
 * @author mak500
 */
import java.util.Vector;
public class TaskInformation {

    /** Creates a new instance of TaskInformation */
    private String taskID;
    private int graphID;
    private double y;
    private double x;
    private int numberOfNetworks;
    public Vector lastPosition;

    public TaskInformation(String taskID, int graphID, double y, double x, int numberOfNetworks) {
        this.taskID=taskID;
        this.graphID=graphID;
        this.y=y;
        this.x=x;
        lastPosition=new Vector();

        for(int i=0;i<numberOfNetworks;i++){
            lastPosition.add(new Integer("0"));
        }
    }

    public String getTaskID(){
        return taskID;
    }
}
public int getGraphID()
{
    return graphID;
}

public double getY()
{
    return y;
}

public double getX()
{
    return x;
}