Insight in Scheduling Choices: A Visualisation Tool for SDF Graphs

Sven Elsert Santema

1st Supervisor: Dr. M. I. A. Stoelinga
2nd Supervisor: W. Ahmad MSc

12th of July, 2016

List of Figures

Abstract

Modern streaming applications are becoming increasingly demanding, requiring higher throughput, smaller buffer sizes and execute on less resources. Synchronous dataflow (SDF) graphs are a widely used formalism for the analysis of data flow in streaming applications, for both single processor and multiprocessor applications Despite state of the art scheduling methods for SDF graphs very little visualisations options are available for people who work with SDF graphs. Interviews have shown that these tools are self made and not very matured, even though they desire to have such tools. Therefore this project has developed a visualisation tool for SDF schedules called SDF Fish. The tool can load in data files of schedules and visualise these. The visualisations exist out of a Gantt-like chart and a snapshot visualisation of the resources. Users can navigate through the schedules by time controls. Furthermore, an automatic play function is featured. In order to provide overview for bigger schedules the tool offers zoom scale and filter capabilities.

Table of Content

Title **Page No.**

1. Introduction

Modern streaming applications, such as Skype [1], are becoming increasingly demanding. Skype started out as a video chat application for two people, but now also supports video conferencing for groups and even supports screen sharing during calls. On the one hand applications like Skype demand a high throughput in order to become real-time. On the other hand, they try to minimise resource requirements (buffer sizes, number of processors, energy consumption). In order to fulfill these demands smart scheduling strategies are needed that balance between these conflicting requirements. Synchronous dataflow (SDF) graphs are a formalism that is widely used for such analysis and generation of schedules with certain optimal properties [2].

1.1. Context

Synchronous dataflow (SDF) graphs are a widely used formalism for the analysis of data flow in streaming applications, for both single processor and multiprocessor applications [2]. A simple SDF graph can be seen in figure 1, this graph exists out of x tasks, also known as actors, that together from the application. An analysis that can be of interest is to see how the graph behaves on a different number of processors. Figure 2 illustrates how the number of processors affects the schedule. A more in depth explanation of SDF graphs can found in the next chapter, Synchronous Dataflow. SDF graphs can be used to obtain and analyse schedules with certain optimal properties, e.g. maximal throughput or minimal resource requirements. Current resource-allocation strategies for SDF graphs have shortcomings. Using the max-plus algebraic semantics leads to a bigger graph, in the worst case this graph is exponentially larger [3]. Another method assumes there are always enough resources to execute as soon as possible. This may not always be the case in real-life applications [4].

Therefore a novel method has been proposed; the usage of timed automata (TA) as an approach of modeling SDF graphs and analysing schedules [5]. By translating the SDF graphs to TA, the state-of-the-art model checker UPPAAL [6] can be used to derive optimal schedules. This new method offers many new benefits, e.g. maximal number of throughput or minimal number of processors required.

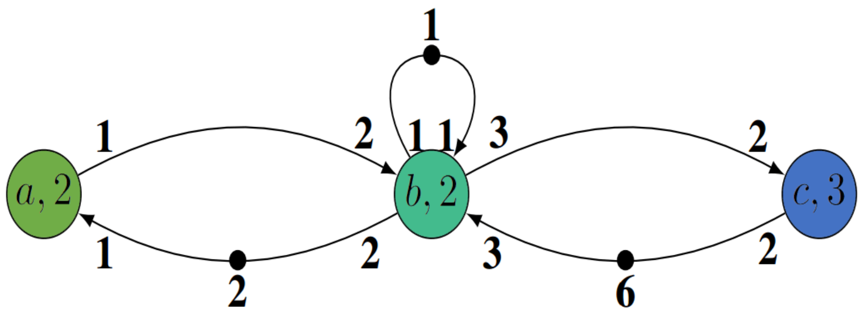


figure 1 *- SDF Graph, from [5].*

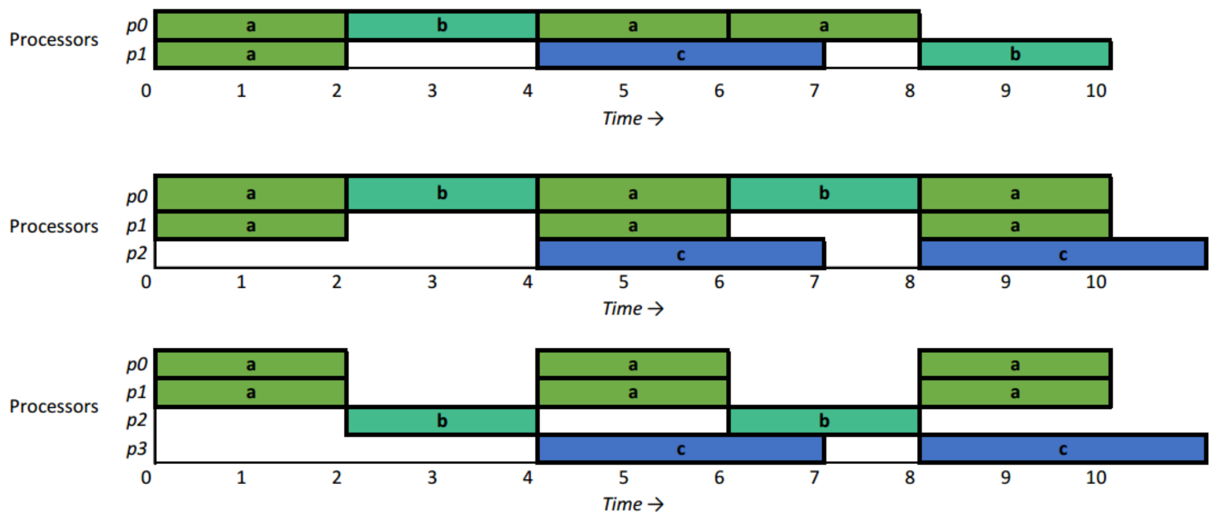


figure 2 *- SDF graph scheduled on two, three and four processors.*

1.2. Challenge

As SDF graphs get bigger and thus generated schedules become longer, it becomes harder to oversee the schedules. Especially the resulting schedule can be difficult to interpret, due to the fact that the outcome is a generated text file of states the model transits through. In the case of small SDF graphs these files are already close to a thousand lines. Currently the text file is converted to a csv file with the timestamps of task activation and deactivations, also known as actor firings. Hereafter, Excel is used to interpret the traces. This poses some problems, since, merged cells are not supported with the csv format. Thus cells have to be merged manually in situations where multiple processors are active and one task spans several others. Furthermore, Excel can only display values in cells, not on edges of these cells. Therefore it is hard to read the ending times of the tasks. Whilst the start and end time of a task are the primary interest.

The objective of this project is to ease this process. Visual representations of data aim to effectively exploit the ability of the human visual system to recognise spatial structure and patterns. Therefore, a well-designed visualisation can be of great help to quickly interpret the large generated schedules. This should reduce the time needed for the interpretation of the trace and give the user a better understanding of the result.

1.3. SDF Fish: A SDF Visualisations tool

In order to provide an quicker and better visualisations of the generated schedules the SDF Fish tool has been devised. Key features of this tool include:

- Ability to load SDF schedules from a text file
- Gantt chart style visualisation of SDF schedule
- Snapshot display of a moment within the schedule
- Zoom, filter, scale abilities for the different visualisations
- Play function for schedules to see changes occur over time
- Wide variety of time controls to navigate through the file

SDF Fish is freely available and builds for different platforms can be obtained via http://www.mudcrab.eu/SDF-Fish/. Furthermore, the program features a web version, this does not require the program to be downloaded to the local machine, whilst still offering all the features except for the ability to load files. The web version can be approached on the same website. The source code of the project is also available on this website under an open source license, if one wishes to expand upon the platform.

The main screen is build up out of six sections, namely; the *toolbar*, a *time control* panel and five *data views* (including the legend), this screen is shown in figure 3. Each view offers different interactivity and functionality to the user. The toolbar offers general option for the program itself such as what file is loaded in the program and settings where the user preferences are stored. The time controls allow users to navigate through the time variable of the data. The legend provides users with an overview of all elements the data is made up off. Furthermore, it provides users with the controls to disable or hide irrelevant elements, so the elements which are important can attract more attention. Lastly, there are the different data views, each view is designed to highlight a different aspect of the data.

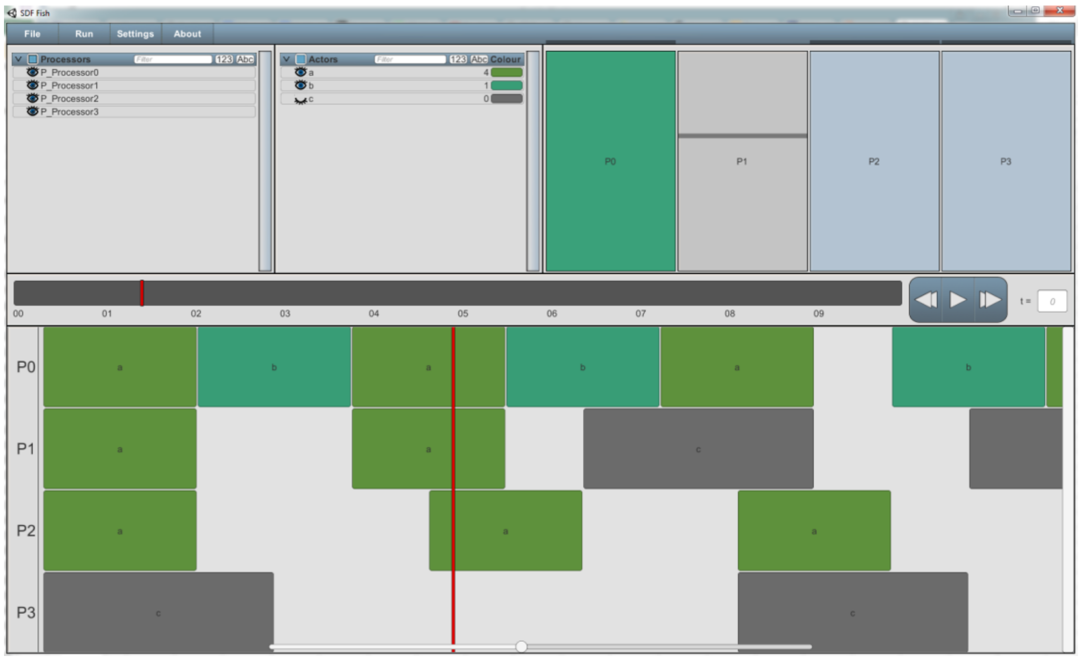


figure 3 *- A screenshot of SDF Fish with a schedule loaded into it.*

1.4. Research Questions

The objective of this project is to create a visualisation tool for SDF schedules. This raised the following main question:

- *How can we ease the understanding and interpretation of SDF schedules by visual means?*

In particular:

- *What information from these schedules are relevant for the user?*
- *What interaction with this data is useful for the user?*
- *What alternatives in visualisations are possible for these schedules?*

1.5. Report Outline

Firstly, Section 2 provides background information and an explanation on synchronous dataflow. Section 3 shows the visualisation tool and Section 4 presents a literature review on the construction of data visualisations. Section 5 and 6 discusses the ideation and specification of the visualisation tool respectively. The realisation phase of the project is explained in section 7. Section 8 discusses the evaluation of the visualisation tool. Finally, Section 9 draws conclusions and outlines possible future research.

2. Synchronous Dataflow

This chapter provides the reader with background information on SDF graphs that is required to understand the context of the project. Firstly, the basic concepts and terminology of SDF graphs are explained and illustrated by an analogy. Furthermore, the reader is introduced how SDF graphs are visualised. Following this the SDF scheduling methods and the visualisation of these schedules are discussed.

2.1. Synchronous Dataflow: Preliminaries

SDF graphs are a formalism used model data flow in streaming applications, generally multimedia applications. SDF graphs are part of the Graph Theory, this is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up out of nodes and connections between nodes, called *edges*. In SDF these edges also have an direction, which indicates the flow from one node to another. An example of a graph can be seen in figure 4. The direction is represented by an arrow, also SDF uses numbers on the end and start of edges to display production and consumptions ratios of the nodes that the edge connect. The nodes in these graphs represent tasks. An ordinary streaming application exists out of a set of tasks, also known as *actors*, these actors need to be executed in a certain order. The execution of such an actor is known as a firing. When an actor completes a firing it produces data on an edge, this is displayed by the base of an outgoing arrow. A unit of data is called a *token*. Furthermore, for an actor to fire it needs tokens on its ingoing edges. These numbers are displayed at the head of the arrow. Thus when an application start also initial tokens are needed to get actors started. The initial tokens on an edge are displayed by a number at the center of this edge. Since actors fire according to a set ratio two phases can be distinguished in a schedule. Firstly, there is an initial phase. After this an periodical phase is entered, in which the actors fire according to a fixed rate. The duration of a firing can vary per actor, this is generally displayed by a number in the node of an actor. An SDF graph is formally defined as a tuple $G = (A, D, \mathrm{Tok}_0, \tau)$ where:

- A is a finite set of actors,
- D is a finite set of dependency edges $D \subseteq A^2 \times \mathbb{N}^2$,
- $\mathrm{Tok}_0 : D \rightarrow \mathbb{N}$ denotes initial tokens in each edge and
- $\tau : A \rightarrow \mathbb{N}_{\geq 1}$ assigns an execution time to each actor.

Furthermore, there are various properties SDF graphs can possess. Two important properties are consistency and absence of deadlock [7], [8]. Consistency of a graph is determined by the consumption rate. A SDF graph that is not consistent requires unbounded memory to execute or deadlocks. An SDF graph deadlocks, when no actor is able to fire, which is either due to inconsistency of due to an insufficient number of tokens in a cycle of the graph. The concepts discussed so far are further elaborated by a factory analogy in section 2.1.1.

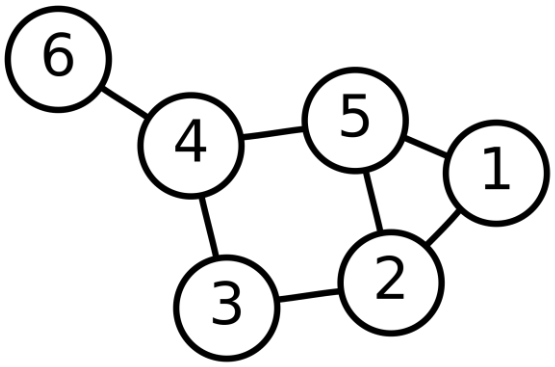


***figure 4** – A graph with six nodes and edges connecting the nodes.*

2.1.1. Skateboard Factory: an SDF Analogy

There is a factory that creates skateboards. This process of creating a skateboard is the application in this analogy. In total there are five tasks that need to be completed in order to build a skateboard. The wheels, trucks and deck need to be created. Furthermore, the wheels need to be attached to the trucks and the trucks have to be attached to the deck. All tasks needed to build a skateboard can be seen in figure 5. A task is executed by a worker and occupies this worked until the task is completed. For the simplicity of this analogy the time it takes to complete a task is the same for all five tasks.

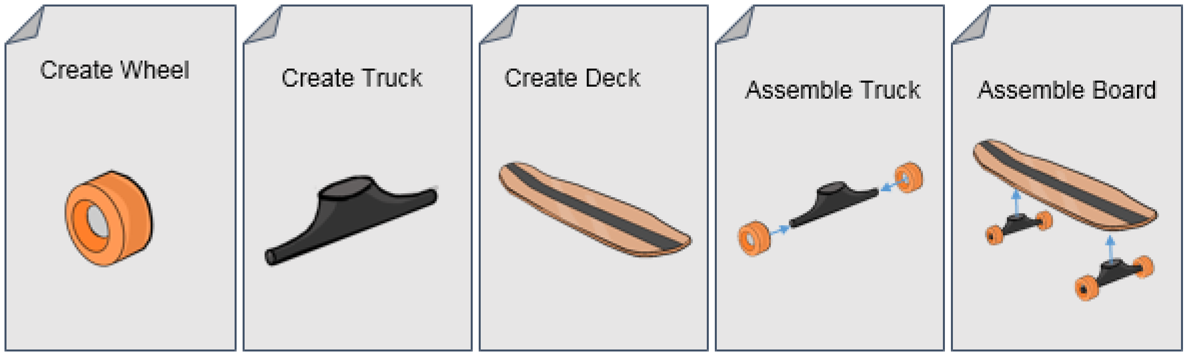


***figure 5** - Five tasks that are involved in building a skateboard.*

Additionally, an execution order between tasks is imposed on the system. In order to assemble the trucks you first need to have two wheels and one truck. Moreover, to assemble the board you need a deck and two assembled trucks. The steps involved in creating a skateboard are illustrated in figure 6. This production process can be translated to an SDF graph, the constructed SDF graph is shown in figure 7. The graph consists out of five nodes, one for every task. Furthermore, edges represent the flow of components to subsequent tasks. For example, in order to assemble a truck, one truck is needed and two wheels are needed. It is important to observe that every time a task is completed only one unit is produced, e.g. completing ‘Create Wheel‘ only produces one wheel.

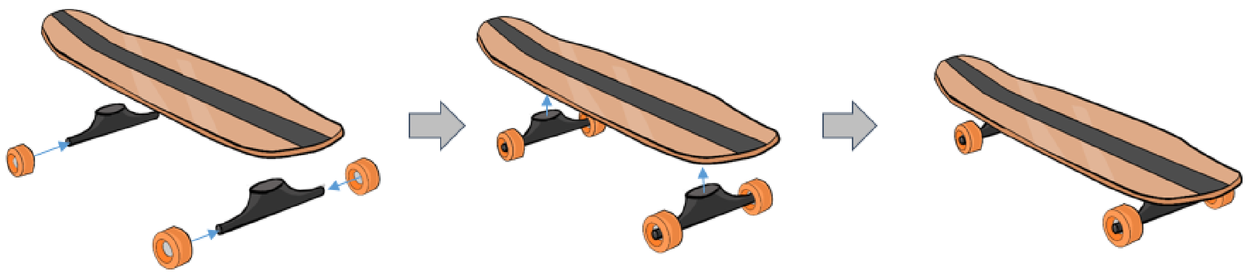


figure 6 *- Assembly order of skateboard production.*

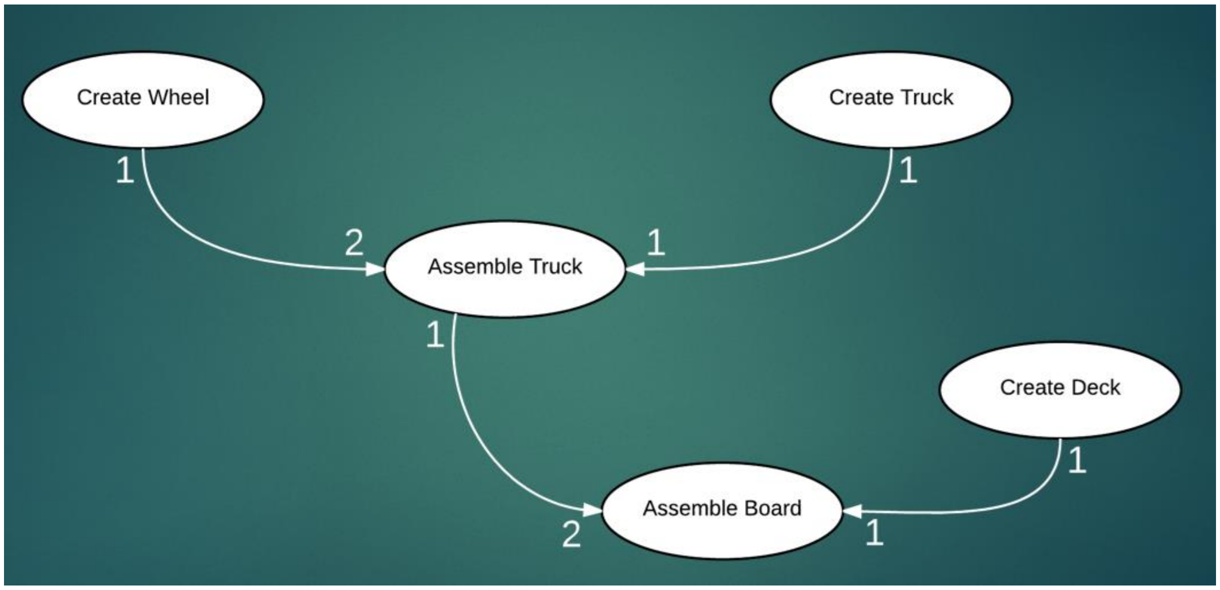


figure 7 *- SDF graph of skateboard creation.*

To make our factory run, a division of the work over the workers is needed. This can described in a schedule. One of the simplest scheduling solutions for the factory would be to hire five worked, one worker for each task. When the application starts, or in other works the factory starts producing. Every worker will complete their task as often as possible. Certainly, this will result in a factory that produces skateboards, particularly after six time units the first skateboard is produced. However also some undesired effects take place, specifically too many unassembled trucks and decks are created. For instance, for every wheel that is created also a truck is produced. But every truck needs two wheels to be assembled. So the longer the factory produces the bigger this overflow becomes. In other words the SDF graph is not consistent.

In order to solve this excess production of certain components constraints can be set to buffer sizes. A simple solution would be to request new units for every unit that is consumed. That is to say that for every truck that gets assembled a new truck and two wheels are ordered. Furthermore, workers can only produce units if these are requested. This solves the issue of having an excess of certain units, and ensures that only the minimum amount of every unit is kept in stock. Also this addition can be translated to SDF, this is shown in figure 8. As is shown in the figure for every truck that is assembled two wheels and one truck are consumed. Furthermore, two new wheels and one new truck are ordered.

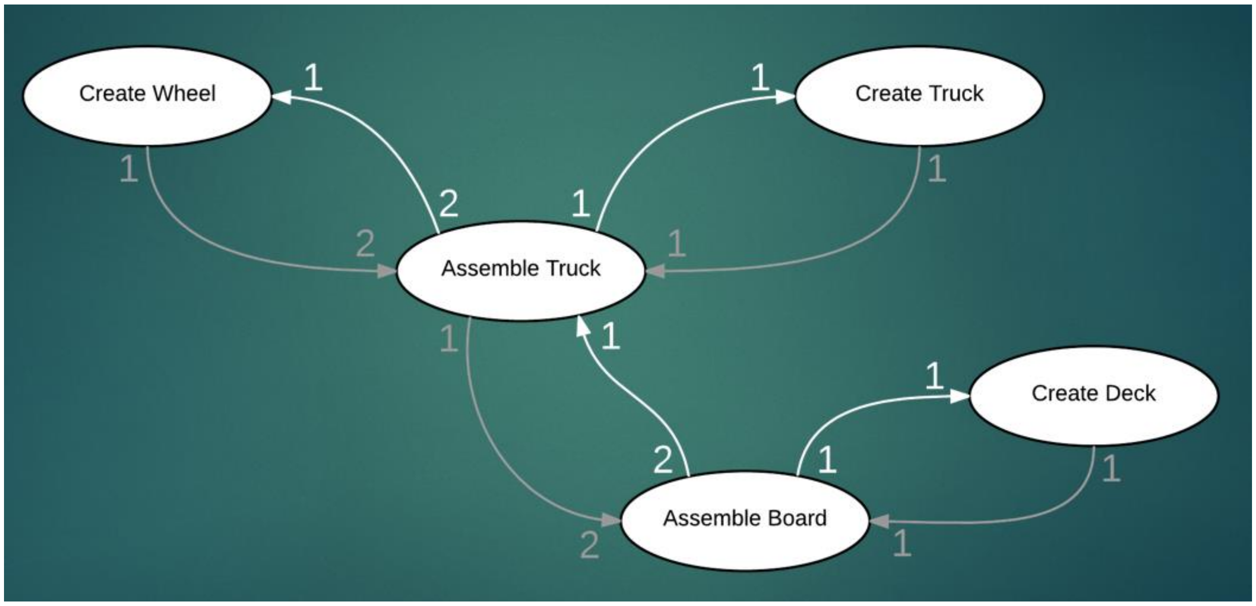


figure 8 *- SDF graph of factory without an overflow.*

Only one important aspects of this graph is missing, a way to get started. The truck assembly can only start if one token is present from create truck and two tokens from create wheel. However, in order to get these tokens, create truck and create wheel have to fire, but this cannot happen since they need to receive tokens from assemble truck. In other words, deadlock occurs in the graph. There are not enough tokens available in the graph to fire any tasks. In order to solve this specific deadlock occurrence, initial tokens can be introduced into the system. These are represented by points in the middle of an edge, accompanied with a number which represents the number of tokens the graph starts off with. In order to let the factory produce one skateboard at a time, an initial distribution of orders need to be setup for every component in the skateboard. This is added to the SDF graph in figure 9.

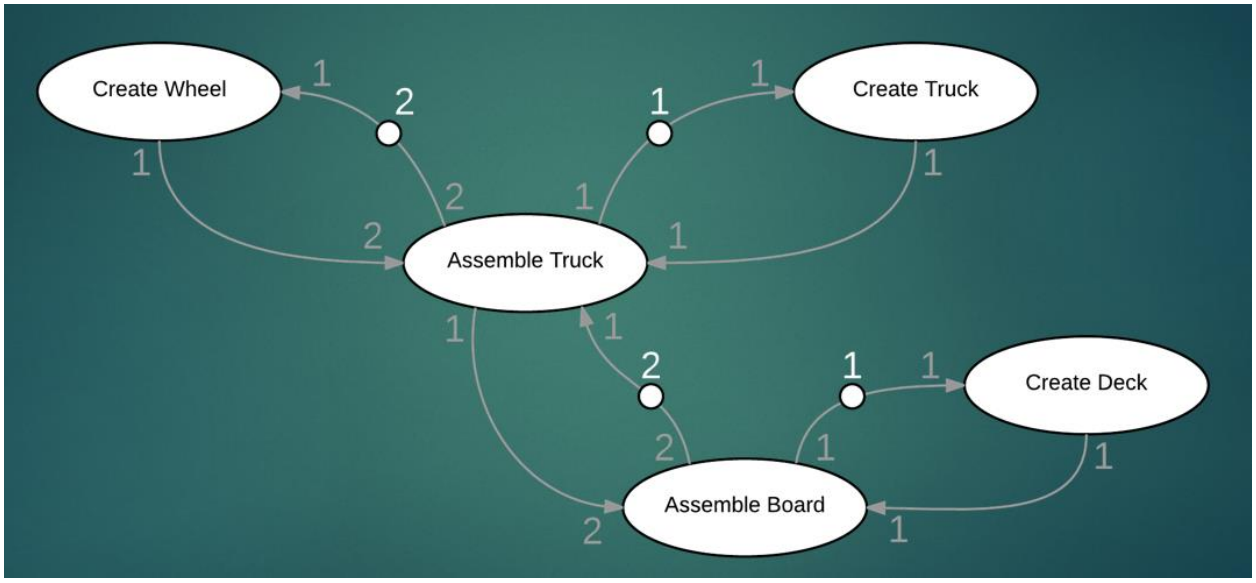


figure 9 *- SDF graph with initial token distribution included.*

In the example discussed thus far all the tasks have the same execution time for simplicity. However, it does not necessarily have to be the case. Generally the execution time is also displayed in the graph, this is represented by a number in the nodes. Figure 10 shows a version of the skateboard factory where all the creation tasks take one time unit, truck assembly two and board assembly four units.

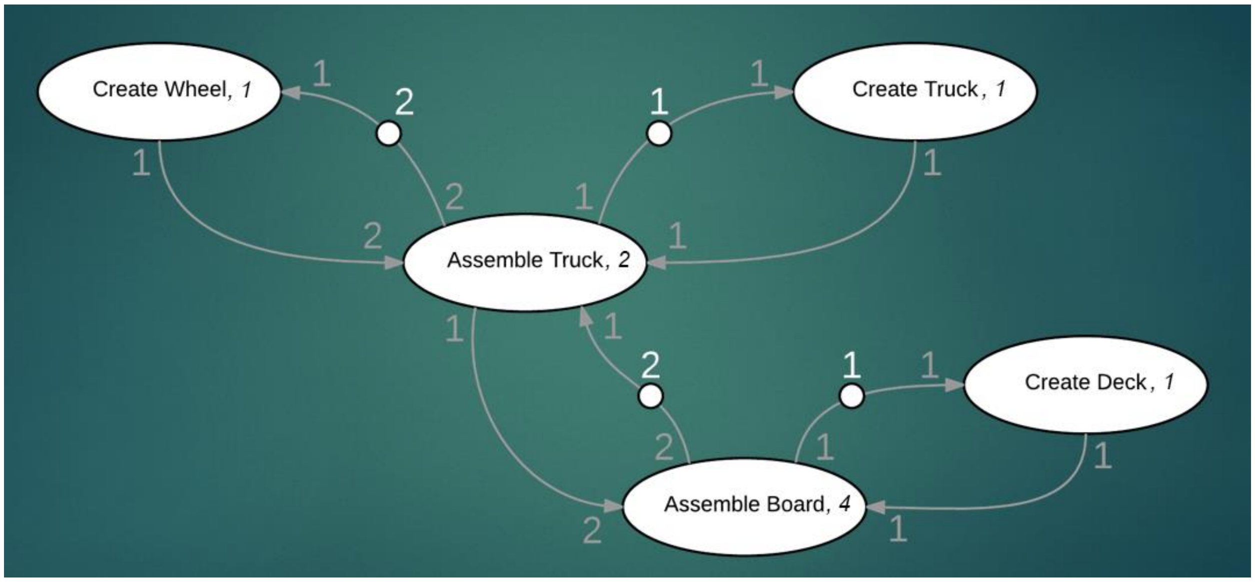


figure 10 *- SDF graph with actor durations*

2.2. Synchronous Dataflow: Scheduling

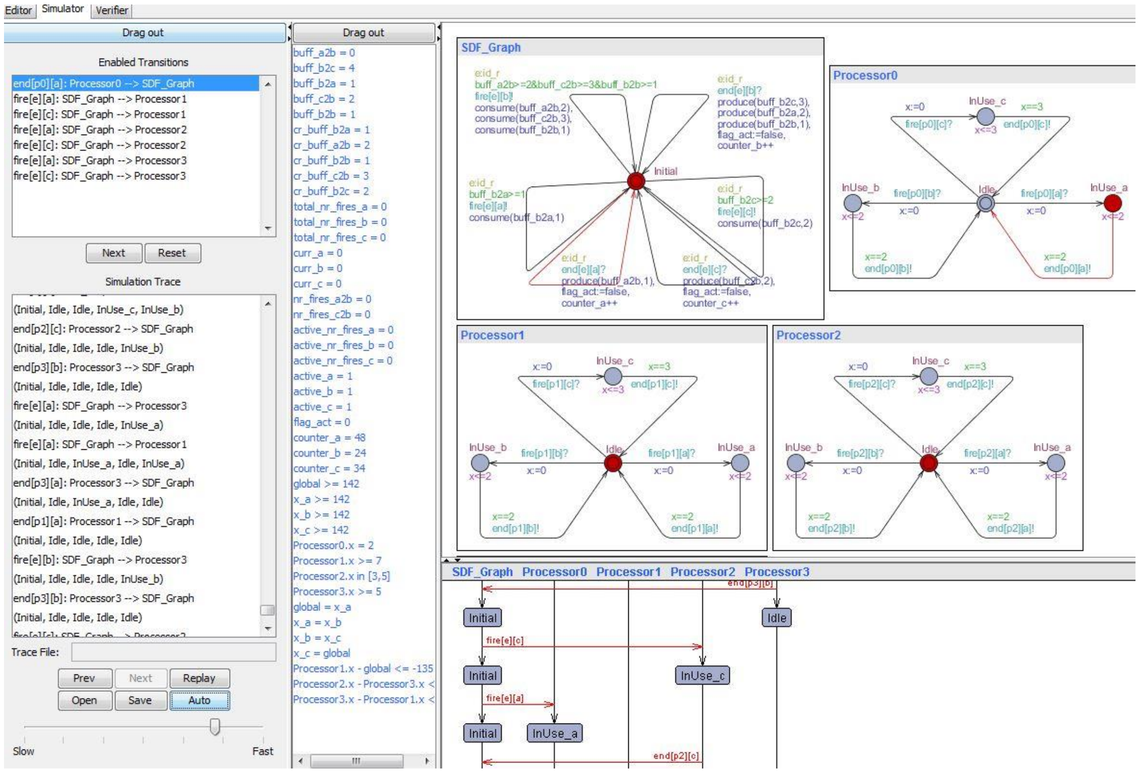
When an actor fires, it fires on a processor. A processor can be seen as a worker which can execute only one task at a time. When every actor is allowed to fire on any resource, the SDF graph is called homogenous. However, in some cases an actor is only allowed to fire on specific processors. This is called heterogeneous SDF graph. Evidently this limits the options to map the firing of actors on resources. The mapping of all the actors on workers over time forms a schedule. Thus a schedule depicts how actors fire over time, usually these schedules are generated with certain characteristics. For example a schedule could be made to execute as fast as possible, which can consume a lot of processors, or on the other hand as resource efficient as possible, which would require a lot less processors. There are various methods for the generation and visualisation of schedules. First, methods for the generation of schedules are discussed. The primary focus here is the method proposed by W. Ahmad [5], since this project revolves around this method. After this several visualisations types for schedules are discussed.

2.2.1. SDF Scheduling Methods

There are various scheduling methods available for SDF graphs, each with their own benefits and shortcomings. One scheduling method makes use of the max-plus algebraic semantics and transformation to homogeneous SDF graphs [9], [10]. This method leads to a bigger graph, in the worst case this graph is exponentially larger than the original [3].

Another method explores state-space of the graph until a periodic phase is found [4]. However, in this search it is assumed that an actor can alway directly fire. Thus it assumes there are always free resources available. However this may not always be the case in real-life situations, where there is always a constraint on the number of resources.

A novel method has been proposed by W. Ahmad et al.[5]; the usage of timed automata (TA) as an approach of modeling SDF graphs and analysing schedules. By translating the SDF graphs to TA, the state-of-the-art model checker UPPAAL [6] can be used to derive optimal schedules. These models are setup in UPPAAL by running a model for every processor and one model for the SDF graph. A screen of this method executed in UPPAAL is shown in figure 11. This new method offers many new benefits, e.g. maximal number of throughput or minimal number of processors required. However, the resulting trace can be rather difficult to understand, an example of a generated trace can seen in the simulation trace window of figure 11. Currently, this trace is converted to a csv file with the timestamps of actor firings. Hereafter, Excel is used to interpret the traces, in figure 12 such a visualisation is shown. This poses some problems, since, merged cells are not supported with the csv format. Thus cells have to be merged manually in situations where multiple processors are active and one task spans several others. Furthermore, Excel can only display values in cells, not on edges of these cells. Therefore it is hard to read the ending times of the tasks. Whilst the start and end time of a task are the primary interest.

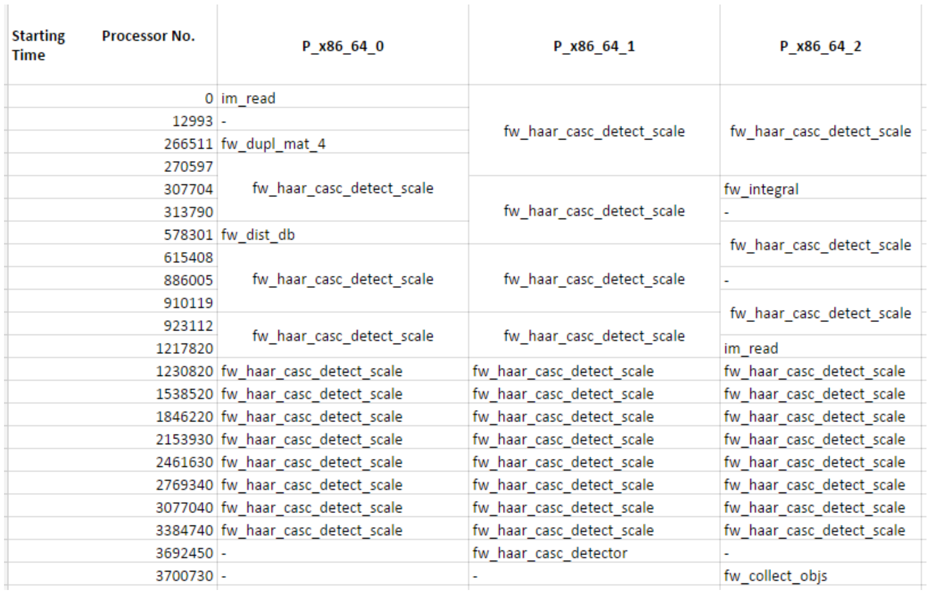


```
end[p0][a]: Processor0 --> SDF_Graph
fire[e][a]: SDF_Graph --> Processor1
fire[e][c]: SDF_Graph --> Processor1
fire[e][a]: SDF_Graph --> Processor2
fire[e][c]: SDF_Graph --> Processor2
fire[e][a]: SDF_Graph --> Processor3
fire[e][c]: SDF_Graph --> Processor3
```

```
(Initial, Idle, Idle, InUse_c, InUse_b)
end[p2][c]: Processor2 --> SDF_Graph
(Initial, Idle, Idle, Idle, InUse_b)
end[p3][b]: Processor3 --> SDF_Graph
(Initial, Idle, Idle, Idle, Idle)
fire[e][a]: SDF_Graph --> Processor3
(Initial, Idle, Idle, Idle, InUse_a)
fire[e][a]: SDF_Graph --> Processor1
(Initial, Idle, InUse_a, Idle, InUse_a)
end[p3][a]: Processor3 --> SDF_Graph
(Initial, Idle, InUse_a, Idle, Idle)
end[p1][a]: Processor1 --> SDF_Graph
(Initial, Idle, Idle, Idle, Idle)
fire[e][b]: SDF_Graph --> Processor3
(Initial, Idle, Idle, Idle, InUse_b)
end[p3][b]: Processor3 --> SDF_Graph
(Initial, Idle, Idle, Idle, Idle)
```

```
buff_a2b = 0
buff_b2c = 4
buff_b2a = 1
buff_c2b = 2
buff_b2b = 1
cr_buff_b2a = 1
cr_buff_a2b = 2
cr_buff_b2b = 1
cr_buff_c2b = 3
cr_buff_b2c = 2
total_nr_fires_a = 0
total_nr_fires_b = 0
total_nr_fires_c = 0
curr_a = 0
curr_b = 0
curr_c = 0
nr_fires_a2b = 0
nr_fires_c2b = 0
active_nr_fires_a = 0
active_nr_fires_b = 0
active_nr_fires_c = 0
active_a = 1
active_b = 1
active_c = 1
flag_act = 0
counter_a = 48
counter_b = 24
counter_c = 34
global >= 142
x_a >= 142
x_b >= 142
x_c >= 142
Processor0.x = 2
Processor1.x >= 7
Processor2.x in [3,5]
Processor3.x >= 5
global = x_a
x_a = x_b
x_b = x_c
x_c = global
Processor1.x - global <= -135
Processor2.x - Processor3.x <
Processor3.x - Processor1.x <
```

figure 11 *- Screen of schedule generation with the method proposed by W. Ahmed et al.[5]. Generated schedule can be seen on the left, models for SDF graph and processors on the right.*



Starting Time / Processor No.	P_x86_64_0	P_x86_64_1	P_x86_64_2
0	im_read	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
12993	-		
266511	fw_dupl_mat_4		
270597	fw_haar_casc_detect_scale		
307704		fw_haar_casc_detect_scale	fw_integral
313790			-
578301	fw_dist_db		fw_haar_casc_detect_scale
615408	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	
886005			-
910119			fw_haar_casc_detect_scale
923112	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	
1217820			im_read
1230820	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
1538520	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
1846220	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
2153930	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
2461630	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
2769340	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
3077040	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
3384740	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale	fw_haar_casc_detect_scale
3692450	-	fw_haar_casc_detector	-
3700730	-	-	fw_collect_objs

figure 12 *- Screen of Excel visualisation of an SDF schedule.*

2.2.2. SDF Schedule Visualisations

SDF schedules are commonly visualised in a Gantt chart-like form, an example is shown in figure 13. The Excel visualisation in figure 12 is also a variation of such a graph, with a notable difference being that the axis are inverted. In these graphs time is represented on the horizontal axis and either the computational machines or actors are mapped on the vertical axis. Bars in this graph represent the firing of an actor. These bars can contain text with the name of the actor or are assigned the colour of the actor that fires. This visualisation type thus makes primarily use of position for encoding. The horizontal position represents the start, duration and end time of the firing. Furthermore, the vertical position represents the processor the firing takes place on. Colour is optional, it can be used to encode the actor that is firing.

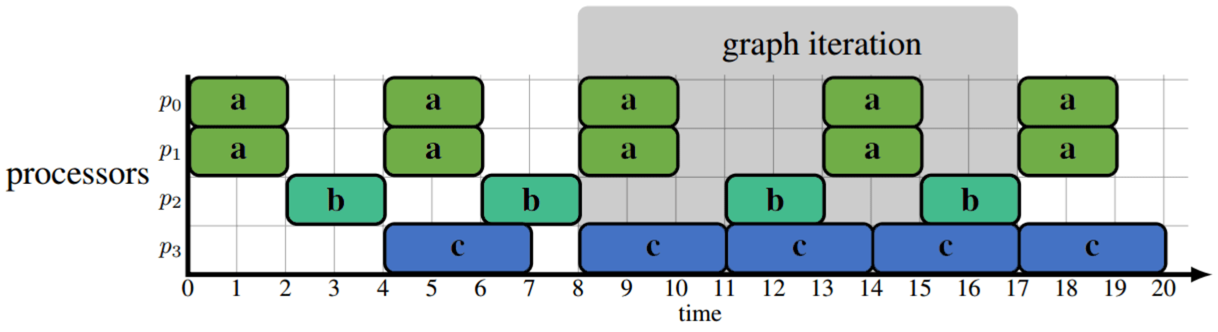


figure 13 *- A Gantt chart approach to visualising SDF Schedules.*

Another approach to visualising SDF schedules utilises the fact that SDF schedules enter an periodical phase, in which actors fire according to a fixed ratio. An example of this visualisation can be found in figure 14. The visualisation considers time to be discrete, this is justified since firings happen in steps. So if an step is equal to the smallest time step, no changes occur within each time step. Therefore time can be considered discrete. Every state (step) is visualised with a point. Points are connected by arrows, thus showing the flow of time, if any changes occurred in this transition they are listed above the arrow. After the initial phase, the period phase occurs, since this is periodical, the graph is looped.

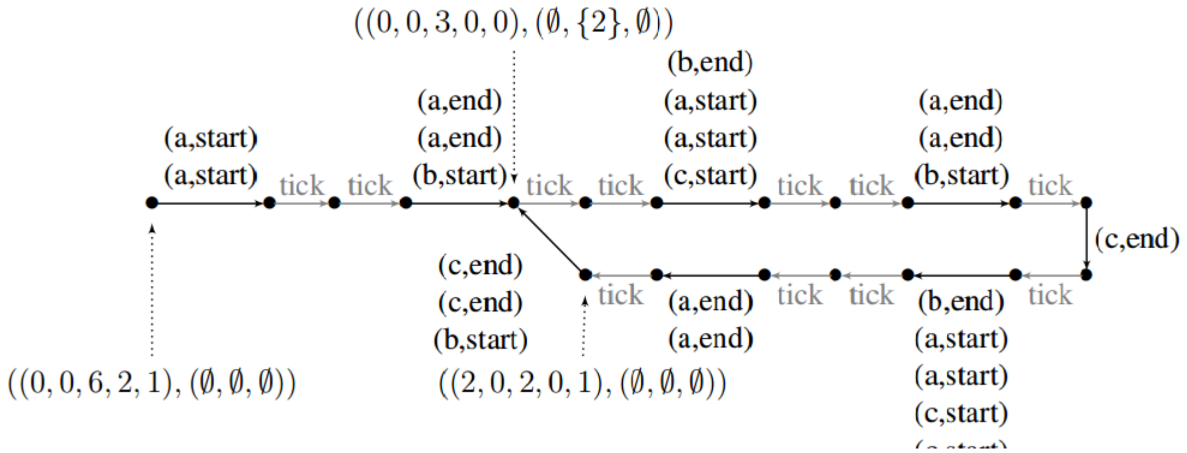


figure 14 - A node approach of visualising SDF schedules.

3. SDF Fish: Visualisation Tool for SDF Graphs

In this section the built product is presented to the reader, this presentation will take as a starting point the perspective of the user. The emphasis of this chapter is to exhibit the build tool rather than to provide an explanation as to why. A more in depth analysis of the completed product can be found in chapter 6, Realisation, hereby more attention is focussed on the design decisions. Following this related visualisations tools are discussed and compared. Finally, a conclusion is presented of this section.

3.1. Tool Overview

In order to provide an quicker and better visualisations of the generated schedules the SDF Fish tool has been devised. Key features of this tool include:

- Ability to load SDF schedules from a text file
- Gantt chart style visualisation of SDF schedule
- Snapshot display of a moment within the schedule
- Zoom, filter, scale abilities for the different visualisations
- Play function for schedules to see changes occur over time
- Wide variety of time controls to navigate through the file

SDF Fish is freely available and builds for different platforms can be obtained via http://www.mudcrab.eu/SDF-Fish/. Furthermore, the program features a web version, this does not require the program to be downloaded to the local machine, whilst still offering all the features except for the ability to load files. The web version can be approached on the same website. The source code of the project is also available on this website under an open source license, if one wishes to expand upon the platform.

Upon launching the program the visualisation interface is loaded. On launch the program does not hold any data yet, therefore all data views are empty at start. In order to get started a user first need to load a desired datafile into the program. This can be done by using the file>load file function from the toolbar in the top. A screen of SDF Fish with data loaded into it is shown in figure 15.

The main screen is build up out of six sections, namely; the *toolbar*, a *time control* panel and five *data views* (including the legend). Each view offers different interactivity and functionality to the user. The toolbar offers general option for the program itself such as what file is loaded in the program and settings where the user preferences are stored. The time controls allow users to navigate through the time variable of the data. The legend provides users with an overview of all elements the data is made up off. Furthermore, it provides users with the controls to disable or hide irrelevant elements, so the elements which are important can attract more attention. Lastly, there are the different data views, each view is designed to highlight a different aspect of the data.

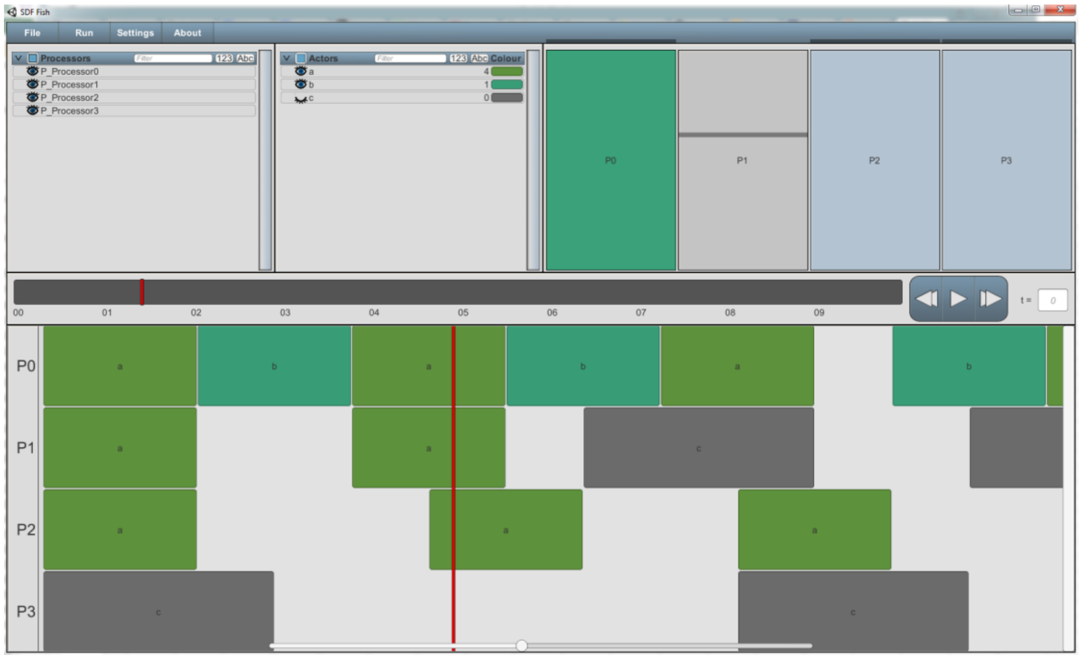


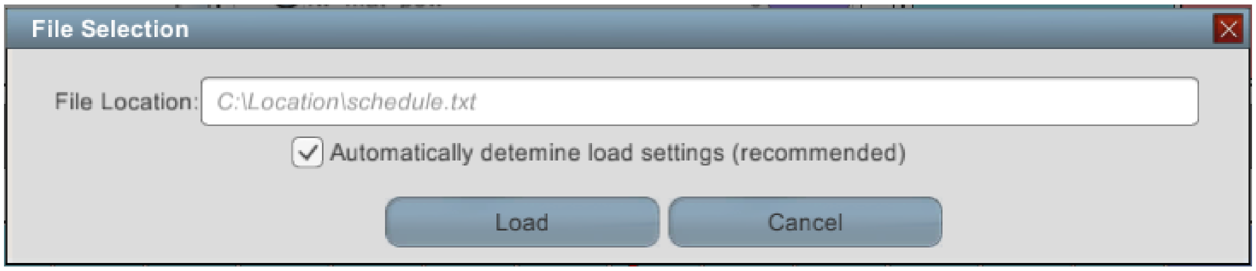
figure 15 *- A screenshot of SDF Fish with an schedule loaded into it.*

3.2. User Input

This section discusses the elements that form the input to the program. This includes the data that is loaded into the program. On top of that it also discusses the sections that have the main aim to provide controls to the user.

3.2.1. SDF Schedule

The most important user input into the tool is the data that is loaded into the program. Files can be loaded into the tool through the tool bar. By pressing the File option a dialogue is opened which requests users to select a data file, this dialogue can be seen in figure 16. An example of a data file is shown in figure 17. The Realisation chapter provides more insight on how data is extracted from these files. On top of this the user is provided with the option to automatically detect file specific parameters or manually fill these in. If chosen for manual selection a new dialogue is opened where users can adjust various parameters before the data is loaded into the displays. These parameters are the time it should take to play a file and the scale of actors in the Gantt chart.


File Selection

File Location: C:\Location\schedule.txt

✓ Automatically detemine load settings (recommended)

Load Cancel

figure 16 *- File selection dialogue.*

```
State:
( SDF_Graph.Initial P_p1_0.Idle P_p1_1.Idle )
global=0 P_p1_0.x=0 P_p1_1.x=0

Transitions:
  P_p1_0.Idle->P_p1_0.InUse_getmb { 1, fire[p_id][getmb]?, x := 0 }

State:
( SDF_Graph.Initial P_p1_0.InUse_getmb P_p1_1.Idle )
global=0 P_p1_0.x=0 P_p1_1.x=0

Delay: 1667
```

figure 17 *- Example of a data file that can be loaded into the program.*

3.2.2. Toolbar

The toolbar offers in total four buttons to the user; *File*, *Run*, *Settings* and *About*. The purpose of this bar is to provide general access to functions that do not belong to any of the other sections. The file controls allow users to load data files into the program. When users select the Run option from the toolbar a dropdown is opened where users can adjust the run speed of the program. Upon loading a file users are asked to set an initial run speed for the file or this is detected depending on the number of actors and the average length. Furthermore, Run offers a quick selection of speed modifiers, such as, twice the speed or half.

The settings options opens the user preferences screen. This screen allows users to change various settings that influence how the different data views behave. These settings are saved between sessions and therefore upon restart the program will load with the same settings. The location these settings are saved is dependent on the operating system that is used. Lastly, the about option will open a dialogue displaying general information about the program and the context in which it is created.

3.2.3. Time Controls

Central in the user interface are the time controls, these controls allow users to navigate through the loaded data file and thus allow users to view different moments in the data. The time controls can be seen in figure 18. This navigation can take place in multiple ways. Firstly, there is a big slider, which shows the user what point in time is currently observed. The start and the end of the slider represent the start and end of the data respectively. The red bar represents the current position, this bar can be dragged in order to change time. Or a position can be clicked in the bar and the time will be changed to this moment.

Besides this slider, users can also make use of the three buttons presented next to the time slider. These buttons allow the user to play the file automatically on the selected speed, playing the file from the current moment till the end. When pressed whilst playing the playing will be paused. The remaining two buttons allow users to jump forward or backward into time. The distance of this jump can be specified in the settings.

Lastly, users can also manually enter a time they would like to observe for precise observations. Users simply enter the desired moment in numbers in the field, and all views will be automatically updated. An additional feature of the entry field is the usage of percentages. Besides times, percentages can also be entered, and the program will jump to this point in time e.g. 0% will bring you back to the start, 50% to the middle.

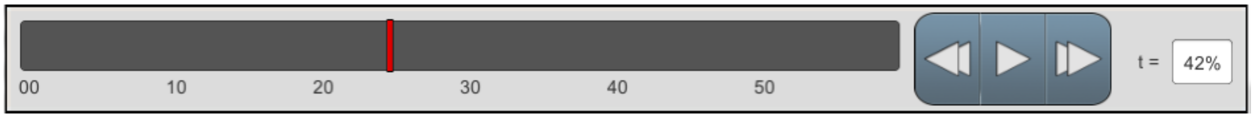
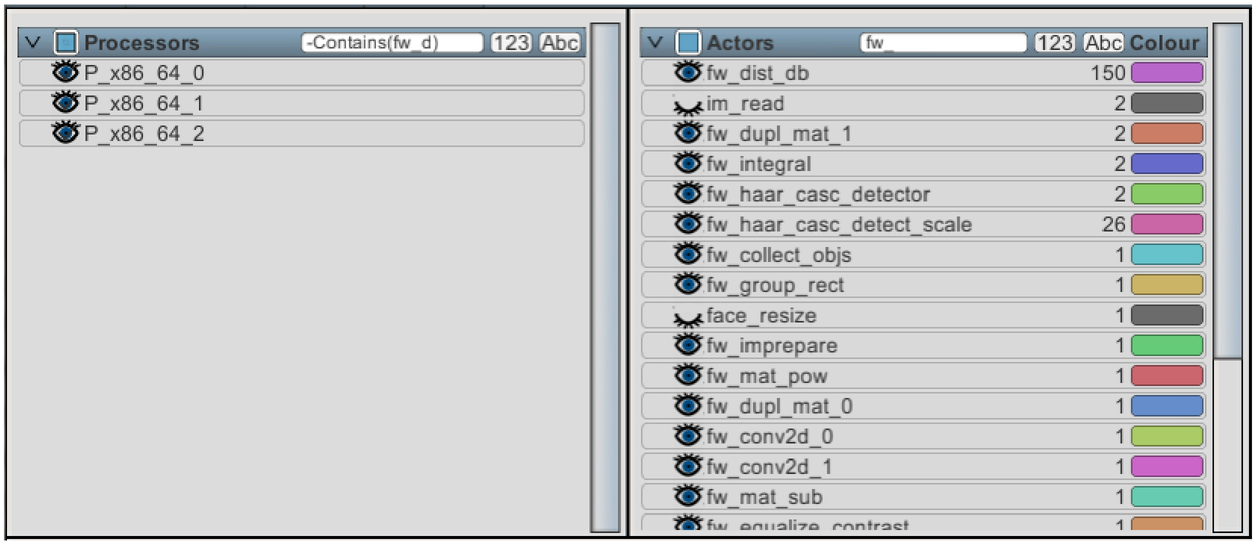


figure 18 *- Time controls in user interface.*

3.2.4. Legend

The legend displays all elements that together form the data, a close up of the legend can be seen in figure 19. It also shows users how this data is encoded in the different display views e.g. what colour a certain task is given. Moreover, this display does much more than solely communicating information to the users. The display also offers powerful control to users over the data. For example, by clicking on a name a dialogue is brought up which allows users to enter a new name for this task, by clicking on the colour a dialogue is brought up in which users can select a new colour for this tasks. Furthermore, users can enable and disable data elements. These elements become hidden from the users view or their colour is taken away thus attracting less attention from the users.



Processors	-Contains(fw_d)	123	Abc
P_x86_64_0			
P_x86_64_1			
P_x86_64_2			

Actors	fw_	123 Abc	Colour
fw_dist_db		150	🟪
im_read		2	⬛
fw_dupl_mat_1		2	🟧
fw_integral		2	🟦
fw_haar_casc_detector		2	🟩
fw_haar_casc_detect_scale		26	🟪
fw_collect_objs		1	🟦
fw_group_rect		1	🟨
face_resize		1	⬛
fw_imprepare		1	🟩
fw_mat_pow		1	🟥
fw_dupl_mat_0		1	🟦
fw_conv2d_0		1	🟩
fw_conv2d_1		1	🟪
fw_mat_sub		1	🟩
fw_equalize_contrast		1	🟧

figure 19 *- A close up of the legend with its functionality.*

As datasets become bigger, thus containing more elements, these element specific options might become less valuable. Therefore the header also contain some powerful tools. Firstly, one can mass enable or disable all elements under this header by a simple toggle. This is represented by the icon of an eye, which represent being able to see it versus not. Secondly, the list of elements can be ordered alphabetically or by execution. Last, perhaps most powerful, a search bar allows users to apply filters to the data. By entering text into these search fields filters the data real time hiding and disabling any element that contains the entered text. This can also be inverted by putting an '!' in front of the text. Finally, also commands can be used in the filter for processors.

Users can type '-Contains(<actor name here>)' and as a result all processors will be hidden that do not fire this task at any point in time. These functions are explained in more detail in example I and II.

EXAMPLE I

Suppose a schedule is loaded into the program with various tasks and processors. One of the tasks, task x, holds our particular interest. Therefore we filter the tasks on "x", thus taking away the colour from all tasks but x. Now if we slide through the schedule we can easily spot where tasks x occurs since it is the only thing with colour that attracts our attention. Allowing us to pause on those moments, and evaluate the situation.

EXAMPLE II

Now suppose we have the same situation as the previous example, example I, only this case the number of processors is a vast time larger. Applying the same method as before would work, however, one still needs to keep track of a lot of processor rows. Ideally processor rows that do not fire the task of our interest would be hidden. This can be achieved by the filter -Contains(x), it will hide all processors that do not fire x at any point.

3.3. Visualisation Displays

This section presents the visualisations that the tool generates dependent on the data that is loaded into the program. One of the visualisations is in the form of a Gantt chart, the other display focusses more on a snapshot view of the data.

3.3.1. Gantt Display

This display evolved around the commonly used and employed Gantt like charts in the field of SDF graphs and their visualisations. The tool offers this view from the perspectives with resources on the vertical axis and time on the horizontal axis. This visualisation can be seen in figure 20. Blocks inside this graph represent when the element is active in time. The redline in the middle of this view represents the current observed moment used in other views.

The view allows user interaction in various ways. Firstly, users can disable elements on the vertical axis by clicking on the label. Also the blocks themselves can be disabled, simply by clicking on them. Furthermore, the scale of the whole graph can be changed with a slider, this provides users with a zoom functionality. The data displayed in this view is kept up to date with any changes that happen in other views. And in the case the number of vertical elements do not fit in the container anymore a vertical scroll is automatically added to the display.

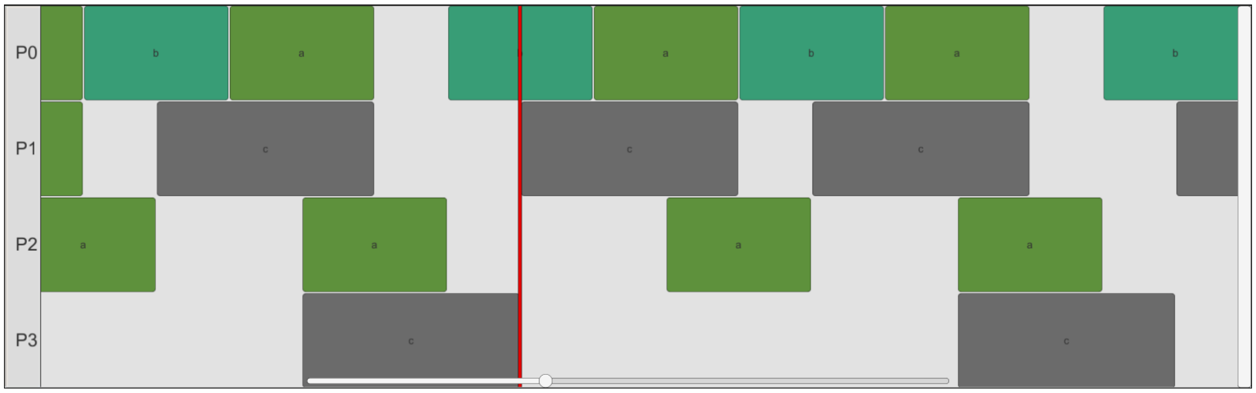


figure 20 *- Gantt like chart generated by the tool with processor cores on the vertical axis and time on the horizontal axis.*

3.3.2. Processor Display

The processor view provides the user with a moment observation. This display creates a bar for every processor, the colour of the bar represents the task the processor in question is currently occupied with. Inside each processor block a small indicator is placed, this indicator moves according to the ratio of time occupied by tasks versus idle time of the processor thus far. A indicator at the bottom of the bar indicates that the processor has not run any tasks at all, the higher the processor gets the more time is spent executing tasks in relation to idle time. This indicator moves real time as the user run the schedule or changes time. This display with its functionality can be seen in figure 21.

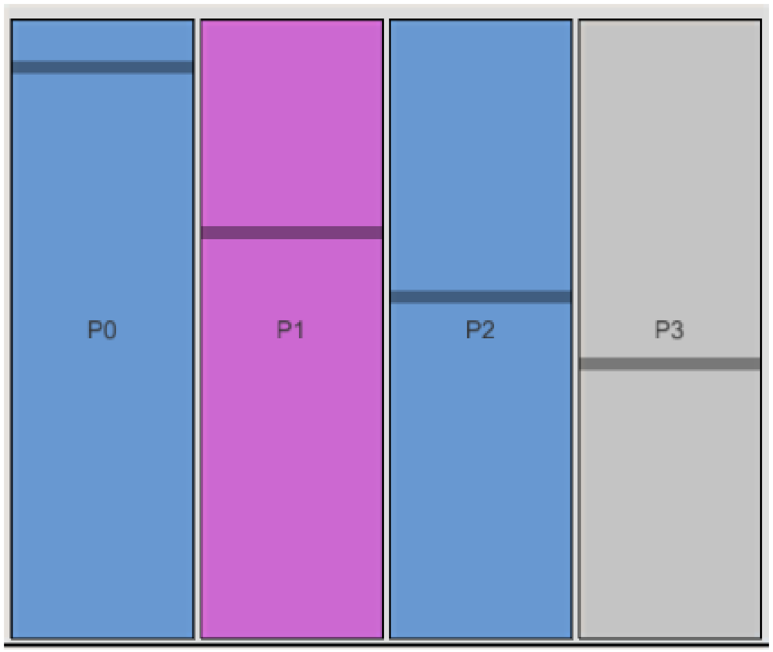


figure 21 *- Processor view and its functionality.*

3.4. Related Work

In this section various SDF visualisation tools will discussed. A total of three tools are assessed, TRACE, VET and SDF3. For each tool the input, output and method will be discussed. After this discussion a comparison follows between these tools.

3.4.1. TRACE

TRACE [11] is a visualisation tool made for Gantt-like charts developed by TNO. The tool is developed for the visualisation of activities restrained by resources and dependencies as a function of time. TRACE can be downloaded as a standandalone for every operating system or as a plugin for Eclispe from the website http://trace.esi.nl/. It requires Java to run. The tool is subject to a license and does not allow for any modifications. Users can select an .etf (enriched text file) to load data in the program. This file contains a timestamped schedule about resources, activities and dependencies. The program offers various visualisation options divided into two categories; Gantt chart and Design space visualisations. For SDF graphs the first category holds the most interest. The Gantt chart display has much resemblance to the SDF Gantt type of visualisations discussed in section 2.2.2. The program also visualises dependencies in this graph and allows users to do a performance analysis such as critical path, latency and throughput based on this graph. A screen of the TRACE program visualising a Gantt chart is shown in figure 22. To support design space visualisations the program provides six statistical graphs; radar graphs, 3D scatter plot graphs, 2D scatter plot graphs, 3D heat graphs and parallel coordinate graphs. TRACE provides various analysis options for users. Based on the Gantt chart advanced property checking can be done, such as critical path, latency and throughput. If a property is violated the tool will notify the user where this violation takes place.

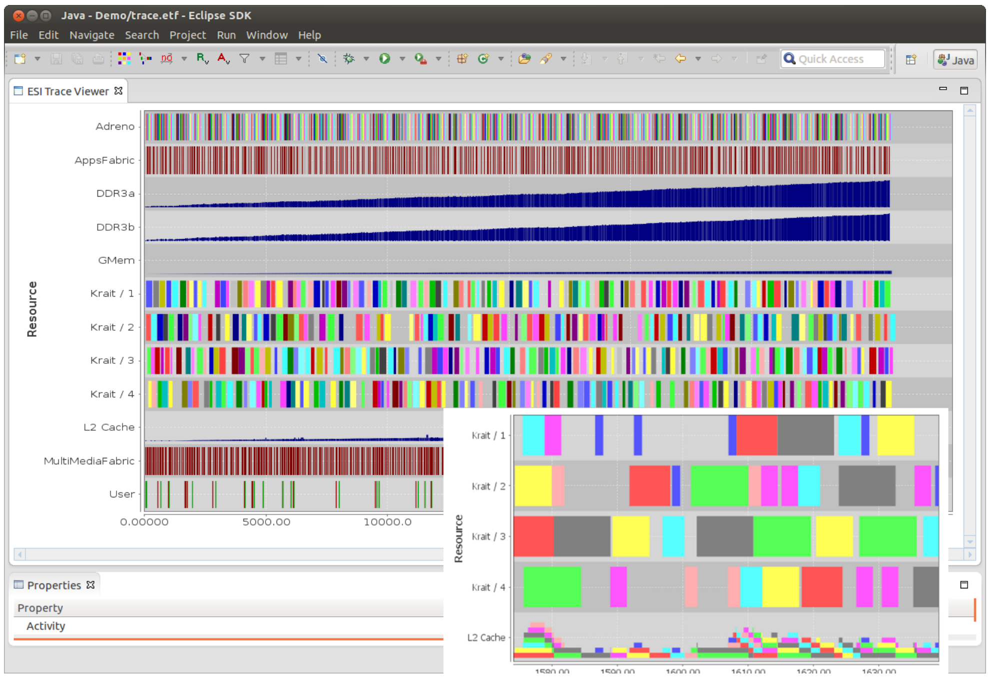
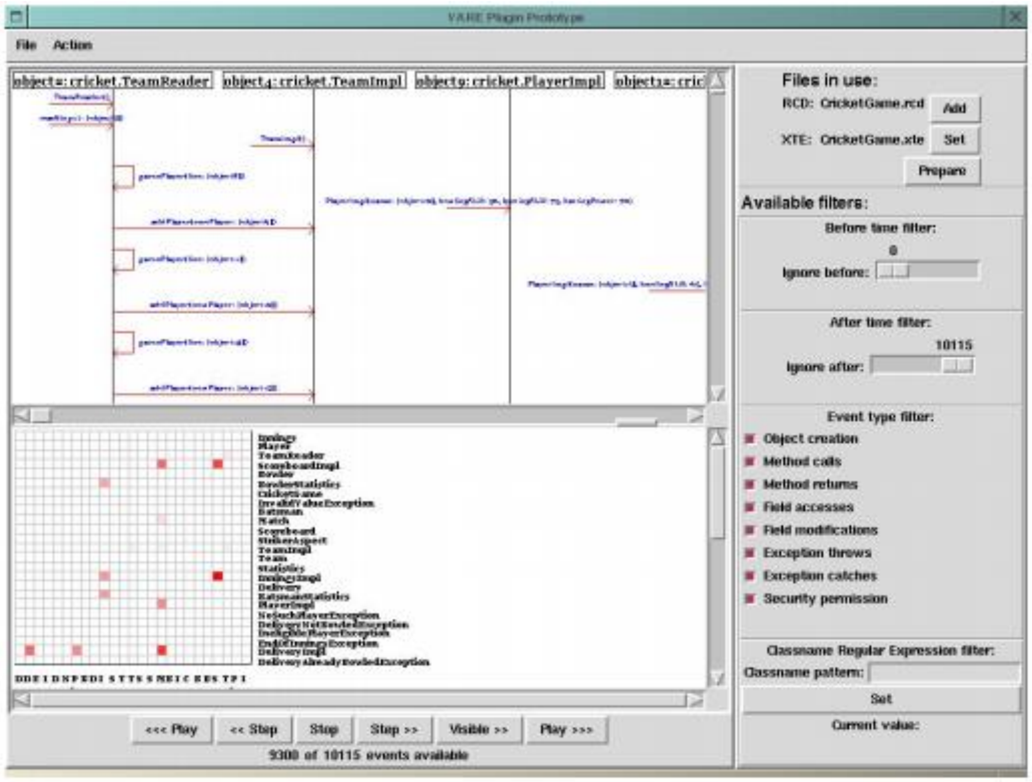


figure 22 *- TRACE visualisation tool developed by TNO.*

3.4.2. VET

Visualisations of Execution Traces (VET) is an execution trace visualisation tool developed by researches from Victoria University of Wellington that helps programmers manage the complexity of execution traces obtained from object-orientated programming languages, such as Java and C# [12]. The tool is based on the design heuristics defined by Card, Mackinlay & Shneiderman[13]. These heuristics aim to show the user everything up front, let the user filter out unwanted information and lastly, show details about data-points on demand. Expert users can also write plugins to add new visualisations and write plugins. Execution traces can be loaded into the program in an XML format, these files can be up to 100mb in size. The program uses the execution data to create a sequence diagram and call graph. A screen of VET displaying both visualisations is shown in in figure 23.



Files in use:
RCD: CricketGame.rcd
XTE: CricketGame.xte

Available filters:

Before time filter:
0
Ignore before:

After time filter:
10115
Ignore after:

Event type filter:
- Object creation
- Method calls
- Method returns
- Field accesses
- Field modifications
- Exception throws
- Exception catches
- Security permission

Classname Regular Expression filter:
Classname pattern:
Current value:

9300 of 10115 events available

figure 23 - *VET visualisation tool developed by McGavin et al. [12].*

3.4.3. SDF³

SDF For Free (SDF^3) is a tool that can generate SDF graphs, with support to analyse and visualise the generated graphs [14] . The SDF graphs can be generated with random properties, e.g. random number of actors or random connections between a given number of actors. The tool is developed at the technical University of Eindhoven. The tool is written in C++ and the source code is freely available. The SDF graphs generated by the algorithm in the program are guaranteed to be connected, consistent, and deadlock-free. Users can specify parameters determining the characteristics of the graph in a configuration file. Besides the generation of random graphs, SDF^3 also offers a library that provides SDF analysis and transformation algorithms. The tool also provides the option to export the generated SDF graph as an XML file. Furthermore, the generated graph can be visualised through the use of the visualisation tool dotty [15], such a visualisation of a generated SDF graph can be seen in figure 24.

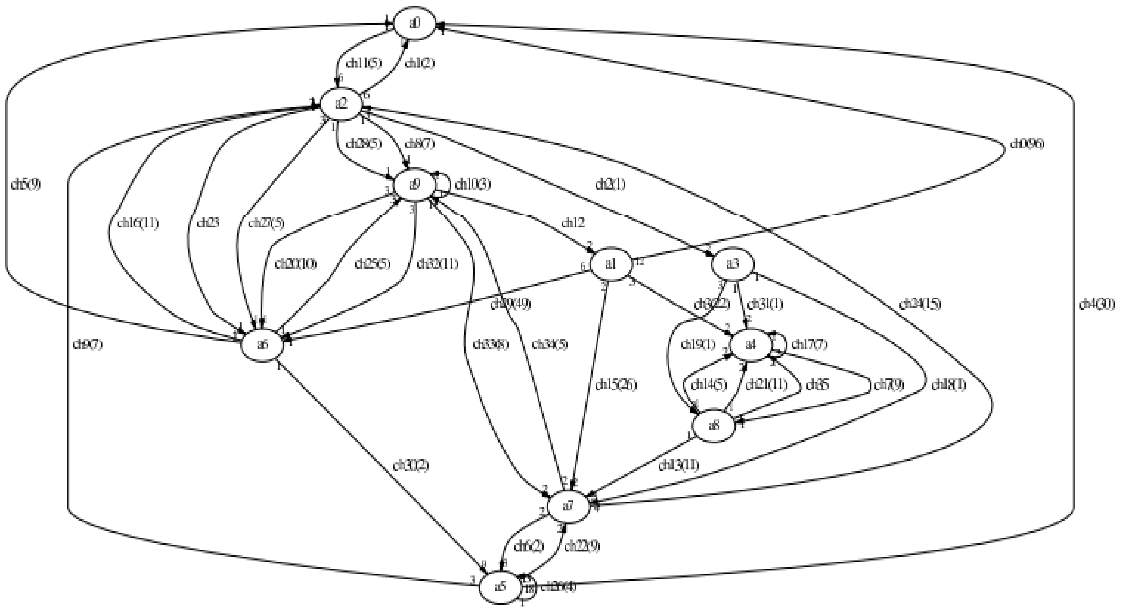


figure 24 *- An SDF graph generated and visualised with SDF^3.*

3.4.4. Comparison of Related Work

The tools discussed so far are all considered related work. Even though the tools themselves and their visualisation are inherently different. Therefore a summary of the comparison between these tools can be found in Table 1. The input, visualisations and output are discussed for every tool that has been discussed.

TOOL	INPUT	VISUALISATIONS	OUTPUT	MODIFIABLE
TRACE	Enriched text files of timestamped schedule	Gantt chart radar graphs 3D scatter plot 2D scatter plot 3D heat graphs parallel coordinate graphs	None	No
VET	XML file of execution trace	Sequence diagram call graph	None	Yes
SDF [3]	Configuration file with parameters	SDF graph (through dotty)	XML file with SDF graph	Yes

Table 1 *- A summary of the characteristics of the discussed tools; TRACE, VET and SDF[3].*

3.5. Conclusion of Chapter 3

In this chapter the visualisation tool has been presented from the perspective of the user. Furthermore, related work has been discussed. SDF Fish allows users to load and visualise SDF schedules. Also zoom, scale and filtering options are offered to the users. The developed tool for this project provides these visualisations in the form of a Gantt-like chart and a snapshot view of the resource usage. This does not overlap with SDF[3] or VET, since these tools only offer different type of visualisations and features. TRACE does also offer a Gantt-like visualisation, additionally it offers advanced property checking. SDF Fish on the hand is more focussed on SDF graphs and user convenience, whereas TRACE attends to any data that has activities on resources as a function of time.

4. Background on Data Visualisations

This chapter presents a literature review that aims to establish ways to construct accurate data visualisations. Literature is searched for guidelines that one should take into account when constructing a data visualisation. Furthermore, the building blocks that form a data visualisation are discussed. Lastly, the use of elements that do not convey data are discussed.

4.1. Literature Review

Synchronous dataflow (SDF) graphs are widely used models for analysing data flow in streaming applications, for both single processors and multiprocessor applications [5]. Examples of practical applications are audio decoders. Current resource-allocation strategies and scheduling of tasks for SDF graphs have shortcomings and may not always be realistic towards real world applications. Using the max-plus algebraic semantics and transformation of SDF graphs to homogeneous SDF graphs leads to a larger graph, in the worst case this graph is exponentially bigger [9], [10]. Another method explores state-space but assumes that there is always a resource available to fire the task on. This may not hold up for real-life applications, where there is always a constraint on the number of resources [4].

Therefore a novel method has been proposed: the usage of timed automata (TA) as an approach of modeling SDF graphs and analysing schedules [5]. By translating the SDF graphs to TA, the state-of-the-art model checker UPPAAL [6] can be used to derive optimal schedules. This novel method solves issues of current resource allocation methods.

However, the generated traces are very difficult and time consuming to interpret. These traces can be thousands of lines long of every state the schedule transitions through. An overview of these states is difficult to obtain, since one would need to remember every state. Visualisations can make the interpretation of data easier and faster. Since visual representations of data aim to exploit effectively the ability of the human visual system to recognise spatial structure and patterns [16]. In order to build such a tool one needs to understand how data visualisations work. Visualisations are constructed by visualisation experts. These experts often rely on their experience and perception to create clear and visually pleasing data visualisations, not following a set recipe for every visualisation. This review aims to bring some structure to this unorganized process by means of assessing literature. Perhaps a set of building blocks or guidelines can be derived from empirical research, these can help a designer in creating visualisations that display the data as accurately as possible. This raises the main question:

- *What guidelines should be taken into account for such a visualisation tool?*

In particular:

- *What building blocks are there to build a visualisation?*
- *What effect do elements have that do not convey the data?*

4.1.1. Method of Literature Review

This section provides an overview on how literature was obtained. First, the different search engines that were used are presented, after this the search terms.

4.1.1.1. Search Engines

To form an answer to the research questions proposed in the introduction relevant literature has to be found. To find as much material as possible searches have been conducted in both Google Scholar and Scopus. Google Scholar orders search results based on how often an article is cited, whereas Scopus orders on relevancy and publicity date. Therefore the first search round was conducted in Google Scholar, this would provide an introduction to the field of research. Since key articles in this field are more likely to end up high in the search results. After this round the same search terms were used to find articles in Scopus.

4.1.1.2. Search Terms

The literature search was started with a general search terms as 'visual communication', 'visual encoding' and 'chart junk'. These terms are related to field of research and the research questions. Secondly, also a search has been conducted by publisher. The initial list of articles revealed some leading researches in this field. Furthermore, the related work sections and references of the acquired papers lead to additional papers. A full overview of search terms used in both search engines can be found in Table 2. For all search terms where applicable, both the UK as US spelling have been used.

● Data Visualisation	● Chart Junk	● Visual encoding
● Information Visualisation	● Visual Communication	● Visualisation Pipeline
● Information Design	● Infovis	● Edward Tufte
● Yuri Engelhardt	● Jeffrey Heer	

Table 2 *- List of search terms used to collect literature.*

4.1.2. Results of Literature Review

Elements that form visualisations can be categorised into two groups; *data ink*, all elements that encode data and *non-data ink*, elements that do not directly encode data. First an analysis takes place on the data ink, literature is assed to try and establish the most basic unit for data visualisations. In other words; trying to establish the most basic building block that creates the data ink. On the other hand, visualisation generally also contain non-data ink. Opinions vary widely on the utility on non-data ink, therefore literature on this topic will be discussed. Lastly, good practice guidelines found in literature are discussed.

4.1.2.1. Building Blocks

Data visualisations consist of building blocks, each building block can vary in numerous visual variables e.g. colour, shape, pattern or size, to communicate data [17]. Since these visual variables are inherently different, they are perfect to communicate data simultaneously, but also because of this they differ in accuracy of interpretation for the user, example III elaborates on this.

One of the first papers to establish this difference in an empirical way is the classical paper written by Cleveland et al. [17]. Cleveland et al. [17] compiled a ranking of elementary tasks by accuracy of interpretation e.g. position, length, angle and colour, this list can be seen in figure 25. Elementary tasks are described as elementary graphical encodings that people use to extract quantitative information from graphs. However, they state that the list is not exhaustive and could be expanded.

1. Position along a common scale
2. Positions along nonaligned scales
3. Length, direction, angle
4. Area
5. Volume, curvature
6. Shading, colour saturation

figure 25 *- Ranking of visual variables by Cleveland et al. [17].*

EXAMPLE III

A simple example of a visualisation is a bar chart. Each bar represents a building block, since it represent an aspect of the data. It uses position along a common scale (some could argue area) and colour or shading as visual variables. The position along common scale lets viewers compare the different values and the shading of each bar encodes the category the bar belongs to.

Although over time the list has been expanded, the ranking itself has not changed much, but more rankings were introduced. One of such papers that expands upon the groundwork laid by Cleveland et al. [17] is Mackinlay [18]. First, elementary tasks are categorised into quantitative, ordinal and nominal variables. Each of these variables has its own ranking since each category conveys a different type of data. The ranking which is established by Mackinlay can be seen in figure 26.

Quantitative	Ordinal	Nominal
Position	Position	Position
Length	Density	Colour Hue
Angle	Colour Saturation	Texture
Slope	Colour Hue	Connection
Area	Texture	Containment
Volume	Connection	Density
Density	Containment	Colour Saturation
Colour Saturation	Length	Shape
Colour Hue	Angle	Length
Texture	Slope	Angle
Connection	Area	Slope
Containment	Volume	Area
Shape	Shape	Volume

figure 26 *- Ranking of visual variables by Mackinlay [18]. The tasks shown in the grey boxes are not relevant to these types of data.*

Another approach suggested by Bertin [19] states that the most basic unit in visualisations is a mark, a mark is defined as something that is visible and can be used in cartography to show relationships within a sets of data. Three types of marks are brought forwards; points, lines and areas. The different ways these marks can vary with are defined as visual variables. This results in visual variables such as placements, size, shape, value, orientation and texture, in Table 3 a visual explanation of these variables can be seen. Every visual variable has five characteristics: selective, associative, quantitative, order and length [19]. A visual variable is selective if by changing this variable a mark becomes easier to select in relation to other marks, associative if the variable allows marks to be perceived as a group. A variable is said to be quantitative if the relationship between two marks can be seen as numerical. Order, when marks differentiating in this value can be seen as more or less compared to each other. Lastly, length is the number of changes in this variable that can be used while still maintaining an accurate distinction between the values. In Table 4 these characteristics are visually shown for position.

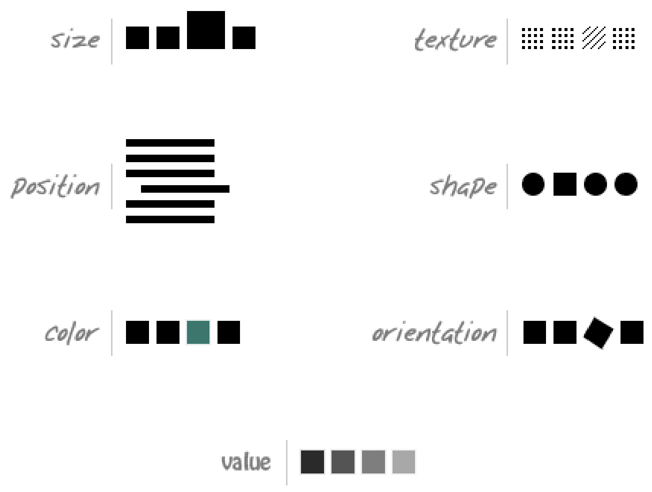


Table 3 *- Visual variables according to Bertin [19].*

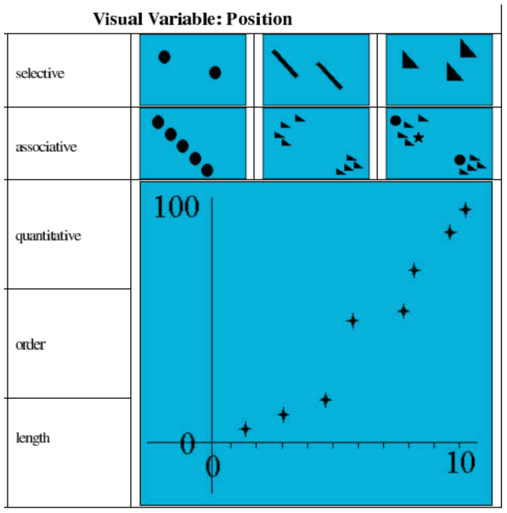

Visual Variable: Position
selective
associative
quantitative
order
length
100
0
0
10

Table 4 *- The five visual characteristics for position.*

Carpendale [20] builds further upon this idea laid down by Bertin [19]. Due to the introduction of the computational display, the set of marks is expanded with surfaces and volumes. These can be seen as expansions on the existing marks with the only difference that they now exist in 3D space. Also motion is suggested as a powerful visual variable.

4.1.2.2. Non-data ink

Data visualisations consist out of more than just building blocks. The elements that do not directly convey data are added to visualisations to create more memorable, appealing or understandable data visualisations. These elements that are not necessary to comprehend the data or elements and can distract the viewer from the data is referred to as chart junk [21], figure 27 shows two versions of a graph, one without and one with chart junk. Some are of the opinion that these elements should be minimised, whereas others see it as a powerful tool to create data visualisations. Moacdieh [22] states that many well-cited theories or guidelines for data visualisation advocate 'minimalism', the absence of all chart junk. However, many designers include a wide variety of visual embellishments in their charts, such as small drawings, large images, and visual backgrounds. The widespread use of embellished designs, such as advocated by Nigel Holmes, raises the question about whether the minimalist position on chart design is really the better approach.

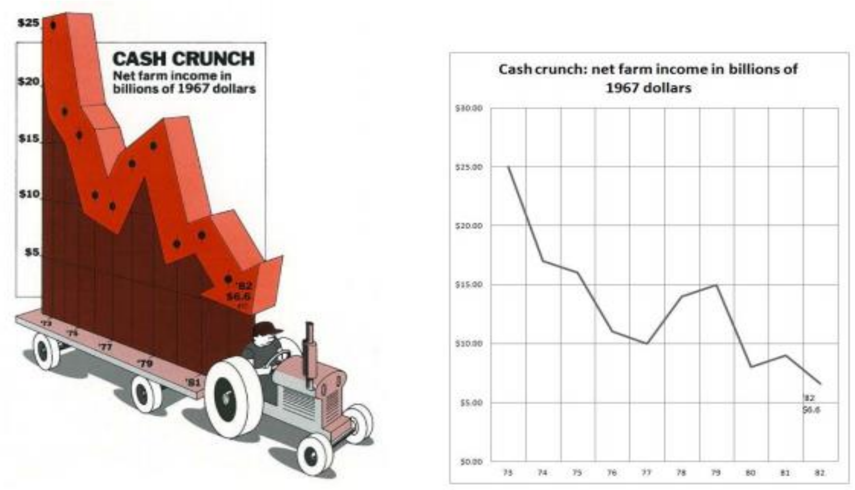


figure 27 *- Example of the same graph, a graph with chart junk version to the left and the plain version on the right.*

There are various empirical studies that put the minimalist approach into question. One of these studies showed that non-minimalist style charts improved short term recall, result in shorter time needed to review the chart while answering questions under 10 and 15 seconds and participants found that non-minimalist style charts are more attractive and memorable [22]. Borkin et al. [23] conducted a study and found out that visualisations containing chart junk in the form of pictograms are more memorable, Bateman et al. [24] conducted a study with similar results. This study suggests that there is no significant difference in accuracy and recall accuracy between the two graphs types. After a period of 2-3 weeks the recall for non-minimalist style charts were significantly better. Participants found non-minimalist style charts more attractive and easiest to remember. Inbar et al. [25] had similar findings and concluded that while the minimalist concept may appeal to designers, it is not endorsed by the public. Gillan et al. [26] state that the minimalist axiom is overly simplistic and unsupported by theory and evidence.

Only one empirical study was found that supports the minimalist approach. Reaction times were fastest in this experiment for the highest data-ink ratio, however accuracy was not affected [27]. An important difference between this experiment and the previously mentioned studies is the fact that this experiment displayed data as numbers for the minimalist visualisation. Therefore the minimalist version could also be considered as a table rather than a visualisation.

4.1.2.3. Design Guidelines

One way to construct visualisations is the bottom up approach, first establish the most basic building blocks, typically points, and then combine these building blocks in addition with visual variables to construct a data visualisation. For instance, one would create points for every data entry, after this these points are assigned visual variables to encode different dimensions of the data. Carpendale [20] applies this method, he constructs visualisation from marks, and distinguishes them by assigning visual variables to marks. Each of these visual variables has certain characteristics. Carpendale [20] states that substantial power comes from choosing which visual variable would be most appropriate to represent each aspect of the data to create the most accurate visualisations. The ability to make these choices can be greatly enhanced by understanding how a change in particular visual variable is likely to affect the performance of an interpretation task of the visualisation. This method provides some insight, though still heavily relies on the designer, since it depends on the understanding of both the data and visual variables. Card et al. [28] have developed a framework to aid this process. They propose to order and categorize variables in a table, this table shows what data variable translates into what visual variables. Designers should realise that it is essential for users to be able to invert this mapping. An example of such a table can be found in Table 5. The table is structured with data on the left and users on the right.

Variable	Data Type	Mark Type	Visual Variables	Position over time	Interaction

Table 5 *- Simplified version of the framework proposed by Card et al. [28] on how data translate into visual variables in a visualisation.*

If these changes have the same effect for every visualisation, a general ranking can be established for visual variables. Thus, further eliminating the dependency of the designers experience on the visualisations' succes. Cleveland et al. [17] takes such an approach. Data visualisations are described as a set of elementary graphical encodings that people use to extract quantitative information called elementary tasks e.g. position, length, direction. Cleveland et al. [17] argue that the goal of data visualisation is to convey data as accurate as possible. Thus, data which is more important should be encoded more accurately. Therefore, this study forms a ranking of ten elementary tasks to guide designers by means of an empirical research, this ranking can be seen in figure 25. This should elicit judgments that are as accurate as possible, and therefore the graph will maximize a viewer's ability to detect patterns and organize quantitative information, because of the ranking the visualisations are no longer dependent on the designers understanding of the visual variables. The designer solely has to determine the importance of each of the data variables.

However, only having one ranking of visual variables may seem overly simplistic since there are different types of data variables. Therefore Mackinlay [18] expands further upon this idea of ranking. Instead of having one hierarchy, data variables are categorised into three categories: quantitative, ordinal and nominal [34]. A variable is said to be nominal when its a collection of unordered items, such as {Jay, Eagle, Robin}. A variable is said to be ordinal when it is an ordered tuple, such as {Monday, Tuesday, Wednesday}. Lastly, a variable is said to be quantitative when it is a range, such as [0, 255]. Each of these categories has its own ranking of accuracy. Mackinlay [18] also introduces a term for the guideline of encoding more important data more accurately, the principle of ordering; encode more important information more effectively.

Another, more top-down, approach is providing the overview first and let users explore the data. Schneiderman [29] introduces a mantra for designing advanced graphical user interfaces. This is the Visual Information-Seeking Mantra: provide an overview of the data first, allow for zooming and filtering, then provide details of the data on demand. The benefits of this mantra are described as attractive because it presents information rapidly and allows for rapid user-controlled exploration [29]. Even though this mantra takes a whole different approach compared to starting with the most basic unit, it can still be applied together with the discussed methods. Since this mantra does not specify how one constructs the actual data visualisation, solely how one presents the data.

The methods discussed so far are abstract and help contribute towards a framework for a data visualisation. There is also literature that provides valuable insight and guidelines for more specific scenarios of visualisations [17], [30]. One of such guidelines that is suggested is the usage of circles and rounded edges, because these tend to be more memorable and visually pleasing [30]. Another example is the guideline that bar charts should always be used over pie charts, since judgment of position along common scale are ranked higher than angle, in figure 28 the visual representation of bar graphs versus pie charts can be compared. Cleveland et al. [17] conducted tests that showed this. This should also hold up to the guidelines proposed by Mackinlay [18], since their principle is based on the same idea, although this is not explicitly stated in their paper

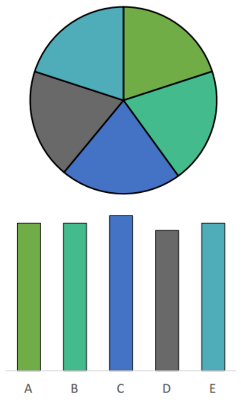


figure 28 *- Pie charts and Bar graphs representing the same data.*

4.1.3. Conclusion of Literature Review

The main objective of this literature review is to assess the literature on construction of data visualisations. Generally, these data visualisations rely heavily on the experience and perception of the designers to create clear and visually pleasing data visualisations. However, if a set of guidelines is established then this process is less dependent on designers, and could help more people to create clear and aesthetically pleasing visualisations.

Various literature has tried to bring structure to this process by first establishing the most basic units that make up a data visualisations. These basic units can convey data by means of visual variables. Having established this list, the most important step for creating visualisations is the matching of visual variables with the data variables. There are various rankings that rank visual variables by accuracy. The goal of a visualisation should be to convey data as accurately as possible, therefore the data should be ranked according to importance, this ranking should be coupled to the ranking of visual variables. Following this guideline implies that the designer only has to rank the data variables by importance. Besides this also very specific rules followed from literature analysis, e.g. usage of bar charts over pie charts, not sharp but rounded edges.

The literature suggests that the minimalist approach to chart design is an useful heuristic but is overly simplistic. Chart junk can lead to better memorability without compromising the accuracy. Furthermore users have shown a preference for the non minimalistic charts.

Everything discussed in this review is applicable for both paper as computer visualisations. With the use of computer displays the set of visual variables can be expanded. One of these visual variables which is introduced by the use of computational displays is motion. Motion could perhaps be a very powerful encoder of data but is only touched upon in the assessed literature.

5. Ideation

In this chapter the process that lead to the initial concept is reported. First of all, the stakeholders are discussed. These include the users and thus largely determine the needs of the program that should automate the visualisation of SDF schedules. Secondly, the SDF data is analysed and current SDF schedule visualisations are explored. These sections together formed an initial concept for the visualisation tool, this idea is discussed at the end of this chapter.

5.1. Stakeholders

There are two groups of stakeholders, firstly there are users of the program, and then there are the developers. The user group contains, embedded system engineers, embedded system students and lastly, clients of those engineers. The second group, includes both me, as the initial developer, as well as any future developers that might work on this project by contributing to the program. A complete list of all the addressed stakeholders in this section:

- Embedded System / Software Engineers
- Embedded System Students
- Clients of Embedded System Engineers
- Developers

5.1.1. Embedded System / Software Engineers

Various dataflow models exist that embedded system/software engineers can work with to model dataflow, such as task computational graphs [10] and SDF graphs. SDF graphs are more expressive, and can efficiently model and analyse multiprocessor applications, like MPEG-4 and MP3 decoders. Currently the people who work with SDF have often devised their own tools. These tools are often not very matured and most of the time only text-based. For visualisations these people usually rely on manually drawn graphs on either whiteboards or in computer software.

The envisioned program will offer a more polished visual toolset to work with. This can help with the analysis and comprehension of the SDF graphs these engineers are working with. Furthermore, they can automate the generation of visualisations of these schedules to a large extent, instead of having to do this manually. Also it will save them the investment of time and skill of having to devise their own tools.

An engineer can use the tool by generating a SDF schedules with their prefered allocation strategy for any SDF graph. After this is completed the schedule has to conform with the format that can be loaded into the program. If successful, the engineer can load the schedule into the program and observe different visualisations of the loaded schedule. The desired functionality for this user can be described as follows:

- Rapid visualisations of the generated schedules
- Visualisations should be clear and conform with standards of SDF schedule visualisations
- Tool should be able to load schedules generated by the user

5.1.2. Embedded System Students

An embedded system student has to get familiar with various dataflow models, since it is a powerful tool for the analysis of streaming applications. After completing their studies this group can turn into an embedded system/software engineer. However this group is here considered as a separate entity since the needs of these students differs from working engineers in the aspect that this group focuses on learning, whereas engineers focus on analysis of SDF. When students have to study SDF the tool could provide student with an interactive visual support next to the existing learning methods. This would create a more ‘hands on’ approach as indicated in one of the user interviews. For the tool to be useful this groups needs to be able to access the tool in educational institutions, but perhaps also on personal machines. The desired functionality for this user can be described as follows:

- The tool should provide insight into SDF schedules, so this user can study them
- The tool should be easy to use, so students do not have to invest in learning the tool too
- Visualisations should be clear and easy to understand.

5.1.3. Clients of Embedded System Engineers

The tool could provide value not only to the engineers who directly work with SDF but also to clients of these engineers. This group hires embedded software engineers to develop a certain application, which might have certain design requirements. The engineer could use SDF to model and analyse the behaviour of the application, and could conclude that the requirements are not feasible. Since the clients often do not have a deep understanding of these techniques it can be difficult to explain why the requirements cannot be met. The visualisations in the tool could be used to showcase the analysis and models. Which can make it easier to communicate to clients why the requirements are not met. It is important to note that these visualisations are of a different nature, since they showcase and explain. Opposed to the visualisations an engineer is interested in, which aim to convey as much data as accurately as possible. The desired functionality for this user can be described as follows:

- Aesthetically pleasing visualisations
- Easy to understand, even if the user does not have any knowledge on SDF

5.1.4. Developers

This group includes anyone who contributed to visualisation tool itself. This can be done by either adding new functionality, or expanding upon existing functionality. Developers invest time and skill into the tool to accomplish this. There are various reasons one could do this, e.g. financial reward, fame but one can also do this to tailor the program to their own needs. A person in this group can, or is likely, also part of one of the user stakeholders which are previously mentioned. The desired functionality for this user can be described as follows:

- Needs to be able to modify the tool, preferable from the source
- Dependencies within the tool should be minimised in order to allow for easy modification

5.2. SDF Data Element Analysis

An SDF schedule describes how actors fire on resources over time, this results in three dimensions of data that hold interest, namely *actors*, *resources* and *time*. An schedule exists of an whole finite number of actors and resources. Furthermore, it takes a certain amount of time to complete the schedule. In figure 29 the three data dimensions and related variables can be seen

Every actor in an SDF graph has dependencies to and from other actors. It consumes tokens from the in going edges to fire, and produces tokens on the outgoing edges when finished. Furthermore, each actor has a execution time, the time it takes to complete the firing. This value can vary between actors. Though this value cannot change over the course of the schedule. An actor firing takes place on a resource.

Resources determine how many actors can fire simultaneously. These resources are generally speaking the processors. A processor can only execute one actor at a time, so when a firing starts the resource is occupied until this actor finishes. Processors can make use of frequency scaling in order to work faster. When working on a higher frequency actors firings will be completed faster, however the processor will also consume more power. Besides this, the switching of frequencies also consumes power.

Time is an important aspect of schedules. Since time connects resources and actors. A schedule is created by telling how actors fire on resources over time. Furthermore, each actor has a completion time and a processor has a frequency. These element affect the time it takes to complete the schedule. But also the scheduling itself can affect the overall completion time of the schedule. Sometimes actors cannot fire because they do not have enough token to start, better scheduling might resolve this issue. Because perhaps there is a way that actors can be scheduled differently, so that enough tokens are present to fire.

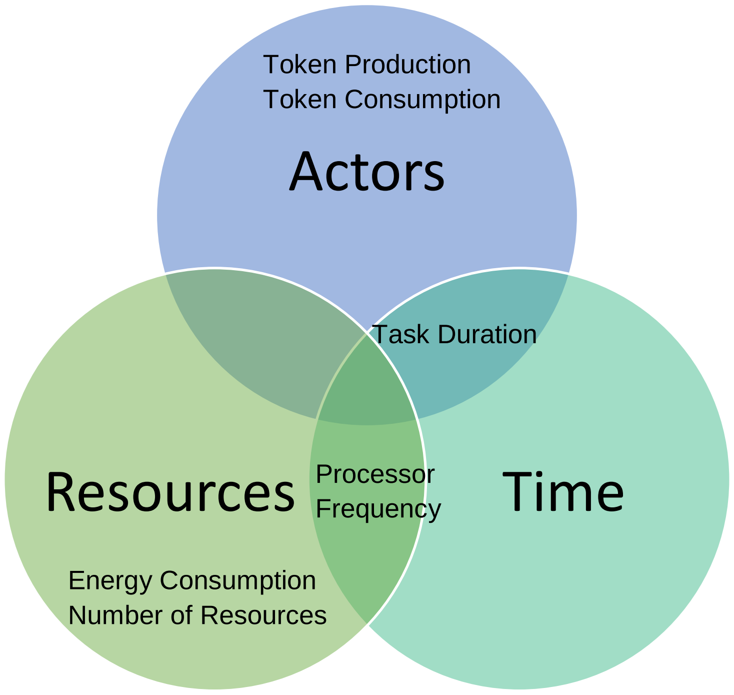
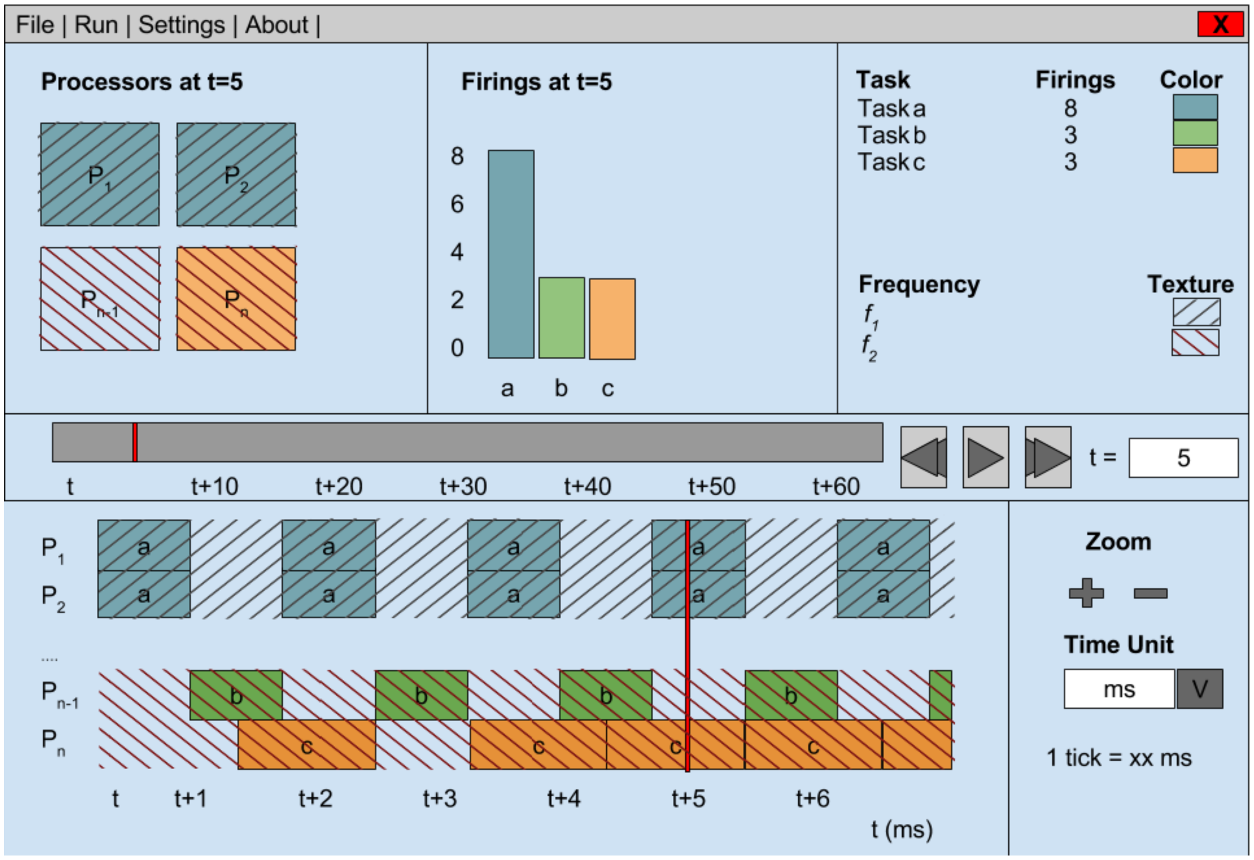


***figure 29** – Data dimensions found in SDF schedules with related variables.*

5.3. Product Idea

The idea is to develop a more automated, and faster visualisation method for SDF graphs and schedules. Users should be able to use their own SDF models and should be able to choose an analysis method freely. Once they have generated a schedule they can load this in a program, after which the program will visualise the data. Since users can be interested in different aspects of the data, the tool will make use of various displays. Each screen can highlight the data from a different perspective. An early concept version as a result of the ideation phase can be seen in figure 30. The tool makes use of the Gantt-like visualisations which are common for SDF schedule visualisations. Furthermore, it offers a moment display and a legend.



Task	Firings	Color
Task a	8	🟦
Task b	3	🟩
Task c	3	🟧

figure 30 *- Early concept version of UI as a result of ideation.*

5.3.1. Interaction Idea

Since the datasets can be very big and are most likely overwhelming, a mantra for designing advanced graphical user interfaces is used. This is the Visual Information-Seeking Mantra as described by Schneiderman [29]. Users should first of all be provided with an overview of the data. Since this might be overwhelming zoom and filter options should be provided. This allows users to remove irrelevant information and prevent data overload. Lastly, details should be provided on demand. In order to meet this mantra the interface should be highly interactive, allowing users to easy navigate, zoom and filter the data intuitively.

Another idea for interaction will be time controls. The idea is to allow users to play schedules and see changes happening over time. This makes use a person's sense of time and causality to develop an understanding of the data. This interaction should allow for quick and easy general understanding of what is going on.

5.3.2. Experience Idea

The envisioned workflow for the tool is to be at the end of the tool chain. Users start out with an SDF graph, after this the user chooses a desired scheduling method and simulates this in a tool of their desire. This simulation should result in a trace file, this trace file should conform with the format used by the tool. If this is the case the generated schedule can be load into the program. The program visualises all data and provides the user with a visualisation of the schedule. The user can visually analyse this to spot any discrepancies. In figure 31 the workflow is explained visually.

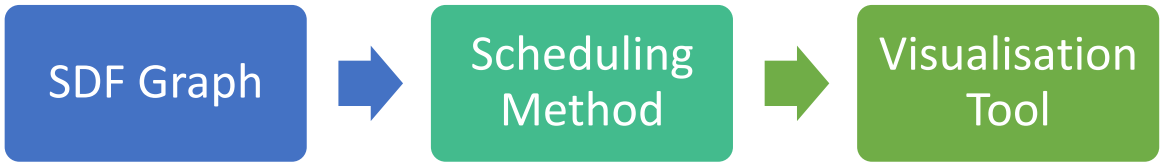


figure 31 *- Envisioned workflow with the tool.*

6. Specification

This chapter aims to specify the product specifications and requirements. Whereas the ideation chapter provided an initial concept to work with, this chapter aims to describe a more detailed prototype to build. This is achieved by looking at the tool from the perspective of the stakeholders. First user scenarios present the envisioned usage of users. Furthermore, user interviews have been conducted with people from industry as well as from an academic background. Together these elements have led to a list of requirements and constraints the tool should fulfill.

6.1. User Scenarios

This section describes the context various stakeholders would use to program in. It goes into more detail as to why, and how they use the program. First, an user scenario of a researchers analysing scheduling methods will be presented. Followed by an user scenario of an embedded software engineer. Thirdly, an user scenario of a student will be described and lastly, an example of a stakeholder who works in the industry is described. A list of all the user scenarios in the form of a list:

- Researcher
- Embedded Software Engineer
- Embedded Systems Student
- Industry Example

6.1.1. Researcher

As a researcher, Adam is working on a novel scheduling method for SDF graphs. He wrote the algorithm for the scheduling operation and applied this to various SDF examples including an MPEG-4 decoder. Now that Adam has generated the schedules according to his novel method, he is interested in certain properties of his scheduling technique like efficiency, throughput and if he is able to spot any points for improvement. Adam has a hard time understanding the 5mb large text-file depicting the schedule. So instead of analysing or working this schedule out by hand. He loads the generated trace into the visualisation tool. The visualisation tool extracts the data from the trace file and updates the visualisation displays. Adam uses the play function once to get an overview of the trace, and perhaps is already able to spot some points of interest. While playing the file Adam sees that there are various moments where all processors except for one are on idle because they have to wait for a certain actor to finish. Adam spots this bottleneck fast because of the visualisations. Now that he is aware of this issue he tries to improve his scheduling method to resolve this bottleneck. Once Adam thinks he found a solution to this scheduling mistake he repeats the process and tries again. After a few iterations Adam is satisfied with his scheduling method and writes a paper about his novel method. In this paper he uses visualisation generated by the tool to demonstrate his scheduling method.

6.1.2. Embedded Software Engineer

Bob has gotten a job from a customer who would like to develop a streaming application. The customer has certain design requirements for the product, e.g. high speed and cannot occupy more than two processors. Bob takes the job and starts modelling the application with SDF. He generates some schedules with various scheduling methods on two processors. After the schedules are generated, Bob finds out that none of them have met the time requirement. In order to find out why Bob loads the traces into the visualisation tool. With the use of various displays Bob comes to the conclusion that he is not able to fulfill both design requirements. Bob sets a meeting with his client to discuss this issue. Once the meeting takes place Bob tells the client that he cannot meet the design requirements and also explains why, with the help of the visualisation tool. The client understands the point Bob made and together they discuss solutions, they agree to make the application run on three processor cores instead of two. After the meeting Bob repeats the analysis for a platform where the application runs on three processors, he comes to the conclusion that this is feasible and now he can fulfill all the design requirements.

6.1.3. Embedded Systems Student

Carl is an embedded systems student at the university. During one his courses he is introduced to the SDF modelling formalism and for this course he has to complete various assignments. In one of the assignments he has to analyse the differences between various scheduling methods. Carl starts out with a simple SDF graph and generates various traces with various scheduling methods. In order to establish the differences between the scheduling methods Carl loads the traces separately into the visualisation tool. The tool displays how the actors are scheduled over time, Carl compares these and is able to spot some differences and uses this to write his analysis on the various scheduling methods. He reports his findings for the assignment and exports views which show the differences to strengthen his results.

6.1.4. Industry Example

Denis is an employee of Recore System for various years now. He is tasked with the analysis of their new platform. This new platform has many processor cores and deploys a certain application. Dennis creates an SDF model of this application and uses this to generate a trace of the application on their new platform. This trace is loaded into the visualisation tool. Denis navigates through the file and uses the search and brush functions extensively to make the large dataset manageable. After a thorough analysis he finds out that one of the buffers between actors is floating out of bounds. Now that he has spotted a problem he tries to find out why, the visualisation tool helps him to find this problem. He manages to resolve this problem. Furthermore, he is now informed about the minimum number of cores required for the application to run smoothly and able to guarantee certain properties.

6.2. Interview Results

In order to get a better understanding of the context and the needs of users interviews were conducted with potential users. In total four users have been interviewed, these people were Robert de Groote, Timon ter Braak, Philip Hölzenspies and Bart Theelen. First the interviewees will be introduced in more detail, after this the findings of the interviews are presented. The full interviews and interview questions can be found in Appendix I.

6.2.1. Interviewees

Here the four people that have participated in the interviews are introduced in more detail. These interviews are ordered chronologically. Most interviewees use SDF in a different context, except for Robert de Groote and Philip Hölzenspies. Their background and (previous) usage of SDF are rather similar.

- Robert de Groote currently holds a postdoc position at the CAES group at the University of Twente. He has worked for over five years with SDF graphs for his PhD thesis on this topic.
- Timon ter Braak works for Recore Systems, a company located in the Kennispark neighbouring the University of Twente. Recore Systems designs heterogeneous many core processor systems. Timon ter Braak encounters SDF graphs on a daily basis during his work.
- The interview with Philip Hölzenspies through was conducted through Skype. In the past Philip Hölzenspies also has done research into SDF graphs at the CAES group at the University of Twente. Currently, he is located in London working for Facebook.
- Bart Theelen works for TNO in Eindhoven. He has also worked with SDF graphs in the past and on top of this worked on the development of a visualisation tool called TRACE.

6.2.2. Current Methods of Visualisation

The current methods of the interviewees to understand and manage the underlying SDF graphs behind their work vary widely, though what they all have in common is that they are not very evolved. Philip Hölzenspies indicated that he does not use any tool whatsoever but uses a whiteboard and markers to work out SDF graphs. All other participants have indicated that they use some sort of tool for the analysis of the SDF graphs. For example, Robert de Groote has developed his own tools in python, but they are solely text-based. Timon ter Braak indicated that Recore Systems has developed its own in house visualisation tools. However these are still in prototype phase and are not matured yet. Perhaps the most developed tool that participants used in their workflow for SDF visualisations is TRACE, Bart Theelen uses this to generate Gantt like visualisations. Though, one must note that this tool is for presenting activities on resources in the form of Gantt chart visualisations not directly tailored for SDF schedules.

All interviewees have indicated that there is room for improvement on their current tool usage. Philip Hölzenspies hints that a tool such as that from Robert de Groote could already be an improvement, however it is still limited because it does not scale well. Robert de Groote also indicated that improvements can be made by adding visualisations: “In SDF you want to be able to tweak values and see what results this has. Doing this visually allows for much faster interpretation of the changes”.

Timon ter Braak and Bart Theelen on the other hand think that there is room for improvement in the type of visualisation. The Gantt type of chart does not display dependencies between tasks. The perspective in this visualisation also matters. The most common Gantt chart in the context of SDF schedules has time on the horizontal axis and resources on the vertical axis. Timon ter Braak, however is of the opinion that the perspective from the tasks is more interesting, thus with tasks on the vertical axis. This could perhaps offer an easier insight in the dependencies between tasks with an addition to visualise dependencies.

6.2.3. Data Variables

The data in SDF schedules can be categorised into two categories. On one hand the schedule can be considered as a whole and data can be coupled to this, e.g. the total idle time of a processor, total ratios between task firings. On the other hand there is data on a specific moment within the schedule, e.g. the task on a processor, frequency of the processor.
Overall the interviewees have indicated to be more interested in the second type of information. Only Bart Theelen mentions that the overall summary statistics could be useful, though Timon ter Braak counters this by saying that the summary statistics are already easily obtained by application statistics. Thus it might be more valuable to display information from the schedule rather than summary information of the schedule itself. An example would be to display task firings in a moment rather than a total overview of how often a task has fired.

When analysing these schedules the user is interested in optimising these schedules. Thus they are interested in variables that provide useful insight on how to optimise, for this it is important to be able to spot bottlenecks. So at first glance the task activation, task duration and task on a processor are important. Different activation moments or processor mapping might resolve bottlenecks and also tasks durations could perhaps be decreased to resolve bottlenecks. However, perhaps of more importance is the relation and dependencies between tasks. Since these are at the core of the problem with bottlenecks when SDF graphs are mapped to resources such as processors. The idle time of a processor could be of some value in this context because it could sign towards a possible area of optimization. However, a low utilisation of a processor does not necessarily mean that the schedule is inefficient or bad.

Task firings and ratios between firings are not interesting in the context of pure SDF because they stay within certain bounds. After an initial transient phase a periodic phase is entered, because of this periodic phase the token distribution and firing ratios can only diverge with a set amount within a period, since the ratios and distributions are the same at the end and start of a period. Thus this information is already encoded into the SDF graph itself and therefore does not hold the prime interest of the viewer. However, Bart Theelen indicates that this information becomes much more interesting if you look at more dynamical models, in these cases it is not fixed and much less implicit in the model.

The frequency a processor is running on could also be of interest for users. Though, Philip Hölzenspies and Robert de Groote point out that this would be a whole different angle from the one discussed so far.

Timon ter Braak acknowledges this point, frequency can be very interesting, however it is not deployed in the context Recore Systems works in. Their current platforms do not have frequency scaling, and even if their platform would support it, they would not directly use it. The previous board developed by Recore Systems had the option to set the frequency, however, this value was set and never changed again. Part of the reason why scaling is not applied is because it is hard to know when you do this efficient, and for the other part the reason is that cores cannot be scaled individually.

6.2.4. Concept Version

From the previous section it appears that the dependencies between tasks are of prime importance to users. The proposed concept version, however, only displays information about the SDF schedule, so in the visualisations there is no information about the original SDF graph available. Robert de Groote and Philip Hölzenspies stress at various points during the interviews that by displaying the original SDF graph you can exactly visualise these dependencies. And if you would take this even further you can provide much more valuable information. For example, with an interactive SDF graph over time you could also visualise what scheduling choices have been made by highlighting what is active and what was ready to fire. With the current Gantt chart view you can only see what decision has been made to generate the schedule. Robert de Groote points out that having both SDF graph and the Gantt chart view would sketch a very complete and comprehensive view.

He proposes to take the implementation of SDF graphs itself a step further. The workflow with the concept version is at the end of the tool chain, the user generates an SDF graph, generates a schedule in a tool and after this, uses this concept tool to visualise this schedule. To drastically simplify this workflow, Robert de Groote (and Philip Hölzenspies hint at the same) propose to shorten the workflow by directly loading the SDF graph or to create this graph in the visualisation program. After which the program can interface with analysis tools that can generate schedules. These schedules are send back to the visualisation tool so that they can be visualised. This way you can still make use of existing visualisation but still maintain information of the original SDF graph. In figure 32 the envisioned and proposed workflow can be seen visually.

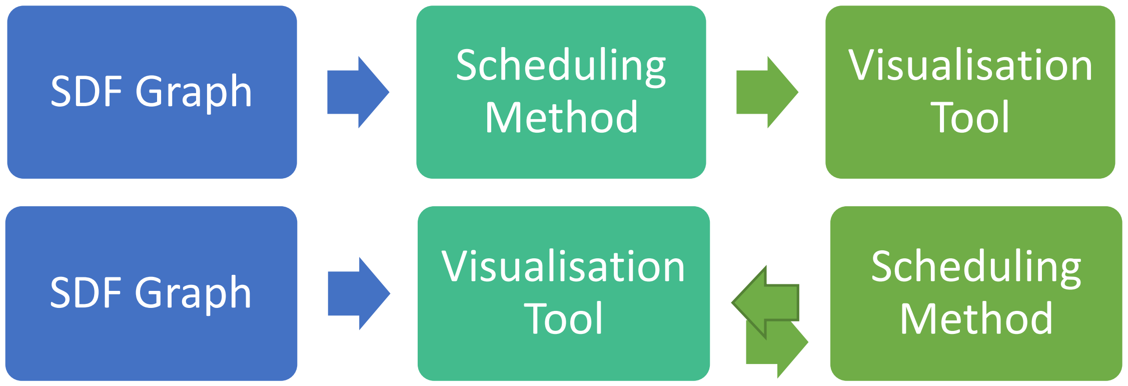


figure 32 *- Envisioned workflow (top) and proposed workflow as by Robert de Groote (bottom).*

Most interviewees see the ability to play schedules with interactive displays as an useful addition, especially if SDF graphs are added as a display according to Robert de Groote and Philip Hölzenspies. However, it should not restrict the users ability to navigate through the schedules on their own. Bart Theelen is most sceptical about the play function, the Gantt chart already displays over time and sketches a pretty complete picture.

Another issue concerning the time has to do with the unit of time. The unit of time is not of much importance to the interviewees, since this is all relative in the displays. It could be practical to be able to view this, though it should certainly not have a dedicated display on the screen. Also, Timon ter Braak pointed out that things can become more complicated when you work with a heterogeneous system. At different clock frequencies ticks will have different durations for processors, and therefore one should be aware of this when data is being displayed.

The tool should be able to load in the schedules. The interviewees seem to be very flexible with the format this data should be conveyed to the program. According Bart Theelen this should be as simple as possible, so text-based, certainly nothing encoded. Both Philip Hölzenspies and Robert de Groote would prefer to see this as tightly coupled with the analytical tool as possible to ensure that are no discrepancies between the two.

Another issue that is brought forth at various points during all interviews is scalability. There seems to be a big gap between the number of actors used in academic context and in industry. The interviewees indicate that in academic context vary roughly from 7 (Robert de Groote) - 25 (Bart Theelen). In the industry however this number is many times higher, in the context of SDF this can be a couple hundreds and in the context of HSDF and CSDF it can easily in the thousands.

6.2.5. Summary of Interviews

All interviewees agree this tool could possible offer an improvement on the current situation. Robert de Groote says:”Definitely, text based is very limited. Visually displaying information is much faster”. Timon ter Braak states that they have not found a solution to this complex problem yet, and this could perhaps be one if it is well executed.

The tool would also help to explain to users why a SDF application acts like this, according to Philip Hölzenspies. Robert de Groote sees good use for this tool in education, it offer a more ‘hands on’ approach. The tool would generate value for all embedded system and software engineers in general according to Philip Hölzenspies, since it could fill the gap between existing high and low level tools. Bart Theelen also sees good use for the documentation of SDF schedules, he thinks that these charts are now often made manually, this tool could automate this.

The interviewees have suggested to make the following changes to the current concept:

- Remove time display from screen
- Remove horizontal time axis
- Increase value of processor view, since it currently does not display any new information
- Display dependencies between actors
- Scalability issues are likely to arise

6.2.6. Alternative Visualisations

Over the course of the interviews the interviewees have brought up various alternative visualisations that could be used. Timon ter Braak for instance indicated that it would be very interesting and insightful to be able switch between having resources and actors on the vertical axis. Timon ter Braak also hinted that concurrency execution diagrams might provide an efficient method for visualising the schedules, especially when used in 3D. He points to an example where this is used in the context of the GO programming language [31]. An example of such a visualisation can be seen in figure 33.

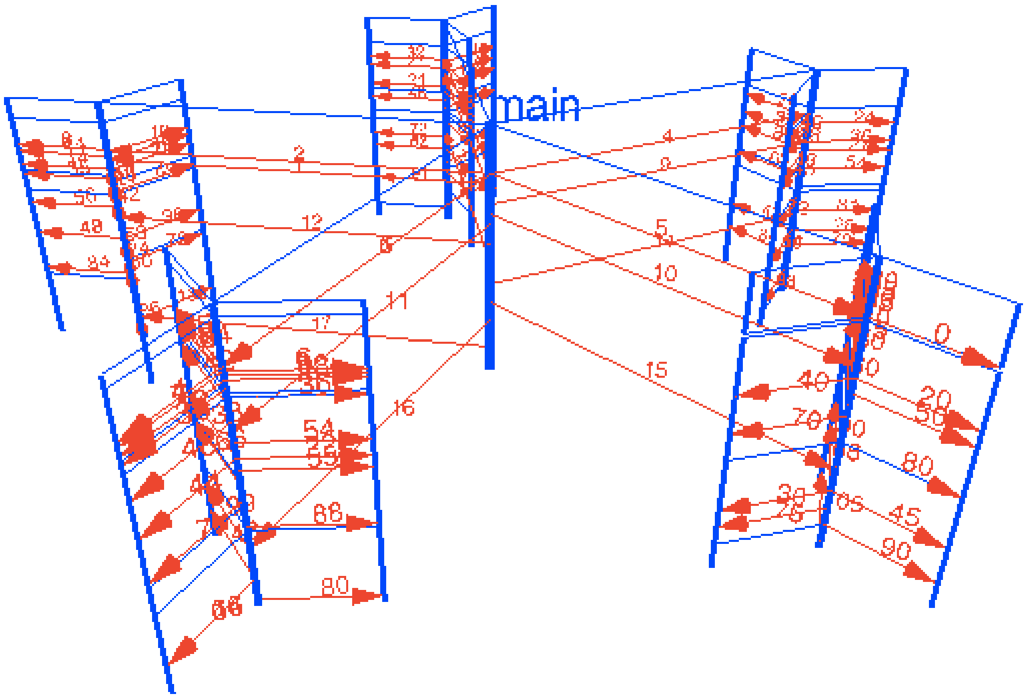


figure 33 *- Concurrency execution diagram of concurrency happening in the Go language.* Image taken from [31].

Philip Hölzenspies thinks some sort of temperature map could provide usefulness. He refers to an article [32] about the ladder of abstraction. The ladder of abstraction is explained as follows, if you move to a new city, you might learn the territory by walking around. Or you might use a map, but far more effective than either is both together, a street-level experience with higher-level guidance. The most powerful way to gain insight into a system is by moving between levels of abstraction. The current visualisation has time specific views, but also more abstract views, such as the Gantt chart. The tool scrolls this chart over time, however, the chart itself is abstracted over time. Some sort of temperature map could provide another display at the same level of abstraction or possibly allow for another level of abstraction. An example of a heatmap can be seen in figure 34.

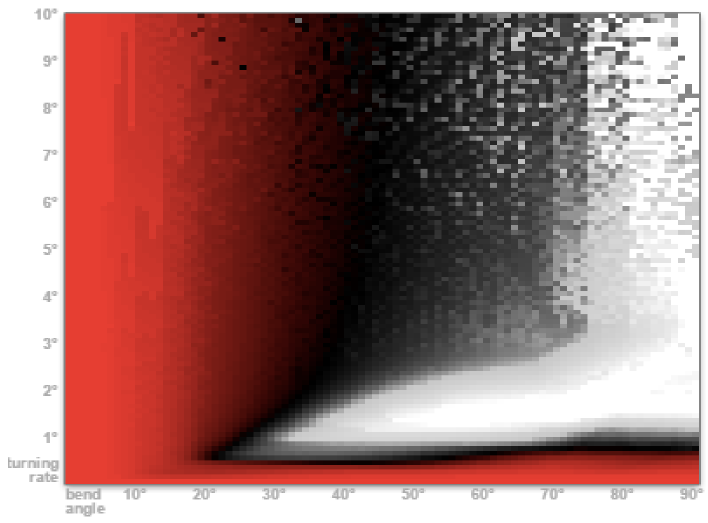


Figure 34 *- An example of an heat map, with three variables encoded into the graph. Bend angle on the horizontal axis, turning rate on the vertical and lastly, the time completion represented by colour. Image taken from [32].*

6.2. Requirements

In order to specify what the program will do a list of requirements is compiled. Each requirement describes a characteristic the program will need to fulfill in order to provide value and utility to the stakeholders. The requirements in this section are composed after the input of potential users through interviews and as a result of the ideation process and background research. The requirements are also an important input into the verification process, for the evaluation, the prototype can be tested whether requirements are met.

- Any user with familiarity to computers should be able to load data into the program without the need of the user manual.
- Any user with familiarity to computers should be able to obtain the program.
- Any user with familiarity to computers should be able to launch the program.
- Any user that knows how Gantt-like chart schedules work for SDF should be able to understand this display in the tool.
- Any English speaking user needs to be able to read and understand all text in the application.
- Users should not have to restart the program to load a new data file.
- The program should be able to visualise SDF schedules, this includes for any number of resources, actors and the firings of actors on resources over time.
- Users need to be able to scroll through time variable, most importantly for displays that are not abstracted over time.
- Users should not feel overwhelmed when using the program.
- Users should be able to extent upon the program.
- The mean time between failures should not be smaller than five minutes.
- The mean time to recovery should not be bigger than one minute.

- The program should not freeze upon loading a data file.
- The program should load a file within one minute.
- When a data file is loaded and not recognized the user should be notified.
- The program should be able to load a file of at least 5mb.
- The program should not impose any roundings to the data that is loaded into the program.
- All displays should be in sync with each other.
- The program should not contain any spelling mistakes.
- The program should respond within a second when the user filters data
- The program should load within 10 seconds.
- The program should not require an internet connection.
- The program should not damage the original data file in general or in the event of a crash.
- The program should be able to load schedules from a text file in the format proposed by W. Ahmad et al. [5].
- The program should be able to run on the most common operating systems, e.g. Windows, IOS and Linux.
- The visualisations should comply with standards for SDF graph and SDF schedule visualisations.
- An average computer needs to be able to run the program.
- The program needs to support at least a minimum resolution of 1024x768.

6.3. Product Specification

The tool should provide a quick and highly interactive visualisation platform. The file that is loaded into the program should be as accessible as possible, thus some sort of text-based format is probably most suitable to accommodate this. This will also allow most users to use the tool with their prefered analysis method since they can output traces into the desired text format and load that into the tool. The initial workflow of tool would be at the end of the toolchain, however the user interviews have brought forth that a more integrated workflow is desired. This means that users use the tool to visualise their schedule and then can make adjustments to their previous steps, to subsequently see the results in the tool. In order to facilitate this the tool needs to be modular and dynamic.

A major issue that most likely is going to arise is scalability. Especially if the tool will have the industry as its target user, since SDF graphs that are used in the industry are significantly bigger in the dimensions of actors and resources.

6.3.1. Modular

The aim of the tool should be to be as modular as possible on all levels. This should allow users to tailor the program according to their needs. Modularity helps with this since it will allow users to disable unnecessary components. Furthermore, since the source will be publicly available specific components could be expanded upon or even new onces could be created without having to know anything of the other components.

6.3.2. Dynamic

In order for the program to provide value and utility to the user it is important that it is dynamic. If the program could only visualise one specific case its utility would be rather shallow. It would provide value in the sense it can be used as a display demo and it might be aesthetically pleasing. However, this would only provide insight in one specific case and thus might be something a person looks at once, but never uses it again. Thus, it would be more of a showcase than an actual tool.

On the other hand, if the program were to be able to load data files, thus being able to visualise a great many cases the program becomes more dynamic, and its appliances becomes much larger. Since now any person can use the tool to visualise their SDF schedules. This however does mean that the program needs to be able to load these data files, and generate different visualisations depending on this data. Thus providing an extra challenge.

6.3.3. Experience Specification

The tool should have a low threshold tool use, the envisioned users are already accustomed to a certain workflow. In order to target these users the tool should not pose a high threshold to start using the tool. Also to facilitate this change users should not get overwhelmed by the tool. Furthermore, the tool should keep up to standards used by other tools.

6.3.4. Interaction Specification

As indicated in the interviews a major issue that is very likely to arise is scalability. A possible solution for this issue could be a highly interactive tool. Powerful search and brush options for users could make these very large data sets manageable. This also aligns with the Mantra proposed by Schneiderman [29] to provide an overview of the data first, and then let users zoom in and view details on demand.

7. Realisation

This chapter discusses the realisation of the tool. In this chapter provides an in depth explanation on how the tool is built. First the tools that were used to realise the tool are discussed. Followed by the Visual implementation, this explains why tool looks the way it looks. Lastly, the technical implementation is discussed. Here is explained how to tool operates.

7.1. Development Environment

The visualisation tool is build in the Unity Engine (henceforth Unity). Unity is a cross-platform game engine developed by Unity Technologies. One emphasis of the game engine is portability. Unity projects can be build to a great number platforms. An overview of all supported platforms is shown in figure 35. The engine is written in the C and C++ language. Developers can write scripts in C# and Javascript. Even though Unity is originally a game engine, it is starting to be used for applications outside the field of gaming, such as architecture, simulation and visualisations. At the time of writing Unity is in version 5.4. The project was started in version 5.3 and upgraded midway to version 5.4, this did not require any changes to the project. The engine can support both 3D as 2D, and since update 4.6 offers a powerful new UI system. I have chosen for this development environment since it offers powerful visualisations options for both 2D and 3D. So in the case that 3D visualisations would be needed this option was available. Furthermore, I have over three years of experience with Unity, this should help to realise a prototype within the timeframe of this project.

figure 35 *- An overview of all the platforms Unity Engine can build for.*

7.1.1. Software

As discussed before Unity is the core software used to realise this project. Other tools that have been used in the realisation of the prototype are:

- Microsoft Visual Studio 2015
- Photoshop CS5
- Google Drawings

7.1.1.1. Microsoft Visual Studio 2015

Microsoft Visual Studio 2015 has been used to write the code in. The program is automatically installed with Unity and is freely available under the community edition. The program supports code completion as well as code refactoring.

7.1.1.2. Photoshop CS5

Photoshop CS5 has been used to create the visuals for UI elements. The usage of layers with different blending options allows for rapid creation of shaded UI elements. On top of this Photoshop offers a wide variety of export settings. Thus providing a large tool set to compress the UI elements as efficiently as possible.

7.1.1.3. Google Drawings

Google Drawings has been used to quickly create UI layouts. It is a free, web-based diagramming tool developed by Google. The power of the tool lays in its simplicity and the alignment guides, snapping to grid and auto-distribution of visual elements.

7.2. Visual Implementation

This section explains why the visual looks the way it looks. It explains how and why UI is structured in this way. Furthermore, the usage of icons and symbols are mentioned and discussed as to why. The algorithm that assigns colours to tasks is discussed in technical implementation, since these are randomly generated, and only have as one of the constraints that they need to be visually pleasing.

7.2.1. Gantt Display

The aim of this display was to bear close resemblance to the Gantt charts used to visualise SDF schedules. These graphs have time on the vertical axis and either resources or actors on the vertical axis. The Gantt charts that are used in the field of SDF are constrained to paper. However, this tool is visualised on a computer, therefore much more freedom and interactivity options are available. A screen of this display in the tool is shwon in figure 36.

Since screen space is limited the display scales in both vertical as horizontal direction. If the schedule becomes too wide for the screen it will simply go beyond it. Users can access the out of view content by using the time controls or by adjusting the scale of the schedule. Vertical scaling happens automatically. Only when the height of a processor row gets too small a scrollbar is added to the side and the processor rows are extent below the screen, thus ensuring a minimum height for each processor row. Processor rows themselves can also be hidden from view, thus allowing more space for the remaining processor rows.

One must note that the labels are always displayed on top of the rows. This is to ensure that the user can always see what rows are currently visible, rather than having to remember this when a file is played. Lastly, a red line is added in the middle of the schedule. This represent the current time, this line has exactly the same appearance as the time indicator in the timeline. This is done to strengthen the idea that they are the same, they both display what moment is currently observed.

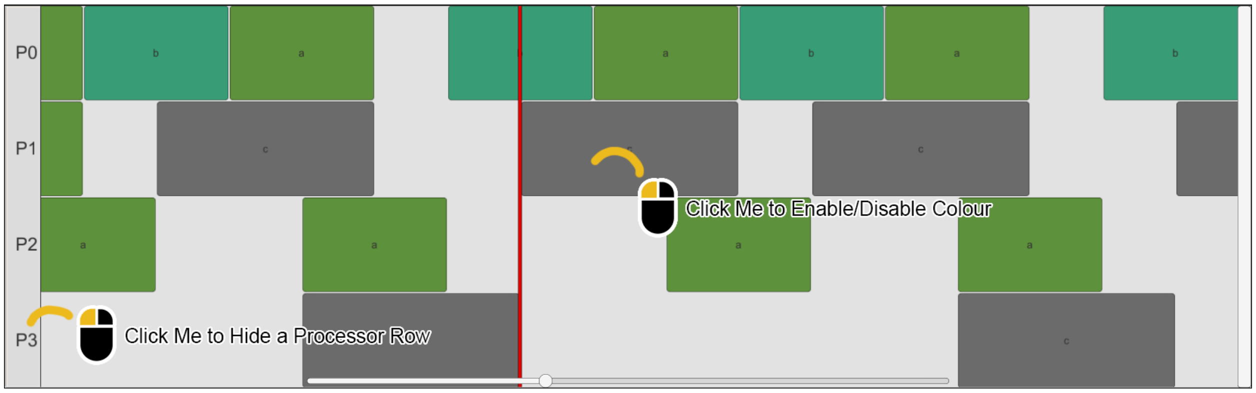

P0
P1
P2
P3
a
b
c
Click Me to Enable/Disable Colour
Click Me to Hide a Processor Row

figure 36 *- Gantt chart generated by the visualisation tool with user interaction options.*

7.2.2. Processor Display

The goal of this display is to provide a precise view of the current occupation of the resources. Therefore each processor is represented by a bar. The name of the processor is shown in the center of the bar. The colour of the whole bar is changed according to the current occupation of the processor. Within the processor bar there is a black indicator line. This shows the utilisation of the processor thus far. If the processor has been idling the whole time, this indicator is at the bottom. On the other hand if the processor has been executing tasks for the entire time the bar is at the top. This view does not offer any direct interactivity to the user. A screen of this view can be seen in figure 37.

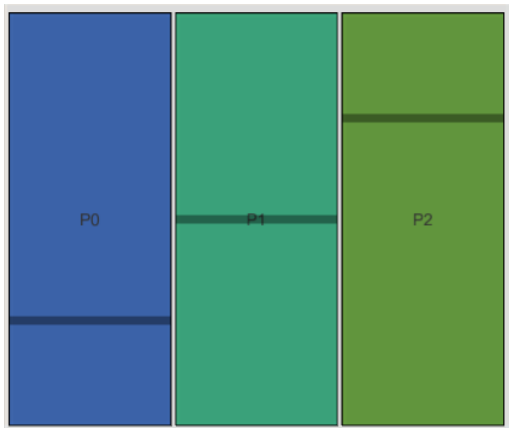
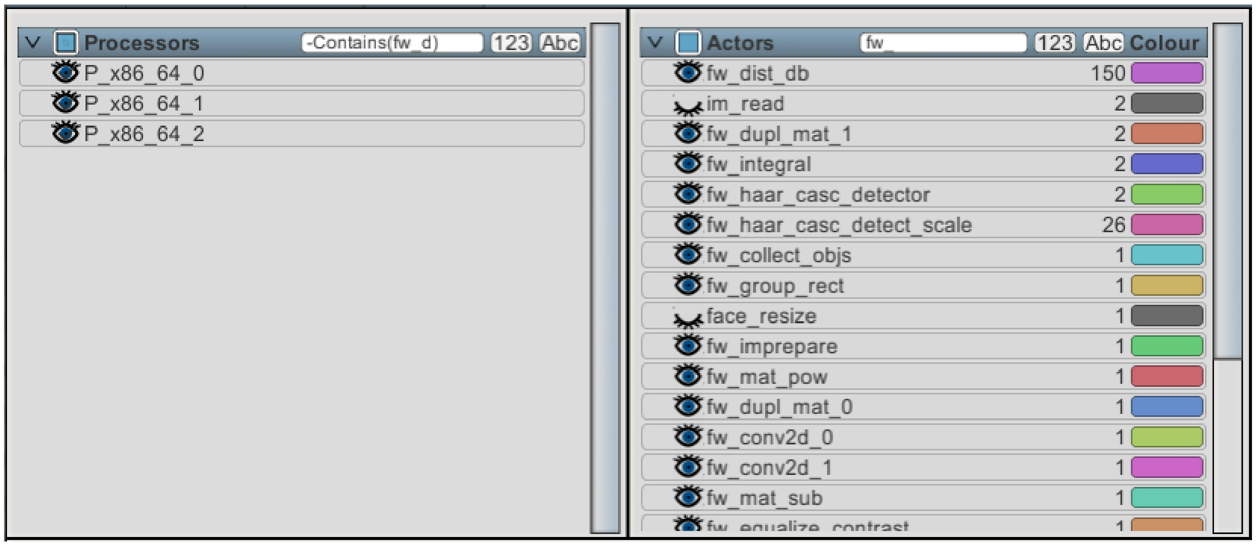


figure 37 *- A screen of the processor display for three processors with an utilisation of 25%, 50% and 75% respectively all executing different actors.*

7.2.3. Legend

The goal of this display is to provide an overview of all the data elements such as processors and actors present in the data. Furthermore, it shows how these elements are encoded in the other displays. The legend contains separate lists for the processors and tasks as is shown in figure 38. Each list can be collapsed with the arrow in the header. Collapsing a list will hide the whole list except for the header itself. On top of that for every data entry the data is displayed. In the case of processors this means showing the processor name and in the case of actors this means showing the name, number of firings and colour. Users can click on the data element to change it, e.g. clicking on a name will bring up a window to change the name, clicking on the colour will allow a user to pick a new colour. On top of that every row starts out with the icon of an eye. If the eye is closed the element is hidden or disabled. Open, on the other hand means it is visible or enabled. The eye can be clicked to toggle the state of the element.

Since these lists can grow rather large, the header offers filter and order functions. The two order functions available are ordering by execution order and ordering alphabetically. These buttons are labelled '123' and 'ABC' respectively, indicating the sort of ordering in the label. Filter options are available by a search bar. Entering plain text in the bar will filter for actors containing this string in their name or display name. On top of that also functions can be used to provide more powerful search options.



	Processors -Contains(fw_d) 123 Abc
👁	P_x86_64_0
👁	P_x86_64_1
👁	P_x86_64_2

	Actors fw_ 123 Abc		Colour
👁	fw_dist_db	150	🟪
⌣	im_read	2	⬛
👁	fw_dupl_mat_1	2	🟧
👁	fw_integral	2	🟦
👁	fw_haar_casc_detector	2	🟩
👁	fw_haar_casc_detect_scale	26	🟪
👁	fw_collect_objs	1	🟦
👁	fw_group_rect	1	🟨
⌣	face_resize	1	⬛
👁	fw_imprepare	1	🟩
👁	fw_mat_pow	1	🟥
👁	fw_dupl_mat_0	1	🟦
👁	fw_conv2d_0	1	🟩
👁	fw_conv2d_1	1	🟪
👁	fw_mat_sub	1	🟩
👁	fw_equalize_contrast	1	🟧

figure 38 *- Legend display of processors (left) and actors (right), with the 'fw_' filter enabled on actors and –Contains(fw_d) on processors.*

7.2.4. Time Controls

This sections provides users with an indication what the current time is that is being displayed, additionally controls are offered to manipulate this value. The general idea of this section is to make it similar to the time controls offered to users of video players, such as those of VLC media player shown in figure 39. A bar that indicates the current moment in respect to start and end. Furthermore, there is a play/pause, forward and backward button. The icons used on these buttons are the same as used in video players. Since the dragging of the time indicator might not offer enough precision, or could be difficult to handle for some users, there is also the option to enter a time. This will set the time to this specific number. The entered text can be a floating point number, thus jump to this time point. However, percentages can also be entered, e.g. entering 50% will set the time to halfway of the schedule duration regardless of how long it is.

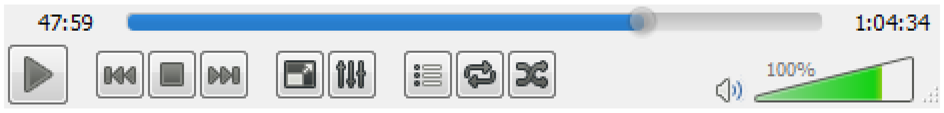


figure 39 *- Default layout of time controls in VLC media player.*

7.3. Technical Implementation

The program is build upon two core principles, modularity and dynamic displays. One should be able to remove or replace a component without affecting the other components. This should allow for easy extensibility and customisation. Therefore the different components of the program will be discussed individually. Before this however, some design patterns that most components rely on are discussed.

The first component to be explained is the file parser, which loads the data from a text file. After this, the data storage method within the program is discussed. Followed by an explanation how time is kept within the program, the clock. Then the implementation of modal windows (pop ups) is discussed. Lasty, all the displays that make up the visualisation tool are discussed.

7.3.1. Design Patterns

In software design a pattern is a general reusable solution to a commonly occurring problem. I encountered and learned about these patterns from the book Game Programming Patterns by Robert Nystrom [33]. Therefore most implementations will be similar as presented in the book. The patterns that will be discussed are:

- Singleton Pattern
- Observer Pattern
- Component Pattern
- Prototype Pattern

7.3.1.1. Singleton Pattern

The singleton design pattern ensures a class has only one instance, and provides a global point of access to it. This is done by keeping a reference to itself, and if this reference is empty, it instantiates an instance of itself. In the context of Unity this pattern is slightly altered, to fit in the component mindset and to be able to inherit from the monobehaviour base class. Singleton patterns are only initialised when accessed. This means it is initialised during runtime. In the implementation of this project the pattern is used for the clock, data and modal panels. Only one clock should be active at a time, and there are multiple displays which need access to it, same applies for data. We also only need one modal panel at a time, since a user should first answer to the current modal window before a new one appears. Furthermore, modal windows should be accessible from anywhere in the code.

7.3.1.2. Observer Pattern

One of the most important patterns used in this project to decouple elements is the observer pattern. The observer pattern lets one piece of code announce that something interesting happened without actually caring who receives the notification. This pattern exists out of three components, an EventManager, instigator and listeners. The EventManager is at the center of all this. It implements the singleton pattern to be available from any piece of code. The EventManager maintains a list (Dictionary) of events to call with a key and offers a start and stop listening function, which require a key and event parameter. Calling either of these functions will add or remove the listener from the list. It is crucial to unregister for events so garbage collection can clean it up, otherwise an reference is kept and garbage collection will skip it. Furthermore, the EventManager has a trigger function which allows anyone to trigger an event by a key.

EXAMPLE IV

After having loaded the data it is important that all displays update and display the new data. In order to ensure this all displays which relate to data register themselves with a function to the key “DataChange” at the EventManager when they are enabled and unregister upon disable. Now if a new file is successfully load and the list of processors, tasks and firings are up to date, the “DataChange” is triggered at the EventManager. The EventManager subsequently notifies all the displays which update their views. A detailed explanation can be seen in figure 40.

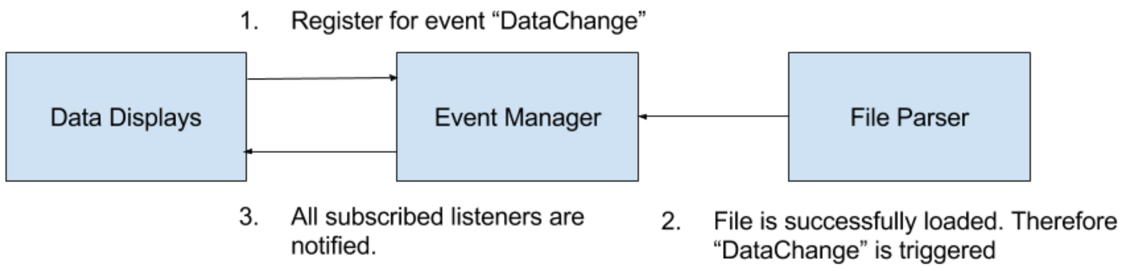

1. Register for event “DataChange”
Data Displays
Event Manager
File Parser
3. All subscribed listeners are notified.
2. File is successfully loaded. Therefore “DataChange” is triggered

figure 40 - Explanation of Observer pattern.

7.3.1.4. Component Pattern

This pattern is not directly implemented into the program by me, but is a core principle of Unity, since it is component based. This pattern allows a single entity to span multiple domains without coupling the domains to each other. To keep the domains isolated, the code for each is placed in its own component class. The entity is reduced to a simple container of components. So instead of having an entity that does both rendering, audio and physics. The entity is transformed to a component container that may or may not hold a rendering, audio or physics component. In practical cases some components will need to talk with each other, but communication between those components can be restricted to only those, rather than having everything be able to communicate with everything. Also a benefit that arises from this implementation is the fact that components can be reused in different locations.

7.3.1.5. Prototype Pattern

Much like the component pattern, the prototype pattern is not consciously chosen to be implemented in the program, but is imposed by Unity. The key idea is that an object can spawn other objects similar to itself. The neat thing about is that the pattern does not just clone the class of the prototype, it also clones the state. So instead of creating a new class and assign all its parameters, we force one object into shape and clone this as many times as are needed.

7.3.2. File Parser

One of the constraints that was set on the project is file format that the program should be able to load data and adjusts its display accordingly.

7.3.2.1. File Format

The program should be able to load text files that are generated by UPPAAL according to the method described Ahmad et al. [5]. An example of such a text file is shown in figure 41. These files hold up to a certain structure, there are three main actions that can be seen, namely State, Transition and Delay. State describes the state the whole system is in, this includes the occupation of the processors, global clock time and buffer sizes. Transitions occur when a state change happens. Hereby is described whether an actor starts to fire or ends with fire[][] and end[][] respectively. Lastly, the Delay statement increases the time of the global clock.

```
State:
( SDF_Graph.Initial P_p1_0.Idle P_p1_1.Idle )
global=0 P_p1_0.x=0 P_p1_1.x=0

Transitions:
  P_p1_0.Idle->P_p1_0.InUse_getmb { 1, fire[p_id][getmb]?, x := 0 }

State:
( SDF_Graph.Initial P_p1_0.InUse_getmb P_p1_1.Idle )
global=0 P_p1_0.x=0 P_p1_1.x=0

Delay: 1667
```

figure 41 *- An example of a SDF schedule generated in UPPAAL by W. Ahmad et al. [5].*

7.3.2.2. Data Extraction Method

The structure as described in the previous section allows for multiple ways of extracting the data. So could one only look at the State statements, since they describe all changes. Or listen for Transitions, since they notify us of changes. To keep track of time there are also two options, one could use the last known global clock value of the last described state. Or one can keep track of this value by counting Delay statements.

By the implementation of the data extraction I have chosen to first compile a list of processors from the first State statement. Furthermore, the clock is initialised on the starting value of the global clock. After this task changes are registered by searching for the Transition statements. The file parser keeps a list of currently firing tasks. Information that is kept is the start time, processor and task the firing takes places on. If an task ends the end time is inserted and the task is moved from the currently active firings to the loaded schedule. The lapse of time is accounted for by keeping track of the last known value of the global clock. This method is chosen over the Delay statement since the global clock is printed more often, therefore less prone to error. In figure 42 is shown what elements of the data file are used to extract the data.

```
State:
( SDF_Graph.Initial P_p1_0.Idle P_p1_1.Idle )
global=0 P_p1_0.x=0 P_p1_1.x=0

Transitions:
  P_p1_0.Idle->P_p1_0.InUse_getmb { 1, fire[p_id][getmb]?, x := 0 }

State:
( SDF_Graph.Initial P_p1_0.InUse_getmb P_p1_1.Idle )
global=0 P_p1_0.x=0 P_p1_1.x=0

Delay: 1667
```

figure 42 *- Data that is read and saved by the file parser to reconstruct the data.*

I have chosen for this implementation since extracting task firings from the State statements would require me to compare what has changed in respect to the last known state. This would require me to keep the complete last known state in memory and furthermore also compare the whole state to the new one for changes. Using transitions on the other hand already notifies us on what processor the change occurs and also what tasks change. Thus providing a more lightweight file loader.

7.3.2.3. File Parser Implementation

The file parser is written in C#, like all components of this visualisation tool. The data file is loaded using a stream reader. A streamreader reads a file one line at a time. The current line that is being read is checked whether it matches one of our extraction points. This is done with regular expressions. Regular expressions are a sequence of characters that define a search pattern, generally used for pattern matching with strings. In Table 6 the regular expressions used in the file loader can be seen.

Statement	Regular Expression
Time	`global=([+-]?\d+(\.\d+)?([eE][+-]?\d+)?)`
State	`^\(  (.*)  \)`
Processor	`(P_\w*).(\w*)`
Task Firing	`(P_\w*).\w*->P_\w*.\w* { 1, fire\[p_id\]\[(\w*)\]`
Task ending	`(P_\w*).\w*->P_\w*.\w* { \w* == \d*, end\[p_id\]\[(\w*)\]`

Table 6 *- Regular expressions used by the file parser to find matches and extract data.*

The file reader checks if the current line is a match to one of the patterns, if so it applies it. Thus extracting the data. Depending on what match was found the file parser updates its data. For example if a clock update is found the clock is updated. Or if a new firing occurs, this firing is added to the currently active firings. Furthermore, if we have not encountered a task by this name before a new task is added to our list of tasks. Every Time a new task is observed, a new colour is generated for this task.

7.3.2.4. Task Colour Assignment

Colours are of great importance in the visualisation, they are used to encode tasks in the different visualisations. Since the number of tasks that are loaded into the file are not fixed, we do not know how many colours we might actually need. There are various solutions that could be deployed to tackle this problem.

One method could be a big list of colours could be preselected and the program (semi)randomly selects colours from this list during runtime. A downside of this is that our list has to be so big that it is always bigger than the number of actors a user will load in, and furthermore all these colours have to be handpicked.

Another method could involve random colour selection, or in other words: let the program pick colours. The difficulty for this method is to generate colours that are both visually appealing and easily distinguishable since they encode data. There are various algorithms that can be used for random colour generation.

The most basic one is just randomly generating RGB values. Of course these colours are hardly ever visually appealing, and also does not fulfill our second requirement. If colours are randomly generated in HSV space they do get more visually appealing, though the second issue is not addressed yet.

The solution to this issue is randomly generating HSV values with the use of the golden ratio. By generating colours in HSV space we get some grip on making colours visually appealing. Whilst on the otherhand the colours are evenly spaced from each other. Since one of the interesting properties of the golden ratio, is that each subsequent hash value divides the interval into which it falls according to the golden ratio. The distribution of values generated between zero and one for both random number generator and using the golden ratio as spacing can be seen in figure 43. A summary of all discussed methods can be found in Table 7. This is based on an article desribed in [35]

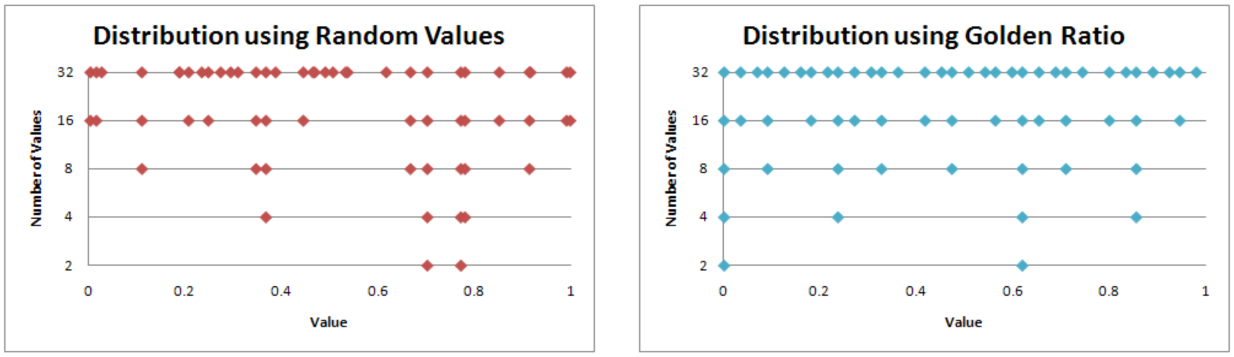


figure 43 *- Distribution of values generated between 0 and 1 using random and golden ratio [35].*

	Endless	**Visually Pleasing**	**Distinctable**	**Sample**
Premade list	x	v	~	🟥🟥🟧🟨🟩🟩🟦🟦
Random RGB	v	x	x	⬜🟩🟦🟩🟪🟦🟪🟨
Random HSV	v	v	x	🟦🟦🟩🟪🟦🟪🟪🟪
Golden ratio	v	v	v	🟪🟩🟪🟦🟨🟥🟦🟧

Table 7 *- Characteristics of different colour selecting methods, plus an sample.*

7.3.3. Data Storage

In order to preserve all information extracted from the data file three lists of data are being kept, one with processors, one with actors and a schedule of how the actors fire on the processors. These are lists of the classes Processor, Task and TaskFirings respectively. Classes are being used to keep all related information together, and allow for additional functions to be added to the class. These objects are stored as classes rather than structs because they are copied and accessed quite a lot. Classes are passed on as parameters by references, whereas structs are copied, which takes significantly more time. Instantiating new classes on the otherhand is more expensive, though this is only done when a new file is loaded.

The data is stored in lists so that language integrated query (LINQ) component can be used. This is part of Microsoft .NET Framework and adds native querying capabilities to .NET languages. What this means is that the data lists can be queried much like SQL queries are used for databases.

7.3.3.1. Actor Class

This class contains all information about the actpr that occur in the loaded data. Furthermore, it contains some functions that allow changes to this task to happen in a centralised place. A simplified version of the code can be seen in figure 44. The actor class stores the names of the actor, whether the actor is enabled and stores information of the colour. The difference between name and displayName is the fact that name is the unique identifier or key for the actor and displayName only relates to the text displayed in the user interface. Furthermore, the class contains functions that can be called to change the parameters, upon calling these functions events are also triggered in the EventManager that allow other elements that are subscribed to this event to update themselves.

```
public class Actor {
       public string name;         // Unique name for this actor
       public string displayName;  // Display name for this actor
       public bool enabled { get; private set; }  // Enabled state of this actor

        // Returns the display colour of this actor
       public Color colour {
              get {
                            if(enabled)   return _baseColour;
                            else          return _hideColour;
              }
       }

       // Constructor
       public Actor(string name, Color color) {}

       // Changes enabled state of this actor to 'b' and triggers all related events.
       public void SetEnabled(bool b) {   }

       // Changes the name of this actor and triggers all related events
       public void ChangeName() {  }
}
```

figure 44 - Simplified version of actor class.

7.3.3.2. Processor Class

The processor class has much resemblance to the task class only in this case relevant information and functionality is kept related to the processors. Like the task class it has the name, displayName and enabled parameters. On top of this it also has an id, when the processors are created the id forms an unique identifier which is incremental from zero. For some views the id is used as identifier instead of the name. A simplified version of this class can be found in figure 45.

```
public class Processor {
       public int id;              // Unique id of this processor
       public string name;         // Name of this processor in the data
       public string displayName;  // Displayname of processor within too

       // The shown/hidden status of this processor
       public bool enabled { get; private set; }

       public Processor(int id, string name) {   }

       // Changes the shown/hidden status to 'b' and triggers related events.
       public void SetEnabled(bool b) { }
}
```

figure 45 - Simplified version of processor class.

7.3.3.3. ActorFiring Class

The ActorFiring class is the glue between processors and actors. It describes how an actor fires on a processor. Therefore each firing needs to have a reference to both a processor and actor. Furthermore, since firings occur over time, they need to keep track of a start time and an end time, thus got parameters for this. The duration parameter has been added for easy access to this value, as opposed to having to access both start and end time and calculate it on the spot. A compact version of the TaskFiring class can be seen in figure 46.

```
public class ActorFiring {
       public Actor actor;                 // The actor that fires
       public Processor processor;         // The processor that the actor fires on
       public float start;                 // Start time of the firing
       public float end;                   // End time of the firing

        // Total duration of the firing
        public float duration {
              get {
                     if(_duration == 0)
                            _duration = end - start;
                     return _duration;
              }
              set {
                     _duration = value;
              }
       }

       private float _duration;

       public ActorFiring(Actor task, Processor processor, float start, float end){ }
}
```

figure 46 - Simplified version of the TaskFiring class.

7.3.4. Clock

The play function is an important aspect of presenting the data in an interactive way. The mechanism powering this is the clock. This class implements the singleton pattern but also has some static parameters. This provides high accessibility of the clock. The clock provides other pieces of code with crucial information such as the current time, maximum time and progress as a value between zero and one. Furthermore, since the time is handled in one place the integrity of the clock is checked, to make sure the time is always between bounds of the data. Changes to time should always occur through this class. When they occur the class communicates these updates to all displays through the event system.

7.3.5. Modal Windows

One way to decouple the program more is the implementation of modal windows. Beside the different views which work independently with the data, in certain cases input of the user is required. However, these input fields should not be available constantly since this would occupy screen space whilst not providing much utility to the user most of the time. A pop-up window could offer a solution, since it only occupies screen space when a response of the user is expected. It is a challenge to make these windows modular. In order for them to be modular, the modal windows should not be coupled to the place they are called form nor to the place they return to, when the input is completed.

This problem even gets more challenging if we are not exactly sure where this input is requested from, or where we need to go afterwards. One of the places this problem is encountered is with error messages or when an user wants to change the colour of an actor. In these cases a new dialogue is brought up notifying the user of the error or requested the user to enter a colour.

The solution implemented to solve these issues are modal windows. To make the modal windows available from anywhere they make use of the singleton pattern. When these windows are called they require input information such as text that will be displayed, what buttons to give the user. Furthermore, an important detail is that actions are passed on to these buttons. This ensures that the program continues when the user is done with the model window. In other words the program can continue. By implementing modal windows like this has a number of benefits:

- Modal windows can be called from anywhere in code
- The modal window can be open for an undetermined period
- When finished with the modal window normal work is resumed
- The modal window is decoupled from the previous action but also from the one that follows

The modal window system has been implemented for error messages, colour selector and name changer. So if one would like to implement a new colour selector, lets say a colour picker. Only the modal window has to be changed to support this functionality. Upon completing the selecting process the program will continue and just request the Color from the selection window.

7.3.6. Visualisation Displays

This section will go over the technical implementation of the different data displays. These displays are discussed in the following order:

- Gantt Display
- Time Control
- Processor Display
- Legend Display

7.3.6.1. Gantt Display

This view is constructed out of a number of rows presenting processors or tasks. Each task is constructed out of several blocks. The graph itself is larger than the display it is shown in, this scroll effect is achieved by using an alpha mask in the viewport for this display. An alpha mask provides us with the control to make an area transparent, thus displaying the underlying object, in this case the underlying object is the gantt chart. The moving effect can be achieved by moving the horizontal position of the graph as a whole.

Childed to the graph are objects for each row, which subsequently have the blocks childed to them. The rows serve as a container element for the blocks, providing one place to deactivate all blocks at the same time. Furthermore, each row takes up the same portion of the height since they expand in the same manner.

The siblings of an object (parent) are positioned according to their sibling index, e.g. first child gets first position and last child gets last position. So, if objects are childed to the row in a chronological manner when they are instantiated, they automatically get displayed in the right order. The width of the blocks are set according to their duration when instantiated. An example of the parent-child structure can be seen in figure 47.

```
➢ Viewport
    ➢ Graph
        ➢ Processor Row
            ➢ Actor 1
            ➢ Idle
            ➢ Actor 2
        ➢ Processor Row
            ➢ Actor 1
            ➢ Actor 2
        ➢ Processor Row
            ➢ Idle
```

figure 47 *- Parent-child structure used to generate Gantt chart.*

As a result of these two design decisions, the graph is fairly cheap on resources during run time. Only one object is moved when the play function is used, the children move with it since their position is relative to the parent. Furthermore, rows can be easily hidden from view, because when an object gets disabled all its children are also disabled. The downside of this method on the other hand is that all the objects have to be created when a file is successfully load. Also, if a file was loaded in memory a lot of objects are tagged for garbage collection, since they are discarded to make place for the new data. This could cause spikes in resource usage, however this is expected to outweigh the delay that would otherwise be imposed during the play function the graph. Furthermore, no notable spikes have been observed during any of the tests or user tests.

7.3.6.2. Time Controls

This view is mainly constructed out of user input elements such as buttons and input fields. The implementation of these elements is fairly simplistic. However an interesting element is the timebar. The time indicator is positioned to the bar according to the relative clock time. Furthermore, users can start dragging the indicator to adjust the time. This in turn updates the clock.

7.3.6.3. Processor Display

This view is initialised in a similar fashion as the gantt chart. For every core that is found in a data file a processor block is instantiated. In these blocks is an indicator for processor usage. This indicator can move relatively to its parent.

However one big difference compared to the Gantt display, is that this display updates real time. Due to the fact that users can jump forward, backwards and above all else drag time around it becomes much harder to predict changes the display should make. Thus it is not possible to create a static list of colours to move through. So instead a much more on-the-go approach is taken. When an event related to time occurs, e.g. play is pressed, time indicator is moved, this display first updates to the current tasks. This is done by recalculating what tasks are currently active and displaying those. Furthermore, if the play function is active a new function call is scheduled to update the colour when the next change should occur. If an event occurs that changes this duration such as a jump backwards, this call is recalculated. On the other hand if nothing happens and the scheduled task fires, we simply calculate and schedule when the next change should occur. Because we schedule the next expected change we do not have to check every frame whether it is time to update the display.

7.3.6.4. Legend Display

These views also depend on the hierarchical structure of parents and children. The headers are always the first sibling of a parent, after this the data rows follow. The header has a component attached to it that allows it to influence siblings, such as hiding them.

Each row has buttons attached to it, rather than influencing the row, the buttons change the data directly. The data elements update all registers event when an change has occurred to itself. So this in turn influences the row, but also all instances of this data are updated in all other views. An example would be when the name of an actor is changed. When the name is entered the display name of this task is changed in the data. The task itself triggers an event at the Event Manager for this specific task. Which in turn notifies all registered listeners for this event. The search filters that can be applied work in a similar fashion. They directly search in the data, as opposed to the legend itself. When text is entered in the filter box, the display names and key names of the data are searched for matches, and updates are made accordingly.

8. Evaluation

This chapter assesses and evaluates the success of product in the realisation phase. The product is considered a success if it brings utility and value to its stakeholders. This is done in two ways. First of all, user tests have been conducted with the envisioned users. These interviews first address the current prototype but also aims to bring forth points for future improvement or expansion. Secondly, validation is done by checking whether all design requirements are met.

8.1. User Testing

In order to validate the build prototype user test have been conducted with potential users. The prototype that was used in the user testing contained all the features that have been discussed, with the exception of the changes that were made as a result from the user testing. Furthermore, the user tests of Timon ter Braak, Philip Hölzenspies and Bart Theelen did not include graphical UI elements. In total four user tests have been conducted. The same people have participated as the ones that participated in the initial user interviews in the specification phase. These people are Robert de Groote, Philip Holzenspies, Bart Theelen and Timon ter Braak. The user test first aimed to make the participant familiar with the tool, and provide an overview of its functionality. This was achieved by providing a list of assignments participants had to try to complete, such as reading out data displays and navigating to a certain point. This was followed up by questions and a discussion on the tool, this can be found in appendix II. The aim of this discussion was to assess the tool itself and it functions, but also to address points of improvement and future work.

8.1.1. Interviewees

Here the four people that have participated in the user tests are introduced in more detail. These user tests are ordered chronologically. Most participants use SDF in a different context, except for Robert de Groote and Philip Hölzenspies. Their background and (previous) usage of SDF are rather similar.

- Bart Theelen works for TNO in Eindhoven. He has also worked with SDF graphs in the past and on top of this worked on the development of a visualisation tool called TRACE.
- Timon ter Braak works for Recore Systems, a company located in the Kennispark neighbouring the University of Twente. Recore Systems designs heterogeneous many core processor systems. Timon ter Braak encounters SDF graphs on a daily basis during his work.
- The interview with Philip Hölzenspies through was conducted through Skype. In the past Philip Hölzenspies also has done research into SDF graphs at the CAES group at the University of Twente. Currently, he is located in London working for Facebook.
- Robert de Groote currently holds a postdoc position at the CAES group at the University of Twente. He has worked for over five years with SDF graphs for his PhD thesis on this topic.

8.1.2 Results of User Testing

The aim of the user testing was to assess the tool itself, but also to find points for future improvement. Therefore first the tool itself is assessed, various issues and points of improvement arose during the user tests. After this the results are presented of what users thought should be the future direction of the tool. Lastly, a summary is presented of the overall thoughts of the participants.

8.1.2.1. Tool Assessment

The overall feedback collected on the implemented functionality of the tool was positive. Bart Theelen describes the UI as intuitive. Philip Hölzenspies states that he would be able to learn how to use the tool by playing around with it, without getting frustrated. Thereby, mentioning that the tool is clear and intuitive. Timon ter Braak states that the tool generates pleasant visualisations. Robert de Groote was pleased with the aesthetics of the application and approved the visualisations. However, users have also addressed various points of improvements. These will be presented chronologically, so first the feedback on the file loading process will be presented. Followed by the feedback on the various displays of the tool.

The first screen that pops up when the program is launched is the startup configuration screen. This screen allows users to tweak various settings such as screen resolution and display settings. Bart Theelen thinks this is an unnecessary step and can become annoying if the program has to be used frequently. After this, users were asked to load a file into the program. This was done through the file loader dialogue, here users have to enter the file location of the data file in a text field. Some users had issues with this workflow. Bart Theelen finds the file loader rather primitive. He and other users expected to be able to select files, as is common in many applications. Furthermore, Robert de Groote attempted to drag and drop the data file on the visualisation tool. Also after entering the location of a file, users pressed enter to proceed. This, however, only stopped editing the text field. Users still had to press the load button with the mouse. Even though the file loader was inconvenient for the users, all participants managed to independently load the files into the tool.

Straight after entering the location of the file a new window is brought up. Asking the user for some load parameters, to visualise the data. It was difficult for users to precisely understand what the values did and what should be entered. Bart Theelen, Philip Hölzenspies and Timon ter Braak all suggested to automate this, or make it clearer what these values exactly do or should be.

The assessment of the legend was overall positive. Robert found it advantageous that the legend for actors and processors are separated. Moreover, he was pleased with the eye symbols used to show and hide elements. Also the random colour algorithm performed as desired according to all participants. In all the user test participants considered them to be clearly distinguishable. Notably, Philip Holzenspies is colour blind but could clearly distinguish between the different colours. Furthermore, the participants thought generated colours looked good. One minor comment that Timon ter Braak was that he expects the filter boxes to be bigger, since they offer a lot of functionality. One major point for improvement that could be made in this section is the colour selector. Currently, this is done by entering the RGB values. First of all, Philip Hölzenspies mentions that the program does not tell the user whether these values are expected as floating point numbers (0-1) or as a byte (0-255). Furthermore, Bart Theelen suggest a colour selector would be much better as opposed to entering raw RGB values.

The time control section was clear to use. However, Timon ter Braak and Robert de Groote think it could increase the ease of use if more controls are given to the user. In the prototype that was used for the user testing one method to control time was to drag the time indicator. Timon ter Braak suggests to extend this by allow users to click on the bar itself to set the time. Robert de Groote additionally suggests to provide dragging function in the gantt chart, and use right click to activate/deactivate actors. Both suggestions do not conflict and would provide more control to the user. During the interview Timon ter Braak brought up another point in regard to the time controls. He thinks the way the time bar and gantt chart scroll is confusing, he describes this as graphically independent. When the time indicator gets dragged over the time bar, the gantt chart scrolls with it. Depending on the scale of the gantt chart, the speed at which either one scrolls does not relate to each other. This is needed since the width of both displays are not equal. However, Timon ter Braak thinks that a scrollbar approach would cause less confusion, where the width of the time indicator is dependent on the width of the Gantt schedule. I tend to disagree with using a scroll bar over the indicator itself. Since scrollbars are used for windows, not scrolling to a specific moment such as a video file. Some displays act as windows, such as the gantt chart. Whereas others, such as the processor display show a moment. Changing to a scroll would cause confusion for the processor display.

No major points of improvement were suggested in the user tests for the Gantt display. The users were generally pretty satisfied with this display. Robert de Groote was fond of the scaling feature this display offered. As were Timon ter Braak and Bart Theelen, but they questioned why the scaling was limited, they would have liked to zoom out even more. Currently, this is limited to 10% of the original and 190% of the original. One weird event did happen. Robert de Groote got confused at one point because the Gantt chart did not scale from the center of the screen, and therefore moved during scaling. However, he was not able to replicate this event.

For the processor views users needed some explanation as to what the grey bar meant. Timon ter Braak found this unpleasant. He thinks the utilisation bar in the processor view is not intuitive, since he had to ask what it meant. On the other side, Philip Hölzenspies states in his interview that this is okay, generally speaking it is common to have the top represent occupied (100%), and the bottom idle (0%), which is the case in the application. Furthermore, Philip Hölzenspies expresses that in his opinion it should be okay to have to learn how the tool works, as long as this provides value to the user afterwards.

8.1.2.2. Future Direction

The main direction participants would like to see the tool move towards is the visualisation of the dependencies between actors. Currently only the result is displayed, namely the schedule. However, according to Robert de Groot, Philip Hölzenspies and Timon ter Braak the most value would be in understand the decisions that were made to generate the schedule. Robert de Groote sees the tool as a first step. Also Philip Hölzenspies and Timon ter Braak would like to see this functionality added to the tool. Future work could further include the dependencies into the visualisations, furthermore, also find a way to load this information into the program.

Another major direction Robert de Groote and Philip Hölzenspies advocate is closer integration with the analysis tool. As mentioned before the tool now visualises the outcome, which is the schedule. However, when this is presented, one might wonder how things would be if an extra processor is added. If the tool could directly communicate this with the analysis tool and present the result in the visualisation tool, the workflow will be shortened and simplified. Since instead of having to generate two schedules and comparing those, the visualisation tool now will display the changes without ever having to leave the tool itself. This is another feature that would provide users with strong methods to get insight in a schedules behaviour.

Beside these major directions, participants also pointed out some additions to the current features that could make the current features more powerful. On of these suggestions, is an export function for graphs. Users already have options to control the colours and labels of the graphs. Therefore an export function would be a good addition according to Bart Theelen and Philip Hölzenspies. The exported images can then be used for the documentation of these SDF schedules in for example papers or presentations.

Another function that Philip Hölzenspies suggest is to make a change to the processor view. Currently this is a 1st order view, namely one can see what tasks are active at that point and the utilisation percentage thus far. However, Philip Hölzenspies would like to see this display in a higher order, for example a graph display the utilisation percentage over time.

Bart Theelen expects a wider variety of input options, this would make the tool easier to use for more analysis methods. The current file format that is loaded into the program is specific to the method proposed by W. Ahmad et al. [5] and its tool usage. Also the file format has a lot of information written in twofold. Therefore, it would be somewhat burdensome to adjust to this format from another method according to Bart Theelen. Furthermore, Timon ter Braak questions the scalability for extremely large schedules, since the data file is structured rather inefficient.

8.1.2.3. Summary of User Tests

All interviewees state that the tool has potential, however, some would not use it in its current state. Bart Theelen argues that the file format is not user friendly enough when it comes to interoperability with arbitrary tools and scalability of this format is also questionable. The file format is too difficult to produce from other tools than SDF models in UPPAAL, therefore a stand-alone version of TRACE might seem favorable, since it is less strict in the format of the data file.
Timon ter Braak also does not think he would use the tool in its current context. The tool offers great options for context visualisations. However, in his work he is particularly interested in the relations between tasks.

Philip Hölzenspies on the other hand would use the tool, especially if the graphs are getting bigger. Small examples can be understood by a person, though bigger ones get too complex. Philip Hölzenspies used to create ASCII art to work these out. However, this tool automates the process and provides a much better view. He also states that there might be similar tools out there but he never used those tools in his analysis. Therefore this tool might still be valuable.

Robert de Groote also sees good use for the tool. Within five minutes of loading a file he already started to address small points of interest he would like to find out more about. Also the scaling of the gantt display is very useful, currently Robert de Groote does not use any visual tools. He states that the tool allows for rapid overview of schedules and for quick search of unexpected situations.

8.2. Requirement Analysis

In section 5.2.1. Requirements a list of requirements has been established the product should fulfill. This list has been established by means of user interviews and a state of the art analysis of SDF visualisation tools. All requirements that were set have been met. Each requirement will be shortly states to what extent they have been met. A more detailed explanation about the requirements and on how they are met can be found in appendix III.

- Anyone with familiarity to computers are able to load data into the program.
- Anyone can obtain the program on the internet
- Anyone can launch the program on his or hers computer
- The Gantt display holds up to the standards in the field of SDF
- All text is in English, no spelling or grammar mistakes have been found
- Users can (re)load data files as often as they please
- The program can load SDF schedules from data files
- Users have various options to control the time that is displayed
- Users did not feel overwhelmed by the program during user testing
- Users can extent upon the program since the source is available
- No breaking failures or crashes have been experienced
- The program loads data files within a matter of seconds
- The program was able to successfully load a file with a size of 5mb
- The program loads data exactly as it is described in the data file
- Visualisation displays are synchronised
- Filter on data are applied instantly
- The program does not require an internet connection
- The program does not damage the original data file
- The program can load data files in the format as discussed in section 2.2.2.
- The program runs on Windows, IOS and Linux
- The program supports a minimum resolution of 1024x768

8.3. Conclusion of Evaluation

The goal of this chapter was to assess and evaluate the success of built in the realisation phase. This has been tested in two ways. Firstly, the product has been tested whether it meets the design requirements. From this analysis it is determined that all these requirements have been met.

Secondly, user tests have been conducted with potential users. The user test have brought forth various points of improvement to increase the user friendliness of the product. Many of comments have resulted in changes of the program, such as:

- The enter key can now be pressed to load a file
- The time bar can be clicked to set the time to this point
- The file load options have been automated
- Scaling limits can now be changed in settings

Besides, minor suggestions on current features, users also have suggested some major features that could improve the program. For instance users would like to see a closer integration with the analysis tool and have displays that provide more insight into the dependencies within the schedule. This would add an enormous amount of value to the program.

Not all participants would adopt the tool in their current workflow, there are various concerns for this. Bart Theelen thinks the file format that is used to load the data in poses to much of a hassle. Timon ter Braak does think the tool offers good context visualisations, however is not certain if it provides the insight he needs. If the tool on the other hand were to display the dependencies, he claims he would most certainly adopt the tool. Robert de Groote and Philip Hölzenspies, do think the tool already provides value in its display. They do still agree with Timon ter Braak, if the tool were to include dependency visualisation and closer integration with the analysis tool, the tool would be likely to be very important. In conclusion, the tool can be considered as a step in the right direction.

9. Discussion

This chapter provides the conclusion of the project as a whole. In this conclusion the research questions are answered. Furthermore, a future work section discusses work that can be done to extend upon this project.

9.1. Conclusion

Despite state of the art scheduling methods very little visualisations options are available for people who work with SDF graphs. This is the main reason they employ very little visualisations in their analysis, even though they desire to have tools capable of generating visualisations. SDF Fish can bridge this gap, providing these people with a visualisation tool for their analysis of SDF schedules. The main objective of this project was to create a visualisation tool that visualises SDF schedules. This poses the core question for this project:

- *How can we ease the understanding and interpretation of SDF schedules by visual means?*

Other particular questions that rise because of this question are:

- *What information from these schedules are relevant for the user?*
- *What interaction with this data is useful for the user?*
- *What alternatives in visualisations are possible for these schedules?*

In order to provide an answer to the question on what information is relevant within the schedules interviews have been conducted with potential users. These interviews have shown that the main interest of users is to optimise schedules. Therefore any information related to this is of importance. So in first regard it is important for users to understand when and how actors fire. On top of that it is important to understand how the schedule is formed. This is mainly the result of the dependencies between actors. Since one actor might have to wait on an other actor in order to fire. These dependencies however are not directly visible in the schedule, but do form the constraints the schedule is bounded to.

The interaction users desire with the data mainly aims to make the data manageable. The interviews with users have shown that as schedules become bigger they get harder to oversee. Interaction between the user and the visualisation will allow users to focus on the areas of their interest. This is realised in the tool by zoom, filter, scale and show/hide features in SDF Fish.

Various visualisation types for SDF schedules are presented over the course of this report. Ultimately, the widely used Gantt-chart visualisations have been chosen to be used for the visualisation tool. Users are already familiar with the Gantt type of visualisation. Furthermore, the literature review did not provide much reason to change away from this data visualisation, since it mostly evolves around position.

To ease the process of understanding and interpretation of SDF schedules the SDF Fish tool has been devised over the course of this project. Key features of this tool include:

- Ability to load SDF schedules from a text file
- Gantt chart style visualisation of SDF schedule
- Snapshot display of a moment within the schedule
- Zoom, filter, scale abilities for the different visualisations
- Play function for schedules to see changes occur over time
- Wide variety of time controls to navigate through the file

The ability to load schedules is perhaps the most important feature. Because of this, the tool is able to generate visualisations for any SDF schedule. Thus providing insight into SDF schedules in general rather than one specific schedule, what would be the case if the visualisation data would have been static.

To ease the understanding and interpretation of the schedules itself the tool makes use of various visualisations. In the current workflow of users for the interpretation of SDF schedules they hardly make use of visualisations. However, visualisations do exist for SDF schedules, they are mostly used for documentation. A common visualisation of SDF schedules are Gantt-like charts, the tool offers to automatically generate such a visualisation.

Also the product was evaluated by user tests and verifying the design requirements. All design requirements are met by the tool. Also the user tests have addressed various issues that have been improved. Furthermore, users see the tool as a step in the right direction. The generated visualisations are useful for the interpretation and analysis of SDF schedules. However, even more value would be presented if the dependencies would also be visualised and a closer integration with the analysis tool.

9.2. Future Work

The work delivered in this project should form the basis for future work. From the start the prototype has been build with the idea of future extensibility. The evaluation also support this idea, the interviewed people in the user tests see this tool as a step in the right direction. They have also indicated various areas in which the tool can be expanded. For instance more visualisations could be added. Furthermore, the data format could be broadened, so that it becomes easier to generate files that can be loaded into the program, but also formats that provide more information to the program could be added. One crucial point that is addressed multiple times over the interviews is the desire to see the dependencies between actors. Currently, this information is simply not available since this data is not present in the data files. In order to add visualisations that can display the dependencies this information also needs to present in the data file itself.

Additionally, closer integration with the analysis tool would be ideal. This allows users to quickly interact with not only the data, but also how the schedule is constructed. Users would be able to quickly explore the effects of adding more resources or changing other characteristics of the program. Currently, for each of these scenarios a new schedule needs to generated and loaded into the program. Especially Robert de Groote and Philip Hölzenspies advocate the need of such a tool.

To provide users with the ability to extent upon the product. The source code is made freely available an open source license. Also, all code has been documented and the workings plus design practices are explained in this document.

Acknowledgements

First of all, I would like to thank my two supervisors, W. Ahmad and Dr. M.I.A. Stoelinga for their time and guidance over the course of this project. Without this guidance I am convinced there would have been various points at which I got too focused on details and lost track of the bigger picture. You drew my attention back to the bigger picture, which has led to the completion of this project.

Also I would like to thank Robert de Groote, Bart Theelen, Timon ter Braak and Philip Hölzenspies for participating twice the interviews for this project. These interviews provided me with insight of situation and the relevancy. Furthermore, discussing this topic with you supplied me with a source of motivation.

References

[1] Microsoft Corporation (2016, July 11). *Skype* [Online]. Available: https://www.skype.com/en/.

[2] E. A. Lee and D. G. Messerschmitt, "*Synchronous data flow: Describing signal processing algorithm for parallel computation*," in *COMPCON '87*, 1987, vol. 79, no. 20 pp. 310-315.

[3] S. Stuijk, "*Predictable mapping of Streaming Applications on Multiprocessors*," PhD thesis, 2007.

[4] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. J. G. Bekooij, B. D. Theelen and M. R. Mousavi, "*Throughput analysis of synchronous data flow graphs,*" in ACSD, 2006, pp. 25-34.

[5] W. Ahmad, R. de Groote, P. K. F. Holzenspies, M. Stoelinga and J. van de Pol, "*Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata*," in ACSD, 2015, pp. 72-81.

[6] G. Behrmann, A. David and K. G. Larsen, "*A tutorial on UPPAAL*," in Formal Methods for the Design of Real-Time Systems: 4th International School on SFM-RT, LNCS, 2004, pp. 200-236.

[7] S. Bhattacharyya, P. Murthy, and E. Lee, "*Software Synthesis from Dataflow Graphs*," Kluwer Academic Publishers, 1996.

[8] E. Lee and D. Messerschmitt, "*Static scheduling of synchronous data flow programs for digital signal processing*," in IEEE Transactions on Computers, 1987, vol. 36 m pp. 24–35.

[9] E. de Groote, J. Kuper, H. J. Broersma and G. J. M. Smit, "*Max-plus algebraic throughput analysis of synchronous data flow graphs*," in 38th EUROMICRO conference on SEAA, 2012, pp.29-38.

[10] R. Karp, "*A characterization of the minimum cycle mean in a digraph*," in Discrete Mathematics, 1978, vol. 23, pp. 309-311.

[11] TNO (2016, July 11). *TRACE* [Online]. Available: http://trace.esi.nl/.

[12] M. McGavin, T. Wright, S. Marshall, "*Visualisations of Execution Traces(VET): An Interactive Plugin-Based Visualisation Tool*," in AUIC '06 Proceedings of the 7th Australasian User interface conference, 2006, Vol. 50, pp. 153-160.

[13] S. K. Card, J.D. Mackinlay, B, Shneiderman, “*Readings in information visualization : using vision to think*,” Morgan Kaufmann Publishers, 1999.

[14] S. Stuijk, M. Geilen and T. Basten, “*SDF^{3}: SDF For Free*,“ in ACSD, 2006.

[15] E. Gansner and S. North, “*An open graph visualization system and its applications to software engineering* ,“ in Software: practice and experience, 2000, vol 30, pp. 1203–1233.

[16] P. K. Robertson, “*A methodology for scientific data visualisation: choosing representations based on a natural scene paradigm*, “ in Proceedings of the First IEEE Conference on Visualization, 1990, vol. 1, pp. 114-123.

[17] W. S. Cleveland and R, McGill, “*Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods*,” in Journal of American Statistical Association, 1984, vol. 79, no. 387 pp. 531-554.

[18] J. Mackinlay, *“Automating the Design of Graphical Presentations of Relational Information*,” in ACM transactions on Graphics, New York, NY, 1986, vol. 5, no. 2, pp. 110-141.

[19] J. Bertin, “*Properties of graphic system*,” *Semiology of graphics: diagrams, networks, maps*, 1st ed. Wisconsin, Wi, University of Wisconsin Press, 1983, pp 41-64.

[20] M. S. T. Carpendale, “*Considering Visual Variables as a Basis for Information Visualisation*,” in PRISM, 2003, pp. 173-193.

[21] E. R. Tufte, “*Theory of data graphics*,” The Visual Display of Quantitative Information, 2nd ed. Cheshire, CT: Graphics Press**,** 1983, pp. 178-184.

[22] H. Li and N. Moacdieh, “*Is “chartjunk” useful? An extended examination of visual embellishment*,” in Proc. of the Human Factors and Ergonomics Society 58th Ann. Meeting, 2014, vol. 58 no. 1 pp. 1516-1520.

[23] M. A. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister, “*What Makes a Visualization Memorable?*,” IEEE Trans. Visual. Comput. Graphics IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 12, pp. 2306–2315, 2013.

[24] S. Bateman, R. L. Mandryk, C. Gutwin, A. Genest, D. Mcdine, and C. Brooks, “*Useful junk*?,” Proceedings of the 28th international conference on Human factors in computing systems - *CHI '10,* 2010, pp. 2573-2582.

[25] O. Inbar, N. Tractinsky, and J. Meyer, *“Minimalism in information visualization: attitudes towards maximizing the data-ink ratio*,” Proceedings of the 14th European conference on Cognitive ergonomics invent! explore! - *ECCE '07,* 2007, pp. 185-188.

[26] D. J. Gillan and D. Sorensen, “*Minimalism and the Syntax of Graphs: II. Effects of Graph Backgrounds on Visual Search*,” Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 2009, vol. 53, no. 17, pp. 1096–1100.

[27] A. J. Blasio and A. M. Bisantz, “*A comparison of the effects of data–ink ratio on performance with dynamic displays in a monitoring task*,” International Journal of Industrial Ergonomics, 2002, vol. 30, no. 2, pp. 89–101.

[28] S. K. Card and J. Mackinlay, “*The Structure of the Information Visualization Design Space*,” in *IEEE Symp. on Information Visualization*, Phoenix, AZ, 1997, pp. 92-99.

[29] B. Schneiderman, “*The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*,” in Proc. Visual Languages, Boulder, CO, 1996, pp. 336-343.

[30] M. Bar and M. Neta, “Humans Prefer Curved Visual Objects,” Psychological Science, 2006, vol. 17, no. 8, pp. 645–648.

[31] I. Daniluk (2016, January 24). *Visualizing Concurrency in Go* [Online]. Available: https://divan.github.io/posts/go_concurrency_visualize/

[32] B. Victor (2011, October*). Up and Down the Ladder of Abstraction* [Online]. Available: http://worrydream.com/#!2/LadderOfAbstraction

[33] R. Nystrom, *“Game Programming Patterns*,” in Game Programming Patterns, 1st ed. United States, 2014.

[34] K. Wilhelm,"Level of *measurement*" in Encyclopedia of Public Health 2, United States, 2008. doi:10.1007/978-1-4020-5614-7_1971.

[35] M. Ankerl (2009, December 9). *How to generate random colors programmatically* [Online]. Available: http://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/

Appendix I

Interview Questions

Introduction

- Ik stel mij zelf voor
- Deelname is vrijwillig en we kunnen op elk moment stoppen met het interview. U hoeft geen antwoord te geven op de vragen.
- Heeft u er iets op tegen als ik het gesprek opneem?

Huidige Werkwijze
Wat vindt u belangrijk aan traces / SDF grafen?

Ik zou eerst graag wat willen weten over de huidige werkwijze. Ik veronderstel dat u met SDF graphs werkt en bijbehorende traces.
Wanneer komt u deze traces tegen? Hoe vaak?
Is het belangrijk dat u de inhoud van deze traces begrijpt? Waarom?
Wat is de huidige methode om deze traces te begrijpen? Hoelang bent u hiermee bezig?
Zijn er verbeterpunten aan te brengen op deze werkwijze? Zo ja waar?

SDF Trace Analysis
Wat voor informatie bent u vooral in geïnteresseerd bij het analyseren van de traces?
Het totale plaatje? Of wat er aan de gang is op een specifiek moment?

De SDF traces bevatten verschillende variable, zou u deze kunnen orderen van belangrijkste naar minst? Of is dit afhankelijk per geval?
Op een specifiek moment:
frequentie van processor,
de taak op de processor,
de duur van een taak
aantal firings van processen tot dan toe,
verhouding tussen firings van processen tot dan toe.
In totaal:
totale verhouding tussen firings van taken
totale aantal firings van taken
verhouding tussen frequenties van processen
verhouding of totale idle tijd

Visualisatie Tool
Dit is een concept versie van de UI die ik voor ogen heb, als gebruiker kun je traces in laden. Vervolgens kun je de data in deze traces ontdekken. Het idee is dat je de traces kan afspelen, vertraagd natuurlijk. Hier kan jezelf de tijd besturen. Deze sectie geeft voor elke processor core weer welke taak erop runt, of dat deze in idle mode is. Deze sectie geeft een staaf grafiek met het totale aantal firings tot dan toe. Hier kunt u zien hoe tasks encode worden in kleuren. Dit is

een Gantt-achtige grafiek van de traces. Elke processor core krijgt een rij, en over de horizontale as komen de taken te staan. De rode lijn geeft het huidige moment aan. Met deze controls kun je de tijd unit veranderen in tijd(ms) of ticks. Verder kun je de Gantt chart zoomen om een groter of kleiner overzicht te krijgen.

In het voorbeeld kunt u de SDF grafiek zelf niet zien, is het belangrijk om de SDF grafiek te zien waar de trace uit komt?

Denkt u dat door het afspelen van de traces een handige functie is? Leidt dit tot een beter begrip van de traces?

Is het handig om de tijdlijn in zowel ticks als ms te hebben?

Denkt u dat dit een verbetering zou kunnen zijn ten opzichte van het huidige process?

Is er vraag naar tool zoals dit? Brengt het waarde naar het publiek? Of mensen in het technische veld?

Wat voor formaat zouden de bestanden hebben die de tool zou moeten in laden?

Outro

- Heeft u nog vragen of opmerkingen?
- Dankjewel voor medewerking
- Interesse in resultaten?
- Bereid mee te werken aan user testing / feedback of prototypes?

Interview Robert

Huidige Werkwijze

Wat vindt u belangrijk aan traces / SDF grafen?

Mijn hele proefschrift gaat over SDF dus tijdens mijn promotie heb ik vijf en een half jaar aan SDF gewerkt, en dus de analyse daarvan. SDF graven stellen een berekening voor, met SDF kan je dus een grafische weergave van het algoritme maken, dus je ziet zeg maar hoe de data loopt.

En wat ik heb gedaan is gekeken naar het tijdsgedrag, want we zijn vaak geïnteresseerd hoe vaak z’n berekeningen uitgevoerd wordt en daar heb je dus heel veel analyses voor. En nu wil ik daar dus mee verder gaan, ik wil dus met een SDF graaf meer berekeningen kunnen visualiseren en inzicht geven in hoe berekeningen aangepast kunnen worden en dat soort dingetjes. Het tijds gedrag is heel belangrijk en dat komt vooral naar voren in die schedules.

Tijd is belangrijk en dat je gelijk een bottleneck ziet. Je wilt kunnen inzoomen op bepaalde stukken van de SDF graaf. Bepaalde stukken voor de SDF graaf zijn bepalend voor de hele SDF graaf. Het stuk wat het langzaamst loopt vormt een bottleneck voor de rest, en die wil je zo snel mogelijk kunnen vinden.

Wat is de huidige methode om deze traces te begrijpen? Hoelang bent u hiermee bezig?

Tekstueel, ik heb een python tool geschreven, en ik kan een graaf construeren in een JSON bestand. Het is niks visueel en blijft daardoor ook vrij klein. En via analyses krijg je dan, tekstueel, een output vanwaar de kritieke punten zijn. Maar dat is wel beperkt, je kunt niet even snel zien wat er gebeurt als ik dit(*lees iets)* aanpas. De traces zelf kijk ik niet heel erg naar, het programma wijst de bottlenecks aan.

Zijn er verbeterpunten aan te brengen op deze werkwijze? Zo ja waar?

Ja, zeker weten, een SDF stelt een berekening voor en die berekening wordt bijvoorbeeld uitgevoerd op processoren - reken elementen - en als je het aantal reken elementen gaat beperken dan moet je een keuze maken van wat je gaat uitvoeren op een bepaald reken element, en die keuze is cruciaal voor de schedules die je krijgt. En als je dus een zo snel mogelijk schedule wilt hebben dan wil je graag een beetje kunnen spelen met dingen; als je een taak verplaatst van dit reken element naar dat reken element, wat voor invloed heeft dat? Visueel zou je dit heel snel kunnen zien, als je een schedule hebt, wil je heel snel inzicht kunnen krijgen in hoe bepaalde keuzes invloed hebben.

SDF Trace Analysis

De SDF traces bevatten verschillende variable, zou u deze kunnen orderen van belangrijkste naar minst? Of is dit afhankelijk per geval?

Vooral de taak op een processor is belangrijk voor mij, denk ik. Het aantal firings van processen tot dan toe is denk ik minder belangrijk voor mij. De verhouding is ook niet heel erg belangrijk. Wat ik hier wel ook bij zou zeggen is als je dus een taak op een processor draait. Oh die staat hier volgens mij, wat is de idle tijd van een processor. Dus per processor hoe vaak die eigenlijk stil staat zeg maar, dus dan kan je processoren die minder werk hebben eigenlijk er tussen uitpikken, dus daar geef je dan meer taken aan. Ik denk dat de firings van taken zelf niet heel erg in geïnteresseerd ben. Het is vooral lokaal meer op processor gericht denk ik. Dus op een specifiek moment zou ik de taak van de processor en frequentie misschien ook wel interessant maar dit is dan een heel ander gebied wat je er bij betrekt eigenlijk. Dus het zullen vooral de eerste 3 zijn. De andere twee niet zo heel erg. En in totaal de idle tijd vooral en deze is misschien ook nog wel leuk. Maar die zijn ook gerelateerd aan elkaar.

Visualisatie Tool

*Dit is een concept versie van de UI die ik voor ogen heb, als gebruiker kun je traces in laden. Vervolgens kun je de data in deze traces ontdekken. Het idee is dat je de traces kan afspelen, vertraagd natuurlijk. Hier kan jezelf de tijd besturen. Deze sectie geeft voor elke processor core weer welke taak erop runt, of dat deze in idle mode is. Deze sectie geeft een staaf grafiek met het totale aantal firings tot dan toe. Hier kunt u zien hoe tasks encode worden in kleuren. Dit is een Gantt achtige grafiek van de traces. Elke processor core krijgt een rij, en over de horizontale as komen de taken te staan. De rode lijn geeft het huidige moment aan. Met deze controls kun je de tijd unit veranderen in tijd(ms) of ticks. Verder kun je de Gantt chart zoomen om een groter of kleiner overzicht te krijge*n.

In het voorbeeld kunt u de SDF grafiek zelf niet zien, is het belangrijk om de SDF grafiek te zien waar de trace uit komt?

Ja ik zelf wel, uit de structuur van z'n graaf kan je best wel veel dingen halen.

Denkt u dat door het afspelen van de traces een handige functie is? Leidt dit tot een beter begrip van de traces?

Nee ik denk dat het afspelen wel nuttig is, maar het liefst dan met de graaf zelf erbij. Dat je ook zien welke taken in de graaf actief zijn. Nu geef je de kleuren aan de processoren maar als je dit ook in de SDF graaf kan doen. Er zelf door heen lopen is misschien minder belangrijk dan het afspelen.

Zou in een optimaal scenario het programma in de visualisatie de bottleneck aangeven? Uhmm ja, precies, maar misschien is het moeilijker om die bottleneck uit het schedule te halen dan uit de SDF graaf zelf vermoed ik.

Is het handig om de tijdlijn in zowel ticks als ms te hebben?

Dat maakt niet zoveel uit, vaak hebben taken een bepaalde tijdsduur in een bepaalde eenheid. Wat deze eenheid is maakt niet uit. Het gaat vooral om iets relatiefs.

Denkt u dat dit een verbetering zou kunnen zijn ten opzichte van het huidige process?

Ja, ja, zeker weten. Omdat het dus allemaal tekstueel is, is het dus heel erg beperkt. Vaak werk je op een dag met een graafje en dan maak je kleine wijzigingen omdat je beperkt bent in wat je kan.

Is er vraag naar tool zoals dit? Brengt het waarde naar het publiek? Of mensen in het technische veld?

Ja ik denk vooral in het onderwijs dat het wat minder stoffig wordt allemaal, als je gewoon dingen kunt laten zien dan geeft het veel sneller een beeld. Ik denk zeker dat het voor het onderwijs handig is.

Wat voor formaat zouden de bestanden hebben die de tool zou moeten in laden?

Het zou het handigst zijn denk ik, maar ik weet niet of dit is wat jij ook wil gaan doen is. Kijk nu heb je dus een trace die je inlaad. En als je een wijziging maakt in de SDF graaf op basis van die trace moet je opnieuw die trace generen en opnieuw inladen en dat proces kun je inkorten door direct een SDF graaf in te kunnen laden, dus bijvoorbeeld een JSON of XML die een SDF graaf beschrijft en dan zou je vervolgens via een interface met een tool, de SDF graaf aan een tool kunnen geven die de trace genereert en vervolgens weer terug geeft. Dus dat je het proces wat inkort. Dan kan je gewoon gebruik maken van bestaande analyses. Maar dan heb je dus ook de informatie van de SDF graaf.

Heeft u nog andere ideeën voor visualisaties?

Op de een of de andere manier hoe druk een processor is of hoe goed deze gebruikt wordt. Dat zou je misschien met een bepaalde kleur kunnen aangeven. Hier zie je bijvoorbeeld dat P1 en P2 de helft van de tijd op idle staan en dat kan je misschien op een bepaalde manier visualiseren. Toch een soort van bottleneck is zoals je hier naar kijkt, deze twee a’s niet kunnen beginnen omdat die b nog bezig is. Dus er is waarschijnlijk een afhankelijkheid tussen die a’s en b. Dit is dus waarschijnlijk een kritiek pad. En dat kun je er dus uit halen als je er goed naar kijkt omdat het aantal processoren nu vrij beperkt is maar als dat er heel veel zouden zijn wordt het al heel lastig. Dus wat je vaak wilt begrijpen van die traces is waarom deze twee a’tjes niet al hier kunnen beginnen en die reden daarvoor wil je eigenlijk zichtbaar kunnen maken. De SDF graaf zelf heeft hier heel veel informatie voor. Ik begrijp dat deze informatie echter niet in de trace file zit.

Ik denk dat de graven voor mij een hele sterke toevoeging zouden zijn. Dat je de structuur van hetgeen dat je aan het mappen bent kan zien. Dat is heel inzichtelijk. Als je de graaf hebt, stelt het een deel voor waar die trace uit komt, het andere deel is hoe de mapping van op de processoren plaats vindt en die kun je ook weergeven. Dat geeft gewoon heel veel inzicht over bepaalde taken die niet niet actief gemaakt kunnen worden. Ik denk dat als je die erbij hebt , de trace en de graaf zelf, dan heb je gewoon een heel compleet beeld.

Trace tool bespreking:

Grappig, ik heb hem van de week voor het eerst gezien. Hierbij heb ik eigenlijk hetzelfde, dat ik de graaf hierbij mis. Dus je krijgt heel veel informatie maar je kan die informatie niet meer relateren aan iets.

En sowieso als je de graaf kunt visualiseren maar misschien ook een graaf zou kunnen maken, dan wordt het wel een ander soort tool denk ik, maar zou wel echt iets zijn wat ik zou gaan gebruiken. Als je graven zou kunnen maken in een klik omgeving.

Hoe groot zijn de graven waar u mee werkt?

Nou.. 7-8 nodes denk ik.

Interview Timon ter Braak

Huidige Werkwijze

Wat vindt u belangrijk aan traces / SDF grafen?

Je hebt daar meerdere gelaagd heden of modellen in. Dus wij werken in eigenlijk met applicaties, nog specifieker aan de real-time laag. Dit is de laag onder applicaties die er voor zorgt dat applicaties goed kunnen executeren. Nou wil je daar wat coördineren en doen dan moet je dus weten hoe die applicatie eruit ziet. Dus daar maken we modellen van, dat doen we ook eigenlijk onderwater, want de user roept de API aan.Onder water in die API hebben wij een model of een structuur hoe die applicatie eruit ziet. Noem dit een applicatie graaf of een taak graaf, dat zijn taakjes die met elkaar communiceren. En dat is nog een redelijk hoog niveau. Wil je daar een model van maken wat formeel is en je meer wilt zeggen over het gedrag van een applicatie als je dat wilt analyseren, dan kom je bij dataflow uit. SDF is een van de varianten daarvan. Dus we hebben eigenlijk altijd een vertaling van het applicatie model naar SDF en weer terug. Wij zijn vooral geïnteresseerd in hoe die applicatie draait, en daar kunnen we SDF voor gebruiken, maar ook andere dingen. Het is interessant hoe de scheduling zit en hoofdzakelijk als je naar de multi-many core platformen gaat, is vooral niet alleen de scheduling van losse taken belangrijk. Waar je nu veel in andere gebieden ziet is dus dat je een collectie aan taken heb en daar zit een belangrijke volgorde afhankelijk in, dit kun je modelleren naar SDF. Dus de scheduling vraag van hoe kan ik taken goed schedulen dat de taken dependencies goed zijn. Als taken vanuit elkaar communiceren, dus dan heb je eigenlijk een volgorde afhankelijk tussen taak a en b en deze hebben een communicatie kanaal nodig en een buffering in dit kanaal. De buffering bepaald in welke mate die twee aan elkaar gerelateerd zijn. Hoe groter die buffer is hoe losser ze van elkaar zitten. Precies dat fenomeen kunnen we altijd heel veel mee spelen, in specifiek hoe je buffer grote kiest. Die keuzes zie je direct terug in het schedule. Dat soort dingen is waar je inzicht in kunt krijgen. Als je all the way gaat dan kun je echt analyse methoden doen en gebruiken om SDF modellen van je applicatie volledig correct is om allemaal door te rekenen en te zien hoe bepaalde keuzes en settings gedaan moeten zijn maar vaak, vooral in onze huidige situatie kiest de gebruiker gewoon een bepaalde waarde en andersom als je een schedule kan visualiseren en je ziet daar gaten in; waarom kan dit niet strakker gescheduled worden, dan kan je terug vinden waar dit aan ligt. Hoofdzakelijk denk ik dus dat er twee misschien wel drie dingen die ik belangrijk vind aan schedules visualiseren is **(1.)** dat je het kan relateren aan een applicatie, die gerelateerd is zodat je inzicht krijgt in hoe je applicatie nou daadwerkelijk executeert op een platform. **(2.)** Je wilt schedules gebruiken om mogelijk inefficiënties of problemen te identificeren, en als laatste **(3.)** optimalisatie, het ziet er goed uit maar je kan plekken zoeken voor verbetering. Misschien een klein beetje krap door de bocht, maar je hebt een applicatie model, als dat executeert wordt dit omgezet naar een SDF graaf.

Bij deze optimalisatie ben je vooral geïnteresseerd in de dependencies tussen dingen. Je kan bijvoorbeeld wel de processor utilisation optimaliseren, maar waar je uiteindelijk in geinteresseert bent is dat de applicatie goed draait en dit mag best betekenen dat de processor soms op idle staat.

Wat is de huidige methode om deze traces te begrijpen? Hoelang bent u hiermee bezig?

Die zijn allemaal nog in prototype fase. Een daarvan wat veel gebruikt wordt is om een executie trace te visualiseren, en dat wordt vaak gedaan door een soort Gantt chart. Zoiets zie je vaak, maar hier is het grote probleem dat je de afhankelijkheden tussen de taken niet ziet. Je kan dat plaatje maken vanuit het perspectief van de cores. Dat vind ik zelf het minst interessant want dat zegt eigenlijk alleen maar iets over de cores. Zelf vind het ik het perspectief vanuit de applicatie interessanter, met de taken op de verticale lijn. En dan zie je wanneer ze geenabled en disabled worden. Daar zit de lastigheid in dat je daar graag de relatie tussen taken wilt zien. Als een taak stopt met executeren en iets anders begint kan dat compleet ongerelateerd zijn, maar het kan ook zijn dat ze data communiceren. Die relaties dat is eigenlijk het meest interessante wat je wilt zien. Alleen het is lastig omdat goed en clean in een 2D omgeving weer te gaven. Dus iets anders wat ik een tijd geleden gezien heb. Dat hebben wij zelf niet ontwikkelt, voor de GO programmeertaal, dat zijn concunrrency execution units. Die met elkaar communiceren en relateren. En daar kun je dus ook gewoon een executie trace uit halen en dat visualiseren. Daar hebben ze een mooie 3D omgeving van gemaakt. Dat is misschien een verrijking aan het inzicht wat er eigenlijk gewenst is aan inzet van visualisaties. De huidige werkwijze; wij hebben zelf een omgeving van 3D visualisatie, maar dat is meer voor demo doeleindes als wij een applicatie draaien op ons platform is dat een blackbox, waarvan de klant geen idee heeft wat er gebeurt. Dus daar hebben we mooie visualisaties om te laten zien wat er gebeurt. En dat is heel leuk, maar voor development en analyses is dat niet zo geschikt.

SDF Trace Analysis

Het totale plaatje? Of wat er aan de gang is op een specifiek moment?

Dat laatste (op een specifiek moment), want ik kan mijn applicatie draaien en aan het einde zien hoelang dit geduurd heeft. Dus deze informatie hebben we eigenlijk al. En ook processor utilisatie kan je ook makkelijk plotten, maar wat hier goed in is de vraag. Daar kan je niet zoveel over zeggen. Terwijl specifieke momenten, kunnen je helpen om er nog meer uit de applicatie te persen. Dan ga je kijken waar de gaten zijn, en waar je nog kan optimaliseren. Worden dingen wel goed gescheduled? Dat zijn allemaal vragen die naar boven komen als je aan het debuggen bent en aan het optimaliseren.

De SDF traces bevatten verschillende variable, zou u deze kunnen orderen van belangrijkste naar minst? Of is dit afhankelijk per geval?

De taak op een processor is belangrijk, maar dat is misschien zelfs secundair, stel je hebt een bepaalde taak en die stopt met executeren. Dan ga je kijken op welke processor draait die. Maar eigenlijk is het al heel interessant om een schedule van de applicatie te zien om te kijken welke taak wanneer actief is. Dat is interessant om al te weten zonder eigenlijk te hoeven weten op welke processor de taak draait. Want het is pas als je zegt dat; hier zit iets geks, of hier zie ik mogelijkheden om iets te optimaliseren, dan wil je ook kijken op welke cores draait het nou eigenlijk en waarom komt dat. Dus die informatie is wel belangrijk. Maar wat het low hanging fruit is om te visualiseren als eerste optie zeg maar hoe de taken op elke processor draait. Dat is leuk en toegevoegde waarden. Maar als je nog een niveau hoger wilt is dit secundair. Omdat je eerst het gedrag van de applicatie wil zien en wil je daar verder in duiken dan ga je naar die mapping of task assignment kijken.

De duur van de taak is ook relevant. En dat is relevant omdat er heel veel mogelijk oorzaken zijn waarom een taak een bepaalde tijdsduur. Dus dat kan zijn de pure computatie tijd die nodig is. Dat kan je vergelijken met wat je verwacht of is alles snel maar een bepaalde taak is langzaam. Dan ga je daar als dat een bottleneck is kijken of je daar de code kan optimaliseren. Maar de duur van een taak heeft ook direct te maken met dependencies. En daar zie je ook een verschil in een SDF als model en de werkelijke en dat verschilt over welke werkelijk je hebt. Als ik dat specifiek toegepast op waar wij mee werken dan heb je een applicatie met een taak a en die communiceert over een buffer met taak b. En stel dat er ook nog een taak c die ook wat naar b toestuurt. In SDF is het zo dat B pas mag beginnen als er voldoende input data op alle edges zijn. Maar wat in werkelijkheid kan gebeuren en bij ons het geval is, dat b nooit tegelijkertijd de data uit beide queues kan halen. Hij zal altijd eentje eerst ophalen en vervolgens de andere. Dus wat je vaak ziet is dat b geactiveerd wordt om te zien of er data beschikbaar is. Zo niet dat stop ik weer, en daarna misschien is de data van c beschikbaar.

Je hebt een applicatie, daar maak je een SDF model van, en een trace van de executie zelf. Ik verwacht van de visualisatie dat hij het schedule van SDF kan laten zien. Dat is wat je in de worst case kan analyseren of synthetiseren. Maar je zou dezelfde methode gewoon het schedule van de daadwerkelijke executie kunnen laten zien. Die ziet er misschien iets anders uit maar die ziet er in wezen op hetzelfde uit.

Visualisatie Tool

Dit is een concept versie van de UI die ik voor ogen heb, als gebruiker kun je traces in laden. Vervolgens kun je de data in deze traces ontdekken. Het idee is dat je de traces kan afspelen, vertraagd natuurlijk. Hier kan jezelf de tijd besturen. Deze sectie geeft voor elke processor core weer welke taak erop runt, of dat deze in idle mode is. Deze sectie geeft een staaf grafiek met het totale aantal firings tot dan toe. Hier kunt u zien hoe tasks encode worden in kleuren. Dit is een Gantt achtige grafiek van de traces. Elke processor core krijgt een rij, en over de horizontale as komen de taken te staan. De rode lijn geeft het huidige moment aan. Met deze controls kun je de tijd unit veranderen in tijd(ms) of ticks. Verder kun je de Gantt chart zoomen om een groter of kleiner overzicht te krijgen.

In het voorbeeld kunt u de SDF grafiek zelf niet zien, is het belangrijk om de SDF grafiek te zien waar de trace uit komt?

Ik zelf vind van niet, het gaat erom wat die SDF modelleerd, dus als je SDF actoren een op een zou vertalen naar taken. Dan zit je SDF model eigenlijk al in dat een bepaalde taak meerdere keren vuurt en terug komt in je schedule. En dat is niet zoveel van toegevoegde waarde om daar een SDF model van te zien. Wat relevanter is dat je relaties tussen actors ziet en ik denk dat dat erg belangrijk is. Ik denk niet dat je die informatie krijgt door een SDF grafiek te laten zien.

Zou interactieve SDF graaf helpen?

Misschien wel maar dat is natuurlijk direct een lastig argument om mee te beginnen. Je zou dat natuurlijk kunnen proberen, maar dat gaat stuk zodra je gaat opschalen. Zelf ben ik altijd bezig geweest met het mappen van taken naar cores. Daar heb ik inmiddels wel een vijftal visualisaties van gemaakt om te zien wat er gebeurt. En als je ook maar een beetje gaat opschalen, gaat van alles stuk. Je kan niet meer laten zien wat je wil zien in het aantal pixels. Dus je komt eigenlijk altijd terecht in semantics of zooming of selectie wil doen op de data die je ziet. Als dit er vijftig of honderd zijn dan kan je niet meer al de hele SDF graaf visualiseren. Als je bijvoorbeeld dit veld(lees:Gantt Chart) ziet zou ik verwachten dat je een search of selectie veld hebt om te kijken wat er gebeurt. Een user is ook niet dom dus die weet of heeft op een moment opgezocht dat taak A en B van elkaar afhangen. Door selectie zou hij zich dan al beter kunnen richten op deze twee taken als hij hier in geinstereest is.

Hoeveel Actors / taken werkt u zowel mee?

Wij zijn nog bezig met het ontwikkelen van ons hardware platform, in de versie waar we alle cores ingevult hebt, zijn dat al 160 cores. Ik heb in een research project van een paar jaar oud ook al applicaties van 100+ taken. Dus daarom zeg ik het is heel leuk om visualisaties te maken voor beperkte groottes, maar ik denk dat de kracht zit dat in een selectie mechanismes om in te kunnen zoomen. Het is dus de uitdaging om de data managable te houden.

Denkt u dat door het afspelen van de traces een handige functie is? Leidt dit tot een beter begrip van de traces?

Je wilt juist ook over tijd informatie kunnen zien. Dus ja het is een handige functie, maar de vraag is of het exact zo zou moeten. Want als je SDF schedules visualiseert en als die periodiek zijn, dan zou je kunnen nadenken dat je alleen de periodieke fase laat zien. Maar juist als je een daadwerkelijke executie trace zou laten zien dan wil je wel over tijd gaan zien. Ik weet niet zozeer of het afspelen zelf daadwerkelijk nodig is. Het ziet er natuurlijk leuk uit, maar ik zou zelf meer verwachten dat je een scrolbar hebt, waarbij de cursor mee gaat. Vaak zal een user zelf scrollen naar het tijd moment waarop hij geïnteresseerd is.

Is het handig om de tijdlijn in zowel ticks als ms te hebben?

Ik denk dat je dat moet kunnen zien of opvragen. Maar zou niet permanent in beeld moeten zijn. Stel dat je een heterogeen systeem hebt, dan heb je cores die op verschillende clock frequenties draait. Als je output uit je platform over hoe dat geëxecuteerd hebt, trace, vanuit die trace krijg je data. Dit moet getimestampt worden. En dan zal je zien dat je de tijd unit nodig hebben om deze twee traces te kunnen laten zien. Dus ik denk dat je onderwater die tijd unit nodig hebt om te visualiseren. En ik kan me voorstellen dat als je ergens op klikt dat je de tijd unit kan zien.

Ik zou de tijd niet op de horizontale tijd verwachten. De nummers zouden veel te groot zijn, en je bent toch geïnteresseerd in relatieve tijd. Dus als je het al wil laten zien moet je een scaling factor toevoegen. Dus dat zegt al niet zoveel. Ik zou dus misschien het tijd aspect een beetje weghalen, en als je op iets klikt meer informatie over de duur van dat blokje krijgen.

Denkt u dat dit een verbetering zou kunnen zijn ten opzichte van het huidige process?
Dat hangt af van de kwaliteit van jou werk. We hebben momenteel een software development environment die we nog aan het ontwikkelen zijn. Wat je hier al wel kan is het assignen van taken naar cores, en daar wordt dan onderwater alle compile en synthese stappen al gedaan. Maar wat we nog wel eens gehad hebben maar nu weer uitgesloopt hebben omdat het niet mature genoeg was, was het visualiseren van een trace. Dus ja we hebben dit nodig, nee we hebben niet exact een idee hoe dit zou moeten. En dat draagt zeker bij.

Bent u vaak geinteresseerd in de frequentie?
Om heel eerlijk te zijn, momenteel niet want we developen onze technologie in FPA dus we hebben geen frequency scaling. Als we daar een ASICS van zouden maken dan zie je in ons gebiedje ook niet direct dat we heel erg voltage frequency scaling doen. Het vorige bord wat door recore systems gemaakt is. Kun je wel clock frequencies zetten maar die worden gewoon vast gezet op een waarde. En daar wordt verder niet aan geschaald. Deels heeft dat een rede omdat je niet weet wat goede scaling zijn, een andere reden is vaak dat de cores niet individueel geschaald kunnen worden. Onze chips zijn heel anders dan wat jij gewent bent van de dikke intel CPUs. Een kwart zo groot als die van intel op een kwart van de grootte 10 cores. Hier heb je niet voor elke core aparte power management of kun je clock scaling doen. Dus stel dat we het hebben en we het doen dan is het al snel voor meerdere cores.

Is er vraag naar tool zoals dit? Brengt het waarde naar het publiek? Of mensen in het technische veld?
Als het helemaal perfect is zouden we dit willen inbouwen in onze software development environment. Waarin gebruikers hun applicatie programmeren, die klikken op een knopje en laden een platform met het resultaat wat gevisualiseerd wordt. Dus het heeft waarde voor zowel interne developers als de eindgebruiker die hun eigen applicatie programmeren. Om te zien of ze het goed gedaan hebben en verbeteringen kunnen maken.

Wat voor formaat zouden de bestanden hebben die de tool zou moeten in laden?
Wat we momenteel hebben is een binairy trace formaat en een parser die het naar verschillende formaten kan uitspugen. Dus het is relevanter welke informatie daarin moet komen, en dan kunnen wij dat wel parsen naar het formaat dat nodig.

Interview Philip Hölzenspies

Huidige Werkwijze
Wat vindt u belangrijk aan traces / SDF grafen?
Ik denk dat je een aantal verschillende niveaus hebt waarin een visualisatie interessant is. Het eerste is natuurlijk het schedule zelf. Ik heb een graaf en daar komt een schedule uit, die blokjes diagram. Dat is dus een standaard visualisatie van een schedule. Dat

gebruiken we vaak in papers maar als je dat interactief hebt kan je er misschien meer mee doen. Interactie zou handig zijn bij het debuggen van een schedule. Maar als je SDF gebruikt om een performance analyse van de applicatie te doen dan zoek je bottlenecks en vraag je af hoeveel zin het heeft om een bepaalde bottleneck te verhelpen. Het is handig om te weten hoeveel problemen je oplost door het verhelpen van een bepaalde bottleneck.
Dus iets over wat ik maar zal noemen; de integraal van traces, dus niet zomaar een trace ziet, maar je het gewicht ziet tussen kritieke punten. Ik heb geen idee of dit mogelijk is. Ik denk dat dat de belangrijkste dingen zijn. Het belangrijke is dus het begrijpen en debuggen van een bepaald schedule en de andere toepassing zou ik zeggen is vervolgens het voorspellen waar vervolgens problemen gaan opkomen en te zien hoeveel zin het heeft om een bepaald iets te veranderen.

Wat is de huidige methode om deze traces te begrijpen? Hoelang bent u hiermee bezig?

Ik ben daar niet zo goed in, gelukkig ben ik redelijk goed in algebraïsch denken. Kijk ik heb er alleen maar mee gewerkt in de onderzoekscontext. Dus dan werk je meer vanuit het analyse probleem waaruit je ziet waar de analyse misloopt. Dit probeer je dan tot een zo klein mogelijk voorbeeld terug te brengen. Vervolgens tekenen we dit op een whiteboard. Deze hele kleine scenarios te krijgen is vooral handwerk. Robert de Groote heeft dus een point en click tool om SDF graafjes te tekenen. Maar daarna schalen doet het niet zomaar, en het werkelijke tunen van een applicaties komen toch andere dingen bij kijken die niet intuïtief zijn. Laat ik zeggen dat zon integraal trace nuttig is, maar ik heb dat nooit zelf gedaan.

SDF Trace Analysis

Wat voor informatie bent u vooral in geinteresseerd bij het analyseren van de traces? Het totale plaatje? Of wat er aan de gang is op een specifiek moment?

Het liefst als je toch aan een visualisatie denkt, zou ik het liefst aan een temperatuur map hebben die over de tijd varieert. Dat het misschien wel zo is dat de bottleneck verschuift. Stel ik heb twee zware computational kernels, en stel die dat die aan elkaar zitten met andere dingen die ook nog moeten gebeuren. Dan wil ik graag zien dat deze delen van de graaf heel erg bezig zijn, en dat er eens in de zoveel tijd maar wat tussen door komt. Deze verbinding bestaat uit actoren die misschien een keer vuren als de twee kernels duizend keer vuren. Dan wil ik natuurlijk zien dat de twee kernels veel meer bezig zijn dan de andere. Dat kan over verloop van tijd, maar misschien is het wel handig om grafiekjes te plotten.

De SDF traces bevatten verschillende variable, zou u deze kunnen orderen van belangrijkste naar minst? Of is dit afhankelijk per geval?

Dit hangt er vanaf vanuit wat voor oogpunt je de SDF graaf wil analyseren. Enerzijds heb je de pure SDF graaf, en hier heb je het nog niet eens aan processoren gebonden, want die bestaan niet in pure SDF. Maar als je actoren op processoren gaat binden ben je vaak geïnteresseerd waarom ergens een bottleneck is, nu bereken je dit zelf door en kom je erop een moment achter dat er te veel of te weinig tokens zijn op een bepaald punt. Hiervoor zou het misschien handig zijn om de Gannt-achtige graaf te hebben met een tijdlijn waarmee je kan zien hoeveel tokens er zijn op een bepaald moment. Je zou met een tijdsbalkje door een schedule kunnen dat je de token distributie op de graaf ziet veranderen. Volgens mij is Robert al een heel eind met deze visualisatie. Maar goed als je er dan processoren bij zet, dan zijn de vragen waarom vuurt actor c niet, toch weer een andere vraag. Want dan zit je ook nog met het feit dat je niet oneindige resources

hebt. Dus de visualisaties daar zijn iets anders, dus daar zou ik wel degelijk schedules en dergelijke, frequenties interessant vinden. Ratio's van aantal afvuringen tot dan toe zijn niet zo interessant, dus als er twee actoren die met een ratio afvuren omdat voor consistente graven zijn ze toch eindig gebound, dus ze kunnen maximaal zoveel vuringen uit elkaar lopen. Die graven zijn periodiek, dat betekend dat een actor maximaal zoveel vuringen kan uitlopen, dus dat soort ratio's zijn niet zo interessant. Nu hebben we een tool die een getalletje dump op de command-line, dan weet je dat en dat is het. Wel interessant is vanuit het resource management perspectief, is wanneer een bottleneck optreedt door het resource management. Dus als heel veel actoren tegelijk enabled zijn zouden kunnen gaan vuren, dan krijg je dat resource management de grootste bottleneck wordt. En dan moet je wat gaan kiezen en welke moet je dan slim kiezen. Dus ik denk ook de maat van parallellisme, hoeveel dingen tegelijk actief zijn over verloop van tijd. Ik denk dat dat een vreselijk interessante variable is, omdat je daardoor het verschil weet tussen resources en applicatie ontwikkeling. Een goede manier om intuïtie te kunnen krijgen, is om te kunnen zien wat er gaat gebeuren als ik deze keuze anders maak. Wat er dus zou gebeuren als ik andere beslissingen maak wanneer ik een keuze moet maken tussen het schedule van taken. En de vraag is of ik dan steeds de vraag moet stellen, of dat ik dat in een keer kan zien.
Een handig artikel is worrydream. Hier gaat het over switchen tussen abstractie levels, dus soms wil je met de tijd kunnen schuiven. Maar het kan ook het geval zijn dat je alle mogelijke paden wilt kunnen zien.

Visualisatie Tool
Dit is een concept versie van de UI die ik voor ogen heb, als gebruiker kun je traces in laden. Vervolgens kun je de data in deze traces ontdekken. Het idee is dat je de traces kan afspelen, vertraagd natuurlijk. Hier kan jezelf de tijd besturen. Deze sectie geeft voor elke processor core weer welke taak erop runt, of dat deze in idle mode is. Deze sectie geeft een staaf grafiek met het totale aantal firings tot dan toe. Hier kunt u zien hoe tasks encode worden in kleuren. Dit is een Gantt achtige grafiek van de traces. Elke processor core krijgt een rij, en over de horizontale as komen de taken te staan. De rode lijn geeft het huidige moment aan. Met deze controls kun je de tijd unit veranderen in tijd(ms) of ticks. Verder kun je de Gantt chart zoomen om een groter of kleiner overzicht te krijgen.

In het voorbeeld kunt u de SDF grafiek zelf niet zien, is het belangrijk om de SDF grafiek te zien waar de trace uit komt?
Ja absoluut, in het bijzonder met highlights van wat enabled is, wat vuurt en wat zou kunnen vuren. Je ziet hier nu alleen maar de beslissing. Ik zou ook niet al te veel moeite doen om alles op een scherm te krijgen. Ik zou daar flexibel in zijn, en soms wil je dingen in meer detail. Op een gegeven moment worden SDF graven groot namelijk en dan werkt dat niet meer.

Hoeveel taken / actors bestaande graven zowel uit waar u mee werkt?
Dat hangt er dus sterk vanaf of je nou echt alleen de applicatie maakt, als je een embedded system engineer bent en je processor of video decoder bezig bent of whatever. Je bent ergens mee bezig. Dan denk je aan hooguit enkele tientallen. Als je aan de analyse werkt, waar de tool ook wel extreem nuttig voor zou kunnen zijn, dan wil je ook CSDF en HSDF naast elkaar kunnen hebben. Die HSDF graven kunnen heel groot worden, met heel groot bedoel ik duizenden. Er zijn trucjes voor om dit te doen, maar geen van allen zullen ze echt schalen om alles te kunnen visualiseren.

Denkt u dat door het afspelen van de traces een handige functie is? Leidt dit tot een beter begrip van de traces?

Zeker met de graven erbij is het nuttig om tijd te kunnen afspelen, dat geloof ik echt wel. Zonder is het tot op zekere hoogte ook wel nuttig. Ik vraag me af of het processor schermpje rechts boven in echt nuttig is. Want in principe is het al in kleurtjes gevangen in het schedule zelf. Misschien heb ik het verkeerd. Wat daar nu natuurlijk nog wel meetelt; voor de processor scheduler class in Waheeds methode zou dit handig kunnen zijn. Maar dat weet Waheed waarschijnlijk meer over. Ik zou het niet direct nuttig vinden. En als je grafiekjes tekent zoals nu, zoals de staaf diagrammen, die vind ik daar niet nuttig. Daar zou ik de trap grafiekjes over tijd doen, dit zegt alleen hoeveel er tot nu toe gezien is. Je kan dezelfde ruimte ook het patroon van de vuringen tekenen.

Is het handig om de tijdlijn in zowel ticks als ms te hebben?

Ik ben daar zeer sceptisch in. Er zijn mensen die daar filosofische een overtuiging over hebben. Voor mij zet je daar gewoon de echte wereld tijd of de grootste gemene deler van de clock frequenties, en dan is wereldtijd hetzelfde als computer tijd. Ik zie niet echt een waarde voor ticks naast echte tijd.

Denkt u dat dit een verbetering zou kunnen zijn ten opzichte van het huidige process?

Oplossing hangt natuurlijk af van wat precies het probleem is. Het zou mij wel helpen om uit te leggen aan gebruikers van SDF waarom een applicatie de performance eisen niet haalt. Of waarom het redelijk is waarom SDF het niet kan vertellen. SDF is altijd een overschatting, dus je bent pessimistisch over het probleem. Dat zou je hier wel mee kunnen zien.

Is er vraag naar tool zoals dit? Brengt het waarde naar het publiek? Of mensen in het technische veld?

Het hele grootte brede publiek moet je niet aan denken want die haakt al af bij het woord synchroon. Embedded software / system engineers, in de breedste zin des woords. Die hebben hier wel wat aan. Ik denk dat het wel een goede tool is in embedded land werken ze heel erg op verschillende niveaus om systemen te modelleren. Op bit software niveau zijn hele goede tools en op software niveau zijn hele goede tools daar tussen is het altijd een beetje afhankelijk van waar je komt. Ieder bedrijf bedenkt hun eigen tools en werken daar maar mee. Ik denk dat dit zeker wel een nuttige stap zou kunnen zijn.

Wat voor formaat zouden de bestanden hebben die de tool zou moeten in laden?

Nou idealiter hetzelfde formaat als de analyse tool die Robert gebouwd. Het mooiste zou zijn als dat een beetje code hergebruikt. Het liefst zie ik die dingen strak aan elkaar verbonden. Want als hij dan wat veranderd krijg je niet gelijk het probleem dat vier maanden later iemand een mailtje stuurt, dat de visualisatie tool niet werkt met analyse tool.

Zelf nog vragen?

Kijk bij dit soort leuke ideeën krijg ik altijd potentieel nog leukere ideeën. Het mooiste zou natuurlijk zijn als ik echt interactief kan kijken waar ik wat kan veranderen, dus stel dat hier een graaf bij staat getekend. En ik kom nu aan een bepaalde throughput. Als het mij nou lukt om a zoveel te versnellen. Het zou mooi zijn als ik die knop pak, de executie tijd terug draai en gelijk het schedule plaatje zie veranderen. Maar waarom dat moeilijk is, UI technisch is dat niet zo moeilijk. Het is natuurlijk moeilijk omdat voor een tikje weinig hoeft uit te rekenen terwijl ook het hele schedule kan veranderen door een tikje. En dat is lastig. Dit zou je kunnen bereiken door de tool geïntegreerd te zien met Robert zijn tool. Zodat ze op lange termijn ook echt gebruikt gaat worden.

Interview Bart Theelen

Huidige Werkwijze

Wat vindt u belangrijk aan traces / SDF grafen?

Daar kan ik vrij kort over zijn want ondertussen is het zo dat ik me daar niet heel erg veel mee bezig houdt. Ik heb daar paar jaar geleden druk mee bezig geweest en heb het ook op verschillende plekken toegepast, maar we kwamen er wel vrij snel achter dat SDF onvoldoende is en graag dynamiek wil kunnen modelleren en vandaar dat we allemaal andere data-flow formalisme zijn tegengekomen, die dynamiek modelleren, wat SDF dus niet doet. En verder SDF al gebruikt in de context in modelleren van applicaties die op multiprocessor architecturen draaien en waarbij het erom gaat dat je dus het potentiële parallellisme identificeert in z'n data-flow graaf. Daarvoor uitermate geschikt blijkt te zijn. Maar als je kijkt naar het modelleren van het complete systeem dan je zie je dat dat data-flow te weinig expressiviteit heeft om allemaal andere mogelijkheden die je nodig hebt te kunnen modelleren. Dus het is vooral geconcentreert op het applicatie stuk. En daar passen natuurlijk Gantt charts, dus schedule graphs dus natuurlijk heel erg goed bij.

Over visualisaties bij SDF graven zijn er eigenlijk twee varianten, de meer traditionele SDF graaf met bijvoorbeeld in Microsoft projects, maar dat is eigenlijk ook een Gannt chart viewer. Daar kijk je eigenlijk vanuit de activiteiten en dan kijk je waar en wanneer die plaats vinden. Terwijl voor de context waarin we nu kijken eigenlijk vooral ook interessant wanneer wel of niet de resource met een bepaalde activiteit bezig is. En dat resource aspect dat is voor laten we zeggen voor mensen in de industrie eigenlijk veel interessanter tegenover het meer traditionele view zoals zon MS project en anders soort projecten met Gantt achtige tools.

Dus u bent al vrij snel in uw process van SDF afgestapt en overgestapt op andere formalismes?

Uuh ja, dat was ongeveer twee minuten. We zijn dus overgestapt op Cyclo-static data-flow.

Wat is de huidige methode om deze traces te begrijpen? Hoelang bent u hiermee bezig?

Ik gebruik inderdaad onder andere TRACE om de activiteiten base te visualiseren en de modellen die daar achter zijn soms data-flow maar soms ook niet. Ik gebruik ook heel veel andere dingen. Maar TRACE is wel een redelijk voorbeeld van ongeveer van wat zinvol en handig is in deze context. Het kan natuurlijk wel een stuk beter. Maar het geeft ongeveer wel aan wat er verwacht wordt qua functionaliteit, dus qua toeters en bellen zeg maar. Er is trouwens ook nog een andere tool, ik weet eigenlijk niet of je het een tool kunt noemen. Op de universiteit van Eindhoven waar data-flow ook heel uitgebreid wordt bestudeerd hebben ze ook een tool gemaakt op basis van XML wat je met een web viewer kunt bekijken. Het doet eigenlijk hetzelfde soort dingen, het laat dus ook activiteiten op resources zien. Die activiteiten zijn eigenlijk dan de bolletjes die data-flow graafen dan en laten zien wanneer gebeurt wat en op welke resource.

SDF Trace Analysis

Wat voor informatie bent u vooral in geïnteresseerd bij het analyseren van de traces? Het totale plaatje? Of wat er aan de gang is op een specifiek moment?

Het is heel erg afhankelijk van wat je probeert te bewerkstelligen met z'n analyse. Kijk als het gaat over bottleneck analyse dan wil je graag weten waar het probleem zit. Hopelijk helpt z'n plaatje hier dan mee om een beetje te zoeken waar je eigenlijk moet zitten. En dat is vaak het gene met het totaal overzicht, ik zie wat raars op een plek. Dan wil je daar kunnen inzoomen en kijken wat daar aan de hand is. Je wilt graag afhankelijkheden kunnen zien tussen taken. Dat gebeurt eigenlijk door een resource als gevolg wat op een andere resource, bijvoorbeeld een andere taak die nog niet kan starten. Want dat zijn zeg maar voorbeelden van informatie die je zou willen visualiseren. Andere dingen die ik me ook kan voorstellen is dat je bijvoorbeeld kunt zien of je bepaalde performances eisen wel of niet haalt, of hebt gehaald. Dat je dit bijvoorbeeld zou kunnen zien door een groene smiley als je het wel gehaald hebt of een rood kruis als het niet gehaald is. Dan wil je graag weten waarom het niet voldoet, en daar wil je dan informatie over hebben. Dit kan gegeven worden op verschillende manieren.

De SDF traces bevatten verschillende variable, zou u deze kunnen orderen van belangrijkste naar minst? Of is dit afhankelijk per geval?

Allemaal en nog veel meer. Dit soort eigenschappen wil je graag weten en in een individuele situatie weten, en graag wil je ook nog wat summary statistics over het geheel hebben. Dus een gemiddelde of een maximum van het hebt om op basis van dat soort informatie een systeem wil gaan engineering.

Dus denkt u dat de ratios tussen afvuringen ook handig zijn?

Vanuit het perspectief vanuit pure SDF is dat eigenlijk al informatie wat je in het model stopt, dus als ik het goed heb, heb je dat soort informatie al. Als je kijkt naar de wat meer dynamische modellen is dat niet fixed en stop je dat veel minder impliciet in het model en is het veel interessanter om die informatie wel te hebben.

Visualisatie Tool

Dit is een concept versie van de UI die ik voor ogen heb, als gebruiker kun je traces in laden. Vervolgens kun je de data in deze traces ontdekken. Het idee is dat je de traces kan afspelen, vertraagd natuurlijk. Hier kan jezelf de tijd besturen. Deze sectie geeft voor elke processor core weer welke taak erop runt, of dat deze in idle mode is. Deze sectie geeft een staaf grafiek met het totale aantal firings tot dan toe. Hier kunt u zien hoe tasks encode worden in kleuren. Dit is een Gantt-achtige grafiek van de traces. Elke processor core krijgt een rij, en over de horizontale as komen de taken te staan. De rode lijn geeft het huidige moment aan. Met deze controls kun je de tijd unit veranderen in tijd(ms) of ticks. Verder kun je de Gantt chart zoomen om een groter of kleiner overzicht te krijgen.

In het voorbeeld kunt u de SDF grafiek zelf niet zien, is het belangrijk om de SDF grafiek te zien waar de trace uit komt?

Gezien die ideeën die ik hier over heb, zou het antwoord ja zijn. Dat kan een zinvolle view zijn. Daarnaast is het bij SDF zo en dat geldt minder voor andere data-flow modellen. In SDF kan het zo zijn dat een actor meerdere keren op hetzelfde moment actief is. Dat is misschien iets dat je op een of andere manier op een speciale manier wilt visualiseren.

Denkt u dat door het afspelen van de traces een handige functie is? Leidt dit tot een beter begrip van de traces?

Ja, ik moet eerlijk zeggen dat ik dat niet zo direct zie. Je kijkt eigenlijk al naar een bepaalde situatie dus dat zegt eigenlijk al alles. Dus als die andere views dan daadwerkelijk wat toevoegen, dat weet ik niet zo. Dat durf ik niet zo te zeggen.

Misschien dat andere mensen wat hebben aan de andere views, maar ik denk dat vooral engineers vooral goed met dat onderste view(lees Gantt view) enorm goed uit de voeten kunnen. Of dat afspeel stuk nodig hebt om daar doorheen te scrollen, ik weet niet of dat heel veel toevoegt. Ik denk dat het wel zinvol is om daar van links naar rechts kunt scrollen, eventueel van boven naar onder als het niet past op het scherm. Deze grafen kunnen enorm groot worden en dan past het niet meer op het scherm. Dus je wilt kunnen zien wat er op elk deel van de tijd gebeurt. Ik denk dat dat afhankelijk is van de gebruiker. Ik zelf durf het niet zo heel erg precieze uitspraak over te doen.

Is het handig om de tijdlijn in zowel ticks als ms te hebben?

Daar moet ik eens even over nadenken. Meestal is het gewoon een tijdseenheid, waarbij de eenheid dan verder niet gespecificeerd is. In de data-flow modellen geef je tijd niet op, dat zou je natuurlijk wel kunnen doen. Pas op het moment dat je daar een eenheid aan toe kent, pas dan kun je dit soort zoom definineeren. Daarnaast is het als je kijkt naar cycles naar seconden dan moet je ook nog weten hoe snel de processor is. Eigenlijk moet je dan ook nog weten in hoeverre processoren in sync zijn op een bepaald moment. Het wordt in ieder geval ietsjes moeilijker. Ik kan me voorstellen dat voor praktisch gebruik om daar een tijdseenheid te kunnen specificeren. Alleen die tijdseenheid zijn bij normaal gebruik niet gespecificeerd. Dit is gewoon een eenheid in tijd. Wat dat dan is, is aan de degene die dat model heeft gesteld. In die zin vrij abstract.

Denkt u dat dit een verbetering zou kunnen zijn ten opzichte van het huidige process?

Een goede vraag. Laat ik het zo zeggen, iedereen maakt van dit soort plaatjes. Kijk maar naar de willekeurige papers en daar vind je ze gewoon. Iedereen maakt deze elke keer met het handje met een teken tool. Maar waarom heb je niet gewoon een tooltje dat dat plaatje voor je maakt. Dat zou dus deze tool kunnen zijn. In die zin zou het kunnen helpen. In de zin voor documentatie zou het kunnen helpen. Het zou ook kunnen helpen om meer inzicht te krijgen om te zien wat er gebeurt, en of dit precies is wat jij wil dat er gebeurt. Ik zie hier wel degelijk veel waarde in.

Is er vraag naar tool zoals dit? Brengt het waarde naar het publiek? Of mensen in het technische veld?

Ook voor de praktische, ik bedoel als je wil weten hoe een systeem zich gedraagt, of er ergens een bepaalde bottleneck is. Dan helpen dit soort plaatjes veel meer dan als je naar een rijtje getallen kijkt. Visueel informatie weergeven is nog altijd handiger dan een platte file met alleen maar getallen erin.

Wat voor formaat zouden de bestanden hebben die de tool zou moeten in laden?

Dan is het een beetje de vraag, wat je met de tool verder wil doen. Als je ook beschikbaar wilt stellen voor andere omgevingen, anders dan alleen maar data-flow. Dan zou ik voor iets kiezen wat in ieder geval simpel is. Wat je ook vanuit andere tool kunt generen. Dus kies een formaat wat niet te moeilijk is, wat tekst is, dus platte tekst. Niet iets gecodeerd, want het is moeilijk dan. En een andere puntje; afhankelijk van in hoeverre je daar tegen aanloopt. Zorg dat het schaalbaar wordt. Ik heb een aantal voorbeeld graven met enkele duizend taken er in. Een SDF is dan nog niet zo heel spannend omdat altijd hetzelfde is, maar als er dynamiek is kun je makkelijk een miljoen blokjes hebben. Dan wordt het toch wel knap om het enig sinds te laten performen. Dus het is eventjes de vraag wat je hier precies mee wil, en wat voor doelgroep je wilt gaan bedienen.

Als deze tool gebruikt voor SDF grafen hoeveel actoren denkt u dat er zowel gebruikt zullen worden?

> Dat is afhankelijk van of je een academicus vraagt of een industrieel persoon. Ik denk voor een academische wereld zal het modelleren in een iets van 20-25 zitten, dat is dan al veel. In de industriele context zijn het er veel meer. Ik heb voorbeelden van 20.000.

Tips:
Ik heb zelf ook wat ideeën hierover, ik ben zelf eigenlijk heel erg geïnteresseerd in de debugger. Dan zie je de graaf zelf met allerlei manieren om daar properties in te brengen. En een debugger die laat wat er gebeurt als je het model gaat executeren of het model uitvoert. Dat kun je inderdaad met een visualisatie die je hier nu voorstelt weergeven. Maar wat je dan bijvoorbeeld ook zou moeten kunnen is op het blokje a te kunnen klikken en dan terug naar de specificatie gaan en zeggen van hier moet toch iets kunnen veranderen. Want die tijd die ik daar heb opgegeven heb ik verkeerd ingevoerd. Ik snap dat het dan gelijk een stuk ingewikkelder en uitdagender wordt.

Appendix II

Users Testing Question
In the user testing folder you will find the file data.txt. This is a schedule generated by UPPAAL from the SDF graph shown in figure 1. The built visualisation tool is designed to visualise files exactly like this.

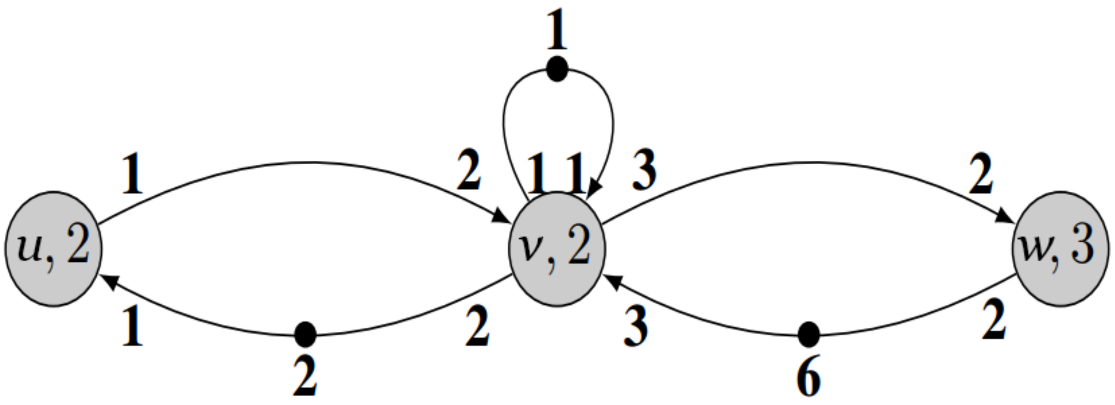
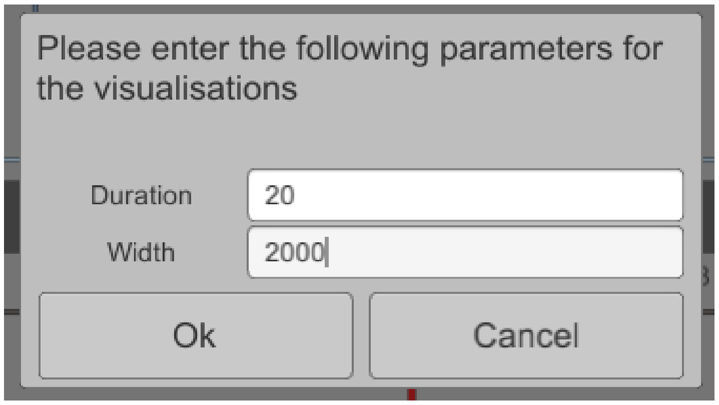


figure 1 - SDF graph used to generate schedule.

Please load this file into the program with duration set to 20 seconds and width to 2000 pixels, feel free to explore a bit.


Please enter the following parameters for the visualisations

Duration 20
Width 2000

The program is build up out of different views. Figure 2 shows where you can find the different views and what they can be used for and what interaction the user can do to control the data. Tip: in the processor filter you can use #CONTAINS-<Task> to filter out processors that only run <Task>.

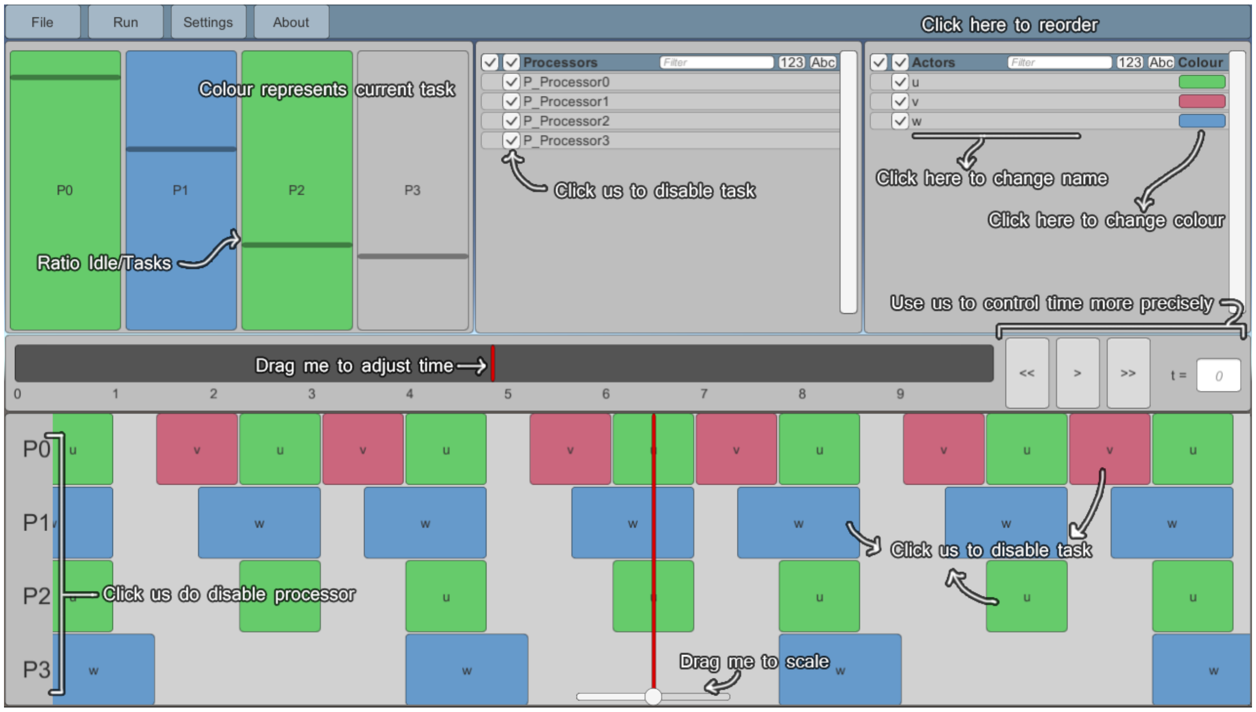


figure 2 - Overview of the visualisation tool and controls that the user has.

<u>Some questions to encourage participants to use different functions of the program</u>

Try to explore the data and give answers to the following questions:

What tasks are active at the start(0%)/ middle(50%) / end(100%) of the schedule?

Which processor is used most/least?

What processors task W runs on.

<u>Questions for after exploration of the visualisation tool</u>

When do you think the renaming and changing colours of tasks could be useful?
Did the tool behave different from what you expected at any point?
Are the visualisations clear?
Do you think the colours of different tasks are distinctive?
Are there any features that are missing?
Would you use the tool again? When?
Do you think this tool is more useful than your current tool?
Is there any functionality that overlaps?
Where you confused in any step of the process?
Is the UI intuitive?
If you could improve any aspect of my tool what would it be?

Suggestion for a name?

Appendix III

- Any user with familiarity to computers should be able to load data into the program without the need of the user manual.

  All participants in the user tests were able to load a data file into the program and thus requirement is met. Even though they did not always succeed on the first try, they managed to recover from their mistake. For example Robert de Groote attempted to drop the file into the window of the visualisation tool. This is very understandable since modern applications tend to take a drag and drop approach. However, unfortunately Unity does not support the dropping of external file into the executing 'game'.
- Any user with familiarity to computers should be able to obtain the program.

  The tool can be downloaded from any platform from www.something.com. Therefore anyone with internet access can obtain the program.
- Any user with familiarity to computers should be able to launch the program.

  Every user was able to launch the program with ease, thus this requirement is met. The program is build to an executable, thus users can launch the program like any other. Furthermore, no install is needed. This lowers the launch threshold even more.
- Any user that knows how Gantt-like chart schedules work for SDF should be able to understand this display in the tool.

  All the participants understood the Gantt chart display straight away and did not ask for further explanation. Furthermore, participants did not experience any confusing with the graph itself. Robert de Groote did indicate that the scale slider acted weird at one point. The scaling did not seem to scale from the center. However, he was unable to replicate this error. In any case the graph itself was clear to all users and therefore this requirement is also met.
- Any English speaking user needs to be able to read and understand all text in the application.

  No mistakes or typos have been addressed during the user tests. Furthermore, all text has been triple checked by me, therefore I consider this requirement to be met.
- Users should not have to restart the program to load a new data file.

  Every user test successfully loaded data files into the program, even when one was already loaded. Furthermore, this has been tested extensively by me, the developer. These tests include both broken files, files that did not hold up to the expected structure, as well as correct data files. In every case the program was able to restore and load a new file successfully if it held up to the structure used by W. Ahmad et al. [5] as discussed in … .
- The program should be able to visualise SDF schedules, this includes for any number of resources, actors and the firings of actors on resources over time.

  Theoretically, this should be possible. The program is written in such a way it adjusts according to the number of resources and actors found in the data file. However, at some point a limit is reached since the resources of a computer the program run on are not infinite. It is hard to establish exactly where this point is. The largest data file that was loaded into the program contained 25 actors over 4 processors. This data file had a size of 5mb and contained 448 firings. The data was displayed correctly, however, the play capabilities of the program started to become less responsive.

- Users need to be able to scroll through time variable, most importantly for displays that are not abstracted over time.
  Users have various options to change the time variable. First of all, users can drag the time indicator. Secondly, users can click in the time bar to set the time to this point. Thirdly, users can use the play, forward and backward buttons. Lastly, users can enter a specific value for time, or even a percentage.
- Users should not feel overwhelmed when using the program.
  None of the participants indicated to be overwhelmed during the user tests.
- Users should be able to extent upon the program.
  Anyone with the programming skill will be able to exent upon the program, since the source for the program is available.
- The mean time between failures should larger than five minutes.
  Every user test took at least 20 minutes during which the prototype was being used. No major failures were encountered during any of them, some minor errors were encount, such as, a slider not scaling with the resize of the window. These error were very minor and generally fixed the same day they were encountered.
- The mean time to recovery should not be bigger than one minute.
  No crashes have been experienced during the user testing or in one of the builds. If one were to happen a person should be able to recover from this within a minute. Since only the program has to be launched and the data needs to be reloaded.
- The program should not freeze upon loading a data file.
  Loading a file can cause the program to freeze temporary if a big data file is selected. However, in all the user tests and also in my own tests the loading was always so fast that this was never noticed. These freezes during loading are not permanent in any case. Therefore this requirement is met.
- The program should load a file within one minute.
  All data files that have been loaded into the program to test it were loaded within a matter of a second. The largest file this has been tested for is 5mb.
- When a data file is loaded and not recognized the user should be notified.
  When a file is not found the user is notified. Also when the format is not recognised.
- The program should be able to load a file of at least 5mb.
  The tool has been tested for a data file with the size of 5mb. The program was able to successfully load and display this into the program. The Gantt display did not have any issues with playing this. The processor view did show some issues. Due to the fact this display is updated real-time and the large data set, at one point the display seemed to skip a task if their duration was relatively short compared to the other actors.
- The program should not impose any roundings to the data that is loaded into the program.
  All data is loaded exactly as it is described in the data file. This has been tested by verifying the displays with the data files, for various data files.
- All displays should be in sync with each other.
  All displays are in sync with each other, even though they do not rely on each other. Only in extreme scenarios, with large data files (5mb) and relatively small actors a small delay between processor view and actor view could be noted.
- The program should not contain any spelling mistakes.
  The program has been triple checked for spelling mistakes. Furthermore, no spelling mistakes were encountered during the user tests.
- The program should respond within a second when the user filters data

The user can apply various filters within the legend display. These filters are already applied when the user is still typing, thus these filters are applied instantly.

- The program should not require an internet connection.
  The program does not require an internet connection for any of its functionality.
- The program should not damage the original data file in general or in the event of a crash.
  No such occurrences have been encountered during the development nor during the user tests. Also the file is only being read, therefore it is unlikely that the file gets damaged if a crash occurs.
- The program is able to load schedules from a text file in the format proposed by W. Ahmad.
  All the files that are currently used for testing and have been used for user tests are files generated by this method. Furthermore, the file loader has been based upon exactly this structure. Therefore this requirement is met.
- The program should be able to run on the most common operating systems, e.g. Windows, IOS and Linux.
  The program is build towards all three of these platforms. No errors were encountered with the Windows or IOS version. Bart Theelen indicated that he could not get the Linux version running.
- The visualisations should comply with standards for SDF graph and SDF schedule visualisations.
  The gantt chart complies with conventions used in SDF. Philip Hölzenspies and Robert de Groote state this explicitly in their user testing interviews.
- An average computer needs to be able to run the program.
  Every computer this tool was tested on runned the tool with an acceptable speed.
- The program needs to support at least a minimum resolution of 1024x768.
  The window is resizeable and the UI scales depending on the window size. The program is able to run with a resolution of 1024x768.