MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Implementation of LaTeX Export Filter into Sequence Chart Studio

BACHELOR'S THESIS

Adrián Farmadin

Brno, spring 2013

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: Mgr. Ľuboš Korenčiak

Acknowledgement

I would like to thank my advisor Mgr. Ľuboš Korenčiak for valuable advice, comments and support during the writing of thesis. I also want to express my gratitude to the members of the SCStudio project. This thesis was created within a joint project of Siemens Convergence Creators, s.r.o. and the Institute for Theoretical Computer Science (ITI).

Abstract

Sequence Chart Studio (SCStudio) is a user-friendly drawing and verification tool for Message Sequence Charts (MSC). The main task of this thesis is implementing new export filter in SCStudio. New export filter will convert diagrams to LaTeX format using new MSC LaTeX package. Before the implementation of new export filter the MSC LaTeX package first needed to be extended to support all shapes and shapes adjustments. Result document will contain configuration header which allows user easy modify diagrams.

Keywords

Sequence Chart Studio, SCStudio, Message Sequence Chart, MSC, LaTeX, MSC package, export filter

Contents

Chapter 1

Introduction

Network protocols set rules how network components can exchange messages. This communication needs to be reliable, secure and fast. This is usually done by modelling, for modelling network protocols to achieve so difficult goals like security and reliability with high demand on speed of data transmission we need a very exact description language. Because many of these protocols are too big and too complicated we need a special tool to verify our designs.

Message sequence chart (MSC) is a description language defined in ITU-T Recommendation Z.120 [1]. It is designed to describe network communication and keep the description as easy as possible. In connection with other languages it can be used to support methodologies for system specification, design, simulation, automated verification, testing and documentation. MSC provides two ways how to represent communication: graphical and textual.

The Sequence Chart Studio (SCStudio) [2] is a user-friendly drawing and verification tool which can work with both MSC representations. Creating diagrams in textual representation is for users not natural neither intuitive, therefore SCStudio uses it only for import and export diagrams. Scstudio is build as a plug-in to Microsoft Visio, this connection allows more user-friendly diagram displaying and manipulating using MSC graphical form.

LaTeX MSC package was created to integrate MSC diagrams in documents and presentations. It offers only textual diagram creation through LaTeX macros on higher level of abstraction rather than supplying coordinates in pixels. This enables users to create diagrams fast, but in big drawings the source code will be still too long, difficult to read and making changes will take too much time.

The main task of this thesis is to develop new export filter for SCStudio which will put together simplicity of creating diagrams in SCStudio and integration in documents thanks to LaTeX MSC package. After analysing SCStudio and the LaTeX MSC package I found out that the package didn't support all shapes and shapes adjustments as SCStudio did. To make the export fully compatible with SCStudio I had to first extend the LaTeX MSC package.

The new version of LaTeX MSC package was created to be compatible with previous versions. To accomplish this goal event a deeper analysis of MSC LaTeX package was

performed. Because of new functionality, all presentation documents were updated.

The result document is well formatted to give user a good overview and enable him easy modify diagrams. Because many users are not familiar with LaTeX MSC package and they often want make same changes to result drawing, it is possible to adjust the layout of shapes without any deeper knowledge of MSC LaTeX package via configuration header. The configuration header contains all parameters used in drawing. The parameters are divided in several categories to facilitate the search and the way how to change the result drawing.

The tests ensure the mapping from SCStudio structure to MSC LaTeX package command is working properly and all known tricky cases are exported correctly. All the test cases must be in SCStudio representation to ensure good testing condition. The tests also ensure the connection with other algorithms work correctly too, because it is supposed the export will be used in many cases in connection with SCStudio algorithms for importing and transforming diagrams. To test this connection test cases are in MSC textual representation.

You find the definition of Message Sequence Chart in Chapter 2. Then in Chapter 3 is the definition of SCStudio, some of its algorithms and how SCStudio stores MSC diagrams. We introduce in Chapter 4 LaTeX MSC package with quick tutorial. All work on implementation is gathered in Chapter 6. Section 6.1 concerns about all modifications made on MSC LaTeX package. Then following Section 6.2 is about implementation the export filter and auxiliary classes. Finally in Section 6.6 is explained how the functionality is tested.

Chapter 2

Message Sequence Chart (MSC)

The following definition of Message Sequence Chart is from ITU-T Recommendation Z.120 [1]. The purpose of ITU-T Recommendation Z.120 is to provide a trace language for the specification and description of the communication behaviour of system components and their environment by means of message interchange. Message Sequence Chart (MSC) is recommended language which provides all these features and thanks to its graphical representation is the communication represented in a very intuitive and transparent manner.

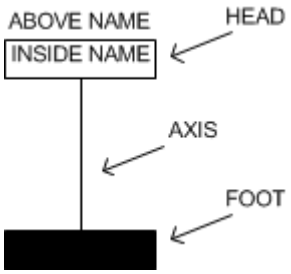
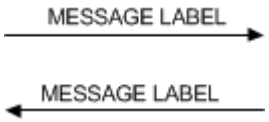
MSC is not tailored for one single application domain. An important area of application for MSC is an overview specification of the communication behaviour for real time systems, in particular telecommunication switching systems. By means of MSCs, selected system communications, primarily "standard" cases, may be specified. Non-standard cases covering exceptional behaviour may be built on them. Thereby, MSCs may be used for requirement specification, interface specification, documentation of real time systems and with connection with other languages for simulation and validation. MSC language is divided in two main parts: Basic MSC (BMSC) and High-level MSC (HMSC). In scope of this thesis we will consider only the visual aspects of MSC diagrams.

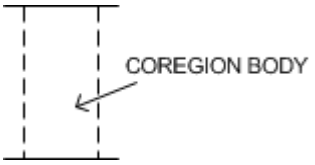
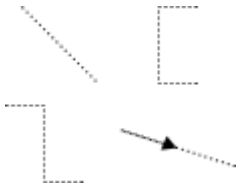
2.1 Basic MSC (BMSC)

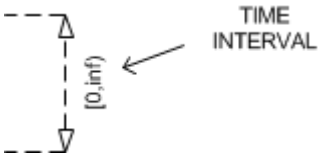
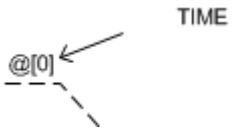
BMSC describes a partial behaviour of a system via message interchange between instances. An instance represents a system component and a message represents information passed from one instance to another one. The sending and consumption of messages are two asynchronous events, where every sending event happens before receive event. Along each instance axis the time is running from top to bottom, however, a proper time scale is not assumed. A total ordering of events is assumed along each instance axis. The semantics of communication represented by diagram is given by the order of events. Order between events and communication can be closer specified with other BMSC symbols described in Section 2.2. All other visual aspects like message slope or distance between events on an instance don't influence the semantics of diagram.

2.2 BMSC symbols

In this section we explain the meaning of some BMSC symbols.

Instance	
	An instance represents a system component. It consist of head, axis, foot and name. All parts except axis are optional.
Message	
	A message represents an information exchange between instances. It is consists of an arrow and label. Arrow determines the direction of communication and label the exchanged information. A message muss be attached to two instancie axes.
Lost message	
	Lost message represents an undelivered message. We know to which sending event and instance message belongs, but we can't determine the recipient.
Found message	
	Found message is opposite to the lost message. It represents message from unknown source.

Coregion	
	Inside of coregion symbol no order of events is assumed. It must be attached to an instance axis.
Order	
	*Order* symbols represents order between events. It can be used to determine some order in coregion or between any two events.
Condition	
	Conditions can be used to restrict the possible communication represented by an MSC. They muss be attached to one or more instance axes. The label of a condition symbol determines a condition represented with this symbol.
Action	
	An action symbol represents an atomic event. The event is described in the action label. An action must be attached to an instance axis.

Comment	
	Comment can be attached to any event.
Time interval	
	Time interval represents time relationship between two events. Each end of the symbol is attached to an event.
Absolute time	
	Absolute time sets time of occurrence of event. It can be attached to the event.

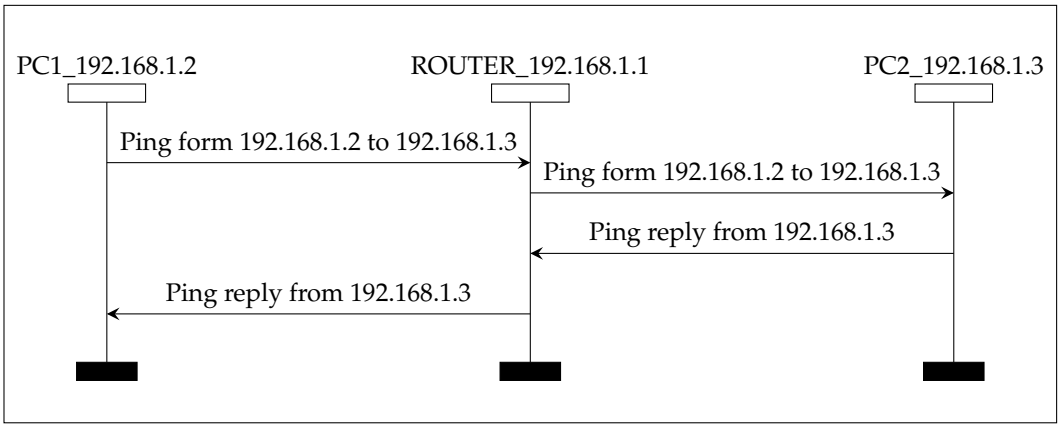


Figure 2.1: BMSC example

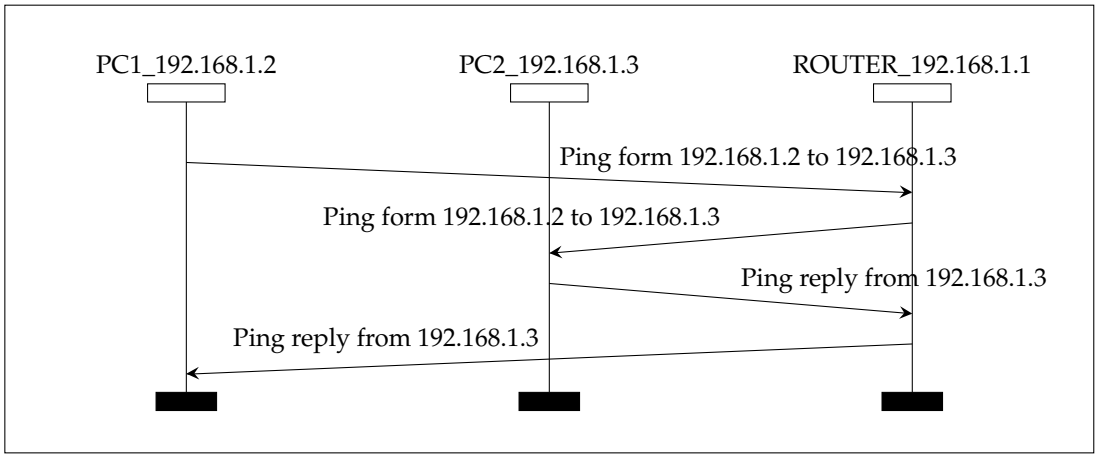


Figure 2.2: BMSC example 2

The BMSC in Figure 2.1 and in Figure 2.2 represent the same communication. They describe simple communication in home network. Instances represent two computers, router and internet. First four messages represent ping from PC1 to PC2. PC1 sends ping request, which goes through router and then the PC2 sends a response.

The diagram is created from three instances. Each instance has a name displayed above its head symbol. The instance names distinguish the network components. In this case there are two computer and router. Each device has assigned IP address. The IP address is a part of instance name. The pings between device is displayed as messages. Point where message connects to an instance is an event. In Figure 2.1 are eight events.

2.3 High-level MSC (HMSC)

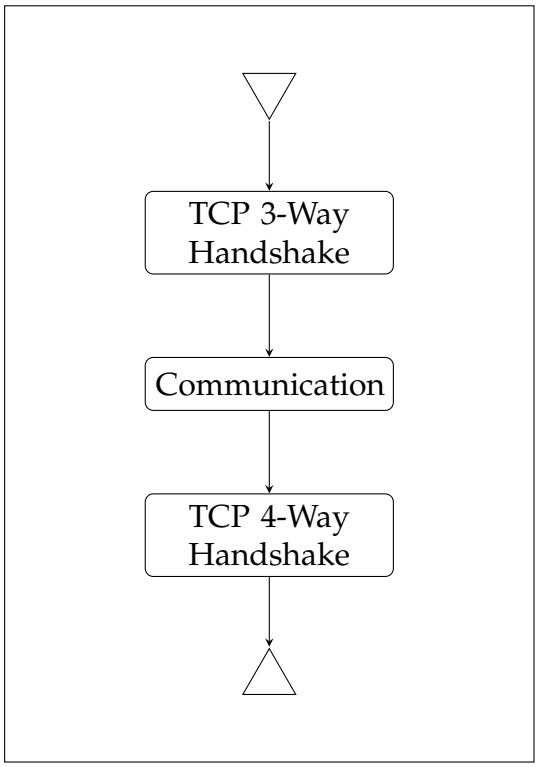
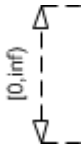


Figure 2.3: HMSC example

HMSC offers higher level of abstraction than BMSC. A HMSC is a directed graph which describes how set of MSCs can be combined. The possible communication among the nodes is displayed as lines connected to nodes, the position of nodes doesn't influence the semantics of diagram. A *flow* is path among the nodes, starting in start node and ending in end node. The incoming arrows are always connected to the top edge of the node symbols whereas the outgoing arrows are connected to the bottom edge. A node can represent a reference to another MSC diagram.

2.4 HMSC symbols

In this section we explain the meaning of some BMSC symbols.

Start symbol	
	There is only one start symbol in each HMSC. It represents flow start.
End symbol	
	It represents the end of a flow. In one HMSC can be more than one end symbol.
MSC reference	
	Can be used either to reference a single MSC or a number of MSCs using a textual MSC expression.
Condition	
	Can be used to indicate global system states or guards and impose restrictions on the MSCs that are referenced in the HMSC.
Connection point	
	Simplifies the layout of HMSCs and has no semantical meaning.
Time interval	
	It is used to enforce the time constraints. It must be attached to one or more reference symbols.

Chapter 3

Sequence Chart Studio (SCStudio)

The following definition of Sequence Chart Studio is from Sequence Chart Studio documentation [2]. Sequence Chart Studio (SCStudio) is a user-friendly tool created as a joint project of Siemens Convergence Creators, s.r.o. and Institute for Theoretical Computer Science (ITI), Faculty of Informatics, Masaryk University. It offers possibilities for drawing and verification MSCs. The graphical front-end is implemented as a Microsoft Visio add-on, but integration with other applications is possible. It is also possible to use SCStudio as stand alone application via command line interface.

3.1 Beautify

SCStudio offers algorithm for transforming diagrams appearance, this algorithm is called *Beautify*. It transforms diagrams to improve they readability. Beautify can transform all visual parameters of a diagrams that do not influence its semantics. User can adjust the transformation parameters through Beautify options. For more information about Beautify see [3].

In Figure 3.1 we can see a BMSC diagram which is hard to read. It is problem recognise to which message belongs which label and the understanding of semantics is difficult. In Figure 3.2 we can see the diagram from Figure 3.1, but transformed with Beautify. The readability was significantly increased and the semantic of the diagram remains untouched.

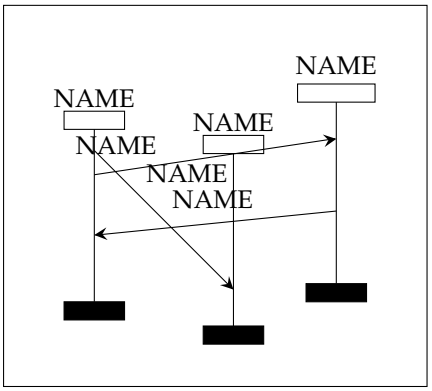


Figure 3.1: BMSC before Beautify

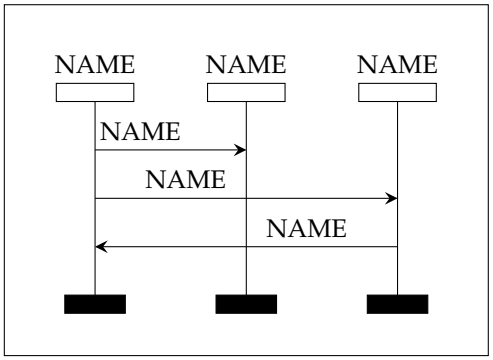


Figure 3.2: BMSC after Beautify

3.2 Z120 import and export

SCStudio includes import and export filter for textual MSC format. In scope of this thesis we will refer this import and export filter as *Z120* algorithm. Z120 works with *mpr* files, which hold diagrams in MSC textual form. Because no graphical information is transferred during importing from or exporting to textual form, SCStudio uses Beautify to complete graphical information.

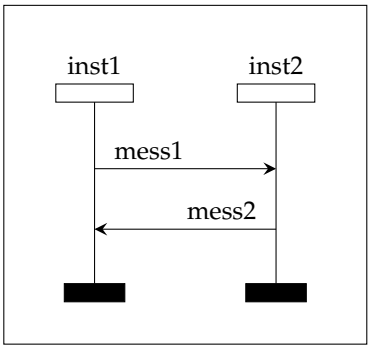


Figure 3.3: BMSC for export to MSC textual format

Listing 3.1: BMSC in MSC textual format from Figure 3.3

```
mscdocument z120.vsd;
msc Page_1;
inst inst1;
inst inst2;
inst1: instance;
out mess1,0 to inst2;
in mess2,1 from inst2;
endinstance;
inst2: instance;
in mess1,0 from inst1;
out mess2,1 to inst1;
endinstance;
endmsc;
```

3.3 SCStudio structure

SCStudio uses it for storing information as an object-oriented model. The base object is `Msc class`. From `Msc` inherit `BMsc class`, which stores BMSC diagrams and `HMsc class`, which stores HMSC diagrams. These two classes are in more detail descripted in sections below.

3.3.1 BMSC class

Main attribute of `Bmsc class` is a list of `Instance class` pointers. `Instance class` represents an instance. Each instance consists of event areas. Every event area is either `StrictOrderArea class` or `CoregionArea class`. Both areas hold events,

difference is in their semantics. In `StrictOrderArea` order of events is strictly given. In `CoregionArea` order of events cannot be determined, if there aren't explicit order symbols.`StrictOrderArea` events are stored as a list. In `CoregionArea` events are stored as a *DAG* - directed acyclic graph. Thus if all events are needed first all minimal events must be found and then listed all relationships of these events. Every relation gives us an order between these events.

An event is abstract class, it can represent any symbol what can be attached to an instance in SCStudio structure. Symbols that can be attached to an event are stored using event attributes.

3.3.2 HMSC class

`HMSC class` represents a relationship between HMSC nodes. Each diagram has one start node. The start node and all others nodes have a list of successors nodes. The current node is predecessor and all nodes in the list are successor nodes. The relation in graphical representation is displayed as an arrow. The arrow beginning is connected to bottom edge of predecessor node and the arrow end is connected to top edge of successors node. The flow ends by end node, which has no successors. Other relation symbols are time symbols. The Time symbols can be only attached to reference nodes.

Chapter 4

LaTeX MSC package

For simple integrating MSC in texts and presentations the LaTeX MSC package was created. It enables to draw MSC diagram in texts using LaTeX macros and commands. If offers higher level of abstraction for drawing. For example by BMSC user doesn't need to know all coordinates where to put symbols. But just the vertical order of symbols on particular instances.

The original macro package [4] was implemented in PSTricks [5], what caused a few limitations. The main limitation is converting drawings in pdf format. For this conversion special tools are needed and whole procedure is complicated. To improve this procedure Tomas Fabry in his bachelor theses [6] re-implemented macro package in TikZ & PGF [7] such that all macros are fully compatible with old package. Diagrams drawn with new package can be converted simply to pdf using pdflatex tool.

4.1 Quick MSC package tutorial

To draw MSC diagrams in your tex document simply include msc.sty file with command `\usepackage{msc}`. BMSC diagram starts with `\begin{msc}{title}` command. The parameter `title` defines the title of the BMSC. Then follow commands to draw diagram.

First we need to define instances in diagram. This is done with `\declinst{nickName}{aboveName}{insideName}` command. The parameter `nickName` defines instance nickname, which will be used in source code to associate components with appropriate instance. The instance nickname must be unique. The second two parameters define visible names for users. The `aboveName` parameter defines instance name displayed above it and the `insideName` is displayed inside the instance head. Instances are ordered in diagram in the same order as they are defined in the document and are displayed from left to right.

To add elements to an instance the special commands are used. All commands have almost the same syntax to make the commands easy to remember and to use. Parameters are ordered as follows, if the element can obtain text, then the first parameter defines it. The next parameter defines instance to which belongs the element. To address the cor-

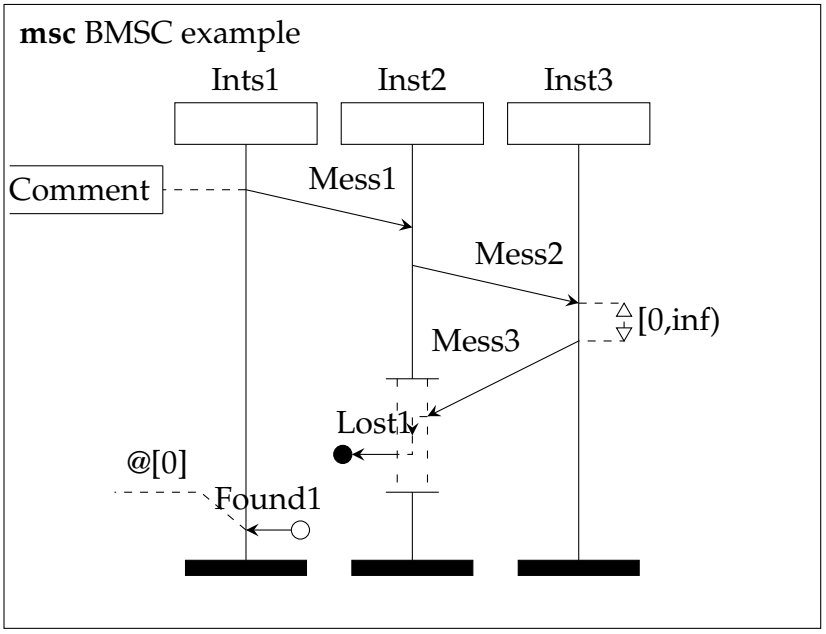


Figure 4.1: BMSC diagram example created with MSC package

rect instance is used instance nickname. Finally if element is associated with more than one instance the last obligatory parameter defines how. Except these main parameter elements macros have optional parameters which specify the element look. The order of optional parameters is not united as order of obligatory parameters, some elements with similar attributes have optional parameters in different order.

The vertical padding in BMSC diagrams is made by level system. The BMSC diagram is divided vertically on levels. First level is automatically created under instance head. Each additional level is created with `\nextlevel` command. New elements are automatically drawn on newest level. The last level for instance foots is created automatically too after using `\end{msc}` command.

To draw diagram in Figure 4.1, first we need to define environment with `\begin{msc}{BMSC example}` command. In this diagram there will be three communication elements, so we need three instances. For every element we define an new instance with `\declinst{nickName}{aboveName}{insideName}` command. The `aboveName` parameter will be the name of instance displayed above instance head and the `insideName` parameter will be empty in all cases, because original diagram has no names inside instance heads. The `nickName` must be unique for every instance. In this case we can use instance `aboveName` as `nickName`. After defining all instances, we must split symbols of the BMSC in levels. On first level we will have an message form *Inst1* to *Inst2*

with an one level slope. To define this message we use `\mess{Mess1}{inst1}[0.3]{inst2}[1]` command. The optional parameter `[0.3]` adjusts the location of message label to improve readability. We attach a comment to send event of this message with `\msccomment{Comment}{inst1}` command. On instance *Inst2* we want have an empty space between the first receive event an the first send event so we create a two new levels with `\nextlevel[2]`. The other messages will be defined in the same way. Between the receive event of message *Mess2* and the send event of message *Mess3* we set a time interval with command `\measure[r]{[0,inf)}{inst 3}{inst3}[1]`. The coregion is defined with command `\regionstart{coregi on}{inst2}` which creates the coregion start on instance *Inst2*. Inside the coregion we want an ordering symbol. The ordering symbol is drawn with `\order[r]{inst2}[y es]{inst2}[1]` command. The optional parameter `[r]` set the direction of order to go from right to left and `[yes]` determines that the order symbol must be inside the coregion. The lost message is drawn with `\lost{Lost1}{}{inst2}`. To close coregion on instance *Inst2* we use `\regionend{inst2}` command. Found message is defined with `\found[r]{Found1}{}{inst1}`. The absolute time on found message *Found1* is set with `\mscmark{@[0]}{inst1}` command. Finally you close the BMSC environment with `\end{msc}` command. Now if you compile your diagram, the result will not match Figure 4.1. The comment intersects with diagram box and message *Mess3* is attached instead of coregion body edge to the inside of coregion body. To remove these drawbacks we make the diagram box wider with `\setlength{\envinstdist}{1.6 \envinstdist}` command and we append an optional parameter `[.\regionwidth]` to command for drawing *Mess3*. The optional parameter sets the padding of the message end to correct the wrong rendering. You can see the result source code in Listing 4.1.

Listing 4.1: Source code of diagram from Figure 4.1

```
\documentclass{article}
\usepackage{msc5}
\begin{document}
\begin{msc}{BMSC example}

\setlength{\envinstdist}{1.6\envinstdist}

\declinst{inst1}{Ints1}{}
\declinst{inst2}{Inst2}{}
\declinst{inst3}{Inst3}{}

\mess{Mess1}{inst1}[0.3]{inst2}[1]
\msccomment{Comment}{inst1}
\nextlevel[2]
```

```
\mess{Mess2}{inst2}[0.3]{inst3}[1]
\nextlevel

\measure[r]{[0,inf)}{inst3}{inst3}[1]
\nextlevel

\mess{Mess3}{inst3}[0.3]{inst2}[2][.5\regionwidth]
\nextlevel

\regionstart{coregion}{inst2}
\nextlevel

\order[r]{inst2}[yes]{inst2}[1]
\nextlevel

\lost{Lost1}{}{inst2}
\nextlevel

\regionend{inst2}
\nextlevel

\mscmark{@[0]}{inst1}
\found[r]{Found1}{}{inst1}

\end{msc}
\end{document}
```

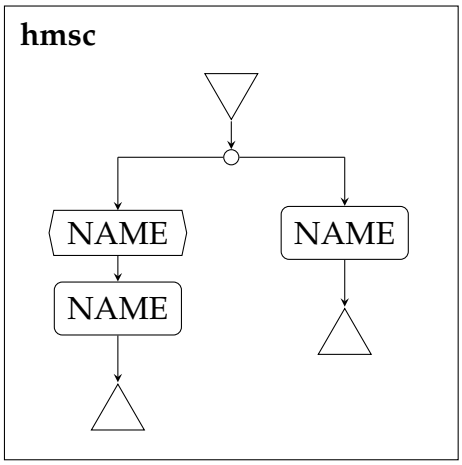


Figure 4.2: HMSC diagram created with MSC package

Drawing of HMSC diagrams is different. On beginning of diagram you must define environment space for yours drawing. It is done with `\begin{hmsc}{title}(x1,y1)(x2,y2)` command where title defines HMSC heading. The other parameters define the size of environment for HMSC elements. The coordinates `x1` and `y1` define lower left point of the environment. The coordinates `x2` and `y2` define upper right point of the environment.

Commands of node main attributes are ordered as follows. First parameter defines symbol nickname. If symbol contains text, next parameter defines the text on symbol. The last one is a point to which attach symbol. Special command with different order of parameters is
`\arrow{startSymbol}[pointList]{endSymbol}` which draws connection symbol between symbols. The `startSymbol` defines from which symbol draw connection, the `endSymbol` defines symbol to make connection with and `pointList` is optional parameter which defines list of points to make the connection symbol bent.

Drawing the diagram from Figure 4.2 is done as follows. First you need to define the HMSC environment with `\begin{hmsc}{}(0,0)(6,6)`. The `(0,0)` sets the begin point of the HMSC environment and the `(6,6)` the end point. Now you can define all nodes in diagram. In the given HMSC we will need these nodes:

- a start node - drawn with `\hmscstartsymbol{nickName}(x,y)`,
- a connection node - drawn with `\hmscconnection{nickName}(x,y)`,
- a reference node - drawn with `\hmscreference{nickName}{text}(x,y)`,
- a condition node - drawn with `\hmsccondition{nickName}{text}(x,y)`,

- an end symbol - drawn with `\hmscendsymbol{nickName}(x,y)`.

After defining all the nodes we can define connections between them. This is done with `\arrow` command. Finally you close the hmsc environment with `\end{hmsc}` and compile source code. You can find the complete source code in Listing 4.2.

Listing 4.2: Source code of diagram from Figure 4.2

```
\documentclass{article}
\usepackage{msc5}
\begin{document}
\begin{hmsc}{}(0,0)(6,6)

\hmscstartsymbol{0}(3,4.5)
\hmscconnection{1}(3,4)
\hmscreference{2}{NAME}(4.5,3)
\hmsccondition{3}{NAME}(1.5,3)
\hmscendsymbol{4}(4.5,2)
\hmscreference{5}{NAME}(1.5,2)
\hmscendsymbol{6}(1.5,1)

\arrow{0}{1}
\arrow{1}[(4.5,4)]{2}
\arrow{1}[(1.5,4)]{3}
\arrow{2}{4}
\arrow{3}{5}
\arrow{5}{6}

\end{hmsc}
\end{document}
```

Chapter 5

Analysis

In this chapter we will make an analysis of the assignment and discuss possible solutions. The assignment is to implement LaTeX export filter. The result of the export filter must be user-friendly and easy to modify. The result document will contain a configuration header to even more facilitate the work with it. The two best possibilities to draw diagrams in documents are Tikz & PGF package [7] and LaTeX MSC package [6]. We address advantages and disadvantages of both.

The Tikz & PGF package is one of the most used tool to draw graphics in LaTeX documents. It can place shapes based on their absolute position or using relative positioning. The relative positioning is done with defining the relationship between shapes. If the export filter would export the diagrams directly to Tikz & PGF commands, then the implementation would be easier. However the result will be not very user-friendly , because in Tikz & PGF the preferred way how to position elements in drawing is with relation between nodes. The elements in result drawing would be strongly linked and deleting or rearranging of the elements would be difficult. The result document will difficult to read as well, because every shape will be listed as several command. To adjust shape in such representation the user must have a very good knowledge of Tikz & PGF package.

On the other hand the LaTeX MSC package is build upon the Tikz & PGF package and it supports only drawing of MSC symbols. The LaTeX MSC package uses higher level of abstraction for positioning symbols. The symbols are placed using a level system. Thanks to the level system user only needs to known the vertical order of symbols on particular instances. The LaTeX MSC package offers way how to make the result document easy to read. Every shape is represented with only one command. The adjustment of shapes is done through predefined parameters. The level based positioning system offers an easy way how to delete or rearrange symbols. The MSC package doesn't support all shapes e.g. it isn't possible to draw time interval symbol in HMSC diagrams. Moreover present shapes didn't have enough adjustment parameters to match the shape appearance in SCStudio and in the ITU-T Recommendation Z.120 e.g. it isn't possible to set slope on a comment symbol or it isn't possible to set colour of an instance or a coregion symbol.

Because the above reasons the MSC LaTeX package method has more advantages

than disadvantages and is more suitable for the export filter. However the MSC LaTeX package doesn't support all shapes and shape adjustment is cumbersome to use in SC-Studio. To make the export result as user-friendly as possible I decided to expand the LaTeX MSC package and use this new version for implementation of the export filter.

5.1 Configuration header

Every printed diagram will contain a configuration header, which allows user to modify diagram without any deeper knowledge of the MSC package. This section of diagram contains definition of all lengths used in it. The lengths are divided into several groups to make the search for appropriate length more easier for the user. The main two groups are *environment* lengths and *shape* lengths. The shape lengths are divided in more subcategories depending on type of diagram. A group of lengths is only displayed when it is used in diagram.

The configuration header facilitates work with MSC package. For example the work with MSC arrow symbols. An arrow often begins on current declared level and ends in level not declared jet. The problem is that we don't know how long will be the vertical distance between the arrow ends. Thanks to configuration header this problem is solved. In configuration header we know the length of every level and we pass the message slope as sum of these levels. Another good example of problem solved with configuration header is with messages going to coregion not defined on current level. Without configuration header in time of printing the message we wouldn't know the coregion width. The result will be message attached instead of coregion body to middle of coregion. With coregion header the value of message padding parameter is set to half width of addressed coregion symbol. This link automaticly changes the message padding with change of coregion width.

Listing 5.1: BMSC configuration header

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Configuration header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Scale:
\def\mscScaleX{1.0}
\def\mscScaleY{1.0}

%%%% Lines width:
\pgfsetlinewidth{0.4pt}
```

```
%%%% Picture box width:
\setlength{\envinstdist}{20mm*\real{\mscScaleX}}

%%%% Picture box y-padding:
\setlength{\topheaddist}{\topheaddist}
\setlength{\bottomfootdist}{\bottomfootdist}

%%%% Instance first and last level height:
\setlength{\firstlevelheight}{5mm*\real{\mscScaleY}}
\setlength{\lastlevelheight}{20mm*\real{\mscScaleY}}

%%%% Levels:
\def\levelC{3mm*\real{\mscScaleY}}
\def\levelB{4mm*\real{\mscScaleY}}
\def\levelA{6mm*\real{\mscScaleY}}
\def\levelD{10mm*\real{\mscScaleY}}

\setlength{\levelheight}{\levelC}


%%%% Slope:
\def\slopeA{0mm*\real{\mscScaleY}}
\def\slopeB{\levelA+\levelB+\levelB}


%%%% Incomplete message slope:
\def\incSlopeA{0mm*\real{\mscScaleY}}


%%%% Coregin first levels:
\def\coregionFirstA{10mm*\real{\mscScaleY}}


%%%% Coregion last levels:
\def\coregionLastA{10mm*\real{\mscScaleY}}


%%%% Width of coregion's body:
\def\coregionWidthA{10mm*\real{\mscScaleX}}
```

```
\setlength{\regionwidth}{\coregionWidthA}

%%%% Width of local action:
\def\localActionWidthA{20mm*\real{\mscScaleX}}

\setlength{\actionwidth}{\localActionWidthA}

%%%% Height of local action:
\def\localActionHeightA{10mm*\real{\mscScaleY}}

\setlength{\actionheight}{\localActionHeightA}

%%%% Length of lost/found message:
\def\lostFoundWidthA{17mm*\real{\mscScaleX}}


%%%% Width of comment:
\def\commentWidthA{15mm*\real{\mscScaleX}}

\setlength{\msccommentdist}{\commentWidthA}

%%%% Comment slope:
\def\commentSlopeA{5mm*\real{\mscScaleY}}


%%%% Width of time interval:
\def\timeIntervalWidthA{7mm*\real{\mscScaleX}}

%%%% Ordering width:
\def\orderingWidthA{3mm*\real{\mscScaleX}}

%%%% Width of instance head/foot:
\def\instanceWidthA{10mm*\real{\mscScaleX}}

\setlength{\instwidth}{\instanceWidthA}

%%%% Height of instance head/foot:
\def\instanceHeightA{3mm*\real{\mscScaleY}}

\setlength{\instheadheight}{\instanceHeightA}
```

```
\setlength{\instfootheight}{\instheadheight}

%%%% Space between instances:
\def\instanceSpaceB{30mm*\real{\mscScaleX}}
\def\instanceSpaceA{35mm*\real{\mscScaleX}}

\setlength{\instdist}{\instanceSpaceB}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of configuration header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The first sections of configuration header in Listing 5.1 are for modifying BMSC environment. These sections are:

- scale - allows scaling diagram in horizontal and vertical way,
- line width - sets line width for drawing diagram,
- picture box width - modifies the distance between the edge of the MSC and the first/last instance axis,
- picture box y-padding - modifies the distance between the edge of the MSC and the instance head/foot symbol.

Then follow sections for modifying diagram symbols. Configuration header offers same options for adjustment symbols as Beautify in SCStudio. Now we explain the main parts of Configuration header from Listing 5.1:

- instance first and last level height - allows to modify level right below instance head or level above instance foot,
- levels - to adjust space between symbols attached to instance,
- slope - this section is for modifying slope of a message, when a message slope goes through several levels, it is listed as a sum of these levels, to preserve the event order in diagram.

Chapter 6

Implementation

The assignment of this thesis is implementation of LaTeX export filter into SCStudio and design tests to ensure correct result of the export filter. The exported diagrams must match Visio representation and result document must be easy to read. Because result document might be too big and for user it'll take too much time to find and change something in the drawing, the result document will contain a configuration header, which can offer to user simple and fast way how to change the drawing.. The configuration header gives user quick overview of used values and offers almost the same possibilities as Beautify to change shape parameters except some tricky cases (e.g. Beautify is able to put two symbols from two different levels on one level).

Final goal of this thesis is to create tests which will test new functionality. The tests try to reveal hidden bugs in implementations and they will ensure that all valid diagrams will be exported correctly.

6.1 Updating of the LaTeX MSC package

To change the LaTeX MSC package first I needed to study Tikz & PGF package and make a deeper analysis of the package. New shape parameters must be passed as optional to preserve compatibility with previous versions of the package. New lengths, command names and order of new command attributes are derived from original ones, to make them similar and easy to remember. The changes in the package are following:

- Package now supports all shapes as SCStudio and some additional. They apperance can by adjusted to match SCStudio shape representation and the Z.120 standard requirements too.
- Each instance and each instance region symbol can now have different shape size.
- Elements which can hold multi-line text are automatically resized to match the size of text field if it is bigger than symbol size.
- Color of symbols can be changed.

With new version of the package the documents about the package needed to be rewritten too. I created a web site [8] where the new version of the package and new documents are published. The web page contains a tutorial and examples of diagrams drawn with the package. I contacted original authors of the LaTeX MSC package, if it is possible to replace original package on CTAN with my version or use some of their documentation. We started correspondence, but I haven't received the final answer before finishing this thesis.

The package development will continue in the future. The next progress will be focused on improving BMSC level system and addressing elements. In the future I want to finish the negotiation with original author of the package about its replacement by this newer version.

6.2 Export filter

In SCStudio every export filter class must inherit from `ExportFormatter class` and implement its methods. `ExportTex class` is new class responsible for LaTeX export. The most important method in `ExportTex class` is `save_msc`. It is called to export diagram. As parameters are taken:

- `std::ostream& stream` - a stream to print export result
- `const std::wstring &name` - name of export result
- `const MscPtr& selected_msc` - pointer to Msc class, from which export begins
- `const std::vector<MscPtr>& msc = std::vector<MscPtr>()` - vector of Msc class pointers to be exported after MSC represented by `selected_msc` pointer.

When `save_msc` method is called, first the link to download Msc5.sty package and it's manual is printed. Than latex document preamble is printed to create valid latex document. Document attributes in this case are:

- `\documentclass{article}` - command to set type of LaTeX document class, default class is article class, because this class is dedicated for articles in scientific journals, presentations, short reports, etc.
- `\usepackage{msc5}` - command for including Msc5.sty package to support macros for drawing MSCs

- `\usepackage[a4paper]{geometry}` - because some diagrams are bigger than A4 paper, this line allows users to change paper size without searching for relevant macro

- `\begin{document}` - command to identify the beginning of the document

Because HMSC reference node stores a reference to another MSC, for printing all MSCs passed in input parameters, we need two lists. One for MSCs waiting to be printed and second for printed MSCs. Before printing MSC we check if it wasn't already printed. If its label is in list with printed MSCs, MSC was printed before. If not it is printed on separate page in result document. To make export parallel capable for actual printing are responsible two classes: PrintBmsc and PrintHmsc. For every MSC a new instance of PrintBmsc or PrintHmsc class is created. After all diagrams are printed the LaTeX `\end{document}` command for identification of the end document is printed.

6.3 DistanceMap class

To store information about diagrams and print them in configuration header in an united way I designed `DistanceMap class`. All length values used in drawing are here stored in a map. Every length value has assigned an unique ID, so they can be listed in configuration header and used by printing symbols. To make the ID unique it is composite of length name and map key of the value.

The `DistanceMap class` supports a function to print length values definitions in configuration header and a function for converting length value to its equivalent ID defined in configuration header. Because SCStudio stores coordinates of diagram elements as a double, to make the work with coordinates easier all diagrams coordinates are used with certain approximation. For the approximation I created `Compare class`. This class compares and rounds all coordinates to the default accuracy 0.1mm, because this is the biggest accuracy in Visio.

6.4 PrintBmsc class

`PrintBmsc class` is responsible for printing BMSCs. BMSC constructor takes two parameters: stream for printing result and BMSC to be printed. Than to print result in given stream, print method needs to be called. This method first analyses given BMSC, to create configuration header.

During the analysis the BMSC is traversed and all information are collected. Traversing starts with instances. Every instance has event area list. Event area can be strict order area or coregion area. The difference between them is explained in Section 3.3.1. Areas

hold a list of events. Because an event represent any element on instance, it needs to be tested which element it is. After knowing event type, the relevant information is stored.

To create level system with MSC LaTeX package, all symbols positions need to be known. By traversing I find all `MscElementPtr` pointers and store them in `ElementListMap`. The `MscElementPtr` is pointer of parent class for all MSC elements. The `ElementListMap` stores elements in a map. The map key is y-coordinate of a symbol. Under a key are symbols with the same y-coordinate. This system correspondents with MSC package level system, all symbol with same y-coordinate are saved under same map key. Before inserting element in `ElementListMap` all attributes of symbol represented by this element are stored in corresponding instance of `DistanceMap class`. An instance of this class is associated with each length. `DistanceMap class` stores name of associated length, its description and values. By traversing the leftmost and rightmost point is found, to compute the environment size.

After all information are collected, they are processed to create configuration header. Level lengths are computed from values in `ElementListMap`. Difference of every two consecutive map keys gives a level length. The level length are divided in seven groups based on elements under map keys. These groups are:

- instance first and last levels - special levels demanded by MSC package when creating a diagram,
- diagram end level - on this level are only instance foot symbols,
- coregion area first and last levels - levels between start or end of coregion and coregions first or last symbol,
- slope levels - levels between send and receive events,
- basic levels - all other levels not fitting described above levels.

Algorithm 6.1: Distinguishing level types

input : $map\langle double, vector\langle MSCelement\rangle\rangle$ $level\_elements$
output: $map\langle double, int\rangle\rangle$ $level\_types$

```
1  for k ← 0 to level_elements.size() do
2      if k == 0 then
3          level_types.insert(pair(level_elements[k].first, INSTANCE_FIRST_LEVEL))
4          continue
5      if (k + 1)! = level_elements.size() then
6          if (k + 2) == level_elements.size() then
7              level_types.insert(pair(level_elements[k].first, INSTANCE_LAST_LEVEL))
               continue
8      else
9          level_types.insert(pair(level_elements[k].first, DIAGRAM_END_LEVEL))
10         break
11     int level_type ← BASIC_LEVEL
12     vector⟨MSCelement⟩ possible_mess_slopes
13     foreach MSCelement e of level_elements[k].second do
14         if e is CoregionArea and e.begin_height() == level_elements[k].first then
15             level_type ← COREGION_AREA_FIRST_LEVEL
16             break
17         if e is SendEvent then
18             possible_mess_slopes.insert(level_elements[k].second[n])
19     foreach MSCelement e of level_elements[k + 1].second do
20         if e is CoregionArea and e.end_height() == level_elements[k + 1].first then
21             level_type ← COREGION_AREA_LAST_LEVEL
22             break
23         if level_type is not COREGION_AREA_FIRST_LEVEL then
24             if e is ReceiveEvent then
25                 level_type ← SLOPE_LEVEL
26     level_types.insert(pair(level_elements[k].first, level_type))
```

The algorithm 6.1 for distinguishing level types takes as input parameter `ElementListMap`. We will reference `ElementListMap` in scope of this algorithm as *map of levels*. The map is by default sorted from first to last level. The first level in map is the instance first level type, determined by `INSTANCE_FIRST_LEVEL` constant. It determines the length between instance head symbol and first symbol in diagram. The last level in map is end level type, determined by `DIAGRAM_END_LEVEL` constant. On this level there are

only instance foot symbols. The second last level in map is instance last level type, determined by `INSTANCE_LAST_LEVEL` constant. It determines the length between the last symbol in diagram and instance foot. If a level doesn't meet any of these types it is more deeply analysed. The decision how to divide remaining levels is influenced with demand to offer same options in configuration header as Beautify menu. The level are divided in types with the following priority:

- 1. If next level contains a coregion and coregion ends on the next level, then the level is of coregion area last level type, determined by `COREGION_AREA_LAST_LEVEL` constant.
- 2. If a level contains a coregion and coregion starts on this level, then the level is of coregion area first level type, determined by `COREGION_AREA_FIRST_LEVEL` constant.
- 3. If a level contains a send event and the next level contains a receive event which belongs to the send event from previouse level, then the level is of slope level type, determined by `SLOPE_LEVEL` constant.
- 4. If a level doesn't fit any of previouse types, then it only determines the distance between ordinary MSC symbols and is of basic level type, determined by `BASIC_LEVEL` constant.

Before the configuration header is printed the BMSC environment is defined. The configuration header is described in Section 5.1. After printing the header the `ElementListMap` is listed and all stored pointers are passed to a system of function. In this system pointers are cast to find the element they are representing. After distinguishing the element, it is printed with correct LaTeX command. By printing the commands all lengths are referenced to their definition in configuration header. After all elements in level are printed, a new level must be created.

6.5 PrintHmsc class

This class takes care of printing HMSC diagrams. The initialization and invoking printing is the same as by `PrintBmsc class`. Only difference is instead of passing pointer to a BMSC a pointer to a HMSC is passed. The printing of HMSCs is easier than BMSCs, because of the SCStudio structure for storing it. Is not so complicated and it stores only basic information about the diagram.

Hmsc symbols have no visual attributes except position and text. The procedure of printing is following: The HMSC structure is traversed three times, because before MSC LaTeX package can draw connection between nodes, nodes need to be defined and

before drawing nodes the actual size of diagram needs to be known. Firstly the structure is traversed only for computing the diagram size. The size is computed from absolute position difference between the most outer symbols. All diagrams have by default 1.5cm border on every side. After knowing the size, the HMSC LaTeX environment is defined and configuration header is printed.

During second traversal all nodes are printed and references hold in reference nodes are pushed to queue for later printing.

By printing a node, its coordinates needed to be modified, because MSC LaTeX package 2D coordinate system has inverted y-axis compared to SCStudio coordinate system. To all nodes a unique number is assigned for later addressing. Because SCStudio stores no graphical information except node position, all node properties are set to SCStudio default ones.

Last time the structure is traversed to analyse relationship between nodes and to print corresponding connections. Because SCStudio stores connection line and arrow as a same symbol, by default connection line is printed as an arrow. As ID for defining nodes associated with connection in LaTeX MSC package commands I used unique node number assigned when printing. Again there is the problem with missing information. The connection symbols don't save information about line break points and the relation symbols don't save information about the distance between nodes and relation. After all relation and connection symbols are printed the HMSC LaTeX environment is closed.

6.6 Testing

One of the tasks of this thesis was to write automated tests for new functionality. The functionality is tested in two ways.

The first one is via pre-designed diagrams created in test source code, where all known tricky cases are tested and we can verify result independently from other functionality.

The first test case test the drawing of MSC instance symbol. This test case is the crucial case, because all BMSC diagrams are based on MSC instance symbols. In this test the proper printing of instances and printing of instances not defined and ending on the same level is tested. The following test cases test the connection between instance symbol and message symbol. The most important think to test here is the computation of slope passing through several levels. The slope must be defined as sum of these levels. The remaining tests test all possible representations of MSC symbols supported in SCStudio.

The second type of tests test the functionality on diagrams imported from mpr files. Although this tests doesn't satisfy good testing conventions, because they are depen-

dent from Z120 and Beautify, they give a good overview about functionality results. In the most cases diagrams to export will be results of Beautify algorithm.

To verify results from tests with known coordinates I compared the results with hand computed results. In second type tests diagram coordinates are created by beautify. So in result it is about to match configuration header symbol parameters with beautify default settings. By changing dependence algorithm, the appearance of imported diagrams changes and the test results get changed too. Therefore with every change in dependence algorithms influencing the appearance of diagrams the test need to be checked for correct results.

Chapter 7

Conclusion

We introduced MSC formalism and SCStudio tool. After analysing the MSC package we encountered problems with shape support. The package didn't supported same shapes as SCStudio and the current supported shapes didn't offer enough adjustment parameter to match the shape appearance in SCStudio. I removed this drawback with new updates of the package. New package version is compatible with SCStudio and ITU-T Recommendation Z.120. All extended commands remained fully compatible with old package versions.

Next we showed how SCStudio structure can be traversed and on this knowledge we based implementation of export filter. Export filter is composed of two main class responsible for export. PrintBmsc class which is responsible for exporting BMSC diagrams and PrintHmsc class which is responsible for HMSC diagrams.

By printing BMSC diagrams a deep analyses of coordinates to create transparent level system and configuration header was needed. Main problem here was how to divide diagrams in levels and distinction in which group created level belongs. In HMSC diagram we encounter problems with missing information about shapes. In this case all missing information were replaced with default ones from SCStudio.

We described what are benefits of configuration header and we analysed its content. We demonstrated on example advantages of configuration header and how to use it.

We explained why are testes divided in two groups and what each group offers. The tests helped reveal some drawbacks in implementation that were removed.

Created export filter meet all targets given in Chapter 1. Because SCStudio didn't support in time of implementation all shapes and their adjustments, I assume SCStudio will be extended and the development of export filter will continue in the future to support new shapes.

Bibliography

[1] ITU Telecommunication Standardization Sector - Study group 17. ITU recommendation Z.120, Message Sequence Charts (MSC), 2011.

[2] Martin Bezděka, Ondřej Bouda, Ľuboš Korenčiak, Matúš Madzin, and Vojtěch Řehák. Sequence chart studio. In *12th International Conference on Application of Concurrency to System Design (ACSD'12)*, pages 148–153. IEEE, 2012.

[3] Milan Malota. Layout Configuration for Message Sequence Charts. Bachelor's thesis, Masaryk University, Faculty of Informatics, 2012.

[4] Sjouke Mauw and Victor Bos. Drawing Message Sequence Charts with LaTeX. *TUGBoat*, 22(1-2):87–92, March/June 2001.

[5] Timothy Van Zandt. PSTricks: PostScript macros for Generic TeX. User's guide. 2007.

[6] Tomáš Fábry. Sequence Chart Typesetting in LaTeX. Bachelor's thesis, Masaryk University, Faculty of Informatics, 2011.

[7] Till Tantau. The TikZ and PGF Packages. Manual for version 2.10-cvs. 2012.

[8] Adrian Farmadin. MSC LaTeX package web site. [online], [retrieved April 22, 2013]. Available at: http://is.muni.cz/www/374320/.

Appendix A

Contents of Attached DVD

The attached DVD contains the following items:

- a PDF and LaTeX version of the thesis;
- source code of export filter (exportTex.cpp, exportTex.h);
- source code of tests (exporttex_test.cpp, exporttex_position_test.cpp);
- source code of SCStudio;
- new MSC LaTeX package
- executable installation file of SCStudio.