Sequence Chart Studio

Martin Bezděka, Ondřej Bouda, Ľuboš Korenčiak, Matúš Madzin, and Vojtěch Řehák
Faculty of Informatics, Masaryk University, Czech Republic
{*xbezdeka, xbouda2, xkorenc, xmadzin, rehak*}*@fi.muni.cz*

***Abstract*—The Sequence Chart Studio (SCStudio) is a user-friendly drawing and verification tool for Message Sequence Charts (MSC). SCStudio supports several checkers that are able to verify properties such as realizability, time consistency, equivalence checking between two MSC diagrams, etc. Some of them are well known, while others are new or non-trivial extensions of existing ones. The graphical front-end is implemented as a Microsoft Visio add-on, whereas the checkers are platform independent. SCStudio is an open source project that provides an open interface for additional modules.**

***Keywords*-message sequence charts; checkers; transformers; sequence chart drawing**

I. Introduction

In many areas of computer science nowadays, the late discovery of a bug may lead to expensive consequences. Specific area is a development of network protocols, which are usually long projects. A tool for automated detection of design flaws can be helpful in these situations.

We have experienced these known facts during our collaboration with CMT (Communication, Media, Technology) department of ANF DATA spol. s r.o. (a subsidiary company of Siemens AG Austria). At the beginning, our partners came up with an old communication protocol containing a flaw that had been discovered only during testing phase (see Section **??** for more details). When we modeled the protocol formally, the error was easy to see and even easy to be automatically detected. Actually, the real problem was a lack of formal specification – only structured English text was used in the design phase of protocol development. According to our experience, there are lots of companies that still do not formalize their specification because they regard this as a too expensive and time consuming process. Therefore, in cooperation with ANF DATA, we have developed Sequence Chart Studio, a tool intended for such companies.

Sequence Chart Studio (SCStudio) is designed for modeling message-based communication expressed in the Message Sequence Chart formalism. Message Sequence Chart (MSC) was initially developed by International Telecommunication Union (ITU) as a graphical language for describing scenarios. MSC offers both textual and graphical representation. For more information about formal syntax and semantics of MSC, see, e.g. [**?**], [**?**].

SCStudio is a user-friendly drawing and verification tool that allows its users to start using formal specification and verification progressively. First, the users will benefit from drawing aids and boost their textual specification by formal diagrams of MSC. This will immediatelly save the time of long textual explanation. SCStudio is a Microsoft Visio add-on, hence, it fully benefits from the natural incorporation into Microsoft Office and all native Visio drawing features. Moreover, SCStudio has a number of its own functions that speed up drawing, such as automatic drawing of a message sequence, message flipping, snapping messages to the nearest instance, redrawing into configurable style, etc.

Later on, the user can employ verification and validation features of SCStudio (e.g. consistency checking of time constraints, race checking, etc.). Some of the checking algorithms are well known in the academic community, some of them are new or non-trivial extensions of existing ones. All checkers demonstrate a counterexample if the check fails. Having lots of MSC diagrams, the company can export all Visio drawings into textual MSCs and use SCStudio checkers as stand-alone programs that read textual MSCs from files. Hence, for textual MSC checking, Visio is no longer needed and checkers can be executed from command line or as a batch. As decribed in the User Instructions document supplied with SCStudio, all stand-alone checkers may be installed on MS Windows as well as on Unix platforms.

SCStudio might be very useful also in testing phase. Each company can easily create textual MSCs from timestamped logs of the implemented system. These MSCs, representing executed tests, can be imported to SCStudio and visualized in graphical form using automatic drawing. Due to SCStudio Find Flow function, the MSCs from test logs can be checked whether they follow the specification drawn during the design phase. The Find Flow function returns coverage of the specification by the test MSCs found in it, and a list of the violating MSCs.

To sum up, SCStudio helps companies to gradually change their own documentation policy and easily benefit from formal description and verification. For this purpose, SCStudio has a standard Windows installation package, well arranged help, a document summarizing basic instructions, and an open bug tracking system. SCStudio is distributed under LGPL license. The installation package and the User Instructions document can be downloaded from http://scstudio.sourceforge.net/.

In the following section, we briefly recall the MSC formalism. In Section **??**, we introduce SCStudio front-

end, some functions that allow easy drawing of MSC, and describe some MSC checkers and transformers. Section **??** lists the most relevant MSC tools and compares them with SCStudio. Finally, we present future work and conclusions.

II. The Formalism

Message Sequence Chart is a formalism for specification of asynchronous message-based communication among system components. It is standardized by the International Telecommunication Union (ITU) as Recommendation Z.120 [**?**]. There are two parts of the specification: Basic MSC (BMSC) and High-level MSC (HMSC), the former specifying individual use cases of a subsystem, while the latter describing higher levels of the system and combining several BMSC diagrams together. SCStudio and its algorithms work seamlessly both with Basic and High-level MSCs.

Figure **??** illustrates a basic usage of MSC. Each of the vertical lines, called *instances*, represents a system component. An arrow denotes a *message* sent between a pair of instances.

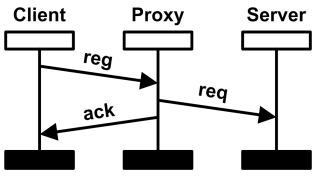

Figure 1. Example of MSC.

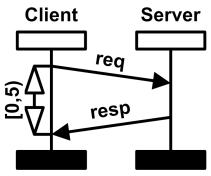

Figure 2. MSC with time.

Formally, each message consists of a *send event* on the sending instance and a *receive event* on the receiving instance. Besides, each message has a *label*. The instance represents top-down flow of time. However, the relative positions of events on an instance do not give any quantitative information about time of their executions. They only imply the order of events on the instance. In addition, each message send event must take place earlier than the corresponding receive event. Sequences of events that satisfy the order specified in the MSC are called *linearizations*. Formally, an MSC specifies a language of linearizations.

The MSC of Figure **??** represents three linearizations. Denoting $s(m)$ the send event and $r(m)$ the receive event of a message with label m, the linearizations are:

- $s(req) \cdot r(req) \cdot s(req) \cdot r(req) \cdot s(ack) \cdot r(ack)$,
- $s(reg) \cdot r(reg) \cdot s(req) \cdot s(ack) \cdot r(req) \cdot r(ack)$, and
- $s(reg) \cdot r(reg) \cdot s(req) \cdot s(ack) \cdot r(ack) \cdot r(req)$.

To allow for specification of time constraints (like timeouts or minimal time required for processing a request), each event is enriched with possible timestamps specifying when the event may take place during a run of the system. Then, a time interval may be set for a pair of events, e.g. see the double-ended arrow in Figure **??**. Such an interval restricts all the possible linearizations, now enriched with timestamps, to just those having difference between the two event timestamps within the interval. For example, in Figure **??**, Client expects a response within 5 time units.

More complex specifications are usually expressed by an HMSC. Intuitively, HMSC can be seen as a finite automaton where every state is labelled by a BMSC. Each run of the automaton represents a BMSC which is the concatenation of the BMSCs along the run. Hence, an HMSC specifies the set of BMSCs represented by all the (accepting) runs. More detailed information can be found in ITU standard and SCStudio documentation.

III. Sequence Chart Studio

The SCStudio front-end is an add-on to Microsoft Visio. The main parts are described in Figure **??**.

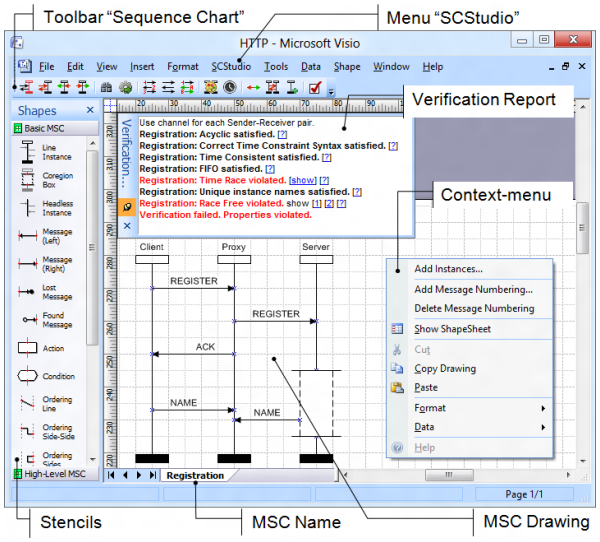

Figure 3. Screenshot of MS Visio with the SCStudio add-on.

On the left side, Basic and High-level MSC stencils are available. From the stencils, individual MSC elements may be dragged and dropped to the diagram. SCStudio functionality is available via the SCStudio menu, toolbar and context menus.

The functionality may be divided into several categories:

- **Drawing aids** for speeding up drawing of MSCs
- **Transformers** changing the diagram in some way:
  - Beautify – redraws the diagram to be well-arranged
  - Tighten Time – restricts time intervals to minimal ranges
  - Repaint – removes all marks from the diagram
- **Checkers** formally verifying the MSC
  - Race checker – checks for race conditions
  - Time Race checker – checks for race conditions considering time restrictions
  - Time Consistency checker – checks whether the MSC is time consistent

  - Strong Realizability checker – checks whether the MSC may be implemented [**?**], [**?**]
  - some trivial checkers used by the checkers stated above
- **Other functions**
  - Find Flow – checks whether a BMSC is contained in another MSC
  - Monte Carlo simulation – prototype analysis of stochastic MSCs

While SCStudio works with the graphical representation of MSCs, it can both import the drawings from and export them to the textual MSC format defined by the ITU-T [**?**]. Moreover, if the diagram is strongly realizable, it may be exported to the DiVinE model checker [**?**], [**?**].

The most interesting features are briefly described in the following text. For even more information, see the SCStudio documentation.

A. Drawing Aids

In practise, there are many stereotypes in drawing MSC. SCStudio helps the user simplify repetitive tasks with several functions that allow to draw some frequently used patterns automatically. First of them is the *Add Instances* function which draws a given number of instances on the active page of the document with given total width or spacing between instances. After instances are created it is possible to generate a sequence of messages among selected instances in given direction by the *Message Sequence* function. Uni- and bidirectional message sequences are both supported. If the user decides rather to create messages manually, SCStudio provides *Message Snapping* function which removes the necessity of connecting every message to the instances manually. If Message Snapping is enabled, all messages are automatically snapped to the nearest instances from the mouse position as soon as the mouse button is released. It is also possible to move or copy messages across instances using keyboard. Another function is *Element Numbering* which can number messages and other types of MSC elements. After specification of the numbering type (e.g. letters) and starting index, all selected elements will be numbered from left to right and from top to bottom. Other drawing aids can be found in SCStudio help or in the User Instructions document.

B. Beautify Transformer

It often happens that a specified MSC is too complex and little redrawing can help with faster and easier understanding the described scenarios. For this purpose, SCStudio offers its *Beautify* transformer that rearranges given MSC according to the user preferences specified in the configuration dialog. For example, an MSC is usually more readable if the order of instances is rearranged such that the number of crossings among messages and instances is minimal. On the other hand, having more MSCs describing communication among the same instances, the user will prefer the same order in all MSCs. Depending on the user configuration, Beautify is capable to rearrange the order of instances, align the length of the instances, and reconcile the slope and indent of messages. The implemented algorithm is based on translation of the requested properties into a linear program which is then solved using an integrated solver. Thanks to Beautify, SCStudio can easily draw MSCs imported from textual form where graphical information is missing. For more information about the translation and all the features of Beautify, please refer to [**?**].

C. Tighten Time Transformer

Time intervals specified in an MSC can be unnecessarily wide. In Figure **??**, the time interval $[0, 5]$ says that all the communication takes at most 5 time units. However, the other interval causes that the second message is sent after at least 3 time units. Hence, the communication will take at least 3 time units and the time interval $[0, 5]$ can be tightened to $[3, 5]$.

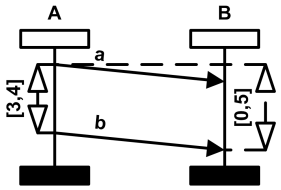

Figure 4. An MSC with a too wide time interval.

The *Tighten Time* transformer of SCStudio changes all time intervals of a given MSC to the strictest values such that the MSC still represents the same set of linearizations. The Tighten Time function can also be used to get a time duration between two events. To know a delivering time of the second message, one would create a new time interval $(-inf, inf)$ between the message events. By running the Tighten Time function, the new interval is changed to $[0, 2]$. This is the best restriction on the delivering time of the second message that can be derived from the original MSC.

Suppose we change the time interval $[0, 5]$ in Figure **??** to $[0, 1]$. Note that there is no time linearization for the new MSC, as the intervals $[3, 4]$ and $[0, 1]$ are in contradiction (a communication could not finish within a time of $[0, 1]$ if its part takes a time of $[3, 4]$). We call such an MSC time inconsistent. SCStudio is able to detect time consistency by the *Time Consistency* checker.

Tighten Time and Time Consistency implementations use the results of [**?**], [**?**], [**?**], [**?**]. For better explanation how we solve problems connected with these functions, see [**?**], [**?**].

D. Race Condition Checker

An MSC contains a race condition if two events are ordered in the MSC, but can occur in the opposite order

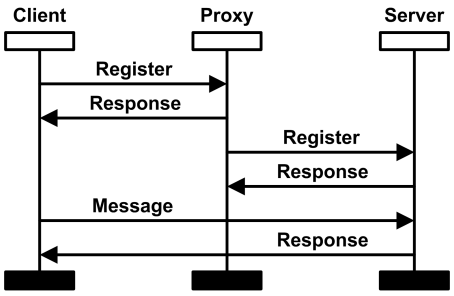

Figure 5. An MSC containing a race condition.

during an actual system execution. We explain the race condition problem using the real protocol flaw mentioned in the introduction. The crucial part of the protocol is depicted in Figure **??**. There are three communicating entities: Client, Proxy, and Server. Client is registered to Server via Proxy. After registration, Client can communicate directly with Server. When the protocol is implemented in a distributed environment, it can happen that delivering the request message from Proxy to Server can take such a long time that the message from Client to Server arrives earlier. This caused a real protocol malfunction.

The race condition problem for MSC was introduced and algorithmically solved in [**?**]. The *Race Condition* checker implements[1] a trace-race algorithm of [**?**]. Trace-race does not differ from race on BMSCs but can be algorithmically checked also on HMSCs (with coregions).

E. Time Race Checker

The Race Condition checker ignores time intervals that can be used to clarify the specification and solve some race conditions. For example, in the protocol of Figure **??**, the designer could use time intervals to express that the delivery of the message from Proxy to Client takes at least 8 time units, while the communication between Proxy and Server is done within 1 time unit. That would guarantee the race could not occur. For taking time intervals into account, SCStudio offers the *Time Race* checker. For more details, see [**?**] and [**?**].

F. Find Flow Function

Deciding equivalency between two scenarios is useful in many situations. Suppose we have an MSC $Spec$ describing some protocol specification and a BMSC $Flow$ representing a scenario from the protocol implementation. The *Find Flow* function indicates whether the specification contains the flow or not. The most challenging task is to return a comprehensive and transparent output of the function, instead of just a simple yes/no answer.

[1]Contrary to the settings of [**?**], SCStudio does not allow open coregions and gates, as all asked industrial partners consider them too complicated for practical use.

Formally, the find flow problem is to decide whether all linearizations of $Flow$ are included in the set of linearizations of $Spec$. The problem can be split into message order checking and time constraint checking.

If both $Flow$ and $Spec$ are BMSCs, the message order checking verifies whether the BMSCs consist of the same messages and represent the same orders of corresponding events, i.e. whether $Flow$ and $Spec$ represent the same set of linearizations. If the sets are not the same, the Find Flow function returns that the $Flow$ was not found and provides a reference to $Flow$ with highlighted differences. To identify the differences, Find Flow implements a non-trivial modification of the diff algorithm for strings published in [**?**]. The Find Flow result for the MSCs of Figure **??** is shown in Figure **??**.

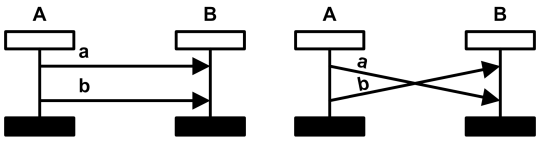

Figure 6. An MSC $Spec$ (left) and an MSC $Flow$ (right).

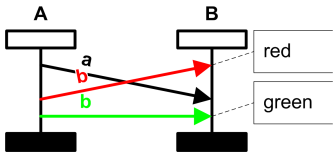

Figure 7. The output of the Find Flow function for the MSCs depicted in Figure **??**. Red message should be removed and the green one should be added in $Flow$ to match $Spec$.

When the message order checking succeeds, the time constraint checking compares time intervals of $Flow$ and $Spec$. Let L_f and L_s be the sets of timestamped linearizations of $Flow$ and $Spec$, respectively. There are three possible results of the constraint checking. First, if $L_f \subseteq L_s$, i.e. the specification $Spec$ contains all runs described in $Flow$, then $Flow$ is found in $Spec$. Second, if $L_f \cap L_s = \emptyset$, i.e. the specification $Spec$ does not contain any of the runs described in $Flow$, then the result refers to $Spec$ with the problematic time intervals highlighted in red. Third, if $L_f \not\subseteq L_s$ and $L_f \cap L_s \neq \emptyset$, i.e. some of the runs of $Flow$ are included in $Spec$ while some are missing, then the result refers to $Spec$ with the intervals causing the missing linearizations highlighted in blue.

An example, where $Flow$ was not found because of time constraints, is depicted in Figure **??**.

If $Spec$ is an HMSC, the message order checking traverses the HMSC and searches for a run the BMSC of which has the same set of linearizations as $Flow$. If there is no such run, the Find Flow function returns that $Flow$ was not found in $Spec$. Otherwise, the time constraint checking is executed

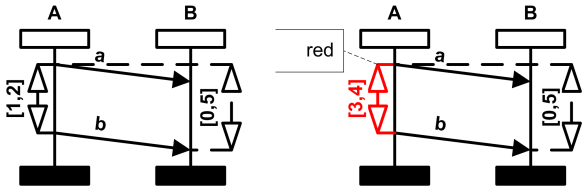

Figure 8. The BMSC on the right is returned when the Find Flow function searches the BMSC on the left in the BMSC of Figure **??**. The problematic time interval $[3, 4]$ is highlighted in red.

on $Flow$ and the BMSC of the run found in $Spec$, and returns the result as already described above. Additionally, if $Spec$ contains the $Flow$, the Find Flow function returns a reference to a copy of $Spec$ where the visited states are highlighted and supplemented with a number of visits.

IV. Related work

There are several tools that work with MSC. The most relevant to the aspects studied in this paper are academic tools Mesa, MSCan, MSC plugin to LTSA and Smyle, and commercial tools Rational Tau, Sandrila SDL and Trace Modeler. We will first review each tool and then provide a short discussion about SCStudio compared to these tools.

Mesa [**?**] provides basic drawing editor for specifying graphical MSCs, supports specification of time constraints, import and export of textual format of MSC formalism and a few verification algorithms. These algorithms check time consistency, process divergence, and local choice properties. Mesa is capable of exporting MSC to the Promela language, which can be used as an input to the SPIN model checker. However, when exporting model in MSC to Promela, Mesa ignores specified time constraints.

MSCan [**?**] provides a large set of verification algorithms that check properties such as strong and weak local choice, local and global cooperativity, and regularity. MSCan offers a counterexample when some of the verified properties are not satisfied. On the other hand, MSCan lacks the specification of time and there is no drawing editor – models are specified in textual form from which they are drawn as a non-editable picture.

MSC plugin to Labelled Transition System Analyser (LTSA) [**?**] allows for creating MSC specifications in a simple graphical editor, obtaining feedback on scenarios that are missing (e.g. due to non-local choice), and adding them as positive or negative scenarios. The plugin is capable of translating an MSC model to the labelled transition system [**?**]. Models can be viewed, edited, analyzed and animated in LTSA. The plugin does not support modeling of any time constructs and, contrary to the ITU recommendation, the messages represent hand-shake communication.

Smyle [**?**] provides an innovative way to design communicating systems. Firstly, the user submits a few examples of valid communication. After that, Smyle tool generates examples and the user decides whether they are valid behaviors of the intended system. In the end, Smyle synthesizes a system model. However, Smyle completely lacks support for time constructs and possibility to check timing aspects of modeled system.

Rational Tau [**?**] is the most advanced of all the MSC tools. It was previously developed by Telelogic, but now it is a commercial product of IBM. Rational Tau supports not only MSCs but also SDL, UML and SOA. It has a good drawing editor and a set of verification algorithms based on state space exploration and simulation of traces of modeled system. It supports specification of time constraints, but we were not able to find any time verification features. Rational Tau supports export of specified models to various programming languages, such as C, C++, C# and Java. It is also capable of importing and exporting MSC models from the Z.120 textual form.

Sandrila SDL [**?**] is very similar to SCStudio. It is also an add-on to MS Visio. It is capable of user friendly drawing of UML, UML2, and ITU-T standardized MSC, SDL, TTCN and URN diagrams. It also supports automated drawing, drawing of timers and time constraints, import and export to standardized MSC textual format. It contains two verification algorithms: a syntax check and a state analysis of process diagrams.

Trace Modeler [**?**] has a user friendly drawing editor for MSCs. It does not support time constructs and verification algorithms. It provides import and export to textual file, but not in standardized textual format of MSC.

Some of the tools contain functions that are same or very similar to the functions of SCStudio. However, none of the tools is a direct competitor to SCStudio in the field of lightweight modeling and verification tools. The commercial tools are really good in drawing MSCs but they are weaker when comparing the verification capabilities. Most of the academic tools are very good in verification because of the fact that most of them support translation of MSCs into other well known formalisms – usually some kind of communicating finite machines. For these formalisms, there are excellent verification tools which can easily detect various design flaws. Usually, the academic tools were developed for this purpose as a proof of concept, that such translation or other functionality is feasible. Because most of the academic tools have not been developed for easy drawing, they have either poor or no drawing support. Almost all of the above-mentioned tools provide no time constructs and no counterexamples when flaw is detected in MSC models.

Obviously, there are tools which also provide functionality not supported by SCStudio. For instance, Smyle supports specification using dedicated learning techniques, or IBM Rational Tau is capable of generating code in various programming languages from the model. We have evidence from companies we were working with that this function-

ality is not crucial for the use case for which SCStudio is developed (specified in the introduction). However, this functionality may be very helpful in some specific cases. In such situations the users of SCStudio can easily export their diagrams from SCStudio to the textual file, which can be imported by the other tools and their functions can be used.

V. Conclusions and Future Work

SCStudio is a drawing and verification tool for MSC. The intention of this tool is to promote formal documentation and verification in industry. Combining a user-friendly editor, functionality supporting correct design and testing, and features requested by industry, it strives to be used as a basic tool for communication protocols development. Its integration with MS Office enables a company to engage formal mechanisms into existing development processes gradually and without too much effort.

There are various possibilities to further target real-world deployment. One of them could be to implement import from *pcap* format to allow analysis of traffic captured from network devices. Another point of extension could be a possibility to export a drawing to LaTeX source code. The Beautify transformer might be extended to allow for storing configuration profiles, which could be easily switched. Regarding the support of MSC elements, inline expressions could be provided to further simplify drawing MSCs. Moreover, conditions are ignored by current checkers and transformers, even though they might be evaluated and a condition checker could be developed.

Acknowledgements

We thank Petr Gotthard, Radek Sedláček, Jindřich Babica, Martin Chmelík, Václav Vacek, Ondřej Kocian, Milan Malota, Martin Vodila, Zuzana Pekarčíková, and other SCStudio developers, testers and collaborators. The work was supported by the Czech Science Foundation, grant No. P202/112/P612.

References

[1] R. Alur, G. J. Holzmann, and D. Peled. An Analyzer for Message Sequence Charts. In *TACAS'96*, volume 1055 of *LNCS*, pages 35–48. Springer, 1996.

[2] J. Barnat, L. Brim, M. Češka, and P. Ročkai. DiVinE: Parallel Distributed Model Checker. In *HiBi/PDMC 2010*, pages 4–7. IEEE, 2010. SW retrieved 19-March-2012 from http://divine.fi.muni.cz/.

[3] H. Ben-Abdallah and S. Leue. Mesa: Support for scenario-based design of concurrent systems. In *TACAS'98*, volume 1384 of *LNCS*, pages 118–135. Springer, 1998. SW retrieved 23-January-2012 from http://tele.informatik.uni-freiburg.de/Mesa/.

[4] B. Bollig, J. P. Katoen, C. Kern, and M. Leucker. Smyle: A tool for synthesizing distributed models from scenarios by learning. In *CONCUR'08*, volume 5201 of *LNCS*, pages 162–166. Springer, 2008. SW retrieved 23-January-2012 from http://www.smyle-tool.org/.

[5] B. Bollig, C. Kern, M. Schlütter, and V. Stolz. MSCan – A Tool for Analyzing MSC Specifications. In *TACAS'06*, volume 3920 of *LNCS*, pages 455–458. Springer, 2006. SW retrieved 23-January-2012 from http://aprove.informatik.rwth-aachen.de/~kern/.

[6] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-State High-Level MSCs: Model-Checking and Realizability. *Journal of Computer and System Sciences*, 72(4):617–647, 2006.

[7] IBM. Rational Tau (Version 4.3) [Software]. Retrieved 23-January-2012 from http://www-01.ibm.com/software/awdtools/tau/.

[8] Y. Inghelbrecht. Object-oriented design with trace modeler and Trace4J. *SIGCSE Bull.*, 41(3):375–375, July 2009. SW retrieved 19-March-2012 from http://www.tracemodeler.com/.

[9] ITU Telecommunication Standardization Sector - Study group 17. Recommendation ITU-T Z.120: Message Sequence Charts (MSC), 2011.

[10] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[11] Sandrila Ltd. Sandrila SDL (Version 4.3) [Software]. Retrieved 23-January-2012 from http://www.sandrila.co.uk/visio-sdl/.

[12] W. Miller and E. W. Myers. A file comparison program. *Software: Practice and Experience*, 15(11):1025–1040, 1985.

[13] Z. Pekarčíková. Computer aided layout of message sequence charts. Bachelor's thesis, Masaryk University, 2011.

[14] L. R. Planken, M. M. de Weerdt, and R. P. J. van der Krogt. P3C: A New Algorithm for the Simple Temporal Problem. In *ICAPS 2008*, pages 256–263. AAAI Press, 2008.

[15] V. Řehák, P. Slovák, J. Strejček, and L. Hélouët. Decidable Race Condition and Open Coregions in HMSC. *Electronic Communications of the EASST*, 29:12, 2010.

[16] M. A. Reniers. *Message Sequence Charts: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1999.

[17] S. Uchitel, R. Chatley, J. Kramer, and J. Magee. LTSA-MSC: Tool support for behaviour model elaboration using implied scenarios. In *TACAS'03*, volume 2619 of *LNCS*, pages 597–601. Springer, 2003. SW retrieved 23-January-2012 from http://www.doc.ic.ac.uk/ltsa/msc/.

[18] V. Vacek. New checkers for sequence chart studio. Master's thesis, Masaryk University, 2011.

[19] Ľ. Korenčiak. Time Extension of Message Sequence Chart. Bachelor's thesis, Masaryk University, 2009.

[20] Ľ. Korenčiak. Effective Algorithms for Time Relation Checking in Message Sequence Charts. Master's thesis, Masaryk University, 2011.