MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Layout Configuration for Message Sequence Charts

BACHELOR'S THESIS

Milan Malota

Brno, spring 2012

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.

Milan Malota

Advisor: RNDr. Vojtěch Řehák, Ph.D.

Acknowledgement

I would like to thank my advisor Vojtěch Řehák for valuable advice, comments and support during the writing of this thesis. I also want to express my gratitude to the members of the SCStudio project and the employees of ANF DATA. Finally, I want to thank to my family for their support.

This thesis was created within a joint project of ANF DATA, spol. s r. o. and the research centre Institute for Theoretical Computer Science (ITI).

Abstract

Message Sequence Chart (MSC) is formalism for specification of communication between entities in the system. For drawing and facilitating work with this formalism Sequence Chart Studio (SCStudio) is being developed within the joint project of ANF DATA, spol. s r. o. and the research centre Institute for Theoretical Computer Science (ITI).

SCStudio provides several transformers which are changing the diagram in some way. One of those transformers deals with well-arranged diagrams.

This thesis addresses problem of well-arranged basic MSC and conflicts which can occur by setting adjustable parameters. Next, a graphical user interface for setting parameters has been revised and improved. In addition, the complexity of implemented algorithm has been described and different method for solution has been provided.

Keywords

Sequence Chart Studio, SCStudio, Message Sequence Chart, MSC, basic Message Sequence Chart, WTL, Windows Template Library

Contents

1 Introduction

1.1 Motivation

In some definitions of the Internet we can see that it is a worldwide network of computers that can communicate with each other using network protocols. For many computer users this definition is not interesting but the Internet works in part because of protocols that govern how the network devices communicate with each other. A lot of questions are connected with network protocols such as questions of efficiency or security. Description languages facilitating specification and description of the communication behaviour of the system components were created for discovering adequate answers. One of those languages is Message Sequence Chart (MSC).

MSC is easy to learn and use, and in connection with other languages, e.g. Specification and Description Language (SDL) [1] and Testing and Test Control Notation (TTCN) [2], can be used for formal and automated validation expressed in the scenario. There are two representation of MSC: textual and graphical. The first one is mainly intended for information exchange between tools. The second one gives overview of the behaviour of communicating instances in two-dimensional diagram.

In general, there are many tools for displaying and manipulating graphs, e.g. Graphviz [3]. However, MSC is very specific type of graphs which makes it hard to use common tools. Sequence Chart Studio (SCStudio) [4] belongs among tools suitable for drawing MSC. SCStudio supports many functions for speeding up drawing, verifying the MSC, or changing the diagram in some way. The functionality called Beautify belongs to the third group and it redraws the diagram to be well-arranged. Well-arranged MSC is a subset of MSC that is easy to read and follows user preferences.

The analysis of well-arranged MSC is described in [5], nevertheless it does not contain analysis of all elements of MSC. The first goal of this thesis is to revise the previous analysis and algorithms for well-arranged diagrams and to extend them with time elements, local actions, and conditions. The second goal is to analyse and implement new graphical user interface for configuration of parameters that influnce drawing.

1.2 Structure of the Document

In Chapter 2, a description of MSC formalism is given and Sequence Chart Studio is introduced. Chapter 3 provides analysis of well-arranged MSC,

definition of conflicts and their solutions. In Chapter 4 algortithms are described. Chapter 5 provides description of Graphical User Interface. Finally, performance issues are discussed in Chapter 6.

2 Message Sequence Chart (MSC)

The purpose of Message Sequence Chart (MSC) [6] is to provide a trace language for specification and description of the communication behaviour of system components and their environment by means of message interchange. Graphical representation of MSC is very intuitive and easy to learn, use and interpret. Because of completeness it must be mentioned that the formalism consists of two parts (basic MSC and High-level MSC), however this thesis mainly focuses on basic MSC and therefore if there is any mention about MSC we speak about basic MSC.

2.1 Basic Message Sequence Chart (MSC)

MSC describes the communication between a number of system components. For each system component covered by a MSC there is an instance axis. The communication between system components is performed by means of messages.

The sending and receiving of messages are two asynchronous events. It holds that all instance axes depict total order. Events of different instances are ordered via messages, i.e. send event is before receive.

In Figure 2.1 there is an example of textual representation. In the first instance there are declared instances of the MSC. Then we can see several blocks with events of each instance; there are two types of events: $send$ (out) and $receive$ (in). Event belongs to exactly one instance and one message. The total order of the events of an instance is represented by the order of the events in blocks of the instance.

Note that textual representation does not contain graphical information, e.g. size of the elements. In the Figures 2.2 and 2.3 we can see graphical representations of MSCs that has the same textual representation. They vary in size of the elements and distances between them. In Figure 2.2 we can see the labels of messages intersect. It holds that appropriate graphical information makes final MSC more readable.

2.2 Sequence Chart Studio (SCStudio)

SCStudio is a user-friendly drawing and verification tool for Message Sequence Charts and UML Sequence Diagrams. The software is freely available under LGPL (GNU Lesser General Public Licence) via SourceForge [7].

SCStudio offers an open interface for additional modules. One of the

```
1  mscdocument Drawing1;
2  msc Page_1;
3  inst A;
4  inst B;
5  inst C;
6  C: instance;
7  out message1,0 to A;
8  in message3,1 from B;
9  endinstance;
10 A: instance;
11 in message1,0 from C;
12 out message2,2 to B;
13 endinstance;
14 B: instance;
15 in message2,2 from A;
16 out message3,1 to C;
17 endinstance;
18 endmsc;
```

Figure 2.1: Textual representation

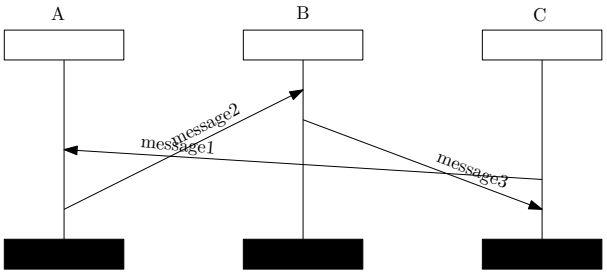

Figure 2.2: Non well-formed graphical representation

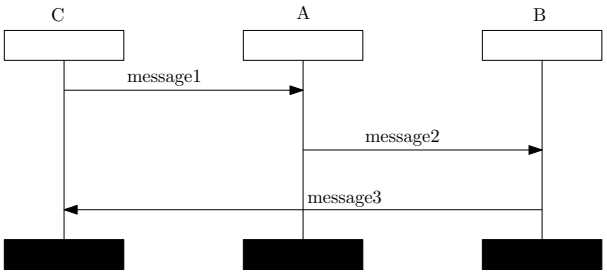

Figure 2.3: Well-formed graphical representation

SCStudio implementations is a module for Microsoft Visio [8]. Currently, it is the only drawing editor of the SCStudio that supports MSC drawing, export from and to ITU-T Z.120 textual format, graphical syntax verification and many other features. For more information please refer to [9].

2.2.1 Beautify transformer

SCStudio also offers Beautify transformer that rearrange given MSC according to the user preferences specified in the configuration dialogs. This transformer is currently used for:

- *redrawing the diagram created in graphical editor*;
  The input of this procedure is a diagram with graphical information created by the user during drawing. The output is the diagram with elements that follows values specified in configuration dialogs.

- *generating graphical information during import pocess*.
  The input is a textual representation without graphical information (see Figure 2.4). The output is the two-dimensional diagram that also respects user preferences (Figure 2.6); without Beautify all elements are projected to the same place (Figure 2.5).

It has to be said that Beautify works properly for both types of MSC diagrams. However, redrawing of High-level MSC is implemented in a very simple way, and thus additional analysis and reimplementation is needed.

Previous implementation of Beautify transformer consists of two algorithms: *instances_sequencer* and *layout_optimizer*.

The first one deals with the sequence of instances and their horizontal and vertical position in diagram. The algorithm didn't always return correct result as it is implemented as a *greedy algorithm*. The algorithm is described in Chapter 4 in detail.

The second algorithm is responsible for arranging events of the instances. This algorithm was based on the principle of *linear programming* (LP) [10]. For more information please refer to [5]. Previous implementation of *layout_optimizer* uses lp_solve, a free linear programming solver [11]. More details of this algorithm are given in Chapter 6.

According to the requirements provided by the company and performance analysis, we found out that the implementation of *layout_optimizer* complicates redrawing and import of big MSCs. For example the MSCs with more than 400 events, processing took several hours and ended without correct result. This problem is also described in Chapter 6 in detail.

```
1  mscdocument Drawing;
2  msc bMSC;
3  inst a;
4  inst b;
5  a: instance;
6  out m,0 to b;
7  endinstance;
8  b: instance;
9  in m,0 from a;
10 endinstance;
11 endmsc;
```

Figure 2.4: Textual representation of a MSC

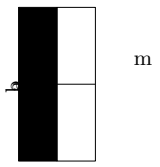

Figure 2.5: Import without Beautify

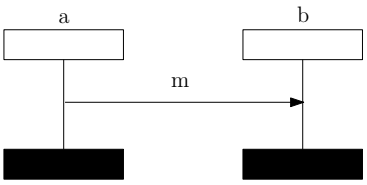

Figure 2.6: Import with Beautify

3 Analysis of well-arranged MSC

It can be said that the factor influencing MSC readability is hidden in distances between elements and sizes of those elements. In the scope of this chapter, these factors are aggregated in term *adjustable parameters*.

Note that the following list does not contain all possible adjustable parameters. It generally holds that there will always be other ways how to define well-arranged MSC, as it depends on user preferences. That is why we have created this analysis of well-arranged MSC in cooperation with employees of ANF DATA, spol. s r. o.

In this chapter, there are adjustable parameters discussed separately. However, many of them are not independent, i.e. assignment of values to these parameters may cause conflict situation. It holds that well-arranged MSC should not contain these situations and thus some parameters cannot be fulfilled at the expense of other parameters. All of these conflict situations are aggregated in term *conflicts*.

The first section introduces elements discussed in this chapter. The second section deals with *arranging of instances*. The next part covers *arranging elements on instances*. In the penultimate section the conflicts are discussed. Finally, in the end of this chapter *arranging of time information* is described.

3.1 Elements supported in SCStudio

According to the Z.120 recommendation, some elements can have different graphical representation (i.e. coregion[1]). In order to make it clear, the following list with the names of MSC elements discussed in this chapter was created:

- line instance, headless instance
- coregion box
- matched message (left, right)
- incomplete message (lost, found)
- local action
- local condition

1. Coregion - an area where events are not ordered.

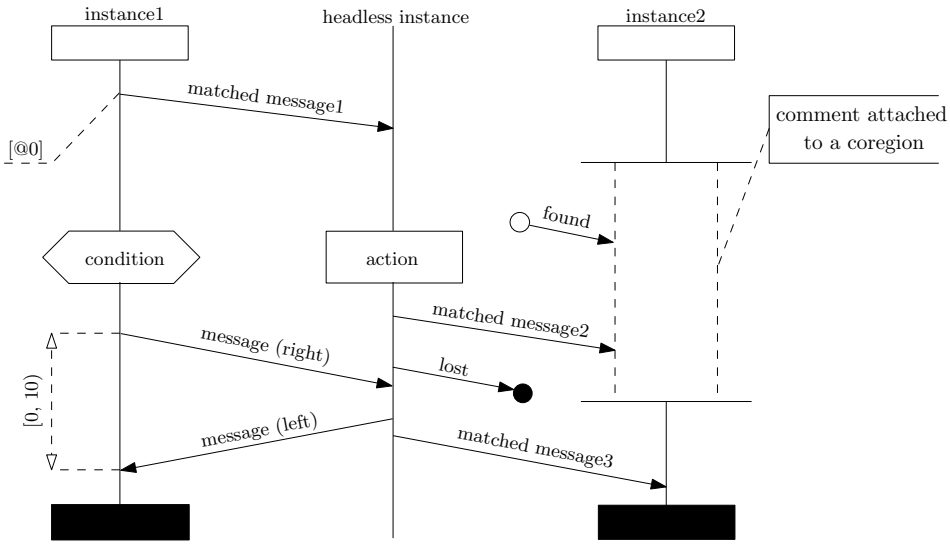

Figure 3.1: Graphical representation used in SCStudio

- time interval
- absolute time

In Figure 3.1 there is a graphical representation used in SCStudio. There are three instances, one of them is headless. There are also symbols for other elements: local condition, local action, comment, and coregion. On the left hand side, graphical representation of absolute time and time interval is shown. Finally, there are all types of messages that are discussed thereafter: matched messages (left, right) and incomplete messages (lost, found).

3.2 Parameters for Arranging Instances

Instances can be considered as the basic building blocks of MSC. Instance is compound of *instance's head*, *instance's foot*, *instance's axis* and a *label of instance* (see Figure 3.2). In the following text, we describe parameters for setting width and length of the instances and the problem of instance placement is formulated.

3.2.1 Length of Instances

The term *length of instance* refers to the distance between upper edge of instance's head and lower edge of instance's foot. Next, there are declared

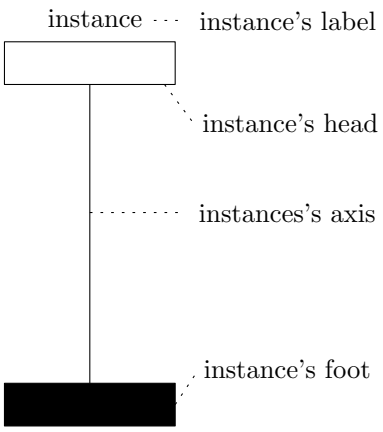

Figure 3.2: Instance's composition

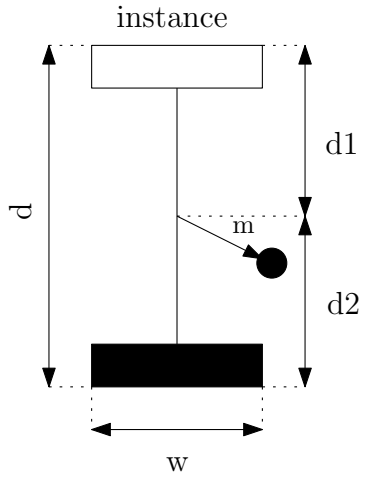

Figure 3.3: d - length of instance, d1 - instance_begin_distance, d2 - instance_end_distance, w - width of instance

two parameters facilitating readability of instances: $instance\_begin\_distance$ and $instance\_end\_distance$. These parameters refer to the distance between the beginning of the axis and first or last event occurred on the instance. If there is no event the length of such instance is set to the sum of those values (see Figure 3.3). According to Z.120, the length of every instance must be big enough to span all elements belonging to it.

For setting length there was declared few possibilities:

- to set length of the instances equal to the *constant given by the user*;
- to set instances' length *according to the arrangement of elements*; it means all parameters for arranging messages, actions etc. are fulfilled and none of the elements overreach the end of the instance;
- *not to set length*. The lengths of the instances remain unchanged.

For the import process where the graphical information is missing the second option is primarily set.

The second option is also connected with different lengths of instances in MSC. That is why the possibility for aligning lengths was introduced; it means all instances' foots are enlarged to the length of the longest instance.

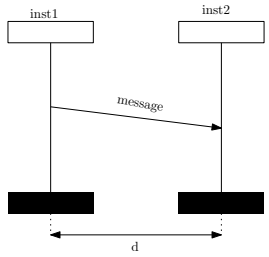

Figure 3.4: d - space between instances

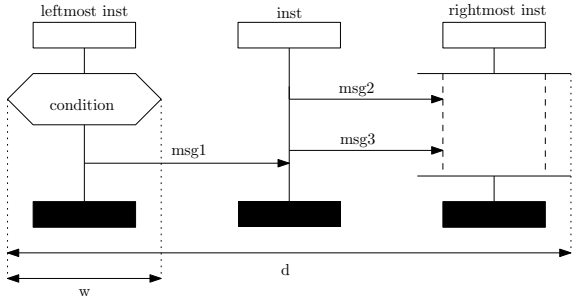

Figure 3.5: d - width of diagram, w - absolute width of instance

3.2.2 Width of Instances

The term *width of instances* assigns the width of instance's head and foot (see Figure 3.3). Regarding Z.120 head and foot of same instance must have the same width.

For setting width there were recommended the following options:

- to set width of instances equal to the *value given by the user*;
- to set width equal to the *size of the maximal width concerning; all heads/-foots* and *minimal width concerning all heads/foots* in the diagram;
- *not to set width*. The widths remain unchanged.

3.2.3 Placement of Instances

The term *placement of instances* refers to the position of each instance and each page sheet where well-arranged diagram should be drawn in; for vertical arrangement of instances we discussed *spaces between instances*, *permutation of instances*, and *placement of leftmost instance*.

The term *spaces between instances* refers to the distances between instances' axes (see Figure 3.4). For this parameter we discussed the following possibilities:

- to set spaces *equal to the constant given by the user*;
- to set spaces according to the user defined constant for *total width of well-arranged diagram* (see Figure 3.5);
- *not to set spaces*.

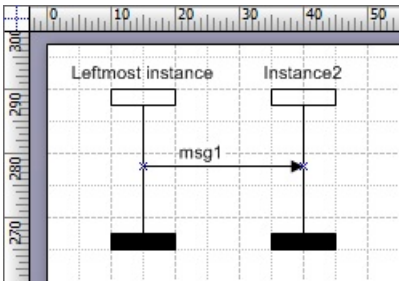

Figure 3.6: Line coordinates

For the explanation of the second option it has to be defined term *absolute width of instance* (see Figure 3.5). This term refers to the maximum width concerning all elements of an instance, i.e. action, condition, coregion, and width of instance.

The second option sets spaces between instances according to Equation 3.1.

$$\begin{aligned} space\_between\_instances = (&total\_width\_value \\ &- absolute\_width(leftmost\_instance)/2 \\ &- absolute\_width(rightmost\_instance)/2) \\ &/number\_of\_instances \end{aligned} \quad (3.1)$$

There is a conflict connected with the spaces between instances. It holds that elements of the different instances mustn't intersect. This conflict is discussed thereinafter.

The term *placement of leftmost instance* refers to the position of leftmost instance on the page sheet expressed by the numerical values on ruler in graphical interface in Visio. In Figure 3.6 there are represented line coordinates and position in Visio.

In order to support possibility to set this position, we discussed the following options:

- to place leftmost instance to the *left upper corner of the page*;
- to *remain position of the leftmost instance* of opened diagram in graphical editor;
- to place leftmost instance *to the coordinates defined by the user*.

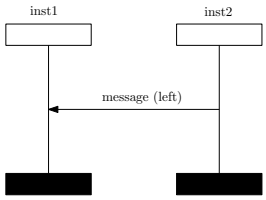

Figure 3.7: Going back message

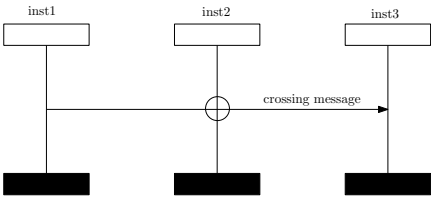

Figure 3.8: Message crossing instance's line

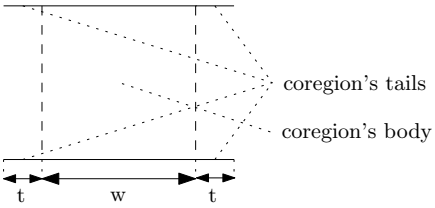

Figure 3.9: Coregion composition, w - width of coregion, t - width of tails

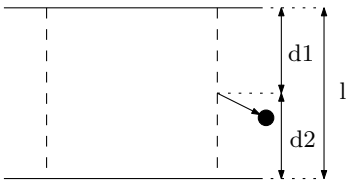

Figure 3.10: d1 - begin coregion distance, d2 - end coregion distance, l - length of coregion

According to the tracking of well-arranged MSC, we find that changing the order of instances can make the MSC more comprehensible. For such situations we discussed another adjustable parameter: *sequence of instances*.

There are two factors influencing comprehensibility of MSC. First one are *messages going back* (see Figure 3.7) and the second one are *messages crossing lines of instances* (see Figure 3.8). Because of that we have discussed two parameters: *number of crossing instances with messages* and *number of going-back messages*.

There are situation when it is not possible to eliminate all crossing and going back messages. The previous implementation allows users to eliminate nothing, the first, the second or both types of messages. If the user wanted to eliminate both types it hadn't been possible to determine which type will be eliminated at the expense of the other. That is the motivation for specifying: *weight_of_crossing_messages* and *weight_of_going_beck_messages*. The higher weight the more message of that type will be eliminated.

3.3 Parameters for Arranging Coregions

In the previous section, we have offered arguments for well-arranged placement of instances in MSC. The purpose of this and the following sections is finding adjustable parameters for arranging instances' elements.

Coregion is compound of *coregion's body* and *coregion's tails* (see Figure 3.9). Because of the shape, there are specified three adjustable parameters: *length of coregion*, *width of coregion*, and *width of coregion's tails*.

3.3.1 Length of Coregion

The distance between upper edge and the lower edge of the coregion is another adjustable parameter. This parameter was called *length of coregion* (see Figure 3.10). As well as the arrangement of instance's length, there are declared two parameters facilitating readability of coregions:

- $coregion\_begin\_distance$ - distance between first event laying on coregion and the beginning of this coregion;
- $coregion\_end\_distance$ - distance between last event laying on coregion and the end of the coregion.

It holds that the length of well-arranged coregions is at least equal to the sum of abovementioned distances.

For arranging coregions' length there is recommended to set lengths *according to the arrangement of elements* laying on the coregion.

3.3.2 Width of Coregion

The term *width of coregion* refers to the width of coregion's body (see Figure 3.9). For setting width there were recommended the following options:

- to set the width according to the *value given by the user*;
- to set width of all coregions occurred on the same instance equal to the *width of the instance's head/foot*;
- *not to set width*; it means the widths remain unchanged.

3.3.3 Width of Coregion's Tails

It holds that each coregion has four tails. According to the Z.120 recommendation, all tails must have the same width. Tails also can have the width

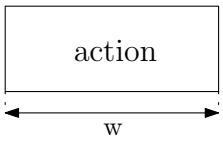

Figure 3.11: Symbol of the action, w - width of action

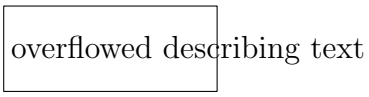

Figure 3.12: Label of action overflowed

equal to the zero. The following options are recommended for arranging tails:

- to set width of tails according to the *value given by user*.
- to set width of tails to the *width of coregion's body multiplied by user given value*.
- *not to set width*; it means widths of tails remain unchanged.

3.4 Parameters for Arranging Actions

The parameter influencing readability of actions is *width of action*. This parameter and the symbol for action is demonstrated in Figure 3.11.

The following possibilities are used for setting width of actions:

- to set the width of actions according to the *value given by user*.
- to set width of all actions occurred on the same instance equal to the *width of the instance's head/foot*.
- to leave width in *original size* found in the diagram created in editor.

Another parameter that was discussed is *height of action*. However, each action has the text explaining the kind of the action inside the rectangle and the possibility to set both variables does not profit. In Figure 3.12 there is conflict situation when the user set both variables too small. That is why height of the action is set according to the size, length and font of the text.

The determination of the actions' height is the very difficult problem. Because each font in different font size occupies different space and the computing of the space can be realized by rendering and measuring the necessary space or by the parsing the font; we have developed two approaches. The first approach is suitable for redrawing diagram created in graphical editor. Based on the behaviour of the actions during modelling we have

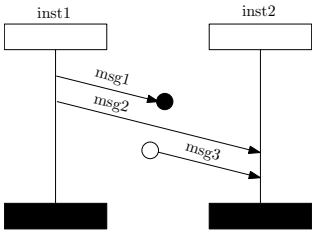

Figure 3.13: Types of messages

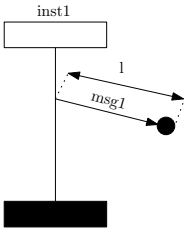

Figure 3.14: l - length of incomplete message

decided to keep the area of the shape same as found in the diagram. The height is computed by the following equation:

$$new\_action\_height = (original\_action\_height * original\_action\_width) / width\_of\_action \quad (3.2)$$

The problem occurs when the user wants to import document in textual format. It means graphical information is missing. We have decided to import actions' descriptions in the monospaced font and in the default size. The determination of the length of the text inside the action is effortless and fast with this approach.

3.5 Parameters for Arranging Conditions

According to Z.120, condition can be shared with the list of the instances. However, as well as the other algorithms, such condition is not fully supported that is why we provide analysis of conditions attached only to the one instance.

The *width of condition* is considered as the adjustable parameter for arranging conditions. It holds that options how to set this parameter are identical to the options specified for *width of action*.

3.6 Parameters for Arranging Messages

There are two types of messages regarding the Z.120 recommendation: matched messages and incomplete messages (see Figure 3.13). The first parameter describing the message is *length of message*. It holds that the lengths of matched messages is determined by the distance between send and receive event. On the contrary length of incomplete message is not determined

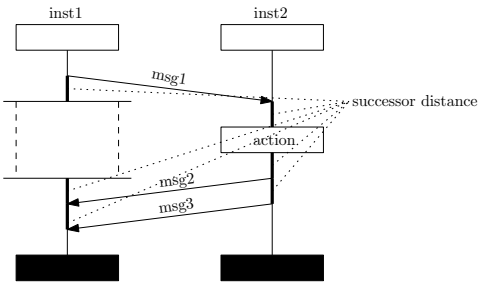

Figure 3.15: Successor distance

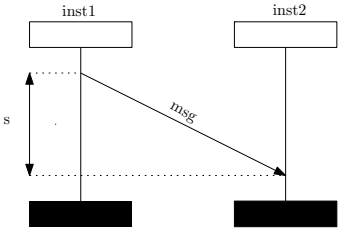

Figure 3.16: s - slope of message

by these events. The term *length of incomplete message* refers to the distance between a send/receive event and a dot (see Figure 3.14).

Next parameter describing the arrangement of elements is the *vertical distance between the messages' edges* (*slope of message*). In Figure 3.16 the slope of message is demonstrated. Regarding [6], messages going upwards are forbidden. The only exception is the cyclic MSC which cannot be drawn without any up-going message [5].

3.7 Parameters for Arranging Elements on Instances

In the previous sections, there are elements discussed separately. In this section, we describe parameters for arranging groups of MSC elements.

The first parameter describing the arrangement of elements is *distance between two elements on one instance* (see Figure 3.15). The term *successor distance* implies to this distance and it means distance between two events, coregions, actions, conditions, or between any two of them.

Next, *instance_begin_distance*, *instance_end_distance*, *coregion_begin_distance*, and *coregion_end_distance* could be also considered as parameters influencing arrangement of elements. However, if MSC contains only instances it could be difficult to determine the length of such instances. The same goes for the length of the coregion without events laying on it. That is why they were introduced as parameters for arranging instances and coregions.

3.8 Conflicts

In this section we define conflict situation that can occur by following certain combination of abovementioned options.

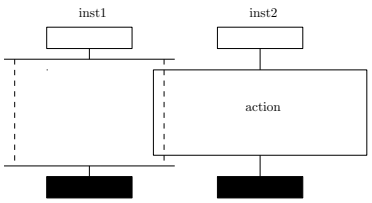

Figure 3.17: Conflict: Spaces between Instance

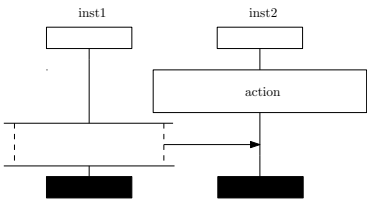

Figure 3.18: Without conflict

3.8.1 Instance Length

The filling of option allowing setting of the *length of instance* according to the user given value may cause conflict. The following parameters are declared for arranging elements on instances: *instance_begin_distance*, *instance_end_distance*, and *successor_distance*.

It is obvious that all of these parameters are not independent, i.e. if the partial distances denoting distances between elements are fulfilled, the length of instance is determined by them. The conflict occurs when the value given by user differs from the optimal value computed from partial distances.

There are two possibilities how to solve this conflict:

- to follow partial distances;
- to follow value for *length of instance*.

Generally, we have decided to follow the partial distances at the expense of the value for *length of instance*. If the user given value is less than optimal value, the warning explaining the problem is provided. If the user given value is greater or equals to the *length of instances* is set to that value.

3.8.2 Space between Instances

Regarding Z.120, the elements of the different instances mustn't intersect. The conflict may occur when the user wants to set spaces according to the given value for spaces or for total width of diagram and the *absolute_width* of neighbouring instances is greater than this value (see Figure 3.17).

Although the previous definition, there can be situation when conflict does not occur (see Figure 3.18).

The set of parameters, that depand on *space_between_instances*, contains: *width_of_instance*, *width_of_action*, *width_of_condition*, *width_of_coregion* and its tails.

In order to produce well-arranged diagram the following constraints should be met:

$$space\_between\_instances > coregion\_width + coregion\_tails\_width \quad (3.3)$$
$$space\_between\_instances > head/foot\_width \quad (3.4)$$
$$space\_between\_instances > action\_width \quad (3.5)$$
$$space\_between\_instances > condition\_width \quad (3.6)$$

The general rule specified in [5] says that instances' heads of the well-arranged diagram are aligned and heads of neighbouring instances must not intersect. That is why Equation 3.4 must be realized always.

In order to solve this conflict there are two possibilities:

- to follow the value given by user for *space_between_instances*;
  The disadvantage of this option is the user defined values for *widths* of other elements are not fulfilled.
- to follow the values given for *widths* of other elements;
  The disadvantage of this possibility is the value for *space_between_instances* is not fulfilled.

As well as the previous conflict, it is recommended to follow widths of elements at the expense of the value given for *space_between_instances*.

3.9 Parameters for Arranging Time Information

In time concepts we can find time intervals and absolute times. Time interval represents the time distance between pairs of events, and thus it uses pairs of events: preceding and subsequent event. On the other hand absolute time is used to define occurrence of event at point in time that relate to the value of the global clock, therefore it is connected to only one event. [6]

3.9.1 Time Intervals

In Figure 3.19 we can see the usage of time interval. Each time interval is connected with two events. Both events determine y coordinates of the interval.

According to the abovementioned analysis and the arranging of elements, each event has coordinates computed before arranging time intervals, therefore the problem of well-arranged time intervals can be restricted to the

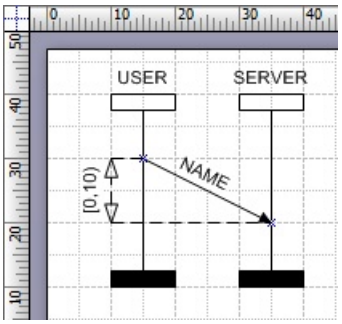

Figure 3.19: Time interval

problem how of setting x coordinates of interval relatively to the position of attached events. For example in Figure 3.19 there are two events – the upper event with coordinates (15, 30) is called preceding event and the lower event (35, 20) is called subsequent.

It holds that there are three possibilities how to arrange one interval to the pair of events:

- to arrange interval on left of the events (see Figure 3.20);
- to arrange time interval between preceding and subsequent event (see Figure 3.21);
- to arrange interval on right of the events (see Figure 3.22).

According to the drawing rules specified in [6], lines in timer symbols can overlap or cross the message symbol. That is why the second option is introduced. However, the second option is suitable only for events that do not occur at the same instance.

The comparison of abovementioned possibilities showed up that the factor influencing readability of well-arranged MSC is hidden in the number of *crossing connecting lines* and *blending intervals*. The last factor is the *length of connecting lines between events and intervals*.

It was said that connecting lines may cross other MSC elements; however, the problem occurs when the time information crosses another element. This situation may cause illegibility of the time information or of the other labels of elements (see Figure 3.23). This situation is the motivation for specifying possibility to arrange time intervals beside other elements. There are two possibilities:

- to arrange time intervals on the left of the leftmost instance;

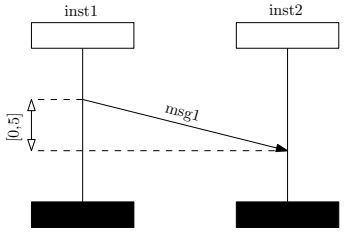

Figure 3.20: Arranging on left of events

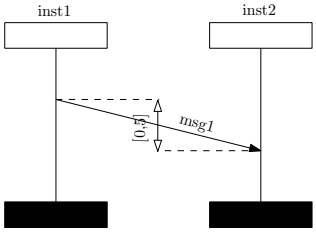

Figure 3.21: Arranging between events

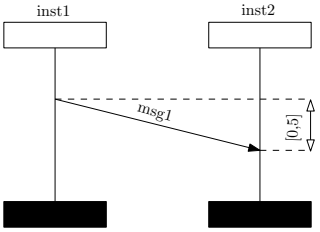

Figure 3.22: Arranging on right of events

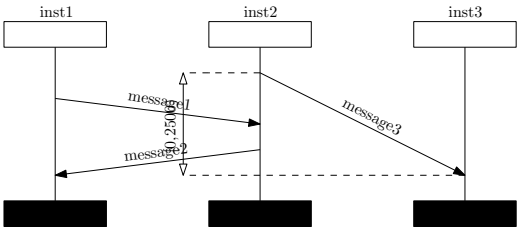

Figure 3.23: Illegible labels

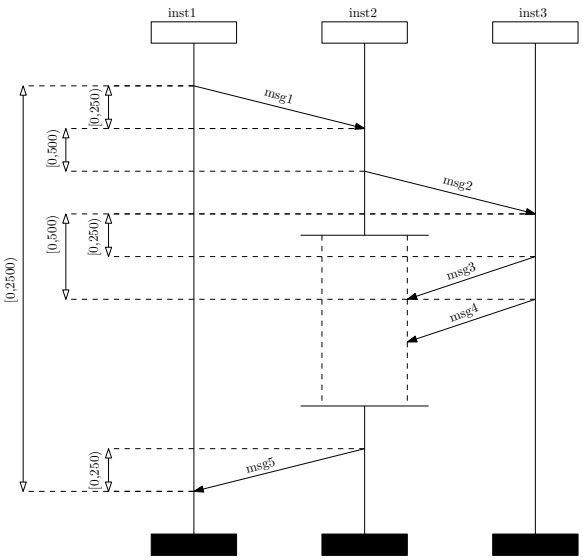

Figure 3.24: Arranging time intervals on left of leftmost instance

- to arrange time intervals on the right of the rightmost instance.

Example of the first possibility is represented in Figure 3.24. As we can see the advantage of this approach is that the communication is separated from time information. On the other hand, diagram with intervals arranged this way contains inappropriately long dashed lines connecting intervals with events which complicates readability, e.g. sometimes it is not possible to determine the attached events of the interval. This problem is the motivation for specifying another option, which says how to arrange time intervals in the diagram.

It holds that arrangement of the intervals in the diagram is difficult problem and thus additional rules must be introduced:

- time interval attached to the events of one right-going message is arranged on the first appropriate position on the left of attached events;
- time interval attached to the events of one left-going message is arranged on the first appropriate position on the right of attached events;
- time interval attached to the events on one instance is arranged on the first appropriate position in the direction of fewer instances, i.e. if the events are on the second instance and diagram contains four instances, the interval is arranged on the left;

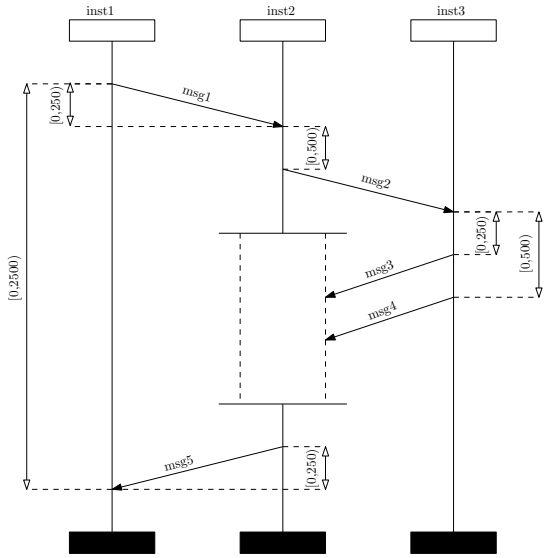

Figure 3.25: Arranging time intervals in the diagram

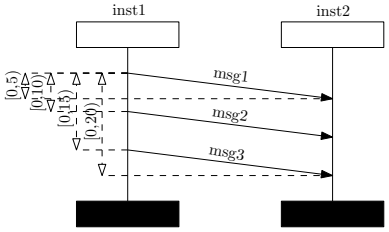

Figure 3.26: Arranging of intervals in unfitting sequence

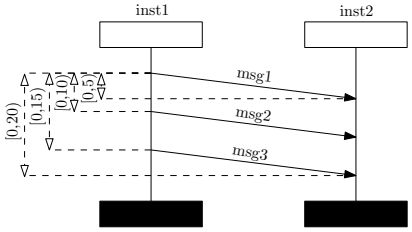

Figure 3.27: Arranging of intervals in suitable sequence

- time interval attached to the events on different instances is arranged on the first appropriate position in the direction of fewer instances.

It holds that previous list serves only to facilitate of problem. The result of application of these rules is in Figure 3.25.

In order to eliminate the number of *crossing connecting lines* of intervals we have to arrange intervals sequentially. Arrangement of narrow intervals before wide eliminates this factor (compare Figure 3.26 and Figure 3.27).

In order to eliminate the number of *blending intervals* we have created another adjustable parameters: *first_time_interval_distance* (minimal distance between the instance and the first interval) and *space_between_intervals* (distance between intervals). Figure 3.28 illustrates both variables.

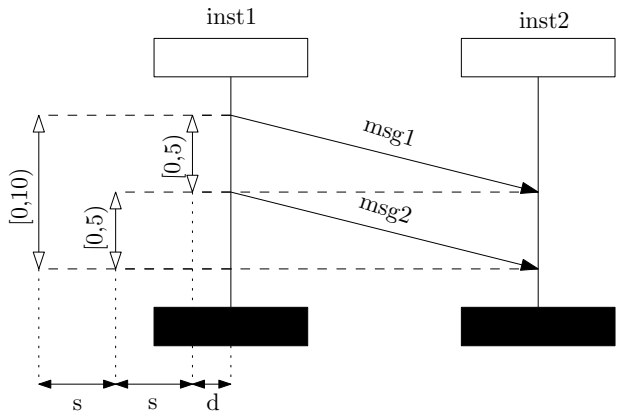

Figure 3.28: Parameters for arranging time intervals;
d- first_time_interval_distance, s - space_between_intervals

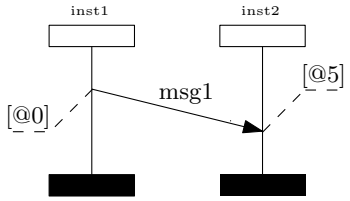

Figure 3.29: Example of absolute time

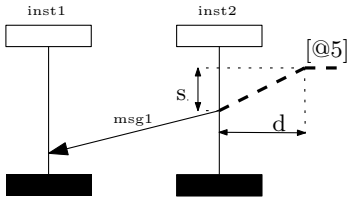

Figure 3.30: s - slope of absolute time; d - horizontal distance between event and time information

3.9.2 Absolute times

Absolute times are connected, unlike time intervals, to only one event (see Figure 3.29). As well as the arrangement of lost/found messages' dots, we can find out two parameters: horizontal and vertical distance between event and time information. The term *slope of absolute time* refers to the vertical distance. Both parameters are shown in Figure 3.30.

The following options are recommended for arranging absolute times:

- to arrange absolute times on the left of the leftmost instance;
- to arrange absolute times on the right of the rightmost instance;
- to arrange absolute times in the diagram.

It holds that rules for setting x coordinates of absolute times on the left (right) follows very similar rules as arranging of time intervals. That is why they are not described again.

The y coordinate of the absolute time is determined by the slope of absolute time, unlike the arrangement of incomplete messages' dots, the slope can be negative. It means the y coordinate of the event is less than y coordinate of time information.

4 Algorithms

In this chapter we describe algorithms for well-arranging MSC. It holds that some of them are kept from the previous implementation and thus we only provide reference to their description.

The first algorithm is called *layout_optimizer*. It deals with arranging elements on instances. The order of the instances and their placement in diagram is solved by the second algorithm called *instance_sequencer*. Both algorithms have two versions, the first one returns correct output, the second one is suitable for arranging big MSCs and thus has better run-time complexity.

4.1 Arranging Elements on Instances

In SCStudio, arranging the elements on instances is solved by *layout_optimizer*. The first version of this algorithm was created by Ing. Petr Gotthar, the second version by Bc. Zuzana Pekarčíková. The first version solves the arrangement of messages; the second version adds arrangement of coregions. The main goal of this thesis is to extend the algorithm to be able to arrange actions and conditions.

The position of the instances determines the position of the events. That is why the task of *layout_optimizer* is to compute y-coordinates of all elements that must be drawn on instances' axes.

The algorithm is based on the principle of *linear programming* and it is described in detail in [5].

4.1.1 Rewriting Problem of Arranging Actions and Conditions into LP Problem

In the following text, rules for arranging actions and conditions are provided.

To achieve well-arranged diagram with actions and conditions we use the following constraints:

- if the action or condition is the first element on the instance it must hold that the distance between upper edge of the shape and the upper edge of instance head is greater or equal to the *instance_begin_distance*;
- if the action or condition is the last element on the instance it must hold that the distance between lower edge of the shape and the instance end is greater or equal to the *instance_end_distance*;

- if the action or condition is the successor of another element it must hold that the distance between the upper edge of the shape and the element is greater or equal to the *successor_distance*;
- if the action or condition is the predecessor of another element it must hold that the distance between the lower edge of the shape and the element is greater or equal to the *successor_distance*.

Note that the expressions in abovementioned items mean the y coordinate of these elements.

4.2 Placement and sequence of instances

The previous implementation used *greedy* algorithm, which is described in [5]. According to the requirements provided by the company, we have implemented algorithm that always returns correct results. The part of the algorithm dealing with sequence of instances is described in Algorithm 4.1.

However, the time complexity of this algorithm is asymptotically equal to $\Theta(m!)$ where m is number of instances. This time complexity is computed as the number of possible permutations of given instances.

The Algorithm 4.2 describes procedure for placing instances in the page sheet. It holds that the coordinate system of SCStudio differs from the system used in Visio. In SCStudio the left upper corner has coordinates (0,0), on the contrary coordinate system in Visio may differ regarding the ruler and the left lower corner of the page sheet has coordinate (0,0). The calculation transferring between both systems is described on lines 5 and 6.

Then, the coordinates for each instance are determined by the *absolute_width* of the leftmost instance and the distance between user given value and the diagram created in graphical editor (lines 16 and 17).

Algorithm 4.1: Setting the sequence of instances: sequence(BMsc bmsc)

input : A BMsc $bmsc$
output: A sequenced vector of instances

```
1  load from register: long m_use_original_order
2  load from register: long m_use_permutation
3  load from register: float m_weight_going_back
4  load from register: float m_weight_crossing
5  set⟨Instance⟩ m_index_order
6  vector⟨int⟩ m_headers, m_headers_final
7  if (m_use_permutation) then
8      unsigned id ← 0
9      foreach instance ∈ bmsc do
10         m_index_order.push_back(instance)
11         m_headers.push_back(id)
12         id++
13 CommunicationGraph m_graph
14 float minimalRankingFactor ← 100000
15 m_graph.create_from_bmsc_with_param(bmsc, id)
16 vector⟨pair⟨int, int⟩⟩ transposition
17 repeat
18     transposition ← get vector of transpositions of permutation
19     Description values for k ← 0 to transposition.size() do
20         swapInstances(m_graph, transposition[k].first, transposition[k].second)
21         k++
22     values ← get number of going back and crossing message to the struct
23     currentRankingValue ← (m_weight_going_back ∗ values.goingBack) +
       (m_weight_crossing ∗ values.crossing)
24     if minimalRankingFactor >= currentRankingValue then
25         minimalRankingFactor ← currentRankingValue
26         m_headers_final ← m_headers
27 until there are no more permutation
```

Algorithm 4.2: Placement of the instances

input : A BMsc in graphical representation $bmsc$
output: A vector of instances with coordinates where should be projected

load from register: long m_use_left_upper_corner
load from register: long m_use_own_coordinates
load from register: float m_x_coordinate_place_value
load from register: float m_y_coordinate_place_value
load from register: float m_grid_origin_x
load from register: float m_grid_origin_y
load from register: float m_page_width
load from register: float m_page_height
$double\ start\_x$, $start\_y$
if $m\_use\_own\_coordinates$ **then**
    $start\_x \leftarrow m\_x\_coordinate\_place\_value$ + $m\_grid\_origin\_x$
    $start\_y \leftarrow m\_page\_height$ - $m\_grid\_origin\_y$ - $m\_y\_coordinate\_place\_value$
if $m\_use\_left\_upper\_corner$ **then**
    $start\_x \leftarrow 0$
    $start\_y \leftarrow 0$
$double\ current\_start\_x \leftarrow$ the x coordinate of leftmost instance's
$double\ current\_start\_y \leftarrow$ the y coordinate of leftmost instance's
$double\ shift\_x \leftarrow start\_x$ - $current\_start\_x$
$double\ shift\_y \leftarrow start\_y$ - $current\_start\_y$
$leftmost\_instance\_width \leftarrow$ the absolute width of leftmost instance
foreach $instance \in bmsc$ **do**
    set begin of the instance $\leftarrow$ get coordinates of begin of the instance + $double\ shift\_x$ + $leftmost\_instance\_width$
    set end of the instance $\leftarrow$ get coordinates of end of the instance + $double\ shift\_x$ + $leftmost\_instance\_width$

5 Grafical user interface

5.1 Motivation

For people who use Sequence Chart Studio and Beautify algorithm, it is necessary to understand the GUI as soon as possible to be able to use this tool in the most effective way. However, developers often design for what they know, not what the users know. This approach can make the user feel incapable of using the product. To avoid these problems we have to reflect perspectives and behaviours of the users, that is why we have designed the GUI in cooperation with the company.

5.2 Analysis of previous design

The previous implementation as is described in [5] consists of two basic dialog windows used for setting adjustable parameters, the first one for the import process and the second one for setting of parameters for the features used on MSC opened in graphical mode.

Both dialog windows provide setting of all parameters in one place. This causes *unfitting size* of each dialog and it also makes it impossible to include all the setting dialogs within one tree structure created for better transparency. Another problem was found in limitation of the user caused by disabling and enabling edit boxes.

5.3 Terminology

For anybody who reads this chapter, it is necessary to understand terminology used in GUI design. The table in Figure 5.1 provides description of each element used in new design.

5.3.1 Windows Template Library

Windows Template Library (WTL) is an advanced set of wrappers and productivity enhancement templates which provide comprehensive support for a wide variety of graphical user interface requirements [12]. The reasons, why the graphical user interface of SCStudio module sits above WTL, are various. The most important one is that the method that gets called is selected

Element	Explanation
radio button	allows the user to choose only one of a predefined set of options
check box	permits the user to make multiple selection from a number of options
edit box	allows the user to determine value of some parameter
spin box	allows the user to adjust a value in editbox by either clicking on an up or down arrow
track bar	allows the user to select a value on a sliding scale

Figure 5.1: GUI design terminology

at compile time, rather than run-time, with benefits in terms of efficiency and code size or uniform intuitive way of programming.

5.4 Arranging Messages and Coregions, Length of Instances, Coregions

Regarding the analysis of well-arranged MSC we have created four dialog windows. The first one was called *Message Position*. There are edit boxes for entering distances between the events on the instances. On the right side there is a group box with radio buttons for sellecting the way the length of the instances is arranged. Downmost group box contains edit boxes for entering values for length of lost/found message and slope of messages. Finnaly, the last group box enables choosing an algorithm for arranging elements on the instances.

Figure 5.2 represents abovementioned dialog window. The improvement compared to the previous design is in edit boxes' enlargements which are connected with spin boxes. This approach allows users to choose the way they want to adjust value in the edit box.

5.5 Arranging Instance Position, Spacing, Leftmost Instance

The second dialog windows is called *Instance Position* and allows to set parameters for horizontal arrangement of the instances (see Figure 5.3).

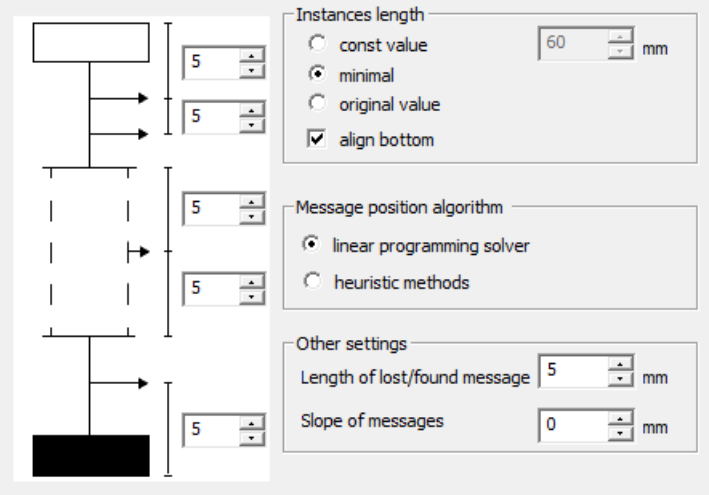

Figure 5.2: Message Position GUI

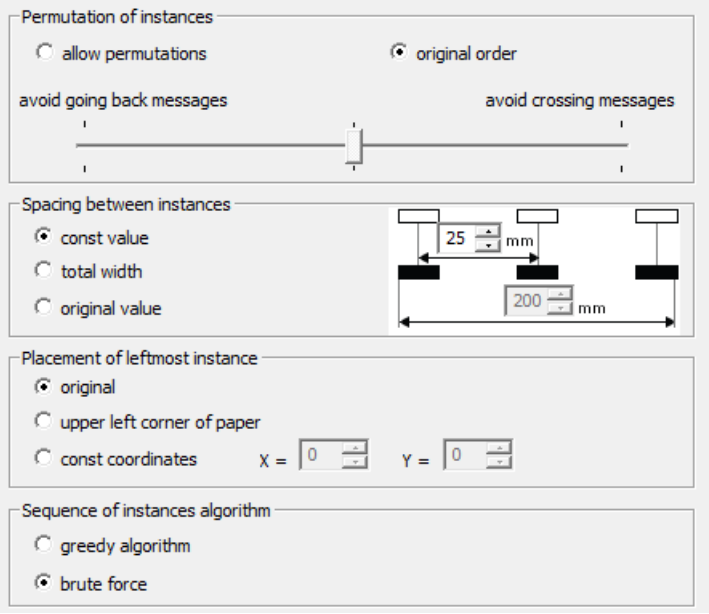

Figure 5.3: Instance Position GUI

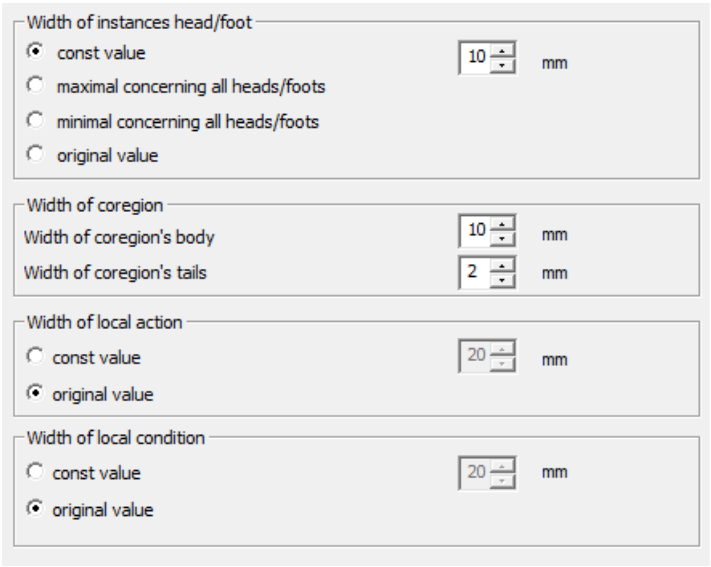

Figure 5.4: Widths of Shapes GUI

There are three groups of graphical user interface elements. The first one allows user to choose if the sequence of the instances remain unchanged or not. There is track bar enabling to choose the ratio of elimination. The second group allows setting spaces. The third group box deals with placement of the leftmost instance. Finnaly, there is a possibility to choose the algorithm for permutation of the instances.

5.6 Arranging Widths of Elements

The next window allows to choose the way the width of elements is set, i.e width of the instance, width of the action, condition and width of the parts of coregion. Illustration of this dialog window with the name *Widths of Shapes* is in Figure 5.4.

5.7 Arranging Time Intervals and Absolute Times

The last dialog window called *Time Objects* contains options for arrangement of time intervals and absolute times (see Figure 5.5).

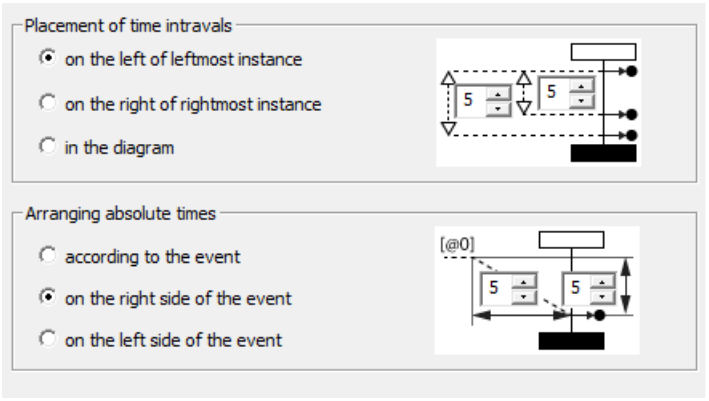

Figure 5.5: Time Objects GUI

5.8 Import process

In Chapter 2 we introduced two ways how the Beautify transformer is used. It was also said that the previous GUI design contained two dialog windows: one for the import process and one for the redrawing process.

In cooperation with the company we have decided to design and implement only one set of dialog windows. However, it holds that some predefined options are not geared to this way of usage.

The solution of this problem is that the options suitable only for redrawing the diagram created in graphical editor are replaced by other options. Following rules describe replacement of inappropriate options for import process:

- the option for not to set the length of the instances is replaced by the option allowing setting the length according to the arrangement of elements;
- the option for not to set space between the instances is replaced by the option that set the spaces equal to the constant given by the user;
- the option for remaining position of the leftmost instance in the diagram is replaced by the option placing the leftmost instance to the left upper corner of the page;
- the option remaining the width of the instances head/foot in original size is replaced by the option that set the width of the instances equal to the user given value.

6 Optimization

This chapter focuses on implementation of the *layout_optimizer* algorithm and its optimization.

First implementation of the *layout_optimizer* as described in [5] used linear programming. We have extended this implementation to support local actions and conditions. However, we found several problems with big MSCs. When the user wanted to import or redraw a document that contained more events, it took too much time and does not work at all for bigger diagrams.

The following table in Figure 6.1 provides measurement of processing times of the diagrams with various numbers of events before optimization. These measurements were executed using the 32 bit computer with Intel(R) Core(TM) i5 CPU M 430 and 3 GB of memory, running on x86_32 Windows 7.

Number of events	Processing time (seconds)
100	5
200	18
300	40
400	74
500 and more	INF

Figure 6.1: Measured data before optimization

The first problem was found in the implementation of Visio module. The screen was updated during series of actions which took too much time. This problem was solved with the *ShowChanges* property, that increases performance during a series of actions [8]. The table in Figure 6.2 provides results after Visio optimization.

As we can see in the abovementioned tables, it is impossible to get results for bigger diagrams if they contain more than 400 events. According to the debugging process we found out that the problem lies in *lp_solve* algorithm which does not return any values for these diagrams. This is the motivation for implementation of the heuristic method allowing importing and redrawing of bigger MSCs.

Number of events	Processing time (seconds)
100	4
200	9
300	22
400	42
500 and more	INF

Figure 6.2: Measured data after Visio optimization

6.1 Heuristic method

The aim of this section is to provide description and limitations of the method that allows importing and redrawing of MSCs containing more than four hundreds events. Note that current implementation only works properly for instances and complete messages, i.e. the support of coregions, actions etc. is not provided.

By using heuristics, time can be reduced when solving problems at the expense of optimality of the results. The way the method works is described in Algorithm 6.1.

The limit of this method is roughly two thousand events due to Visio which is not able to map shapes to the values provided by the heuristic fast enough.

The algorithm uses Depth-first search (DFS) algorithm to traverse all the events of the given MSC to determine the order of the events. Then it backtracks and computes y coordinates of traversed events. The table in Figure 6.3 provides results after final optimization.

Number of events	Processing time (seconds)
100	3
200	5
400	12
800	44
1000	61

Figure 6.3: Measured data after optimization

Algorithm 6.1: Heuristic method

input : A map of event indexes m_events
output: An array with y coordinates where the events should be projected

```
1  load from register: float m_successor_distance
2  load from register: float m_send_receive_distance
3  double* m_var[]
4  foreach event ∈ m_events do
5      if all successors of the event are traversed then
6          int index ← event.get_index
7          m_var[index] ← 0 if event has successor on instance then
8              int successor ← successor.get_index
9              if m_var[index] ≥ m_var[successor] then
10                 m_var[index] ← m_var[successor] − m_successor_distance
11         if event is a send event then
12             int matched_event ← matched_event.get_index
13             if m_var[index] ≥ m_var[matched_event] then
14                 if m_var[index] ≥ m_var[matched_event] then
15                     m_var[index] ←
                       m_var[matched_event] − m_send_receive_distance
```

7 Conclusion

The thesis is being developed within the scope of the SCStudio project. According to the tasks and previous analysis we have established and described options for adjustable parameters for newly supported MSC elements. We have also found out that some parameters are not independent and assignment of values or fulfilling certain combination of options may cause conflict situations and thus we provide solutions how to solve these situations.

Regarding requirements provided by the company we have dealt with the validity and efficiency of the algorithms for well-arranging MSCs. Before our implementation there ware two algorithms: one for the import process and one for well-arranging MSC in graphical mode. Now, there are two algorithms for both features. The first one provides optimal results. The second one has better run-time complexity, providing suboptimal results. It is a big step forward for the SCStudio project because it is now possible to import and redraw big MSCs.

Next, we have designed and implemented new graphical user interface. The four new dialog windows were introduced, each of them used for different group of adjustable parameters.

The part which does not work properly is the support for properly placing the time elements and absolute times.

Finally, the development of the well-arranging algorithms will also continue in the future. The next progress will be focused on adding support for other elements to the algorithm for importing and redrawing of big MSCs. As regards user-friendly graphical user interface, it is recommended to add feature that allows saving current settings and loading them when it is required (profiles).

Bibliography

[1] ITU Telecommunication Standardization Sector - Study group 17. ITU recommendation Z.100, Specification and Description Language (SDL), 2002.

[2] ITU Telecommunication Standardization Sector - Study group 17. ITU recommendation Z.140, Z.141, Z.142, Testing and Test Control Notation version 3 (TTCN-3), 2003.

[3] Ellson, J. and Gansner, E. R. and Koutsofios, E. and North, S. C. and Woodhull, G. Graphviz and dynagraph - static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE*, pages 127–148. Springer-Verlag, 2003.

[4] M. Bezděka, O. Bouda, Ľ. Korenčiak, M. Madzin, and V. Řehák. Sequence Chart Studio. In *Proceedings of the 12th International Conference on Application of Concurrency to System Design (ACSD'12)*. IEEE, 2012. Accepted for publication.

[5] Z. Pekarčíková. Computer Aided Layout of Message Sequence Charts. Bachelor's thesis, Masaryk University, Faculty of Informatics, 2011.

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

[7] The Sequence Chart Studio: SVN. [online], [cited April 14, 2012]. Available at: < `http://sourceforge.net/projects/scstudio/` >.

[8] G. Wideman. *Microsoft® Visio® 2003 Developer's Survival Pack*. Trafford Publishing, 2003.

[9] ANF DATA spol. s r.o. and Masaryk University the research center Institute for Theoretical Computer Science (ITI), Faculty of Informatics. The Sequence Chart Studio. [online], [cited April 15, 2012]. Available at: <`http://scstudio.sourceforge.net/`>.

[10] G. B. Dantzig and Thapa M. N. *Linear Programming: 1: Introduction*. Springer-Verlag, 2003.

[11] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve 5.5, Open Source (Mixed-Integer) Linear Programming system. [Software], [cited April 15, 2012]. Available at: <`http://lpsolve.sourceforge.net/5.5/`>.

[12] E. O'Tuathail. WTL developers guide. [pdf], [cited April 15, 2012]. Available at: <http://www.assembla.com/code/cristianadam/subversion/nodes/WTL%20Developer's%20Guide>.

[13] C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity relationship diagrams. *Journal of Systems and Software*, pages 163–173, 1984.

[14] L. B. Protsko, P. G. Sorenson, J. P. Tremblay, and D. A. Schaefer. Towards the automatic generation of software diagrams. *IEEE Transactions on Software Engineering*, 17(1):10–21, 1991.

A Contents of Attached DVD

The attached DVD contains the following items:

- a PDF and LaTeX version of the thesis;
- source code of well-arranging algorithms (instance_sequencer.cpp, instance_sequencer.h, layout_optimizer.cpp, layout_optimizer.h);
- source code of dialog windows;
- executable installation file of SCStudio.