Property Verification with MSC

Emmanuel Gaudin and Eric Brunel

PragmaDev, France
{emmanuel.gaudin,eric.brune}@pragmadev.com

Abstract. In the development process the very first phase focuses on the requirements. Most of the requirements are dynamic and describe how the system reacts to a set of stimuli. Not all the possible reactions are listed in the requirements but some mandatory reactions are described that can be seen as properties. Later in the development process is a real system or a representative model of the future system. At that point it is possible to gather execution traces of the real system. Based on the work of the European PRESTO project this paper describes the work that has been done to use the same kind of model in both cases and match one against the other.

Keywords: MSC, PSC, Sequence Diagram, Property verification, Trace, Artemis

1 PRESTO Presentation

The PRESTO project started on April the first 2011, and its duration is 36 months. It is co-funded by the European Commission under the ARTEMIS Joint Undertaking Programme. The ARTEMIS JU aims to achieve effective coordination and synergy of resources and funding from the industry, the Framework Programme, national R&D programmes and intergovernmental R&D schemes, thus contributing to strengthening Europe's future growth, competitiveness and sustainable development.

The partners involved in this project are Teletel (Greece), Thales Communications (France), Rapita Systems (UK), VTT (Finland), Softeam (France), Thales (Italy), MetaCase (Finland), INRIA (France), University of L'Aquila (Italy), Miltech Hellas (Greece), PragmaDev (France), Prismtech (UK), Sarokal Solutions (Finland).

PRESTO stands for imProvement of industrial Real time Embedded SysTems development prOcess, from a technical point of view the project aims at improving test-based embedded systems development and validation, while considering the constraints of industrial development processes. This project is based on the integration of test traces exploitation along with platform models and design space exploration techniques

The expected result of the project is to establish functional and performance analysis and platform optimisation at early stage of the design development.

F. Khendek et al. (Eds.): SDL 2013, LNCS 7916, pp. 19–35, 2013.

The approach of PRESTO is to model the software/hardware allocation, by the use of modelling frameworks, such as the UML profile for model-driven development of Real Time and Embedded Systems (MARTE). The analysis tools, among them timing analysis including Worst Case Execution Time (WCET) analysis, scheduling analysis and possibly more abstract system-level timing analysis techniques will receive as inputs on the one hand information from the performance modelling of the HW/SW-platform, and on the other hand behavioural information of the software design from tests results of the integration test execution.

In order to verify the functional and non-functional properties, two approaches have been taken into consideration:

1. Verification on the model
   Model checking was proposed in the 1980s independently by Clarke and Emerson [1] and by Quielle and Sifakis [2]. It aims at testing the correspondence between a logical formula against a mathematical structure, i.e., a model. Model checking is an important member of the family of formal methods, together with testing and deductive verification, all aimed at improving the reliability of systems.
   In recent years model checking has gained popularity due to its increasing use for software system verification even in industrial contexts [3,4]. However the application of model checking techniques is still prevented by the state explosion problem. As remarked by Gerald Holzmann in [5] no paper was published on reachability analysis techniques without a serious discussion of this problem. State explosion occurs either in systems composed of (not too) many interacting components, or in systems where data structures assume many different values. The number of global states easily becomes enormous and intractable.
2. Verification on the traces
   In that approach the system is considered as a black box and a set of typical and representative execution traces are gathered. The functional properties and non-functional properties are then verified on theses traces. The main interest with that approach is that the traces can come from a simulated model or from a real target. This will help to make sure the model is representative of the target.
   Anca Muscholl and Doron Peled have investigated in [6] the automatic verification (model checking) of MSCs, as well as the expressiveness of MSCs, in particular the ability to express communication protocols. Jindrich Babica has discussed Message Sequence Charts properties and checking algorithms in [7]. In [8] the Live Sequence Chart (LSC) language introduces the distinction between mandatory and possible on the level of the whole chart and for the elements messages, locations, and conditions in an MSC. Its primary objective is the application of LSCs in the context of formal system verification.

Of particular novelty in PRESTO is the exploitation of traces for the exclusion of over-pessimistic assumptions during timing analysis: instead of taking all possible inputs and states into account for a worst-case analysis, a set of relevant traces is analysed separately to reduce the set of possible inputs and states for each trace.

As a result the work presented here aims at using the MSC as a basis for expressing the properties, tracing, and verifying the properties on the traces.

2 ITU-T Message Sequence Charts

2.1 Scope

The purpose of the MSC (Message Sequence Chart) [9,10] is to provide a language for the specification and description of the communication behavior of system components and their environment by means of message interchange. Since in MSCs the communication behavior is presented in a very intuitive and transparent manner, particularly in the graphical representation, the MSC language is easy to learn, use and interpret. In connection with other languages it can be used to support methodologies for system specification, design, simulation, testing, and documentation.

2.2 Basic Concepts

Figure 1 illustrates an MSC.

Agent Instance. An agent instance starts with an agent instance head followed by an instance axis and ends with an instance tail or an instance stop as shown in the diagrams below.

Message Exchange. A message symbol is a simple arrow with its name and optional parameters attached to it. The arrow can be horizontal, or the arrow can go down to show the message arrived after a certain amount of time or after another event. A message cannot go up.

When the sender and the receiver are represented on the diagram, the arrow is connected to their instances. If the sender is missing it is replaced by a white circle (found message); if the receiver is missing it is replaced by a black circle (lost message). The name of the sender or the receiver can optionally be written next to the circle.

Timers. An agent instance can start a timer that will automatically send back a message when it times out. While the timer hasn't timed out, the instance can cancel it. Specific symbols are available for timer start, cancel and time out, always attached to the instance performing the action. Figure 2 shows timer elements.

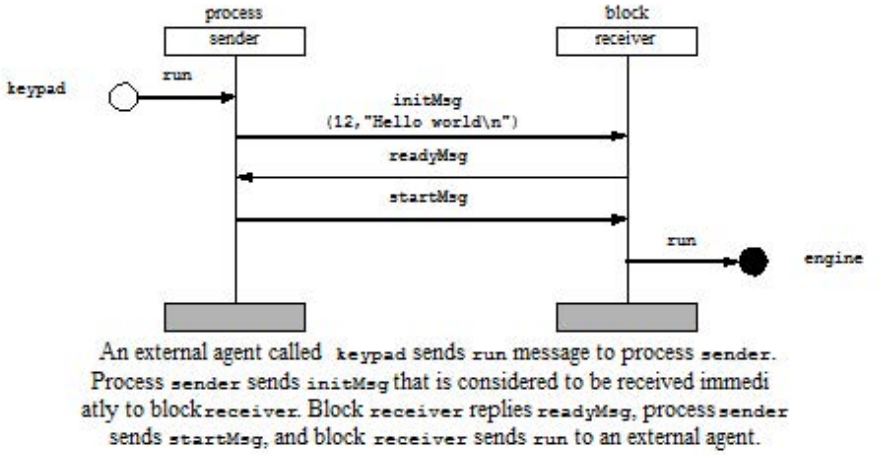

Fig. 1. Message Exchanges in MSC

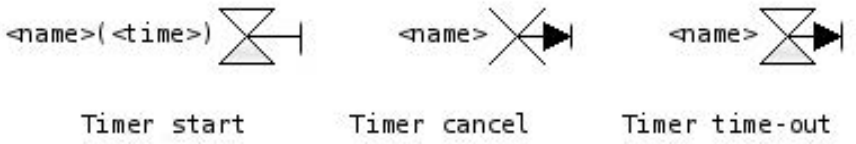

Fig. 2. Timers in MSC

Semaphore Extension. The SDL-RT MSC [9] also introduced the support for semaphore representation. In practice this is rarely used for requirements.

2.3 Inline Expressions

Special semantics can be added to MSC diagrams by the means of inline expressions. These can enclose one or several parts of the diagram and specify that:

- they are optional (opt);
- one or the other part can happen (alt);
- the part can be repeated (loop);
- the parts happens in parallel (par);
- the ordering within the part is not significant (seq).

An example of an alternative inline expression is given on the diagram in Fig. 3:

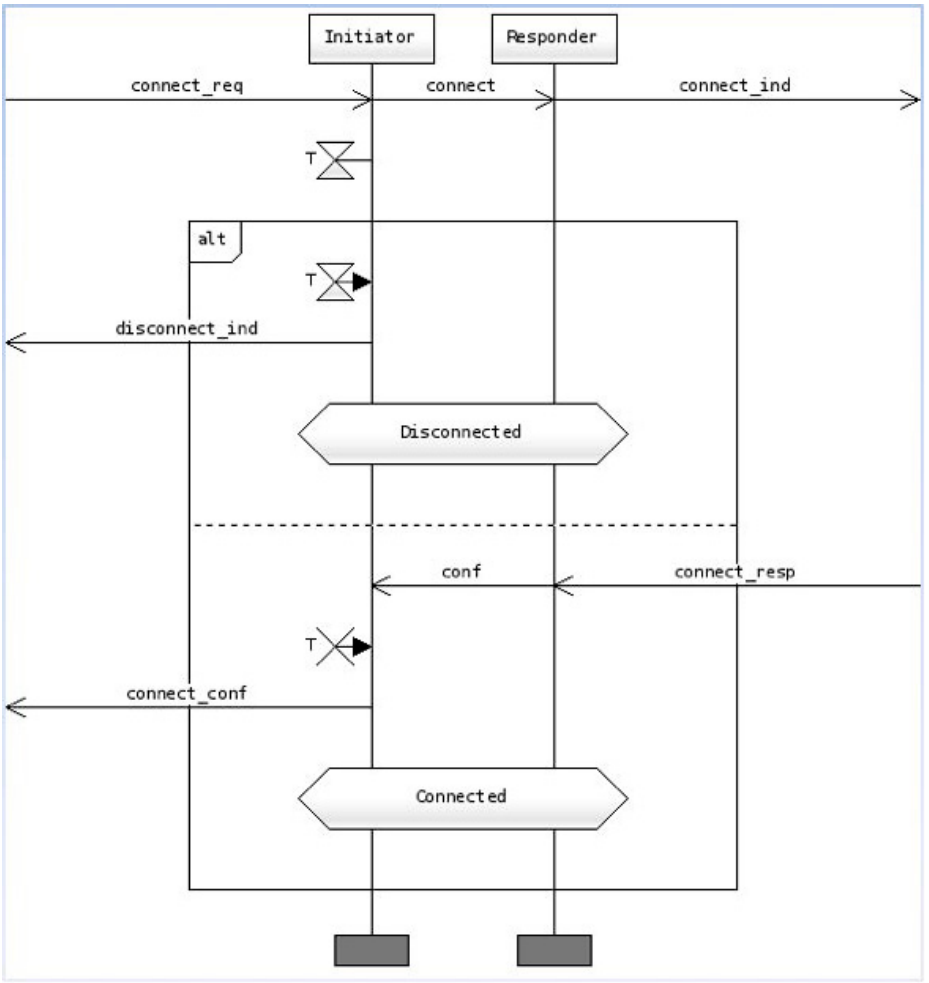

Fig. 3. Inline alternative expression in MSC

2.4 Time Constraints

It is possible to express a relative time constraint in the MSC diagram, specifying a constraint on the time between two events in the diagram. That would define a typical non-functional property of the system. See Fig. 4.

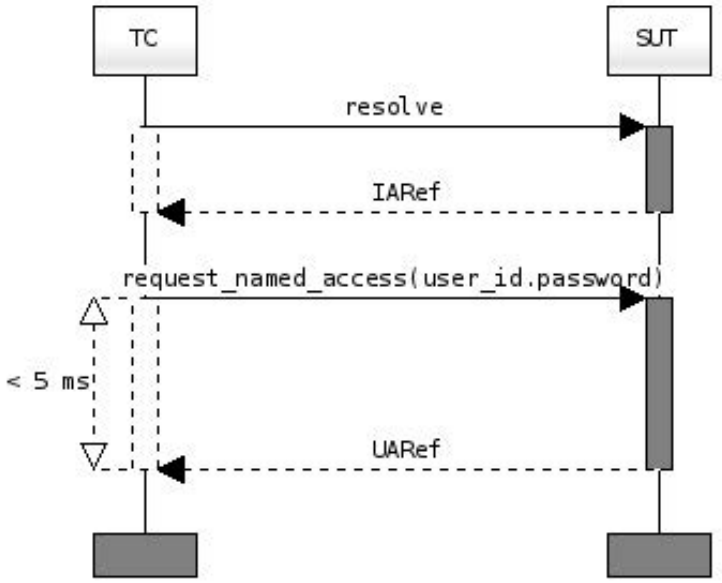

Fig. 4. Time constraint in MSC

3 Property Sequence Chart

Property Sequence Chart (PSC) is a simple but expressive formalism that has been proposed to facilitate the non trivial and error prone task of specifying temporal properties in a correct way and without expertise in temporal logic. PSC is a language that extends a subset of UML 2.0 Interaction Sequence Diagrams or the ITU-T Message Sequence Chart. Further details might be found in [9].

Within the PSC language, a property is seen as a relation on a set of exchanged system messages, with zero or more constraints. PSC may be used to describe both positive scenarios (i.e., the "desired" ones) and negative scenarios (i.e., the "unwanted" ones) for specifying interactions among the components of a system. For positive scenarios, PSC allows to specify both mandatory and provisional behaviours. In other words, it is possible to specify that the run of the system must or may continue to complete the described interaction.

Figure 5 shows the available symbols in PSC diagrams.

Instances are represented as in MSC diagrams. The parallel, alternative and loop operators are represented the same way as the par, alt and loop inline expressions in MSC diagrams respectively. The relative time constraint has the same representation and semantics as in MSCs.

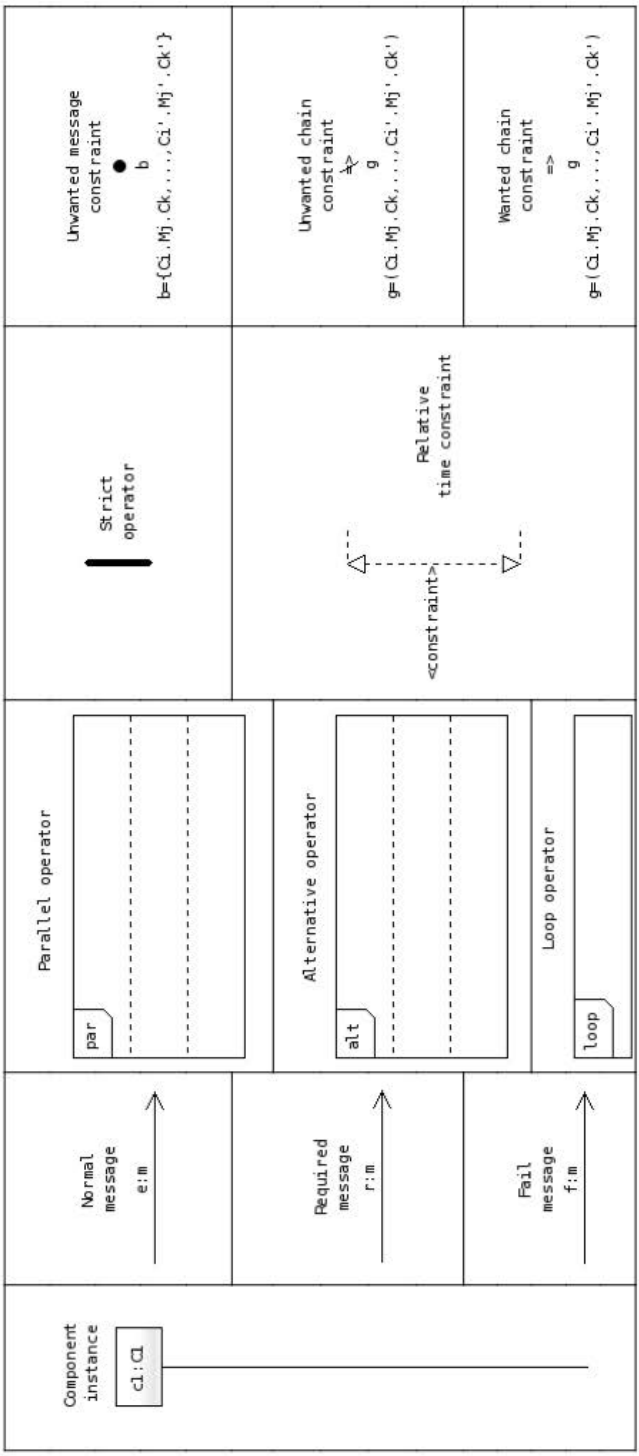

Fig. 5. PSC Graphical notation

Messages in PSCs have two representations:

- An arrow going from the sender to the receiver, just as in MSC diagrams;
- A textual representation, with the format "<sender instance name>.<message name>.<receiver instance name>". This representation is used in constraints, explained below.

Unlike messages in MSC diagrams, message arrows in PSC diagram can have three kinds:

- A regular message, identified by the prefix "e:" for the message text, is a precondition for what follows.
- A required message, identified by the prefix "r:" for the message text, is a message that must occur if the preconditions are met. Required messages must always appear after all regular messages.
- A fail message, identified by the prefix "f:" for the message text, is a message that must not occur if the preconditions are met. Fail messages must also always appear after all regular messages.

When describing a property, the default ordering is the loose ordering: anything can happen between a message specified in the PSC and the one following it. For cases where a strict ordering is necessary, i.e when a message in the PSC must be directly followed by the one following, the strict operator can be used, either on a message send or a receive. See Fig. 6.

The PSC diagrams also allows to set constraints on the messages. These constraints are shown as symbols at the beginning or end of message arrows with an associated text. These constraints can have three types:

1. An unwanted message constraint denotes a set of messages where none should happen before or after the message it is attached too, depending on whether it appears at the beginning or the end of the arrow.
2. An unwanted chain constraint denotes a sequence of messages that should not appear as a whole before or after the message it is attached to.
3. A wanted message constraint denotes a sequence of messages that must appear as a whole before or after the message it is attached to.

A simple example of a PSC diagram is shown in Fig. 7: According to the semantics described above, the property can be read as follows:

- If a message "login" is sent from UserInterface to ATM (normal message "e:login"),
- If a message "wReq" is sent from UserInterface to ATM after the login, without a "logout" message sent from UserInterface to ATM in between (normal message "e:wReq" with the unwanted message constraint "UserInterface. logout.ATM");
- Then a message "uDB" must be sent from ATM to BankDB, unless a message "logout" has been sent from UserInterface to ATM before (required message "r:uDB" with unwanted message constraint "UserInterface.logout.ATM").

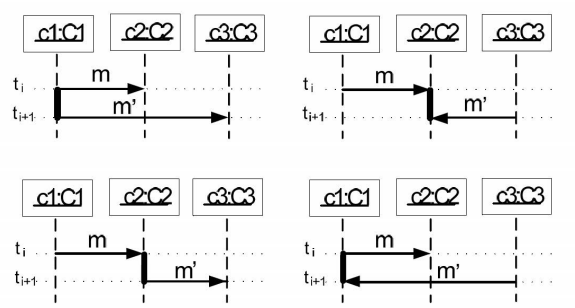

Fig. 6. PSC strict operator example where m' must immediately follow m

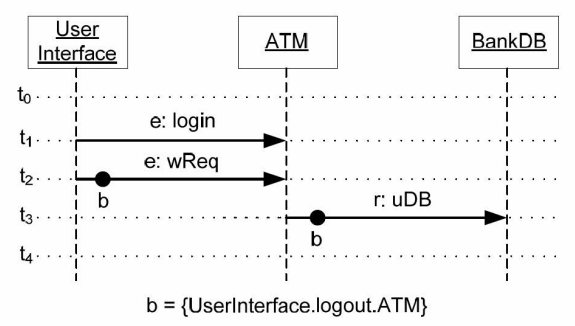

Fig. 7. Example of a property expressed in PSC

The textual notation of PSC together with denotational and operational semantics of the language can be found in [11,12].

4 Using the Same Representation

PSC representation is very close to the MSC representation. In the frame of the PRESTO project the idea is to use the same tool to express both the properties and view the execution traces.

The following sections describe the PRESTO enhancements to the new MSC tracer by PragmaDev. MSC tracer can be used by designers and testers of embedded systems to visualise the flow of control of a system. The benefit of the enhancements is to be able to express a property at the same level as the trace that will be generated by the execution or the simulation of a system. The new research enables the user of the software to write non-functional properties and functional properties at a high level using the well known MSC standard.

5 Functional Property Verification

5.1 Verification for Consistency between Properties, and Collected Traces

The goal of these techniques is to verify that the execution traces conforms to the identified properties. This verification activity checks the consistency between the running system (represented as the observed traces) and the system requirements. In this context, three levels of diagrams will be considered:

1. Requirements
2. Properties
3. Traces

Traces are real execution traces or simulated traces on which a set of requirements or a set of functional properties must be verified. A requirement is basically an expected behavior of a system. It may contain alternatives or loops.

In the example in Fig. 8 the expected scenario is either Stimulus1, Reaction1, EndOfScenario sequence, or the Stimulus1, Reaction2, EndOfScenario sequence.

The traces shown in Fig. 9 will therefore both verify the requirements.

This was a basic approach but the PRESTO project context showed that when it comes to a property, things are slightly different. A property will basically say that if a specific set of events occur, then the following event must or must

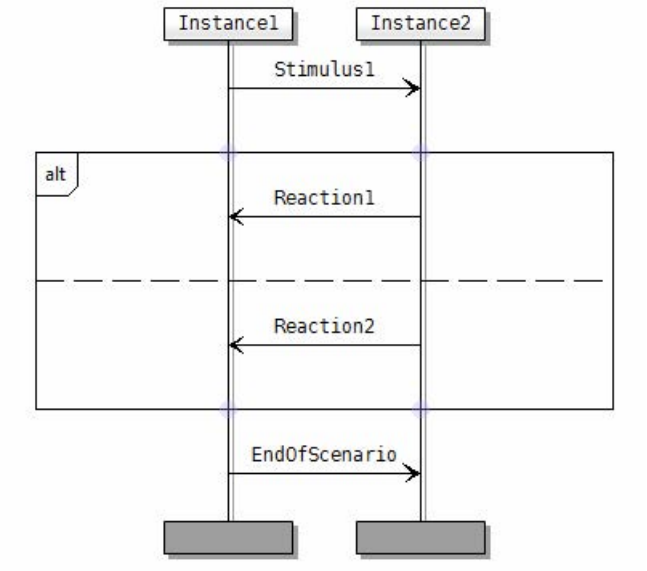

Fig. 8. A simple requirement

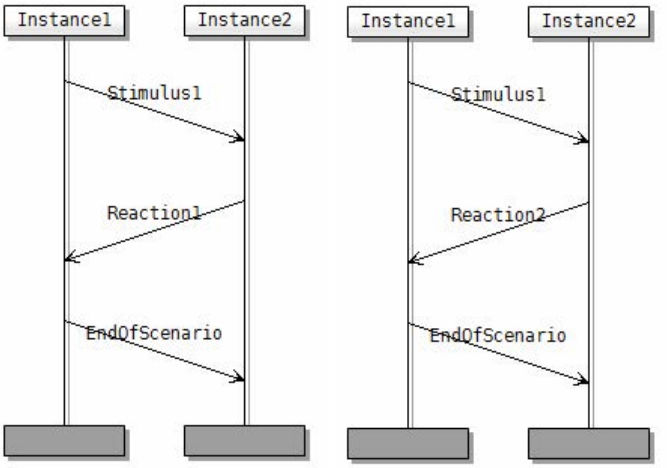

Fig. 9. Two simple traces

not occur. This is very close to what is already available in an MSC. It is just a question of marking the events as a condition, a required event, or a failed event. For that matter the PSC (Property Sequence Chart) complementary notation to the MSC has been adopted.

Therefore the events in the scenario will look like the ones in Fig. 10.

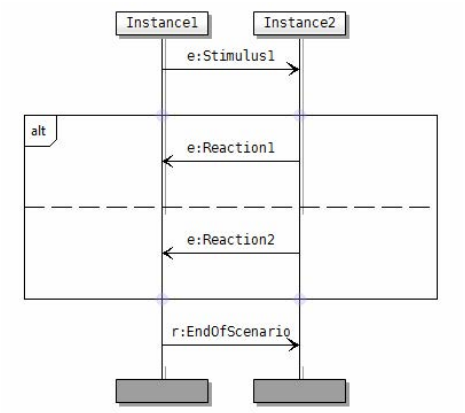

Fig. 10. A property using the PSC notation

That scenario means that if the sequence Stimulus1, Reaction1, or Stimulus1, Reaction2 occurs then the EndOfScenario must occur to verify the property.

This will have very little impact on the look and feel of an MSC but that changes the semantics of the diagram. Therefore three levels of checking have been considered:

- "Basic MSC diff" that makes a simple difference between two diagrams. At this level any logical or graphical difference is considered. A diagram containing an alternative with m1 at the top and m2 at the bottom, and a diagram with m2 at the top and m1 at the bottom are considered different.
- "Spec vs trace" that will handle alternatives and loops in the spec. The two diagrams in the example above would match in that configuration. It is typically intended to deal with a specification diagram without any property against a real execution trace.
- "Property match" that will verify a certain set of events will lead to a required set of events as described in a PSC.

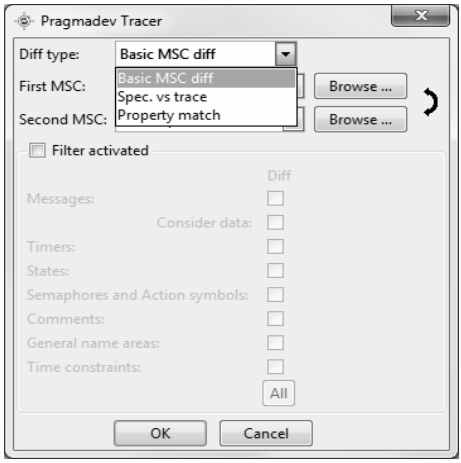

Fig. 11. The PragmaDev Tracer diff options

This has been implemented in the free PragmaDev Tracer prototype; the compare window allows to choose among the different options and appears as in Fig. 11.

Once the property and the trace have been selected, the tool checks the properties on the trace. As a result both diagrams are opened and a third window displays the result of the verification as shown in the example in Fig. 12.

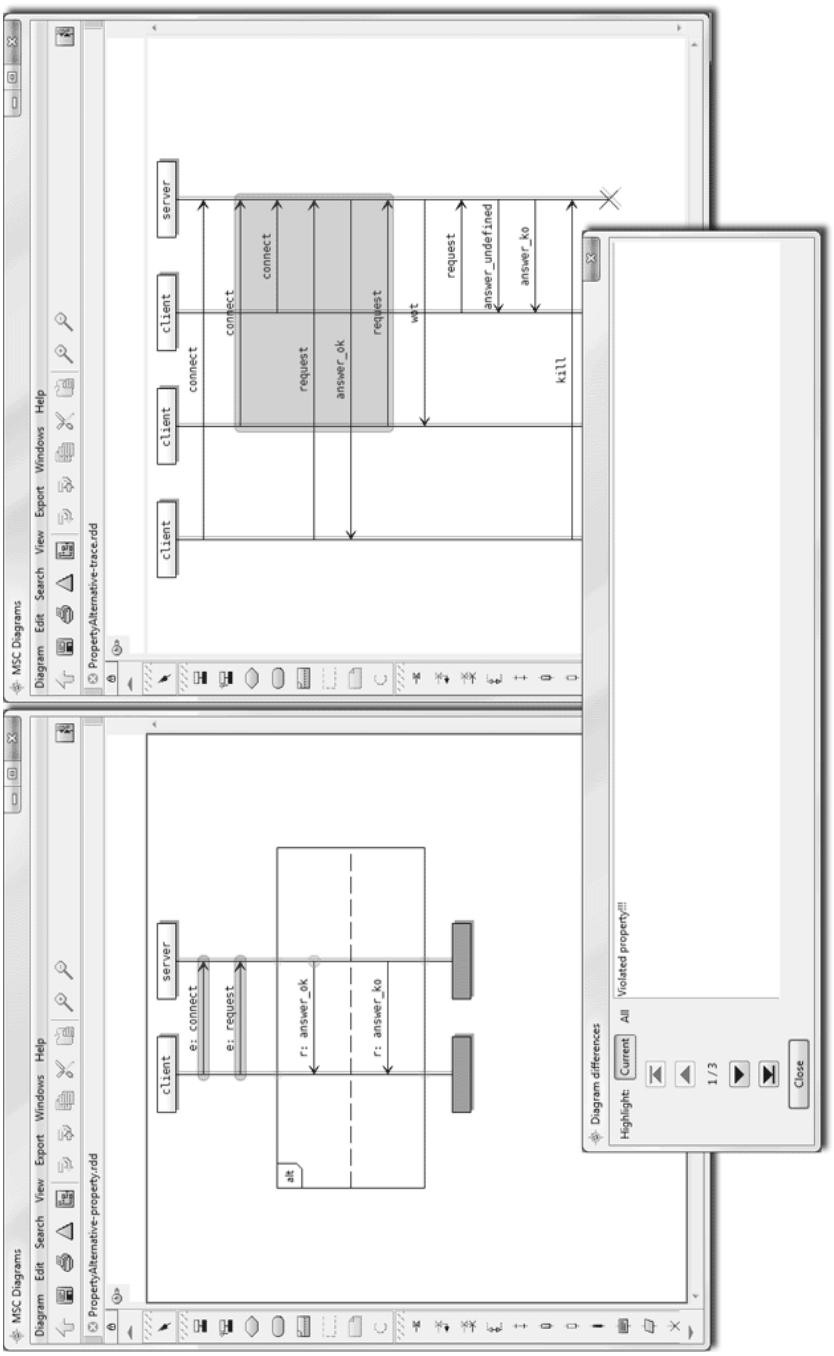

Fig. 12. The property is violated

6 Non-functional Properties

The basic idea is to write non-functional properties at a high level using international standard MSC. One of the main non-functional properties that can be expressed in an MSC is a time constraint in which a set of events must take place in a given amount of time. In the example in Fig. 13 the non-functional property states the exchange of messages between InstanceA and InstanceB must take place within 5 units of time.

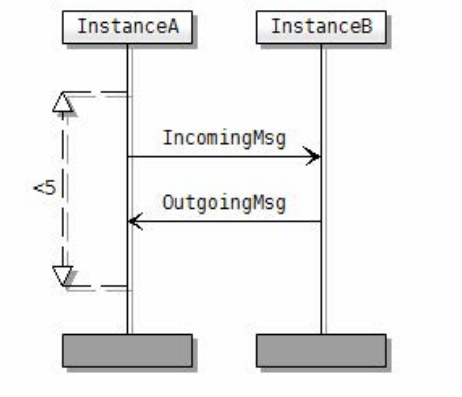

Fig. 13. MSC Message deadline

6.1 Traces

In the trace coming from a real target or from a simulated target, events come with timing information. In the example in Fig. 14 the IncomingMsg is sent at 100 and received at 102. The answer OutgoingMsg is sent at 104 and received at 106. The overall sequence is therefore done in 6 units of time. The new PragmaDev Tracer developed in the context of the PRESTO project can now compare the time constraint in the requirements with a real execution trace.

6.2 Property Verification

In our example in Fig. 15 the time constraint is not fulfilled in the execution trace. PragmaDev Tracer will show clearly the time constraint in the property diagram and the corresponding events in the trace for analysis.

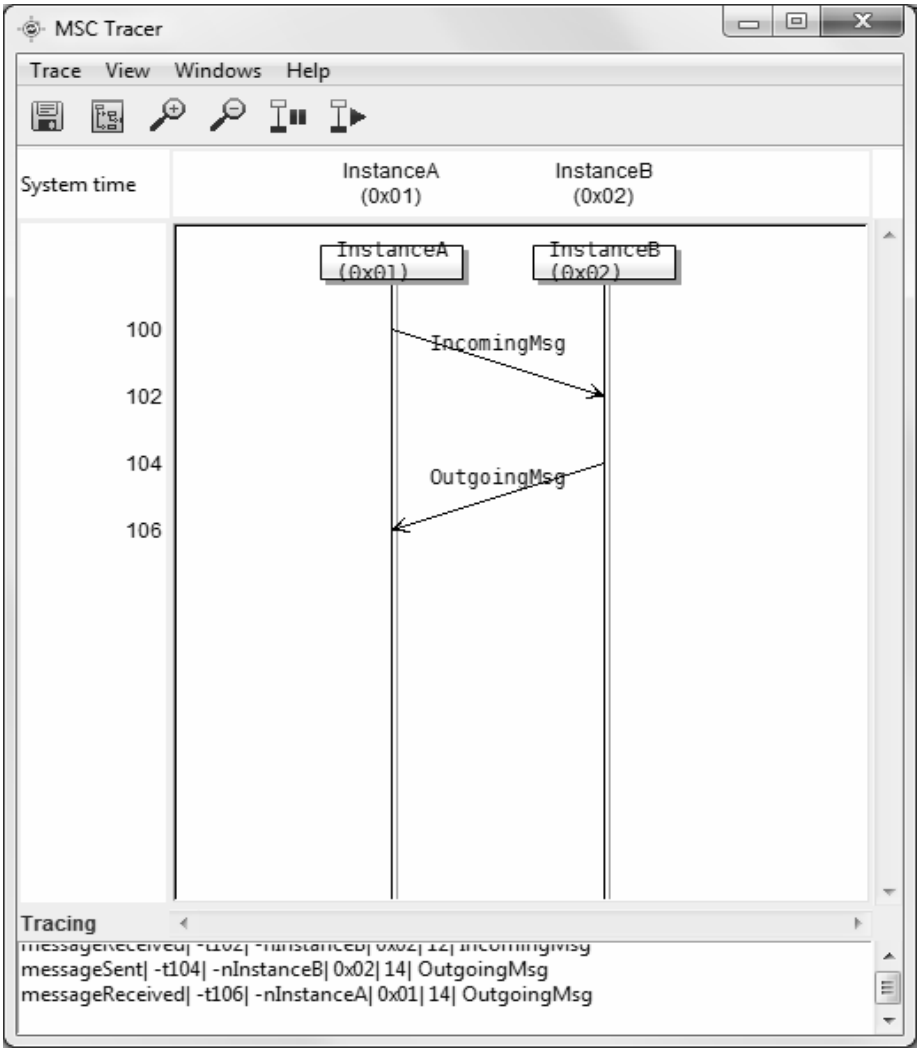

Fig. 14. Screenshot of a real execution trace

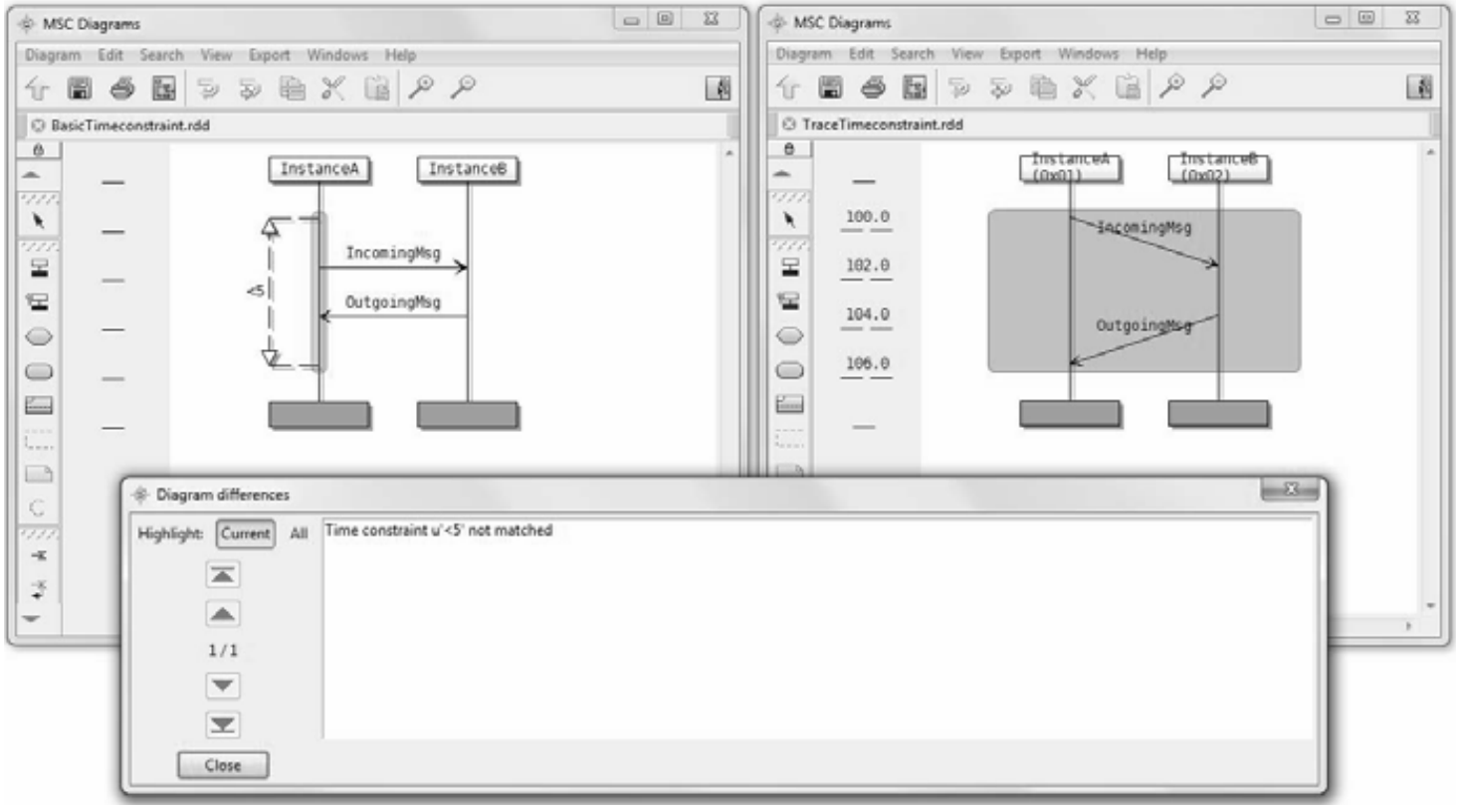

Fig. 15. The timing property is not fulfilled

7 Conclusion

The ITU-T MSC and the PSC are two very close notations that can be used to trace a system behavior and to express properties. The possibility to use both notations in the same tool that will eventually match the properties on real or simulated traces will definitely simplify the verification process.

Thales Italy, one of the partners of the PRESTO project is currently experimenting the new tracer on a real industrial use case. Based on this experiment the tracer that is free to download [13] is constantly evolving.

References

1. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
2. Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
3. Compare, D., Inverardi, P., Pelliccione, P., Sebastiani, A.: Integrating model-checking architectural analysis and validation in a real software life-cycle. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 114–132. Springer, Heidelberg (2003)
4. Holzmann, G.J.: The SPIN Model Checker – Primer and Reference Manual. Addison Wesley (2003)
5. Holzmann, G.J.: The logic of bugs. In: Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering, SIGSOFT 2002/FSE 2002, pp. 81–87. ACM (2002)
6. Muscholl, A., Peled, D.: Deciding Properties of Message Sequence Charts. In: Leue, S., Systä, T.J. (eds.) Scenarios. LNCS, vol. 3466, pp. 43–65. Springer, Heidelberg (2005)
7. Babica, J.: Message Sequence Charts properties and checking algorithms. Master Thesis at Masarykova Univerzita Fakulta Informatiky Brno (2009), http://scstudio.sourceforge.net/files/thesis_babica09.pdf
8. Brill, M., Damm, W., Klose, J., Westphal, B., Wittke, H.: Live Sequence Charts. In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) INT 2004. LNCS, vol. 3147, pp. 374–399. Springer, Heidelberg (2004)
9. Specification & Description Language - Real-Time (2006), http://www.sdl-rt.org/
10. Internation Telecommuication Union: Recommendation Z.120 (02/11) Message Sequence Chart (MSC), http://www.itu.int/rec/T-REC-Z.120
11. Autili, M., Inveradi, P., Pelliccione, P.: Graphical scenarios for specifying temporal properties: an automated approach. Automated Software Engineering 14(3), 293–340 (2007)
12. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite-State Verification. In: Proceedings of the 21st International Conference on Software Engineering, ICE 1999, pp. 411–420. IEEE Computer Society (1999)
13. PragmaDev Tracer, http://www.pragmadev.com/product/tracing.html