MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Grafické rozložení sekvenčních diagramů

BACHELOR'S THESIS

Zuzana Pekarčíková

Brno, 2011

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: Ing. Petr Gotthard

Acknowledgement

I would like to thank my advisor Petr Gotthard for helping me during the writing this thesis. I also want to express my gratitude to Václav Vacek and Radek Sedláček for their time and suggestions during the analysis of BMSC. 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

This thesis addresses the problem of drawing well-arranged Basic Message Sequence Chart (BMSC). The problem is completely analysed. The parameters for setting distances between any elements in BMSC are described. Also the parameters for setting the size of these elements are defined. Next, a graphical user interface for setting these parameters is designed. And finally the algorithms for arranging elements in BMSC are introduced.

Keywords

Basic Message Sequence Chart, BMSC, well-arranged drawing, linear programming.

Contents

Chapter 1

Introduction

Message Sequence Chart (MSC) is a formalism for describing communication between the entities in a system. It is especially suitable for description of network protocols. It is standardized by the International Telecommunication Union (ITU-T) in Z.120 [3] recommendation.
This formalism consists of two parts: Basic MSC and High-level BMSC. Basic MSC (BMSC) can represent only a finite communication. On the other hand, high-level MSC (HMSC) can represent infinite communication. It has a hierarchical form which is made of references to BMSC and HMSC. Both types of MSC have the graphical and textual representation.
SCStudio is a tool for verifying and drawing MSC. It supports conversion from graphical to textual representation of an MSC and vice versa. After importing an MSC to the graphical interface, the elements are focused into one point. That was the motivation for drawing a well-arranged MSC.
In this thesis we discuss the problem of 'well-arranging' of BMSC, considering both theoretical and practical aspects.
The thesis is structured as follows: first we describe BMSC elements and discuss their drawing possibilities. In the second chapter we focus on analysing BMSC in order to reach good characteristics of a well-arranged BMSC. This chapter is divided into five sub-sections in which we decompose individual aspects of a well-arranged BMSC, and establish some configuring parameters. In the third chapter we introduce a user-friendly interface intended for setting these parameters. We also describe the behaviour of the system for redrawing BMSC into a better-arranged form. And finally, the fourth chapter is devoted to the implementation of algorithms intended for solving the arrangement of elements.

Chapter 2

Basic Message Sequence Chart

From [1] the following information are used. BMSC consists of set of processes (called also as instances) which are communicating via messages. An event is a send or a receive of a message on the process. The process line can consist of two forms: strict order area, coregion area. Events which belong to the strict ordered area are totally ordered from the begin of instance to the end of process. Events in coregion are not ordered in default. However general ordering relation can add ordering on the events in coregion. Messages also denote ordering of events. The send event of a message is ordered before receive of the same message. The other types of messages are incomplete messages: found and lost messages.

In graphical representation processes are usually vertical lines. They can be drawn in two ways: as a single line or with boxes on the top (*head*) and bottom(*foot*). The first type is referred to as *headless instance*, the second type as *line instance*. Coregion is represented as a box with dashed lines in place of instance line.

Messages are drawn as an arrows between processes. The place where the arrow connects to the process line is graphical representation of an event. The event where the arrow head is pointing to is the receive event, the other one is send event. If message is lost it is pointing to the black point. If message is fount it is leaving from white point.

Chapter 3

Analysis of Well-Arranged BMSC

For anybody who is reading or developing Basic Message Sequence Charts it is important to understand diagram as fast as possible. That is why the analysis of factors influencing BMSC readability was prepared in cooperation with the employees of ANF Data. This analysis can be used as a foundation for developing the feature well-arranging the BMSC. All factors which were discussed during the analysis are described in this chapter. There are also suggested adjustable parameters for each factor. In this way, a user can influence a form to which a BMSC will be drawn by the feature.
It holds there are different situations in which the feature well-arranging the BMSC can be used. And depending on situation, the way of setting parameters may differ.
The first possible usage of the feature is when the user works with *BMSC in graphical form and wants to change the appearance of it*. And since the graphical form stores the appearance of elements, it is possible to set parameters according to the *value of their occurrence in diagram*. On the other hand it is also possible not to use the graphical information and set parameters to *constant values given by user*.
The structure of BMSC in its textual form does not include any information of elements' appearance. So without using the feature, everything would be drawn in one place. Therefore the other usage is the *after importing BMSC from textual into graphical form*. The feature spreads the BMSC elements up according to the adjusted parameters. In this case any graphical information can be used, so all parameters have to be set to *constant values given by user*.

The first subsection covers *arranging messages and coregions on instances, length of instances and length of coregions*. These three factors are together in one subsection because they are cohered with each other. The second subsection deals with the *positions of instances*. The *size of instances' heads and foots* and *the width of coregions* are written in continual two subsections. And the last factor, *a form of labels*, is analysed in subsection 3.5.

3.1 Arranging Messages and Coregions, Length of Instances and Coregions

The begin and end of instance, events and coregions' edges can be considered as elements of instances. Then it can be said that the factor influencing BMSC readability - 'arranging messages and coregions' refers to the setting distances between the elements on instances. This factor is described together in one section with two other factor - 'length of instances'

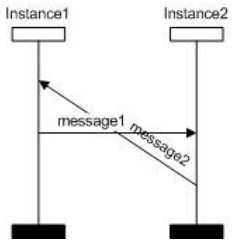

Figure 3.1: Cyclic BMSC

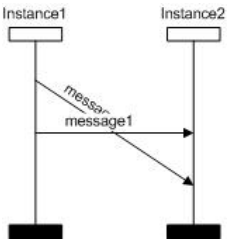

Figure 3.2: Non-fifo BMSC

and 'length of coregions'. All of these factors are cohered with each other: The length of coregion is created by distances between elements related with it. And together the distances on instance create the whole length of this instance. On the other hand, as it is described latter in text, distances can be counted from the length of instance.
In the following text are at first described adjustable parameters for setting the distances between elements on instances and then the possibilities of setting the instances length.

3.1.1 Parameters for Arranging Messages and Coregions

The first parameter which describes the arrangement of instances' elements is the *vertical distance between the messages' edges*. For the complete messages it is space between the received and and the sent event and for incomplete messages it is space between a dot and a received/sent event, Figure 3.3. By setting this space it is possible to coordinate messages' slope and in this text, the term 'slope of message' implies to this distance.
According to Z.120, messages going upwards are forbidden. So the messages will be either horizontal or going downwards. The only exception is the cyclic BMSC (Figure 3.1) which cannot to be drawn without any up-going message.
The slope of messages in BMSC can vary or can be equal. Sometimes, by using different slope, it is possible to express any property of messages. For example, in Figure 3.2, there is a non-fifo BMSC, and different slope means different delays of message1 and message2. Another example is when a BMSC is cyclic (Figure 3.1). For its expressing, there has to be (at least one) message with a different slope. In these and similar situations, messages with different slopes are required, and it is also not possible to draw them with the same slope for all messages. On the other hand, sometimes it is possible to draw all the messages with the same slope or with different slopes. In such cases, the user can choose what he likes better and by using parameter *slope of messages* he can set the the value of the same messages slope.

Next parameters describing the arrangement of elements are *distances between two elements on one instance*. Because there are different elements on instance, there are also different parameters for setting the distances between them:

- *head_distance* - distance between the beginning of an instance and the first element

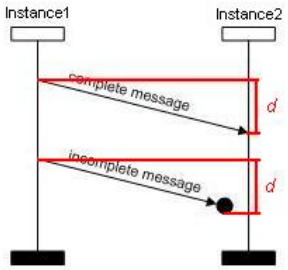

Figure 3.3: *d* - the distance refers to the slope of message

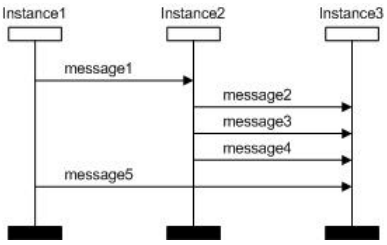

Figure 3.4: Equal slope of messages but different distances between events

laying on this instance

- *foot_distance* - distance between the last element laying on an instance and the end of instance
- *coregion_begin_distance* - distance between the beginning of a coregion and the first event laying on this coregion
- *coregion_end_distance* - distance between the last event laying on a coregion and the end of the coregion
- *successor_distance* - distances between two events/coregions or between event laying above/under a coregion and this coregion

As well as for the slope of messages, distances between elements on one instance can be set to equal or different values throughout the BMSC. However, because the instances can have different number of elements, in some situations it is not possible to keep both the same slope and the same distance (Figure 3.4). From the figure it is obvious that the same slope of messages has bigger benefit, its setting is prior to setting the spaces between elements on one instance. And the abovementioned parameters refer to the minimum values of distances. They are always at least as big as these parameters and as close to them as possible (with respect to slope of messages).

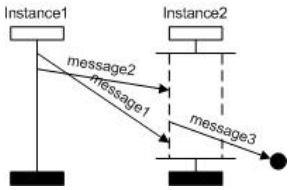

Figure 3.5: Bad jointed messages to coregion

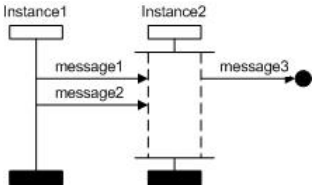

Figure 3.6: Good jointed messages to coregion

The last thing which influences the arrangement of messages and coregions is the *way of connection between messages and coregions*. In SCStudio, a rectangle is used as a coregion symbol. Therefore, situations in which a message jointed to any coregion cuts this coregion into two parts can occur, like message3 in Figure 3.5. It can be solved by attaching messages to correct side according to orientation of message (Figure 3.6).
Also, as it is described in Chapter 2, the order of events in coregion is not strict. Therefore the events in the coregion can be drawn in arbitrary order without changing the meaning. So, in the example in Figure 3.5, the sequence of messages 1 and 2 can be changed in order to not cross each other (Figure 3.6).
For setting those situations a user can choose if he likes detects them or not.

The abovementioned parameters describe the positions of events and coregions on instances. They cohere with each other and so they can be set together to following options:

- *Do not change arrangement of messages and coregions*
  This setting can be used only for modifying graphical form of BMSC because it works with graphical information.
  The user can set the positions of events and coregions by his hand as he likes. The BMSC-well-arranging feature will not change it.

- *Set arrangement of messages and coregions according to given constants*
  The BMSC-well-arranging feature can be used in any way (after import from textual representation of BMSC or reorganization its graphical form). The user adjusts all distances between elements. After using the feature, messages jointed with coregions will not cross and will be attached to correct sides, spaces between elements on instances will be at least as big as parameters and all messages will have the same slope as given constant. The only exception of the slope holds for the cyclic or non-fifo BMSC in which there will be at least one message with different slope.

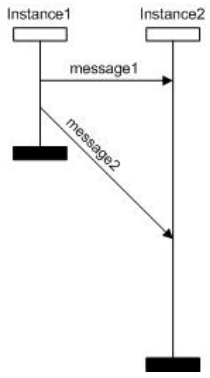

Figure 3.7: A user would like to short Instance2 without arranging messages

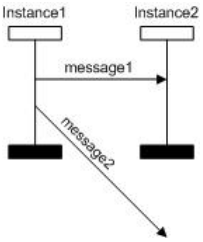

Figure 3.8: Shortening length of Instance2 without arranging messages, individual spaces on instance stay untouched

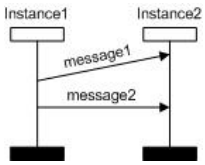

Figure 3.9: Shortening length of Instance2 without arranging messages, individual space on instance proportionally short

3.1.2 Length of Instances

The term *length of instance* refers to the length of axis and it is the next settable parameter for well-arranging BMSC. Regarding Z.120, the length of every instance must be big enough to all coregions and events belonging to it could be drawn in.
For setting length of instance it must hold that messages and coregions are arranged. It is in reason to avoid conflict situations like the following one: In Figure 3.7 there are two instances. If the length of instance2 is cut to the length of instance1 without arranging messages, one event is drawn under the end of instance (Figure 3.7), which if forbidden by Z.120. This situation can be solved by the proportional shortening of distances between the elements on instance2, as it is shown in Figure 3.8. However it causes that one of the message is up-going - which is also forbidden by Z.120.

For setting the length of instance there is declared new parameter - $instance\_length$. In the following text there are described possibilities of setting this parameter:

- The first option is setting the length of instance *according to the arrangement of messages and coregions*. It means that neither the events nor the coregions overreach the end of the instance and all distances between elements are fulfilled. In addition, Because there are five types of individual distances between elements, the advantage is that a user can better coordinate the space on instances.

The length of instances can be different throughout BMSC. However, there is no special reason for different length of instances for this case. Therefore they are align to the bottom according to the furthermost instance.

- The next possibility is setting instances' lengths equal to the *constant given by user*. It is good especially when a user would like to print a diagram. He can set the length of the diagram equal to the size of paper.
  The parameters for arranging elements on instances are uniform and their size is counted from arrangement of messages and coregions to be as big as possible and no event/coregion is drawn under the end of instance.

- The special case of previous setting is setting the length *according to the occurrence of the first instance in diagram*. 'The first instance' refers to the leftmost one. If there are more instances laying on the leftmost position, then the topmost of them is selected.
  The advantage is that a user does not have to know the exact value in millimetres, besides, he can see this length in diagram and the rest of the instances is set automatically by the feature.

- The last option is *to not change the length at all*. It is useful when a user would like to change another factor influencing BMSC readability. In this situation, a user can arrange the messages and coregions but he does not have to. If messages and coregions are arranged, this setting becomes only a special case of setting the length to the constant value.

Only the first two options do not work with graphical information of BMSC. For setting the feature well arranging BMSC after its import form textual mode is used the first option. The reason why it is not used the second option is, that a user does not always know how many messages lay on instances. So it can happen that he enters very small constant and a BMSC will not be well-readable.

The length of instances can be also set to other different values (average length, length of instance chosen by a user,...). Because of space restrictions of this thesis, they are not described here.
For setting length of instances there can occur the situations like in the Figure 3.10. To avoid this situations it must hold that the length of instance is at least as big as together the length of head and food.

3.1.3 Length of Coregions

Length of Coregions is another adjustable parameter. As for instances, even for coregions it holds that the length must be at least so big that all events belonging to it could be drawn in.
In the following text there are described two possible settings of the coregion length.

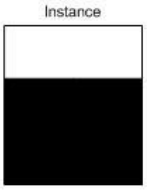

Figure 3.10: Instance is too short

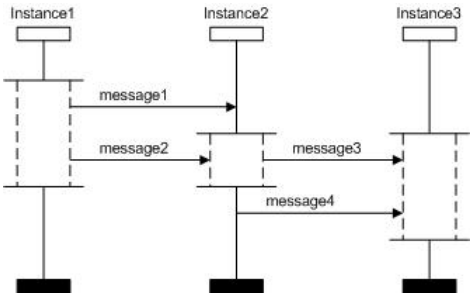

Figure 3.11: Contradiction of constant length of coregions

- Setting the length of coregions *according to the positions of events belonging to them*. The length of coregion is big enough for all events belonging to it being drawn on it, and so the individual distances on coregion are fulfilled. The advantage is that the user can coordinate the space on coregions.

- Next possibility is to set the length of coregion to the original values. For this setting it holds that a user can neither arrange the messages and coregions nor change the length of instances. It can be used if a user would like to change the the length of each coregion separately.

Further possibilities could be: setting the length to the constant value or its special case - according to its occurrence in BMSC. However, in contrast to the instances, it is not always possible to set the same length for all the coregions. One of these situations is shown in Figure 3.11. This option could be changed to setting the minimum length of coregion and each coregion would be at least as big as this value. However, because generally the setting does not bring a big profit for readability of BMSC it is not part of options.

3.2 Heads and Foots of Instances

According to Z.120, if head and foot are both part of the instance, they must have the same scale. Because their shape is rectangular, the parameters adjustable by user are the *width* and the *length* of this rectangle. They have the same possibilities of setting, and so the length

and the width are analysed together in the following text. However for this thesis the feature does not set the length of heads/foots because it is not part of BMSC structure in SCStudion. It holds that the first setting does not need any graphical information of BMSC. Therefore it can be used either as a setting of the feature arranging a BMSC after being imported or as a setting of the feature arranging BMSC opened in graphical mode. On the other hand, the latter settings work with BMSC appearance, so they can be used only for arranging BMSC in graphical form.

- *constant value given by user*
  The size of the width and the length of all heads/foots will be set to the constant given by user.
- *original size* There are some situations in which a user would like to express any property of instance. He can do it by changing the scale of head/foot. The example is shown in Figure 3.23. The situation where a server has a special status in this BMSC is represented by bigger head/foot. A user can manually set different sizes of heads/foots throughout BMSC. The feature, for redrawing BMSC into well-arranged form, does not change this setting and keep the original sizes.

- *setting the size according to the first occurrence of head/foot in BMSC* 'The first head/foot in diagram' refers to the head/foot laying on the leftmost, topmost instance. The value of it is set to all heads/foods.

It has to be mentioned that there are also other possibilities. For example, setting the scale to the value of the smallest (biggest) head or foot in diagram. Or setting it to the average value counted from all heads/foots in BMSC. However, finding the smallest (biggest) head/foot can be difficult for the user. And the average value can be different from all sizes of heads/foots in BMSC, so a user can hardly imagine how big it is. Therefore, they are not recommended and so they are not described in details.

According to Z.120 the label of instance can be placed inside instance head. Then there can occur a conflict situations in which label of instance cross the bounder of instance's head. It can be solved either by enlarging the scale of instance head or by shorten the size of labels. When a user works with BMSC in graphical mode and this conflict occurs, it is recommended that the feature asks a user what he likes better. The reason is that changing the size of heads/foots and changing the font of instances' labels is equally difficult. On the other hand for the feature used after it had been imported from textual form it is recommended not to ask a user but enlarge the sizes of heads to no conflict occur. The reason for not asking is because of the speed of the importing process and the reason for the enlarging the size of heads/foods is the readability of instances labels.
Because for now in SCStudio the labels are not placed inside instances' heads, this situations can not occur.

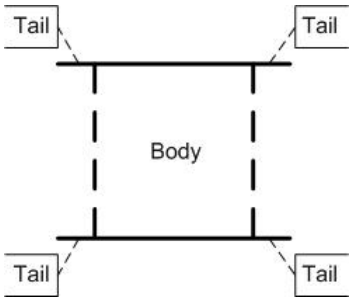

Figure 3.12: Parts of coregion

3.3 Width of Coregions

As it is shown in Figure 3.12, the symbol of coregion can be divided into two parts, *body* and four *tails*. It holds that all tails of one coregion must have the same size.
To describe the width of Coregions there are introduced two parameters, the *width of body* and the *width of tails*.
Because these two parameters are cohered with each other (they together create the width of whole coregion), in some situations it would be useful if setting of one parameter influenced the setting of the other one. Because a main part of coregion is its body (messages are joined there), the setting of body will influence the setting of tails. However, sometimes it would be more suitable to set every parameter separately.
All the following settings are suitable when the feature well-arranging BMSC is used for modifying an opened BMSC in graphical form. However, when the feature is used after BMSC had been imported from textual form, not all of mentioned settings can be used. The most suitable and simplest possibility for this situation is setting both types of width to the same constant value given by user.

Body of coregion
There are three different points of views on coregion. At first it can be considered as a self-contained unit or as a part of instance and also as a part of whole BMSC. Depending on the point of view, there are different possibilities of setting the body width.

1. *Original body width of each coregion*
   When a user wants to set different width of body for each coregion, he can do it manually. If he wants to redraw a BMSC to be more readable, he can choose this possibility and the width of coregions will not change.
   This setting is good especially when a user wants to express some coregions which do not lay in the same instance. (Figure 3.13)

2. *The same width of body for all coregions on one instance*
   All coregions laying on one instance will have equal value of width. This value can be either actual width of any coregion from the set of coregions, or it could be calculated

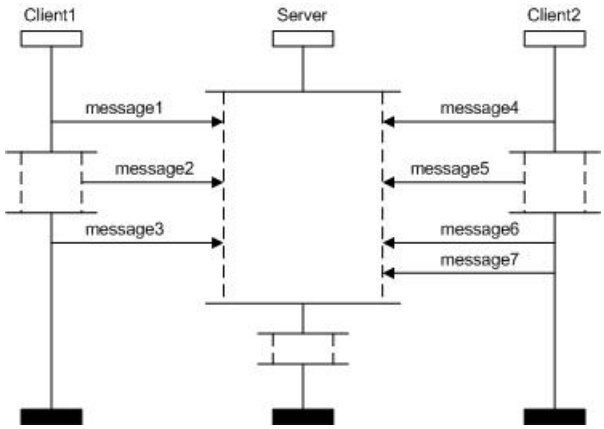

Figure 3.13: Original body width of each coregion

according to some applications within the instance.
By setting the same width to all coregions on one instance a user can visually separate this instance from another. For example Figure 3.23. This setting is not used for the feature because its usage is suitable only for special situations, not widely.

- Width of the first coregion body in one instance
  'The first coregion' is the topmost coregion laying on each instance. The width of this coregion will be set to all coregions in this instance. Coregions in different instances can have a different width.
  The advantage of this case is that a user can *set* the width of the first coregion in one concrete instance as he likes. The other coregions on this instance will adapt to it. And this holds for all instances throughout BMSC.
  The advantage of this case is that a user can *see* the width (it is the minimum width) which will be set to all coregions in one concrete instance. And this holds for all instances throughout BMSC.
- According to width of instance head
  Width of coregion is counted according to the width of instance head.
  This setting illustrates a dependence of coregions on instances. The instance is superior because it can exist without any coregion but it is not possible vice-versa.
  The advantage of this setting is that by changing the width of instance's head, the width of coregions laying on this instance will change automatically.
  According to method of how to calculate the width of coregions from instance head, the setting can be divided into two types:
  - settings by coefficient
    Coefficient c given by a user describes how many times the width of coregion is bigger than the width of instance head. If we refer to the width of instance

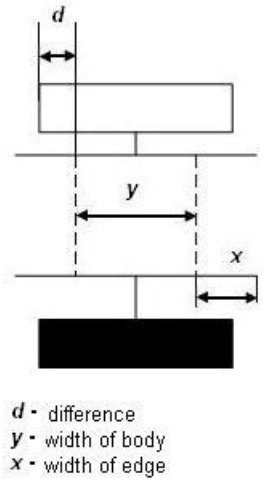

Figure 3.14: Two types of coregion width

head as to h and to the width of coregion's body as to y, we can write:
$y = c.h$.

– settings by difference
Difference d describes vertical distance between the vertical edge of coregion and the vertical edge of head (Figure 3.14). And the width of coregion can be calculated as:
$y = h + 2d$

3. *the same width for all coregions over the BMSC*
The last possibility is when the same value to all coregions in BMSC is set. Again, according to the way of obtaining this value, there are three different alternatives:

- Constant value of width given by user
The width of all coregions is set to a constant value which is entered by user.
- Width of the first coregion body in whole BMSC
'The first coregion in BMSC' is found on the leftmost instance which contains coregion and the topmost coregion on this instance. Width of this coregion is set to all coregions in BMSC.
The advantage of this setting is that a user sets a width only of one coregion, he can see the value of it in diagram and the rest of coregions width will be set automatically.

Again it holds that the mentioned settings are not complete. From the collection of coregions it could be selected coregion by using other criteria. However, for simplicity there are

described only the basic possibilities.

Tails of Coregion

Because the structure of BMSC in SCStudio does not include the width of tails, for this thesis the feature does not deal with setting this type of coregions' width.
As for instance's head and foot, also here it holds that a format of them must be equal. The possibilities of settings are:

- Constant width of tails given by user
  This possibility sets the value for all tails, it is equal to the value of a constant. It is entered by user.
- Multiple given by user
  As it is written at the beginning of this subsection, it is good if width of tails depends on the width of body. An advantage of this solution is that it is enough to change the width of body and the width of tails will change too.
  The size of tails is a multiple (c) of the coregion body's width. If we refer to the width of a coregion body as y and to the width of tails as x, we can write an equation for counting the width of tails as:
  $x = c.y$
- Constant width of tails given by user
  All widths of tails will have the same value. It is necessary to bear in mind that it does not mean that coregions will have the same format. They still can have different bodies.
- Original width of coregion's tails
  A user can set the width of tails as he likes. He does it by hand and the feature for redrawing BSMC will not change width of any tail.

3.4 Placement and Sequence of Instances

This section introduces parameters describing how to arrange instances on the plane. At first there are written parameters influencing the sequence of instances. In the next subsection vertical and horizontal spaces between instances are described.

3.4.1 Sequence of Instances

This section focuses on the order of instances. In some situations, changing the order of instances makes the BMSC structure more comprehensible.
Figure 3.15 shows four processes in which messages exchanges occur. At first, it is not so clear that there are two sets of processes which are not related. However, after switching Instance2 and Instance3 (Figure 3.16), it becomes obvious that processes 1 and 3 do not depend

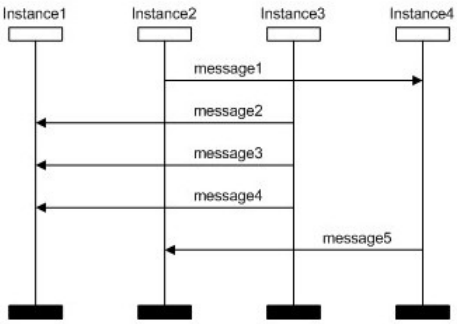

Figure 3.15: Unordered BMSC

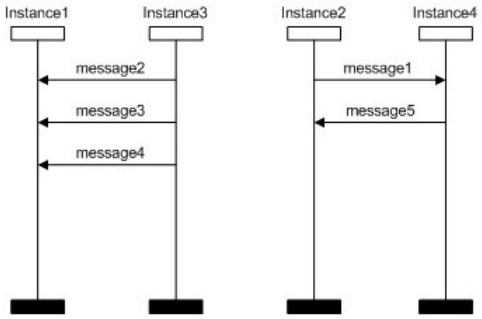

Figure 3.16: Communication processes close to each other

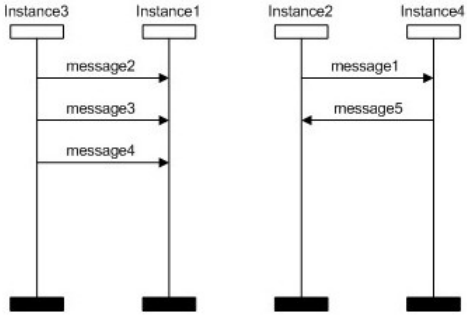

Figure 3.17: Ordered BMSC

on processes 2 and 4. According to this observation, *the more processes communicate between each other, the more important it is for them to be close to each other.*
It needs to be said that the 'distance between two instances' does not mean an actual distance in millimeters, but rather a number of other instances which are between them.

Next, in Figure 3.16 it can be seen that the messages being sent from Instance3 to Instance1 are drawn from right to left. However, people are used to read from the left side to the right side. So the sequence can be again modified as it is shown in Figure 3.17. And from this observation there is declared new parameter the *orientation of messages* which influences the desired sequence of instances.

In general, the process can communicate with more than two other processes. In these situations, it has to be decided which of the instances will be placed further and which one closer to this process. Let's analyse another example which is shown in Figure 3.18.
In this example, there are three possibilities it which the instance will be further from Instance2. The cases, in which Instance1 and Instance3 are the further ones are identical, so we

Figure 3.18: Order of instances with 3 crossings

Figure 3.19: Order of instances with 1 crossing

can consider only two different possibilities shown in Figure 3.18 and Figure 3.19.
The communication between processes 2 and 4 is stronger than the communication between processes 2 and 3. The power of communication is determined by the number of messages which are being exchanged between the instances.
From Figure 3.18 and Figure 3.19 it is obvious, that the relation between the power of communication and the distance between the instances can be expressed by a number of *crossings of instances and exchanging messages* which is the next defined parameter.

So, in the end, there are two parameters influencing the order of instances: 'number of crossing instances with messages' and 'number of going-back messages'. In general, there are situations in which it is not possible to repair only one of those factors (Figure 3.20 and Figure 3.21). Every person could consider different sequence to be good, and thus there is a possibility of eliminating:

- *the number of crossing instances with messages*
- *the number of going-back messages*
- *both number of crossing and going-beck messages*
  If it is not possible to eliminate both parameters, the feature choose one of them. Otherwise eliminate both of them.
- *neither the number of crossing nor going-beck messages*
  The sequence is not changed.

Textual mode of BMSC does not save the sequence of instances. For simplicity, the feature arranging BMSC after importing from textual form does not ask a user what he wants to eliminate, but computes the best sequence and eliminates the both parameters.

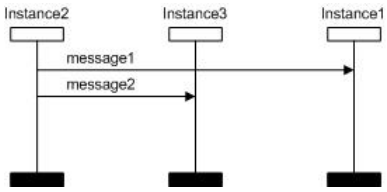

Figure 3.20: BMSC where crossing is not fullfilled

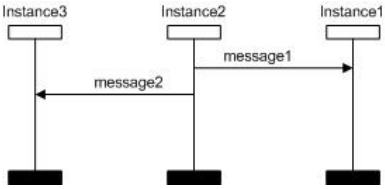

Figure 3.21: BMSC where orientation of messages is not fullfilled

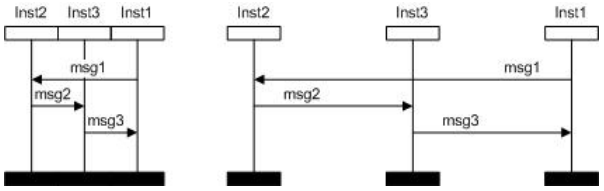

Figure 3.22: Different space between instances

3.4.2 Space between Instances

Vertical spaces
Nonzero setting of a vertical distance between the instances' heads does not contribute to the readability of BMSC. Therefore all the instances' begins are set to the same level. For arranging the BMSC opened in graphical mode, this level is stated by the position of the first instance.
When the feature is used for arranging BMSC after its import from textual mode, the position of the first instance is set by the program to the left top corner.

Horizontal distances
As it is seen in Figure 3.22 and Figure 3.23, the readability of BMSC is different with different setting of horizontal distances. Each of them can be suitable in different situations. Therefore for this distance it is declared new parameter - *space between instances*.

- *Setting the space to constant given by user*
  The user enters the constant value of a space. The distances are the same between each two instances. This setting is used for the feature arranging BMSC after it had been imported because the following ones are not suitable.

- *Setting the space according to the total width of instances*
  The user sets the width between the leftmost and the rightmost instance. The space between each two instances is computed automatically. This setting is useful, for example when the user likes to print the diagram. He sets the value of the total width to the width of paper. However, because a user can mark all instances and shorten the width of their spacing very easily and because in compare with the abovementioned

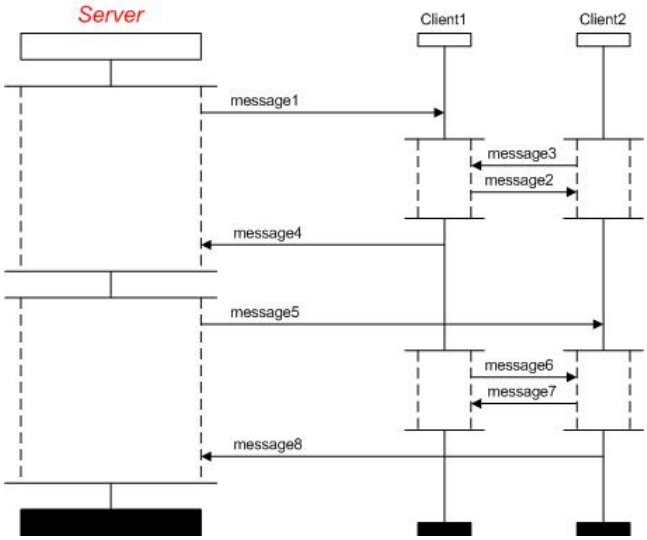

Figure 3.23: The instance Server is separated from instances Clients

setting it does not bring anything else, it is not part of the setting the feature used in any way.

- *original spacing of instances*
  By setting different gaps between the instances it is possible to express some properties of certain instances. As shown in Figure 3.23, the server is further from clients. It implies that it has special status, whereas clients are of equal importance. The spaces between the instances a user sets by himself and the feature does not change them.
  When the sequence of instances is changed, this setting does not make sense, so for such setting changing sequence of instances is forbidden.
  Because it works with the appearance of BMSC it can not be used when the feature arranges the BMSC after its importing from textual mode.

- *minimum spacing according to the labels of messages*
  The distance between each pair of instances is set to the minimum value, so that no label of the exchanging message crosses any of these instances or is behind(before) the message.
  This option is not part of the feature interface, because a size of messages labels in pixels are not included in data structure of BMSC in SCStudio.

It can happen that the unwanted situations occur: label of any message overreaches this message. The example is shown in Figure 3.24. To solve this problem, the spaces between instance can be enlarged or the font of labels can be changed. The recommendation is always to enlarge the spaces because a user can easily change the spaces between instances by marking all instances, however setting the font is complexer. The very similar problem

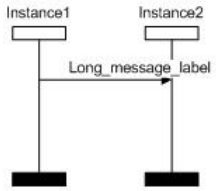

Figure 3.24: Message label overflies edge of the message

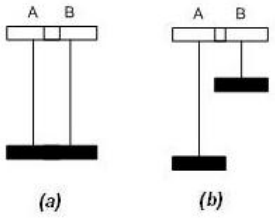

Figure 3.25: Conflict situations: head vs. head

is when labels of two instances are crossing. And it is recommended to solve it in the same manner. Because the size of labels in pixels are not included in BMSC data structure in SC-Studio, the feature not to solve it.
It can also occur the situations in which any part of the instance head, foot or coregion is overlapping the part of other instance. The examples are shown in Figure 3.25, Figure 3.27, Figure 3.26. The solution of this problem is the same as for the abovementioned conflicts, however because the width of coregions and heads/foots is part of BMSC structure in SC-Studio, the feature used in any way detects those conflicts and repairs them. It holds that if the spaces of instances are uniform throughout BMSC, the all spaces are changed. On the other hand if the spaces are different throughout BMSC the space between two instances enlarge only if its necessary.

3.5 Labels

In BMSC only two elements are jointed with labels, instances and messages. A format of these labels also influence the readability of BMSC. So in this section the parameters sufficiently describing them are mentioned. For the thesis the features does not deals with the labels.
Because the labels of both instances and messages are writings, the first parameter is a *font of these writings*. The setting of font is the same for both messages and instances so it is together described in the following text. Next parameter is a *position of labels*. Setting of this parameter is different for instances and messages so it is described individually.

Font of Labels

- *Original labels font of instances/messages*
  In situations when a user would like to express any property of instance(-s)/message(-s) he can do it by changing the font of labels by hand. (Figure 3.23) With using this setting of labels the feature for redrawing BMSC will not to change the font. For the usage the feature after BMSC import from textual form it can not be used.

- *Choice from dialogue*

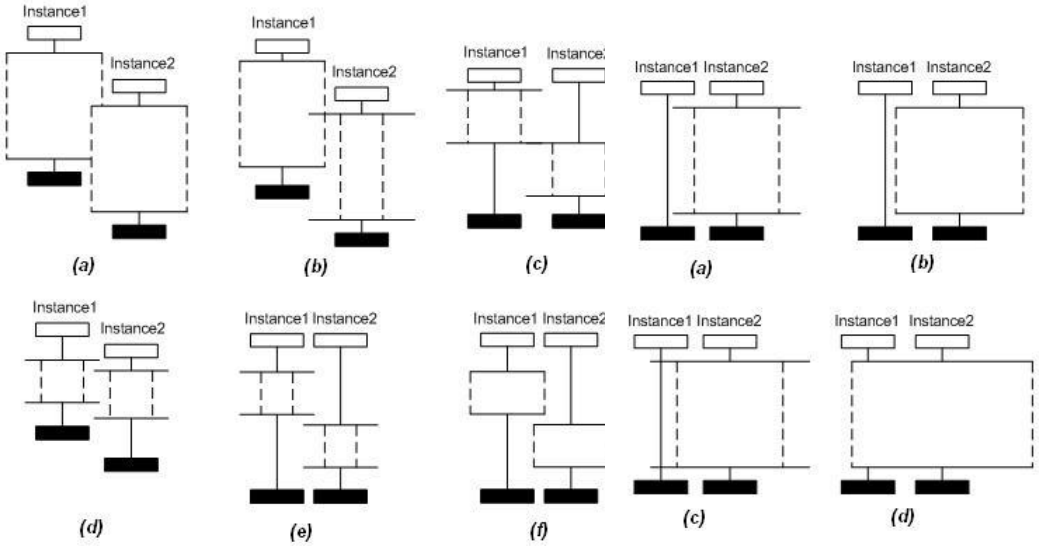

Figure 3.26: Conflict situations: coregion vs. coregion

Figure 3.27: Conflict situations: head vs. coregion

A user set the font as he likes and then it will be set to all labels of instances/messages. This setting is good for uniformity of BMSC. This setting can be used for the feature used in any way.

Position of Instances Labels

According to Z.120, position of instances labels has only two possibilities:

- *Inside instances head symbol*
- *Above instances head symbol*

Position of Messages Labels

Z.120 does not define the exact position of message label. So it should be anywhere. For managing both directions of the position, there are introduced two parameters *alignment* and *vertical distance from message*.

For labels of messages there can occur a conflict situation when a label crosses other elements in the BMSC, except for the elements which are crossed by a message of this label. Conflict of crossing labels and any element in BMSC is recommended to solve only as a static check, because crossing any graphical objects in plane is a difficult problem. From distances between elements on instance and from the slope of message a rectangle is calculated (Figure 3.28), in which label must lay on. This check is calculated only from messages whose slope is the same.

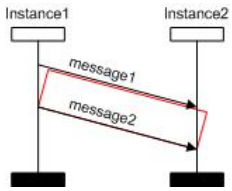

Figure 3.28: A rectangular useful for static check of message label

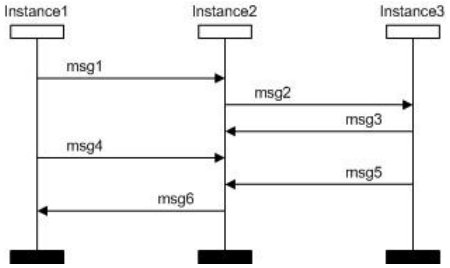

Figure 3.29: Alignment of messages labels to send event

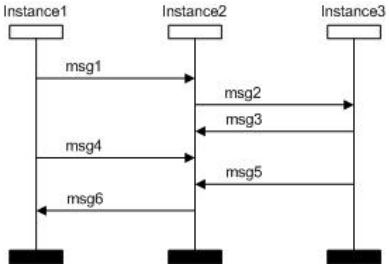

Figure 3.30: Alignment of messages labels to left side

Messages Labels Alignment

There are two different points of view at labels of messages which can influence the setting of alignment. The first is that a label is a part of message which has the direction from send to receive event. And the second point of view is the label as part of whole BMSC. Regarding the point of the view there are next settings:

- *alignment to send/receive event*
  Example of this setting can be seen in Figure 3.29 where the labels are align to the send event. In this setting all labels of messages are close to send/receive event. Alignment is comprendious in regard of messages.
- alignment to right/left side
  Left alignment of labels is shown in Figure 3.30. All labels of messages are close to left/right side. Alignment is comprendious in regard of whole BMSC.
- centre alignment
  Alignment is comprendious in regard of both messages and whole BMSC. The labels are in the middle of the message. Example is shown in Figure 3.31.
- original alignment
  The base setting. A user sets labels anywhere he likes by hand and the feature for redrawing BMSC into more tabular form will not change the horizontal position of message labels (Figure 3.32).

For alignment to any side or event it is necessary to say how far the label will be from instance (we mean horizontal distance). For this distance, there is defined another parameter

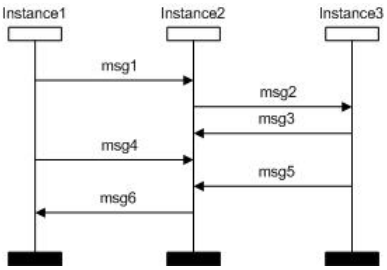

Figure 3.31: Center alignment of messages labels

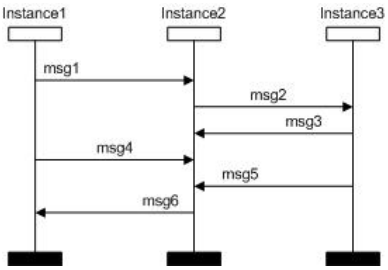

Figure 3.32: Original alignment of messages labels

- an *indent*. The indent has to be equal throughout BMSC (otherwise this setting would come under *original setting alignment by user*). And the value of indent can be taken from indents in BMSC or given by user:

- *Constant value of indent given by user*
  A user enters a value of indent which will be everywhere throughout BMSC. This value can be arbitrary number.

- *Minimum indent from all indents in BMSC*
  If indents in BMSC are not equal the minimum one is found and set throughout BMSC.

- *Maximum indent from all indents in BMSC*
  The same as for minimum only the maximum value is found.

Vertical Distances between Labels and Messages

This distance is between message and a bottom of label of this message. There holds the same rules as for indent (which can be named in other way: horizontal distance label from message). The only one difference is that there are allowed different distances throughout BMSC.
Following division describes possible settings of vertical distances between messages and their labels. The first three possibilities are homogeneous and the last one is heterogeneous.

- *Constant value of vertical distance given by user*

- *Minimum vertical distance between label and message, regarding all distances in BMSC*

- *Maximum vertical distance between label and message, regarding all distances in BMSC*

- *Original vertical distances between label and message*
  The distances are set by user's hand. The feature will not change these distances.

Chapter 4

Graphical User Interface

This chapter deals with the interface for the feature arranging elements in BMSC. It should be clear and simple so that a user could easily achieve his goal. There are proposed windows for setting parameters of factors influencing BMSC readability, which were described in Chapter 3 and are parts of BMSC structure in SCStudio. To make the interface clear and simple, these windows do not include all possibilities of setting. The reasons why there are used exactly these options are described in analysis in details.

As it is written in Chapter 3, the feature can be used either for arranging BMSC immediately after its import from textual mode or for arranging elements of BMSC opened in graphical mode. Because each of these usages has different way of setting, one window, marked as window$_1$ (Figure 4.1), is proposed for setting of the feature which is used after BMSC import. The second window, marked as window$_2$ (Figure 4.2), for setting the feature arranging BMSC opened in graphical mode.
Window$_1$ includes only parameters adjustable to constant values given by user. In addition, window$_2$, includes settings of these parameters according to any graphical information of BMSC. Window_1 has a picture of simple BMSC with highlighted distances and small boxes, for entering constant values, next to these distances. The same picture is in window$_2$. However, there are more possibilities of setting parameters which are grouped in rectangles on the right side. According to used setting, some parts of interface can be inactive or active.

4.1 Arranging Messages and Coregions, Length of Instances, Coregions

Regarding window_1, there are boxes for entering distances between the elements on instances and the length of incomplete messages. The feature arranges the messages and coregions on instances according to the individual spaces. The length of incomplete messages is equal to the value of parameter, the length of coregions depends on the arrangement of messages attached to it. The length of instances is the same throughout the BMSC and its value is as small as possible no event nor coregion lies over the end, and the minimum distances, defined by parameters, are observed.

In window_2, the first group of interface involves setting parameters necessary for arranging messages, coregions and setting the instances' length: $send\_receive\_distance$, $head\_distance$, $foot\_distance$, $coregion\_begin\_distance$, $coregion\_end\_distance$,

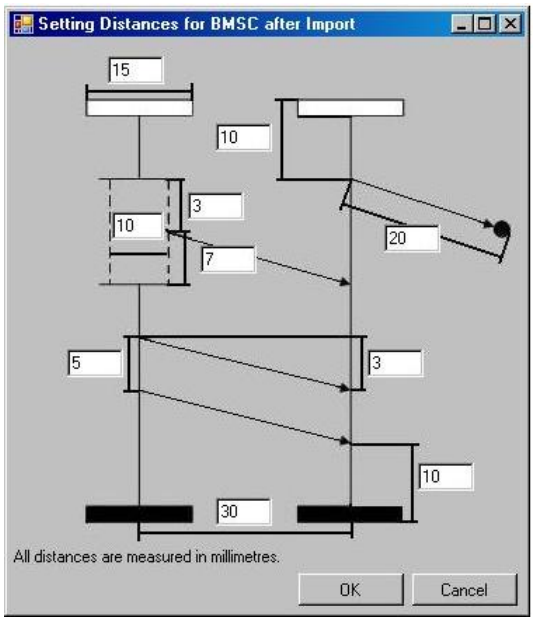

Figure 4.1: Window1 for setting of parameter for the feature used after importing BMSC from textual mode

$successor\_distance$, $incomplete\_msg\_length$, $instance\_length$. For each of these parameters there is a small box next to the highlighted distance which is represented by parameter. There are following possibilities of setting this group:

- Turn off the whole group
  Neither of abovementioned parameters can be set. All small boxes for entering these parameters are inactive as well as the possibilities of setting the length of instance. The arrangement of messages and coregions, the length of instances, coregions and incomplete messages not changed.

- Turn on the whole group
  The small box related with the parameter $incomplete\_msg\_length$, and also the possibilities of setting of the instances' length, are active. After a BMSC well-arranged, the length of all incomplete messages equals to the value of the parameter, messages and coregions are arranged, and the distances between them depend on the way of setting the instances' length:

  – Constant value
    Small boxes for entering parameters $send\_receive\_distance$, $head\_distance$, $foot\_distance$, $coregion\_begin\_distance$, $coregion\_end\_distance$, $successor\_distance$ are inactive, whereas the box related with the parameter

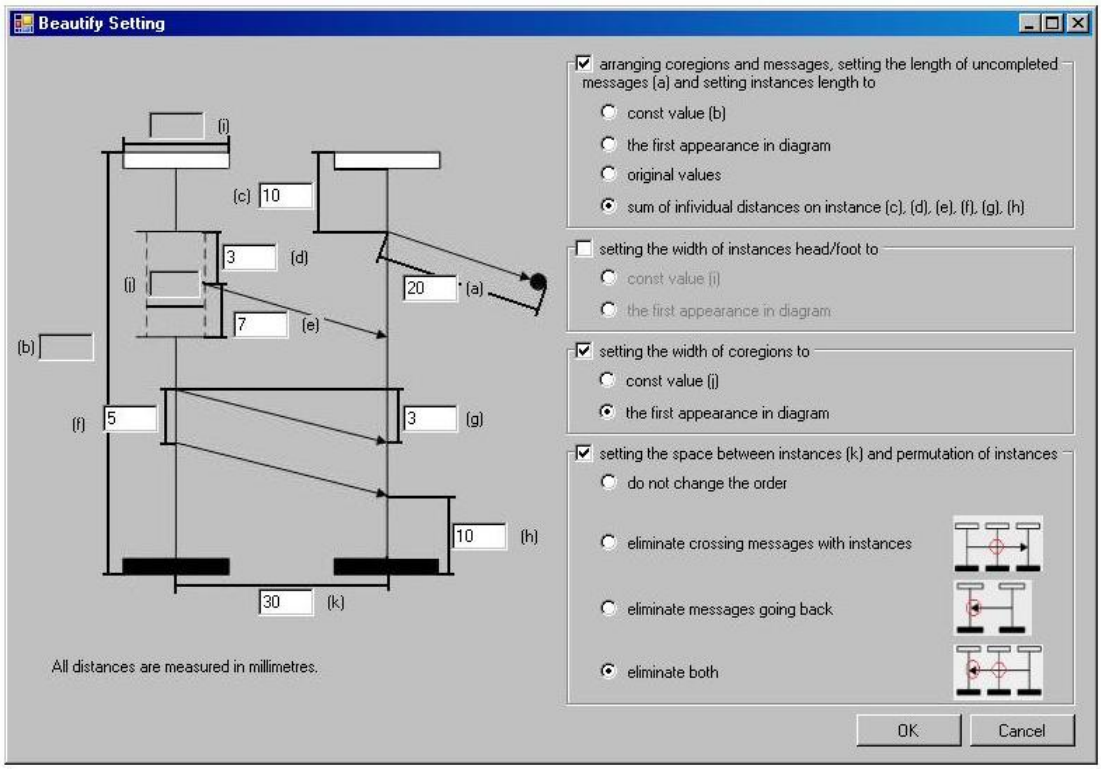

Figure 4.2: Window2 for setting of parameters for the feature used on BMSC opened in graphical mode

$instance\_length$ is active. The feature sets the lengths of all instances equal to $instance\_length$. The messages and coregions are uniformly arranged along the whole length of instances and if it is possible, the messages are horizontal.

- The value of the first occurrence in diagram
  All small boxes which are related with the length of instance are inactive. The length of all instances equals to the length of the leftmost instance in BMSC before redrawing. Messages and coregions are uniformly spread up on the instances and messages are horizontal if possible.
- The original value
  All small boxes related with the instances' length are inactive. The lengths of instances do not change. Messages are horizontal if it is possible, and the distances between the elements on an instance are as big as possible so that neither event no coregion overlay the end of the instance.
- Sum of individual distances on instances
  The box for entering the parameter $instance\_length$ is inactive whereas boxes related with parameters describing individual distances on an instance are active. Messages and coregions are arranged according to the values of parameters, the lengths of instances are set according to this arrangement and all the instances are aligned to the bottom according to the furthermost instance.

4.2 Heads and Foots of Instances

The $width\_of\_heads\_and\_foots$ in the window$_1$ can be set by entering constant value into the box related with this parameter. All heads and foots have the same size after the BMSC is imported from textual mode and it is equal to the value of parameter.
The window$_2$ contains the group box for all possibilities of setting the width of heads/foots:

- Group box for setting heads/foots is un-checked
  The small box for entering constant is inactive as well as the possibilities of setting the heads/foots. The feature does not change the value of heads and foots.
- Group box for setting heads/foots is checked
  The following possibilities of setting the heads/foots are active:
  - Constant value
    The small box for entering the value is active. The parameter $width\_of\_heads\_and\_foots$ equals to the entered value.All heads/foots are set by the feature to the same size according to the parameter.
  - The value of the first occurrence in diagram.
    The small box is inactive and the size of all heads/foots is set by the feature to the value of the first head/foot in BMSC.

4.3 Width of Coregions

For setting the width of coregions, the same rules as for setting the heads/foods of instances are valid.For the simplicity, the width of coregions is not described.

4.4 Placement and Sequence of Instances

In window$_1$, the space between two instances is highlighted in the picture. Next, there is a small box for entering the value of parameter $space$. By this parameter a user can influence the space between instances. The feature orders instances to have minimum crossing messages and minimum going-back messages. The space between each two neighbouring instances is equal to $space$. The first instance's begging is set to the position $x = 10mm, y = 10mm$. Beginnings of the rest of instances are to the same y coordinate.

The last group box in window$_2$ deals with the setting of the order and the space of instances. The possibilities of setting are:

- The group box for setting the space and the sequence of instances is un-checked
  The small box for entering the value of parameter is inactive. Also the possibility of influencing the instances' order is inactive. The feature changes neither the order nor the spaces between instances (according to begins of instances), set for each instance the begin and the end on the same x coordinate.

- The group box for setting the space and the sequence of instances is checked
  The box for entering the parameter $space$ is active as well as the possibilities of influencing the instances' order. The feature spaces instances according to value of parameter. The order of instances depends on choosing the following setting:

  - Original order
    The order is not changed.
  - Eliminate crossing messages with instances
    The feature tries to order instances to minimum messages cross with the instances.
  - Eliminate going-back messages
    The order is without going-back messages.
  - Eliminate both crossing and going-back messages
    The order deals with the minimum number of both going-back messages and crossing messages with instances.

Chapter 5

Algorithms

In this chapter there are described algorithms well-arranging elements in BMSC. At the beginning of each algorithm there are loaded parameters describing the options from registry.

Each of them solves any factor influencing BMSC readability. At the beginning of each algorithm, parameters describing an option of solved factor are loaded from register. The values of these parameters than define the run of algorithm.

The first algorithm is called *instances_sequencer*. It deals with the order of instances and their horizontal and vertical position in diagram. Because the width of coregions or the width of heads/foots can influence the space between instances, next part of the algorithm is setting the width of these elements. Depending up a way of setting the instance length, is the size of the length solved in *length_optimizer* or in *layout_optimizer* or for option 'original length', the instances length does not change and so it is not solved anywhere. Layout_optimizer is also responsible for arranging messages and coregions on instances.

5.1 Sequence and Placement of Instances, Width of Coregion and Head/Foot

The algorithm is described in 1

Correctness and Running Time of process The algorithm does not always return the correct result. The reason is because it uses basis of *greedy* algorithm, event thought it does not have a form of greedy algorithm. The example of incorrect result is in Figure 5.1. Algorithm should to find the best possible order according to eliminate going-back messages and crossing messages with instances. The instances were added to the ordered array in sequence: A, B, C, D. The problem is that algorithm decides according to the best solution at this time. To try to avoid this situations it was added to the algorithm lines $15, 16$.
The time duration of this algorithm is asymptotically equal to $O(m^4)$ where m is number of instances. This time duration is caused by lines $9 - 14$ in algorithm2. The function add is called m-times, and from the algorithms it is obvious that the asymptilic duration of function add is $O(m^3)$. However because the time duration depends on the size of ordered array and during the each iteration the real time duration is smaller.

Algorithm 1 Setting the sequence and placement of instances and the width of coregions and heads/foots according to the options set in interface

load the following parameters from register: long m_setting_space, float m_space, long m_original_order, long m_crossing_order, long m_going_back_order, long m_setting_head_width, long m_use_const_head, float m_head_value, long m_setting_coregion_width, long m_use_const_coregion, float m_coregion_value
set coordinates start_x and start_y (as start point of instance sequence) to the x and y coordinate of begin of leftmost instance
if ($setting\_head\_width$ and !(use_const_head)) **then**
    find the leftmost instance, to the m_head_value set its value of the width
end if
if ($setting\_coregion\_width$ and !$use\_const\_coregion$) **then**
    find the leftmost instance containing coregion, find the topmost coregion on this instance, save it's width to m_coregion_value
end if
for all Instance $instance \in bmsc$ **do**
    if $setting\_head\_width$ **then**
        $instance$.set_width($head\_width$);
    end if
    if $setting\_coregion\_width$ **then**
        for all Coregion $coregion \in instance$ **do**
            $coregion$.set_width($coregion\_width$)
        end for
    end if
end for
if !$setting\_space$ **then**
    set the end of each instance to the same x coordinate as its begin, y coordinate of begin to the start_y and y coordinate of end equal to value to instance length does not change
    enlarge the space between each two neighbouring instances only if heads/foots or coregions of them are overlapping
else
    enlarge the value of m_space only if heads/foots or coregions of any neighbouring instances are overlapping
    if m_original_order **then**
        set the space between begins of neighbouring instance to m_space, x coordinate of end to the same value as x coordinate of begin of the same instance, y coordinate of begins to the start_y and y coordinate of ends to the values to instances length does not change
    else
        call the function *sequence(bmsc, m_crossing_order, m_going_back_order)*. With respect to new order set the space between begins of neighbouring instance to m_space, x coordinate of end to the same value as x coordinate of begin of the same instance, y coordinate of begins to the start_y and y coordinate of ends to the values to instances length does not change
    end if
end if

Algorithm 2 sequence(unsigned $crossing\_penalization$, unsigned $going\_back\_penalization$)

1: set<Instance> $unordered \leftarrow bmsc$.get_instances()
2: vector<Instance> $orderd$
3: find two instances, $instance1, instance2$, which are exchanging the most number of messages each other
4: $ordered$.insert($instance1$)
5: $unordered$.erase($instance1$)
6: add($ordered, instance2$)
7: $ordered$.insert($instance2$)
8: $unordered$.erase($instance2$)
9: **while** !$unordered$.empty() **do**
10: find the instance, $instance$, which has the most number of exchanging messages between it and the instances from $orderd$
11: add($ordered, instance, crossing\_penalization, going\_back\_penalization$)
12: $ordered$.insert($instance$)
13: $unordered$.erase($instance$)
14: **end while**
15: **if** $m\_crossing\_penalization$ and $(m\_going\_back\_penalization)$ and number of going_back messages is bigger than going_forward messages **then**
16: $ordered$.reverse()
17: **end if**

Algorithm 3 add(vector<Instance> $orderd$, Instance $instance$, unsigned $crossing\_penalization$, unsigned $going\_back\_penalization$)

1: $orderd$.push_back($instance$)
2: **for all** possible positions of $instance$ in $orderd$ **do**
3: cound_value($orderd, crossing\_penalization, going\_back\_penalization$)
4: **end for**

Algorithm 4 cound_value(vector<Instance> $orderd$, unsigned $crossing\_penalization$, unsigned $going\_back\_penalization$)

1: **for** int $i = 0; i < ordered$.size(); $i++$ **do**
2: **for** int $j = i + 1; j < ordered$.size(); $j++$ **do**
3: $value \leftarrow value +$ $going\_back\_penalization$*(number of going-back messages between $ordered[i]$ and $ordered[j]$) + $crossing\_penalization$*(number of crossing instances with messages exchanged between instances $ordered[i]$ and $ordered[j]$)
4: **end for**
5: **end for**

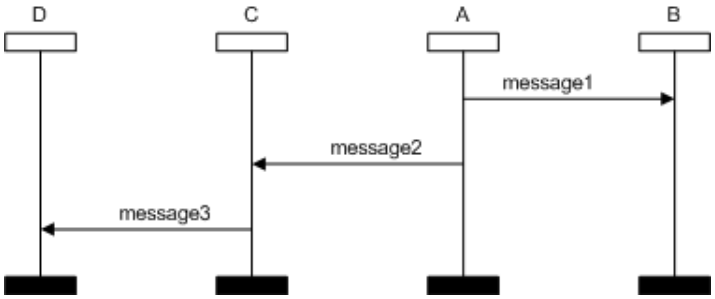

Figure 5.1: Possible result of $process$

5.2 Length of Instances

1. Parameters describing the way of setting the instances' length and the value of parameter $instance\_length$ are loaded from registry.

2. For options 'neither to arrange elements nor to set the instance length' and 'original lengths' the algorithm successfully ends, because it has nothing to solve. The algorithm ends also for option 'setting according to the individual distances on an instance'. In this case the length is solver by the algorithm described in next subsection $layout\_optimizer$.

3. For the option in which setting the length 'to the value of the first instance in diagram' is founded, the leftmost instance and its value is set to the parameter called the $instance\_length$.

4. The length of all instances is set to the value of $instance\_length$. It holds that if the option 'setting the length to the constant value' is chosen, the $instance\_length$ parameter still contains value loaded from registry.

For running time of this algorithm holds that the the first two points take $O(1)$. The third point takes $O(n)$, where n is the number of instances and the last point takes also $O(n)$. So the total running time of algorithm is $O(n)$. It always ends, because the cycles in points $3)$ and $4)$ go through the instances in BMSC, whose number is finite.

5.3 Arrange Messages and Coregions

In SCStudio, arranging messages and coregions is solved by algorithm layout_optimizer. The first version of this algorithm was created by Ing. Petr Gotthard. This original version solves the arrangement of messages. The task of this thesis is to describe this algorithm and extend it by arranging coregions and solve the problem of overlapping messages. Both the original and modified algorithms can be found on attached CD.

It holds that the position of elements (events and coregions) on instance is relative, considering begin of this instance. The x coordinate of elements has to be equal to zero because according to Z.120, elements must be drawn on instance's axis. Therefore the term 'position of elements on instances' refers only to y coordinate of these elements.
The algorithm is based on the principle of *linear programming* (LP). In the first subsection the basis of LP is described. Next subsection deals with transcription of the problem of arranging messages into the problem of LP, and last subsection describes innovations of algorithm.

5.3.1 Linear Programming

This subsection uses results in [2]. Linear programming is a part of mathematical programming. It finds an optimal solution for minimizing or maximizing problems where the resources are limited or where it is necessary to fulfil some specifications. We are able to transform a problem into the problem of linear programming if it is possible to express the objective by a linear function. Resources or specifications have to be defined as linear inequalities.

Formulation of general linear programming [4]
For given real numbers $a_1, a_2, ..., a_n$ and a set of variables $x_1, x_2, ..., x_n$ the function f, called also an *objective function*, is defined as:

$$f(x_1, x_2, ..., x_n) = a_1x_1 + a_2x_2 + ... + a_nx_n = \sum_{i=0}^{n} a_ix_i$$

and the linear inequalities, called *linear constraints*, as:

$$f(x_1, x_2, ..., x_n) \leq b$$

$$f(x_1, x_2, ..., x_n) \geq b$$

where b is a real number and f is a linear function.
Strict inequalities are not permissible. The entries of linear programming (LP) are the linear function and linear constraints. The goal of LP is to minimize or maximize the linear function subject to a finite set of linear constraints.

5.3.2 Rewriting Problem of Arranging Messages into LP Problem

As it is mentioned in Chapter 3, adjustable parameters describing minimum distances between elements on instances are defined. The real distances are always at least as big as these parameters. However, there is also a requirement of bringing the spaces as close to the parameters values as possible. It holds that the setting of vertical distance between messages' edges to exact (or close) value of parameter is prior to setting of other distances. The reasons have been described in former analysis.
The arranging problem has a character of a LP problem because it is necessary to minimize

spaces between the elements (with different priority) and the distances must be at least as big as adjusted parameters.
This problem is transformed into the problem of linear programming, and then the program called *lp_solve* is used. It is a Mixed Integer Linear Programming solver. It computes the optimal solution and from this solution the answer for question 'how the messages could be arranged?' is obtained.

Definition of the problem of linear programming for arranging messages

To define the problem of arranging messages as a LP problem, it has to be rewritten into linear objective function and linear constraints. Because the purpose of layout_optimize is to find out the y coordinates of events, the unknown variables of function and constraints refer to these coordinates.

Constraints for arranging messages
In following text, constraints for the problem of arranging messages are written. It holds that the real space between two elements must be at least as big as adjusted parameter referring to this distance. There are four parameters related to the position of events on instances: $head\_distance$, $foot\_distance$, $successor\_distance$, $send\_receive\_distance$.
Regarding Z.120, there are other rules for events on instances: The events must be drawn on instance axis to which they are attached, so the events have the value of y coordinate minimum equal to y coordinate of instance starting point and the maximum equal to y coordinate of the instance end. Going-up messages are forbidden, so the received event has y coordinate bigger or equal to y coordinate of the sent event. And the order of events can not be changed.
To achieve all of abovementioned rules, there are written constraints for events and instances in BMSC:

$\forall$*instance, event; event* $\in$ *instance* $\wedge$ *event is the first element on instance:*

$$event - instance\_begin \geq head\_distance \tag{1}$$

$\forall$*instance, event; event* $\in$ *instance* $\wedge$ *event is the last element on instance:*

$$instance\_end - event \geq foot\_distance \tag{2}$$

$$\forall event;\ \exists event\_successor:\ event\_successor - event \geq successor\_distance \tag{3}$$

$$\forall complete\ messages:\quad receive - send \geq send\_receive\_distance \tag{4}$$

The domain of $successor\_distance$ is $(0, \infty)$, because two events can not be drawn in one place, and the domain of the rest of distances is $[0, \infty)$.
Note that by expressions $instance\_begin, instance\_end, eventandevent\_successor$ it is meant the y coordinate of these elements.

The constraints $(3) and (4)$ are unrealisable for cyclic BMSC. In cyclic BMSC, there is always at least one message going up. Therefore, for one message in cycle, instead of constraints $(3) and (4)$ it is claimed:

$$event - event\_successor \geq successor\_distance \tag{6}$$

$$send - receive \geq send\_receive\_distance \tag{7}$$

For incomplete messages it holds that the position of a *dot* is not a part of the LP problem. It is because it can be easily set manually according to the matching event. For vertical distance between the event and the dot it holds: $dot - matching\_event = send\_receive\_distance$.

Minimize objective function for arranging messages
As it is said at the beginning of this subsection, there is requirement for minimizing distance between the edges of complete messages (distance of incomplete messages are set manually). From constraint (3) it is obvious that this distance is represented by $(receive - send)$. So the first part of minimizing objective function is $minimize(receive - send)$ (For one message of a cycle in BMSC, received and sent events are reversed).
To minimize the rest of the distances it is necessary to put the events as high as possible, so it is necessary to minimize events using the first part of the objective function, it is enough to minimize only one event of a complete message and the matching event is done automatically. Regarding the sign of events in the first part of an objective function, the second part is $minimize(receive)$. For incomplete messages it is necessary to minimize the event which creates the third part of the objective function: $minimize(event\_of\_incomplete\_msg)$.
After joining all the parts, whole 'minimize objective function' is:

$$minimize(\sum_{i=1}^{n}(2receive_i - send_i) + \sum_{j=1}^{p} event\_of\_incomplete\_msg_j)$$

where n is the number of all complete messages and p is a number of all incomplete messages.

5.3.3 Corrections and Improvements of LP Problem for Arranging

Definition of the problem of linear programming for arranging coregions and messages attached to coregions

Coregion is another element which can lay on an instance. To coordinate its position on instance, there are used the parameters describing minimum distances between elements. In addition, to coordinate the position of events inside coregion, there need to be two more parameters describing the minimum distances between edges and events inside coregion.

All of these distances must be fulfilled and as close to the value of parameters as possible.
In following text, an extension of definition LP problem for arranging coregions and events inside coregion are described. For each event inside coregion, and for each edge of coregions, there are new variables established.

Constraints for arranging coregions
For all events of complete messages inside any coregion there is the constrain for distance between a coregion and its matching event set using the constraint (4).
It holds that the order of events in coregions is not established, and so they can change the sequence. However, the order can be set using the relation of events relation. Considering distances between ordered events in coregion, the same rules as for events outside coregion hold. They are defined according to constraint (3).
Regarding the analysis, the events inside coregions are separated into two sets according to the orientation of its messages. The distance between two events in one set must be at least equal to the parameter called $successor\_distance$. And also the distances between the coregion edges and the events must be fulfilled. Since the coregion lies on instance axis, the same rules as for events are valid. So the distance between the beginning of coregion and the beginning of instance, and between the end of coregion and the end of the instance must be at least equal to parameter describing this distance.
Regarding Z.120 it holds that all events, belonging to any coregion must be bigger then coregion begin and lower than coregion end. Also the coregion begin is lower than coregion end.
All of abovementioned requirements can be written in following linear constraints:

$\forall$ *instance, coregion_begin; coregion_begin* $\in$*instance* $\wedge$ *coregion_begin is the first element on instance:*

$$coregion\_begin - instance\_begin \geq head\_distance \tag{7}$$

$\forall$ *instance, coregion_end; coregion_end* $\in$*instance* $\wedge$ *coregion_end is the last element on instance:*

$$instance\_end - coregion\_end \geq foot\_distance \tag{8}$$

$\forall$ *coregion,*$\forall$*event; event*$\in$*coregion* $\wedge$ *event is the first one on coregion:*

$$event - coregion\_begin \geq coregion\_begin\_distance \tag{9}$$

$\forall$ *coregion,*$\forall$*event; event*$\in$*coregion* $\wedge$ *event is the last one on coregion:*

$$coregion\_end - event \geq coregion\_end\_distance \tag{10}$$

$\forall$ *event*$_1$, *event*$_2$*; both events lay on one side of the same coregion* $\wedge$ *no relation between them:*

$$|event_1 - event_2| \geq successor\_distance \tag{11}$$

∀ *coregion_begin, event; event is right above the coregion_begin:*

$$coregion\_begin - event \geq successor\_distance \tag{12}$$

∀ *coregion_end, event; event is right under the coregion_end:*

$$event - coregion\_end \geq successor\_distance \tag{13}$$

Extension of minimize objective function for arranging coregions
By the reason coregions have to be evenly arranged on instances, y coordinate of coregions' edges must be minimized. And it is also required for length to be as small as possible, with subject to defined constraints. Therefore, it is required to minimize the distance between the beginning and the end of coregion: $minimize(coregion\_end - coregion\_begin)$. And also, as for the events on coregions, it is requested to put it as high as possible. For moving coregions upwards, similar rule as for completed messages holds: It is enough to move only one edge, the other will move automatically using the previous part. Due to the sign of variables represented coregion edges, the next part of objective function is $minimize(coregion\_end)$. The part of objective function for arranging coregions is:

$$\sum_{k=1}^{s} 2coregion\_end_k - coregion\_begin_k \tag{B}$$

where s the number of coregions
Together, the part of (A) and (B) create the whole linear objective function. For minimizing events inside coregion, the same rules as for events outside coregion are valid. Therefore they are already included in the part (A) of objective function.

Repairing the objective function

As it is written at the beginning of this subsection, setting minimal distances between the sent and received events of one complete message is prior to setting minimal spaces between the elements on an instance. However, during the development of layout_optimizer it was found out that in some situations this rule is not observed. The example of this situation is represented in Figure 5.3: The $head\_distance$, $foot\_distance$, $successor\_distance$ are equal to $5mm$ and $send\_receive\_distance$ to $0mm$. The value of the objective function for this example is $25mm$. However, the distance between the edges of $message4$ is equal to $5mm$ even though it could be equal to $0mm$, as it is seen in Figure 5.2. The parameters for the second example are the same as for the first example. The value of objective function is $30mm$, which is bigger than in the first example. The value of objective function in the first example

is smaller because of smaller distances between events e_4, e_6 and $instance\_begin$ which, in sum, bring greater benefit for objective function than observing minimal distance between edges of $message4$. Therefore it follows that the form of objective function does not observe the rule for bigger priority of minimizing send-receive distance of one complete message.
Setting minimum distances between the elements on an instance are in objective function represented by:

$$\sum_{i=1}^{n} receive_i + \sum_{j=1}^{p} event\_of\_incomplete\_msg + \sum_{k=1}^{s} coregion\_end_k$$

To reduce the priority of minimize these spaces, the abovementioned part is changed to:

$$\sum_{i=1}^{n} \frac{receive_i}{m} + \sum_{j=1}^{p} \frac{event\_of\_incomplete\_msg}{m} + \sum_{k=1}^{s} \frac{coregion\_end_k}{m}$$

where m is number of all variables in LP.
The reason this setting of objective function solves the problem is demonstrated in the following images. In Figure 5.5, the distance between the edges of the red message is bigger greater by d than in the Figure 5.4. Distances between received events of k green messages (or coregions) and the $instance\_begin$ is smaller also by $\frac{d}{m}$ than in Figure 5.4. If the value of objective function for Figure 5.4 is marked as G, the value of objective function for Figure 5.5 (marked as H) can be written as:

$$H = G + d - k(\frac{d}{|number\ of\ messages\ and\ coregions|})$$

It is desired to prove that value of G is always less than the value of H. So:

$$H > G$$

after substitution:

$$G + d - k(\frac{d}{|number\ of\ messages\ and\ coregions|}) > G$$

after repairing:

$$1 > (\frac{k}{|number\ of\ messages\ and\ coregions|})$$

which holds because k is mostly equal to $(number\ of\ messages\ and\ coregions - 3)$. It is because 1 message is reserved for the red message and 1 is reserved for a coregion or a message laying on instance above the red message.

Solving of overlapping messages and coregions

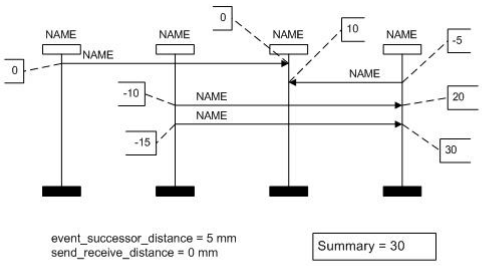

Figure 5.2: Non-overlapping messages

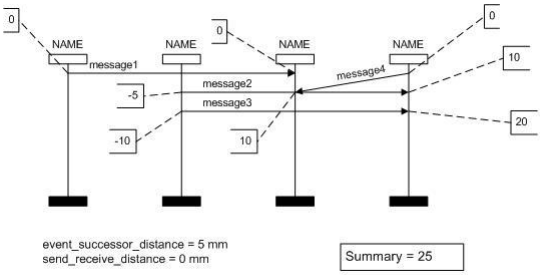

Figure 5.3: Overlapping messages

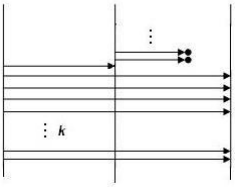

Figure 5.4: The BMSC with horizontal messages

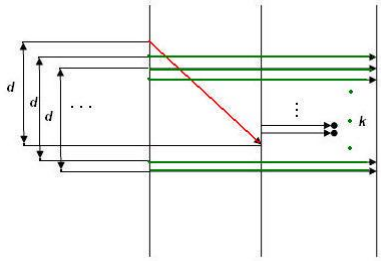

Figure 5.5: The BMSC with one oblique message

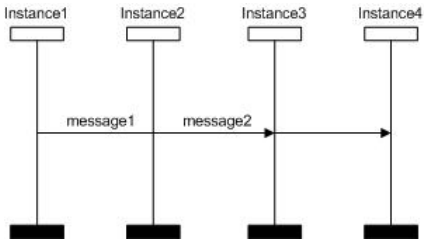

Figure 5.6: Overlapping of message1 and message2

The conditions which were defined by now do not solve the 'overlapping messages and coregions'. The example of overlapping can be seen in Figure 5.6. To solve this problem, a new constraint must be added for variables (events or coregions' edges) laying on different instances. It must be true that the events and coregions' edges which lay on instances crossed by any message must have different position from the sent event of this message. And the minimum distance between them is the $successor\_distance$. Therefore, to the constraints of LP following constraints can be added:
∀*send event of complete message, instance, event; event* ∈ *instance, instance is crossed by message:*

$$|event - send| \geq successor\_distance \tag{14}$$

and for coregions' edges:
∀*send event of complete message, instance, coregion; coregion* ∈ *instance, instance is crossed by message:*

$$|event - coregion\_begin| \geq successor\_distance \tag{15}$$

$$|event - coregion\_end| \geq successor\_distance \tag{16}$$

5.3.4 Description of Layout_optimizer in SCStudio

- In the first step, the algorithm layout_optimizer finds out the way of setting the instances' length. If the option *setting according to individual parts* is chosen, then all the parameters describing individual spaces on instances are initialled by values loaded from register. On the other hand, if the length of instances is already set, individual distances are not known and must be calculated. Because they depend on the instances' length and also on the arrangement of messages and coregions (which is not established yet), are calculated in the end. As initial values of these distances following are valid:
  $send\_receive\_distance = 0$ - the messages are horizontal if it is possible
  $head\_distance = foot\_distance = coregion\_begin\_distance = coregion\_end\_distance =$ 1 - for uniform arranging

- In next step, events and coregions' edges are indexed and the above-defined LP problem for arranging messages and coregions rewritten to the form for lp_solve: *Objective function:*

$$minimize \quad \mathbf{c}^T\mathbf{x}$$

subject to:

$$A\mathbf{x} \geq \mathbf{b}$$

where vector **x** represents y coordinates of all events and coregions' edges, **c** is a vector of coefficients of objective function, matrix **A** and vector **b** are coefficients of constraints.

- Lp_solve is counted. It returns vector **x** with y coordinate for all variables.
- If the option for setting the length according to individual parts is not chosen, the rate of initial spaces is calculated. Its value depends on the arrangement of elements and the length of instances. It must be as big as possible for no element laying over the edge of instance. If the abovementioned option is chosen, the rate is equal to 1.
- Positions of coregions and events are set according to the rate of initial spaces and the values returned by lp_solver.

Chapter 6

Conclusion

In this thesis we have first analysed the structure of BMSC and established some adjustable parameters for elements of BMSC. There are described the possible settings of these parameters and also their usage. We found out that some of these parameters relate to each other and so they can influence the possibilities of settings. The analysis introduces some differences of setting the feature used after the import of BMSC form textual mode and the feature used for well-arranging BMSC in graphical mode.
Next we have dealt with user-friendly interface for the feature for redrawing BMSC. The two windows were introduced, each of them for setting the parameters used of the feature in different situations. We described the behaviour of the system for setting each of the possible options.
In the last chapter is described the algorithm for arranging elements of BMSC. We have transformed a problem of arranging event and coregions on instances into the problem of linear programming and solved it by using lp_solve. It was extended by other constraints to avoid the situations of overlapping messages and coregions. The next algorithm described in this chapter, solves the placement and order of instances. It holds that the vertical distances between instances' heads is equal to zero, so they are all set on the same level. According to the loaded parameters, the algorithm computed the space between instances and the order of them. It holds that whenever heads or coregions of any couple of instances are overlapping, the space is enlarged. The last algorithm deals with the length of instances. However if the length of instances should to be set according individual distances on instances, this problem is moved to the abovementioned algorithm - layout_optimizer.
We have suggested the problem of goodarrangement of HMSCs in future regarding this problematics.

Bibliography

[1] J. Babica. Message Sequence Charts properties and checking algorithms. *Bachelor thesis*, Faculty of Informatics, Masaryk University, Brno, 2009.

[2] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[3] ITU Telecommunication Standardization Sector Study group 17. ITU recommandation Z.120, Message Sequence Charts (MSC), 2004.

[4] S.G. Nash, A. Sofer, and JR Shinnerl. *Linear and nonlinear programming*. McGraw-Hill New York, 1996.

Appendix A

Contents of Attached CD

The attached CD contains the source code of the algorithms: instance_sequencer, length_minimizer, layout_optimizer, beaufitySettingsDlg, importSettingsDlg.