Drawing Message Sequence Charts with LaTeX

Sjouke Mauw and Victor Bos

Abstract

The MSC macro package facilitates LaTeX users to easily include Message Sequence Charts in their texts. This article describes the motivation for developing the MSC macro package, the features of the MSC macro package, and the design of the MSC macro package.

1 Introduction

The Message Sequence Chart (MSC) language is a visual formalism to describe interaction between components of a system. The language is standardized by the ITU (International Telecommunication Union) in Recommendation Z.120 [4]. An introductory text on MSC can be found in [5]. MSCs have a wide application domain, ranging from requirements specification to testing and documentation.

An example of a Message Sequence Chart is given in Figure 1. The MSC shows an ftp login session to a CTAN archive. Three players, called *instances*, are involved in the session: *User*, *ftp client*, and *CTAN* at location *ftp.tex.ac.uk*. The instances are denoted by vertical lines. Interaction between instances is denoted by labeled arrows. For instance, the arrow *ftp.tex.ac.uk* is a message from *User* to *ftp client*. Sending and receiving of a message are special types of *events*; each message has a *send* event and a *receive* event. Later we will see other types of events. Events occur on instance lines. Events are ordered in time and for each instance, time is supposed to run from top to bottom. Furthermore, the send event of a message never occurs after the receive event of the message. For example, from Figure 1, we can derive that the *ftp.tex.ac.uk* message occurs before the *connect* message, because the receive event of the first message occurs before the send event of the second message.

In order to include MSCs in LaTeX documents, we have developed the MSC macro package. The current version of the MSC macro package supports almost the full MSC language as defined in the standard. In this article we will describe the motivation of the MSC macro package, the features of the MSC macro package, the design of the MSC macro package, and the limitations of the MSC macro package. This paper does not describe all features of the MSC macro package. For a thorough treatment of the MSC macro package we refer to the user manual [2].

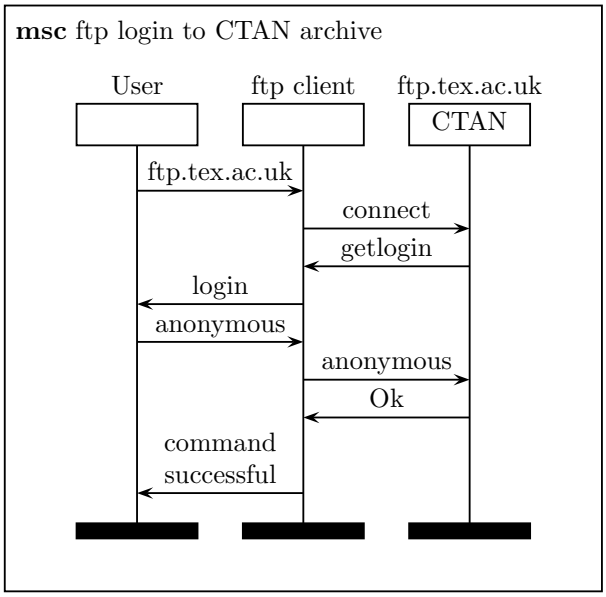


Figure 1: An ftp login to the *CTAN* at *ftp.tex.ac.uk*.

2 Motivation

Several commercial and non-commercial tools are available, which support drawing or generating Message Sequence Charts. However, these tools are in general not freely available and often not flexible enough to satisfy all users' wishes with respect to the layout and graphical appearance of an MSC. Furthermore, they often do not allow the user to include LaTeX code in the MSCs. Another drawback of these tools is that quite often they restructure MSCs automatically. Though for simple MSCs this might be what the user wants, for more complex MSCs the result of automatic restructuring is usually not desired.

Therefore, people often use general drawing tools, such as *xfig* (see `http://www.xfig.org/`) to draw MSCs. However flexible this approach is, it has some drawbacks. First of all, general drawing tools have (and should have) a low level of abstraction; their interface is defined in terms of *coordinates*, *points*, *lines*, *polygons*, etc. To draw MSCs, the user would probably be more comfortable if the interface was defined in terms of *instances*, *messages*, *actions*, etc. For example, if you are drawing a message in an MSC using a general drawing tool, you would probably have to draw a *line* with an arrow head from a *position* (x_0, y_0) to a *position* (x_1, y_1), instead of drawing a *message* from an *instance* i_0 to an *instance* i_1 of the MSC.

Another drawback of using general drawing tools is that they usually do not provide libraries of MSC symbols. Therefore, if you have to draw many MSCs, it will take much effort to get a set of consistent looking MSCs. Furthermore, if you want to change a parameter of the MSCs, e.g., the width of the instance head symbols, you would probably have to edit all MSCs manually.

For these reasons, we developed the MSC macro package for LaTeX. The macros in the package enable a textual representation of an MSC in a LaTeX source document. By compiling the LaTeX document into PostScript, a graphical representation of the MSC is generated.

The design requirements for the MSC macro package were:

1. The package should follow the ITU standard with respect to shape and placement of the symbols of an MSC.
2. The interface of the package should be at the right level of abstraction.
3. There should only be a limited amount of automatic restructuring and layout of the MSCs.
4. The appearance of (sets of) MSCs should be configurable by an appropriate set of parameters.
5. The MSC macro package should run on standard LaTeX distributions.

3 User interface

In this section we will briefly describe the user interface of the MSC macro package. We will do this by giving examples and showing the LaTeX code that produced the examples.

MSC environment MSCs are drawn in the `msc` environment. The syntax of this environment is `\begin{msc}[titlepos]{title} ... \end{msc}`. The title of the MSC is defined by the `title` parameter. The optional parameter `titlepos` determines the position of the title. By default it is `l` (left aligned). Other possible values are `c` (centered) and `r` (right aligned).

Instances Instances are declared with the

```
\declinst[*]{nn}{an}{in}
```

command. The starred version produces a *fat instance* which will not be discussed in this paper. The `nn` parameter defines a *nickname* of the instance. Nicknames identify instances and are used to draw messages and events. The `an` parameter defines the *above name* of the instance. This is the text to be placed above the instance head symbol (the rectangle at the top of an instance). The `in` parameter defines the *inside name* of the instance. This is the text to be placed inside the instance head symbol. Both the inside name and the above name may be empty.

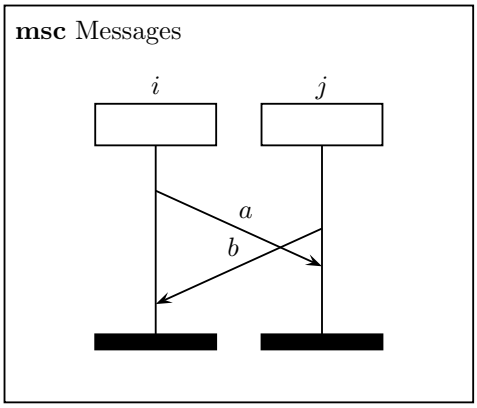
Messages Messages are drawn with the

```
\mess[pos]{txt}{s}{r}[offset]
```

command. The optional `pos` parameter defines the position of so-called *self messages*: messages from an instance to itself. The default value of `pos` is `l` (to the left of the instance) and another possible value is `r` (to the right of the instance). The `txt` parameter defines the label of the arrow representing the message. The `s` parameter is the nickname of the instance on which the send event occurs, i.e., the nickname of the sender. The `r` parameter is the nickname of the instance on which the receive event occurs, i.e., the nickname of the receiver. The optional parameter `offset` defines the number of levels the receive event is shifted vertically with respect to the send event. Levels are discussed in the next paragraph. Offsets are useful if two instances send messages to each other and then wait for the messages to be received. For example, Figure 2 shows messages a and b between instances i and j. The receive event of message a occurs after the send event of message b and vice versa. Both messages have `offset` $= 2$ in order to place the receive events two levels below the send events.

Levels The height of an `msc` environment is determined by the number of *levels* and a fixed amount of vertical space above and below the first and last level, respectively. Levels are created by the `\nextlevel[num]` command. The optional parameter denotes the number of levels to be added; its default value is 1. Levels are used to order events in time. Recall that time runs from top to bottom, i.e., it runs from higher levels to lower levels. Events in the same level are drawn at equal vertical distance from the top of the MSC. The send event of a message will always be drawn in the current level. The receive event of a message can be drawn in another level using the `offset` parameter of the `\mess` command. Note that levels are not part of the MSC language, they are just an implementation means to draw MSCs.

Using the commands described so far, we can generate the MSC of Figure 1. The LaTeX input to generate that MSC is given below. The length `\instdist`, used in the last `\mess` command, defines the distance between instances of an MSC and is one of the parameters to configure the MSC macro package. Here, it is used to create a `\parbox` that is




```
\begin{msc}{Messages}
\declinst{i}{$i$}{}
\declinst{j}{$j$}{}
\mess{$a$}{i}{j}[2]
\nextlevel
\mess{$b$}{j}{i}[2]
\nextlevel[2]
\end{msc}
```

Figure 2: Using non-zero message offsets.

15% smaller than the distance between the instances of the MSC.

```
\begin{figure}[htb]
\begin{center}

\begin{msc}{ftp login to CTAN archive}
\declinst{usr}{User}{}
\declinst{ftp}{ftp client}{}
\declinst{ctan}{ftp.tex.ac.uk}{CTAN}

\mess{ftp.tex.ac.uk}{usr}{ftp}
\nextlevel
\mess{connect}{ftp}{ctan}
\nextlevel
\mess{getlogin}{ctan}{ftp}
\nextlevel
\mess{login}{ftp}{usr}
\nextlevel
\mess{anonymous}{usr}{ftp}
\nextlevel
\mess{anonymous}{ftp}{ctan}
\nextlevel
\mess{Ok}{ctan}{ftp}
\nextlevel[2]
\mess{\parbox[b]{.85\instdist}
  {\centering command successful}}{ftp}{usr}

\end{msc}
\end{center}
\end{figure}
```

Actions Actions are events that can be used to model internal activity of a particular instance. Actions are defined with the `\action{txt}{nn}` command. The `txt` parameter defines the text to be placed inside the action symbol. The `nn` parameter is the nickname of the instance that executes the action. The action will be drawn at the current level with its top aligned with send events at the same level.

For example, suppose *CTAN* has to do some computations in order to determine if the anonymous login is allowed. The computation could be modeled by a *check* action, as depicted in Figure 3. The LaTeX code for the MSC of Figure 3 is:

```
\begin{msc}{Action}
\declinst{ftp}{ftp client}{}
\declinst{ctan}{ftp.tex.ac.uk}{CTAN}
\nextlevel
\mess{anonymous}{ftp}{ctan}
\nextlevel
\action{Check}{ctan}
\nextlevel[2]
\mess{Ok}{ctan}{ftp}
\end{msc}
```

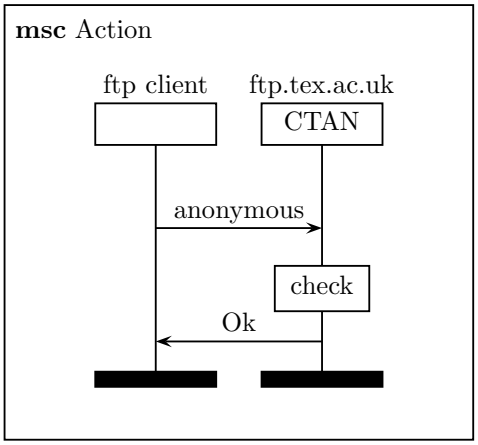


Figure 3: An MSC with an action.

Regions Another way to model internal activity, or inactivity, is by using *regions*. Regions are defined by the `\regionstart{regtype}{nn}` and the `\regionend{nn}` commands. The `regtype` parameter defines the type of the region: `activation`,

`coregion` (which will not be discussed in this paper), or `suspension`. The `nn` parameter is the nickname of the instance on which the region should be drawn. If an instance is active, e.g., doing some computations, this can be modeled by an *activation region*. If an instance is inactive, e.g., waiting for results, this can be modeled by a *suspension region*. For example, the computation of the *CTAN* could be modeled by an activation region. Furthermore, the *ftp client* is inactive during this computation, which could be modeled by a suspension region. Figure 4 shows the resulting MSC. The LaTeX code for the MSC of Figure 4 is:

```
\begin{msc}{Regions}
\declinst{usr}{User}{}
\declinst{ftp}{ftp client}{}
\declinst{ctan}{ftp.tex.ac.uk}{CTAN}
\regionstart{activation}{ftp}
\mess{anonymous}{usr}{ftp}
\nextlevel
\regionstart{suspension}{ftp}
\regionstart{activation}{ctan}
\mess{anonymous}{ftp}{ctan}
\nextlevel[2]
\mess{Ok}{ctan}{ftp}
\regionend{ctan}
\regionstart{activation}{ftp}
\nextlevel
\mess{\parbox[b]{.85\instdist}
  {\centering command successful}}{ftp}{usr}
\regionend{ftp}
\end{msc}
```

Note that the space between the activation region of the *ftp client* and the *anonymous* message from the *ftp client* to *CTAN* is very small. In the next paragraph we will show how redefining one of the MSC *parameters* can increase this space.

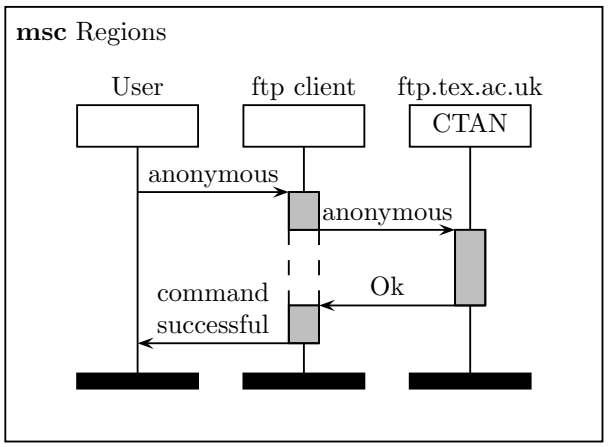


Figure 4: An MSC with activation and suspension regions.

MSC parameters The MSC macro package has almost 30 parameters to change the layout of MSCs. For example, the width of instances, the distance between instances, the distance between the head symbols and the MSC frame, and the width and height of action symbols can all be changed. These parameters are represented by the LaTeX lengths `\instwidth`, `\instdist`, `\topheaddist`, `\actionwidth`, `\actionheight`, respectively. For instance, in the MSC of Figure 4, the distance between the instances should be slightly bigger, in order to increase the space between the activation region of the *ftp client* and the *anonymous* message from the *ftp client* to *CTAN*. Figure 5 shows the same MSC, but now the distance between instances is increased by 10%. The LaTeX code for this MSC is the code for Figure 4 in which just after the line `\begin{msc}{regions}` the line `\setlength{\instdist}{1.1\instdist}` is included.

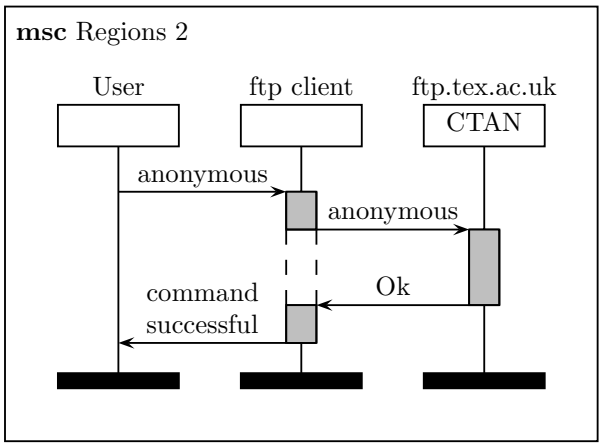


Figure 5: An MSC with larger distance between instances.

The location where an MSC parameter is changed in the LaTeX source document determines its effect. Since the MSC parameters are normal LaTeX macros or LaTeX lengths, the normal LaTeX scoping rules for these entities apply. For example, if a length parameter is changed outside any LaTeX environment, its effect is visible for all `msc` environments defined after the change. However, if it is changed inside an `msc` environment, its effect is only visible for that MSC.

Since there are many parameters to configure the MSC macro package, there are three predefined parameter settings to generate small, normal, or large MSCs. The command `\setmscvalues{parset}`

can be used to change the selected parameter settings. The `parset` parameter should be `small`, `normal`, or `large`. The default setting is `normal`.

4 Implementation

In this section we will describe some aspects of the implementation of the MSC macro package.

Drawing MSCs In general, and as shown by the examples of the previous sections, an MSC consists of a number of vertically oriented instances that are connected by horizontally oriented messages. So, the width of an MSC is related to the number of instances and the height of an MSC is related to the number of (ordered) messages. Based on this observation, there are several implementations possible.

To define the width of an MSC, we could use an additional parameter of the `msc` environment. However, this strategy has some drawbacks. First of all, an extra parameter, the horizontal position, is required to declare instances. Furthermore, this parameter probably changes whenever a new instance is added to the left of an existing instance. Finally, the user should calculate the value of this parameter carefully in order to get evenly spaced instances.

Therefore, we chose to compute the width of an MSC based on the number of instances declared by the user and the, user definable, `\instdist` length that defines the distance between instances. This decision does not violate requirement 3 of Section 2, no automatic structuring and layout, since the number of instances is under control of the user. Furthermore, the user can adjust the space to the left of the first instance and the space to the right of the last instance by redefining the length parameter `\envinstdist`.

The messages are partially ordered based on the relative position of their send and receive events on instances. We could have decided to provide commands to order the events and then let the package compute the final layout of the MSC. However, apart from the fact that this computation is not trivial, this strategy fails with respect to requirement 3: no automatic structuring and layout.

Another strategy is to use an extra parameter of the `msc` environment to define the vertical size of an MSC. There are several drawbacks to this approach. First of all, the vertical size has to be computed. Secondly, commands to draw messages, actions, regions, etc., should have one or more additional parameter to indicate the vertical position at which they should be drawn. Finally, if a new message is to be added somewhere in the MSC, the vertical placement parameter of commands below the new message should probably be updated.

Therefore, we chose to only provide a command, `\nextlevel`, to advance the current height of the MSC. By increasing the current height between two messages, the partial order can be defined. Furthermore, one can easily add new messages to the MSC at any vertical position without having to change parameters of existing messages.

These decisions resulted in an `msc` environment in which the MSC is drawn in a top-left bottom-right fashion.

Nicknames As explained above, the MSC macro package uses nicknames to identify instances. If an instance is declared, the following attributes are associated to its nickname:

- The inside name,
- The above name,
- The width of the instance line,
- A flag indicating if it is a normal or a fat instance,
- The left, center, and right x-position of the instance,
- The y-position from which this instance still has to be drawn,
- The style of the instance line, and
- The style of the region of the instance.

The `\declinst` command defines the attributes using the following TEX code pattern:
`\expandafter\def`
`\csname inst`⟨*attrnickname*⟩`\endcsname`
`{`⟨*value*⟩`}` where ⟨*attrnickname*⟩ is the concatenation of the attribute, e.g., `abname` (above name), and the nickname and where ⟨*value*⟩ is the value of the attribute. For instance, the declaration

```
\declinst{usr}{User}{}
```

defines the following commands:
`\instabnameusr`, `\instinnameusr`, `\instbarwidthusr`, `\instisfatusr`, `\instxposusr`, `\instlxposusr`, `\instrxposusr`, `\instyposusr`, `\instlinestyleusr`, and `\instregionstyleusr`.

For each instance attribute, there is an internal command to read the value of the attribute. For example, to read the value of the above name of instance `usr`, one should use `\msc@instabname{usr}`. For some attributes, like the current y-position, there is a command to change the value of the attributes. For example, to change the y-position of instance `usr` to the value `y`, one could use `\msc@setinstypos{usr}{y}`.

Underlying drawing engine The MSC macro package uses the pstricks package, see [6] or Chapter 4 of [3], to draw lines, arrows, and frames. This package is now commonly available in LaTeX distributions, so relying on this package does not violate requirement 5. A drawback of pstricks is that it is incompatible with PDFLaTeX. Consequently, our MSC macro package is incompatible with PDFLaTeX, too. However, there are other ways to generate pdf from LaTeX documents. One option is to first convert the dvi file into PostScript, e.g., using *dvips*, and then convert the PostScript file into pdf, e.g., using the *ps2pdf* utility included in ghostscript distributions (`http://www.cs.wisc.edu/~ghost/`).

5 Availability

The MSC macro package is freely available at CTAN, see directory `macros/latex/contrib/supported/msc`, and at `http://www.win.tue.nl/~sjouke/mscpackage.html`. It is distributed under the *LaTeX Project Public License*, see `http://www.latex-project.org/lppl.txt`. Documentation of the package consists of a user manual [2] and a reference manual [1]. These documents are included in the distribution.

6 Conclusions

The MSC macro package enables users to include MSCs in LaTeX documents. Furthermore, the MSCs have a consistent layout that can be configured by an appropriate set of parameters. The package supports almost the complete ITU standard of the MSC language, including MSC documents and high level MSCs (which were not discussed in this paper).

1. The abstraction level of the MSC macro package is as desired.
2. The user has full control over the relative position of instances, messages, etc.
3. Changing MSCs, e.g., adding extra instances or messages, is easy and does not require computations by the user.
4. The MSC macro package is highly configurable. There are about 30 user definable length parameters and a small number of text parameters.

The developers of the MSC macro package consider the package more or less complete. Therefore, the only changes to the package will be bug fixes and/or code documentation.

References

[1] Victor Bos and Sjouke Mauw. *A LaTeX macro package for Message Sequence Charts—Reference Manual—Describing MSC macro package version 1.5*, April 2002. Included in MSC macro package distribution.

[2] Victor Bos and Sjouke Mauw. *A LaTeX macro package for Message Sequence Charts—User Manual—Describing MSC macro package version 1.5*, April 2002. Included in MSC macro package distribution.

[3] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

[4] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, 2001.

[5] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on message sequence charts (msc'96). In *FORTE*, 1996.

[6] Timothy van Zandt. Pstricks, PostScript macros for Generic TeX. User's Guide, available at every CTAN site, (`CTAN:graphics/pstricks/`), 1993.

◇ Sjouke Mauw
Computing Science Department
Eindhoven University of Technology
P.O. Box 513
NL-5600 MB, Eindhoven
The Netherlands
`sjouke@win.tue.nl`

◇ Victor Bos
Software Construction Laboratory
Turku Centre for Computer Science
Lemminkäisenkatu 14 A
FIN-20520, Turku
Finland
`v.bos@abo.fi`