

Notational and Methodical Issues in Forward Engineering with MSCs

– Position Paper –

Ingolf Krüger*

Institut für Informatik,
Technische Universität München,
D-80290 München, Germany
<http://www4.in.tum.de/~kruegeri/>

1 Introduction

Message Sequence Charts (MSCs, for short) and similar notations for component interaction have received considerable attention in both academia and industry over the past few years. Despite this growing interest in MSCs their methodical potentials for forward engineering are still almost entirely unexploited in practice. We attribute this to the following observations: 1. the popular MSC notations lack adequate expressiveness and semantic foundation for specifications typically arising during forward engineering (overlapping specifications are just one example); 2. the separate methodical roles of MSC specifications within the development process – ranging from exemplary to complete behavior specifications – are not transparently accessible within MSC specifications; 3. refinement notions for MSCs – if present at all – focus mainly on structural aspects instead of addressing also property and message refinement; 4. procedures for transforming MSC specifications into automata lack adequate positioning within the development process.

Based on our approach at a more seamless integration of MSCs in the development process for distributed, reactive systems described in [Krü00] we discuss these points in more detail in the remainder of this position paper.

The focus of MSCs is on the graphical representation of collaboration scenarios and interaction patterns; this complements the strengths of automata models, such as ROOMCharts [SGW94] and statecharts [HP98], whose major application context is the complete specification of individual component behavior. In this sense, MSCs and automata span two different coordinates of the modeling task. MSCs represent projections of the complete system behavior onto particular (possibly partial) *services* or *use cases*, whereas automata represent projections of the complete system behavior onto individual components.

*This work was supported with funds of the Deutsche Forschungsgemeinschaft within project InTime, and by the Bayerische Forschungstiftung.

One of the most common applications for MSCs within the development process is to use them as informal documentation to illustrate partial interaction sequences. This way of using MSCs has been triggered by their roots in telecommunication protocol illustration, as well as by the increasing popularity of scenario- and use case-oriented development methods.

Yet, using MSCs for informal documentation purposes alone means throwing away much of the “design knowledge” contained in the depicted interaction patterns. This becomes particularly obvious in view of the increasing importance of component-oriented development and its supporting software architectures. Here, the development focus is on the identification of an adequate component structure of the system under consideration, as well as on the specification of the structural and behavioral interfaces between the identified components. MSCs contain information on the system structure (component distribution, message signature) and behavior (interaction pattern). Exploiting this information is one important step in a constructive, methodical MSC usage.

Clearly, however, the application of MSCs within a forward engineering approach calls for their more seamless integration into the development process compared to what is available today. Treating MSCs as a “full-fledged” description technique from which there is even a direct transition to individual component specifications certainly assigns a more prominent role to MSCs, as compared to the one they have traditionally played. The methodical aim, then, is not only to capture or document requirements and system traces by means of MSCs, but also to manipulate the captured requirements by refining them (to make the specification more and more specific), or even to use MSCs directly for the construction of corresponding component specifications and implementations.

Establishing seamlessness in the integration of MSCs into the development process, therefore, requires considering the following development tasks (see also [SGW94]): *scenario elicitation* denotes the capturing of interaction requirements in the form of scenarios, *scenario composition* or *scenario completion* denotes the composition of different scenarios to transit from particular instances of behavior to complete behavior descriptions, *scenario refinement* denotes the adjustment of the level of detail of a scenario, *scenario transformation* denotes the derivation of other forms of specifications, in particular those for individual components, from scenarios.

Clearly, this requires a much more thorough understanding of MSCs than is needed if they only represent exemplary patterns of behavior. In particular, the following four major topics are, in our view, essential for successfully applying MSCs in forward engineering:

- **Precise and Expressive MSC Syntax and Semantics:**

The MSC notation employed must provide the required expressiveness for capturing *and* composing scenarios. Systematic forward engineering with MSCs is possible only if the developer can express the relevant interaction properties of the system unambiguously within the notation under consideration.

- **MSC Refinement:**

The notation and its supporting method must allow us to switch easily between the levels of detail within a given specification; this enables systematic development steps leading from more abstract to implementation-oriented specifications.

- **MSCs for Exemplary and Complete System Behavior:**

Most informal uses of MSCs are based on a purely exemplary interpretation of the depicted interactions: the system may (but need not) exhibit the corresponding interaction pattern. Part of the transition to an implementation of the system is, however, a specification of complete system behavior. Therefore, the notation or the accompanying method (or both) must provide means for distinguishing clearly between exemplary and complete behavior specifications.

- **MSC Transformation:**

Transformation procedures allowing us to *generate* component implementations from a specification of interaction requirements in the form of MSCs are the “back end” of forward engineering: they link global interaction and local state transition specifications to provide component prototypes (semi-)automatically.

In the following sections we discuss the requirements these topics pose at adequate notational and methodical support for MSC usage in forward engineering. In Section 2 we focus on syntax and semantics, whereas we stress methodical issues in Section 3. Along the way we sketch our own approach to providing a seamless integration of MSCs into the development process for distributed systems. For a more detailed discussion of this approach we refer the reader to [Krü00, Krü99, KGSB99, BK98].

2 Notational and Semantic Considerations

An important aspect of using MSCs for forward engineering is their adequate expressiveness; only if the MSCs can express the relevant structural and behavioral properties of the system (part) under development will they be accepted and considered useful as a means for system specification beyond informal requirements documentation. Besides notation to capture the requirements under consideration we believe that also the existence of a precise semantics foundation for the notation is indispensable.

To name just a few notational elements we consider important for MSC specifications of nontrivial size and scope, we mention the following: repetition (bounded and unbounded finite repetition, infinite repetition), composition operators (for guarded alternatives, concatenation, parallelism, overlapping interactions, referencing, hierarchy), control and data state indicators, preemption/exceptions, progress/fairness constraints.

These notational elements are particularly necessary for supporting the transition from exemplary scenarios to complete behavior descriptions. Whereas this is obvious for repetition and most of the composition operators, a few words are in order to discuss overlapping interactions, state indicators, preemption, and fairness constraints.

In typical MSC specifications the individual MSCs are non-orthogonal. Often, the same component appears multiple times within different MSCs, playing either the same or different roles within the corresponding interaction patterns. This calls for a composition operator within the MSC notation enabling the developer to express that certain parts of the operands’ interaction patterns overlap.

State indicators allow the developer to establish a link between (global) component interactions and local component behavior. This link is important for obtaining manageable results

in the transition from MSCs to automaton representations of individual component behavior (see Section 3).

A notational concept for preemption is required particularly in application domains where exceptional behavior is a major part of the system specification. This is, for instance, true for technical systems in telecommunications, avionics, and in the automotive sector. Without an adequate preemption notation MSC specifications cannot mature beyond being exemplary documentation in these domains. A corresponding statement also holds for progress and fairness constraints.

As just one example MSC notation displaying substantial potential for improvement with respect to notation and semantics foundation we mention the UML's SDs (cf. [RJB99]). SDs have syntax for alternatives and parallelism; yet, this syntax doesn't scale well for nontrivial interaction patterns. The SDs' notation for repetition is not precisely fixed. Moreover, SDs provide no explicit notation for referencing and hierarchy, which significantly reduces their suitability for practical applications in forward engineering. Non-orthogonal interactions, preemption and fairness are also not explicitly supported within SDs. On the positive side we note that SDs offer notation for state indicators. Yet, the lack of a formal semantics for SDs, including the absence of a formal mapping between the state information contained in SDs and corresponding statecharts for individual components, reduces the potential for a seamless SD usage within the development process further.

An analysis of existing MSC notations [Krü00] shows that, in fact, none of them includes all of the "features" listed above. This makes it difficult, if not impossible, to employ these existing MSC notations for forward engineering of nontrivial specifications. Therefore, in [Krü00] we have developed an MSC dialect which explicitly addresses these features. The semantics of this dialect bases on a precise, mathematical system model for reactive systems, which allows us to integrate the notions of interaction and state in MSC specifications. With a few exceptions, the syntax of the chosen MSC dialect adheres to the ITU standard MSC-96 [IT96, IT98]. To solve the deficits identified above, we go beyond the standard syntax and semantics, and add several composition operators for MSCs. These operators allow us to deal systematically with overlapping and exceptional interaction patterns. Moreover, we can use them to add liveness and fairness constraints to MSC specifications.

3 Methodological Considerations

After having discussed some of the notational and semantic issues in using MSCs during forward engineering we now turn our attention to the corresponding methodical aspects. In this section we consider the various possible interpretations of MSCs within the development process, the necessary refinement notions, and the transformation from MSCs to individual component specifications.

The systematic application of MSCs in different development phases can require substantially different interpretations of the same set of MSCs. During requirements capture, for instance, we are typically interested in semantically loose specifications – there is just too little information available to fix the intended system behavior precisely already. With respect to MSCs this means that during requirements capture we need a way for allowing additional components, as well as additional message occurrences within the system under development.

In the transition to an implementation of the system during later development phases, on the other hand, we may want to fix the system’s behavior precisely by explicitly disallowing any other than the depicted interaction patterns. Simulation runs or counter-examples generated from model checkers – represented using MSCs – may also require a “tight” instead of a loose MSC interpretation to be meaningful.

Therefore, we distinguish the following MSC interpretations, listed in increasing degree of tightness: *existential*, *universal*, and *exact* (cf. [Krü00, KGSB99]). The existential interpretation forms the basis for using MSCs as scenario specifications: the depicted behavior can, but need not occur; it also may be interleaved with arbitrary additional interactions. An MSC under universal interpretation specifies behavior that must occur eventually in any system execution. The exact interpretation fixes the system’s behavior to match precisely what the MSC specifies; this interpretation is the foundation for using MSCs as a description technique for complete component and system behavior. These different interpretations make explicit the different methodical roles MSCs can play during the development process.

Especially in application domains where the correctness of specifications is essential, systematic forward engineering must provide means for establishing a traceable link between requirements captured during analysis and properties of the implementation. In [Krü00] we formally define, and thus make methodically accessible, three separate forms of refinement: *property refinement*, *message refinement*, and *structural refinement*. Property refinement allows us to add details to an MSC specification by reducing the set of behaviors represented by an MSC; this corresponds to reducing the amount of underspecification contained in the MSC. Message refinement allows us to replace a single message by an entire protocol in the refined specification (see also the notion of “call compression” in [KM96] and the “subscenario concept” in [KSTM98]). Structural refinement enables us to make the hierarchical distribution structure of individual components explicit. These refinement notions significantly enhance approaches addressing only instance decomposition [IT96] known from the literature.

Once a sufficient degree of detail has been achieved in an MSC specification we may try to exploit the information captured within the MSCs, and automatically derive implementations for the components participating in an interaction pattern. Over the past few years several different approaches and algorithms have been developed for this purpose (cf. [Krü00] and [AEY00, HK99, KM93, KMST96, MZ99] for further references). In our approach we derive an automaton for an individual component specification from a given MSC specification by successively applying four transformation steps: **1. projection** of the given MSCs onto the component of interest, **2. normalization** of the MSCs, i.e. adding missing start and end guards (state indicators), and splitting MSCs with more than two guards at an intermediate guard, **3. transformation** into an automaton by identifying the MSCs as transition paths, and by adding intermediate states accordingly, and **4. optimization** of the resulting automata.

One distinguishing “feature” of this approach is that the developer can guide the quality of the inference algorithm’s output by supplying design knowledge in the form of state indicators before the procedure starts. This is a prerequisite for obtaining manageable result automata.

One of the more promising applications of such synthesis or inference procedures is the automatic derivation of automata for interface specifications as required in component oriented development. Here, the size of the corresponding MSC specification is likely to be such that the resulting automata remain manageable.

4 Conclusions and Outlook

In the preceding sections we have discussed several notational and methodological issues in the application of MSCs within forward engineering. We have identified adequate expressiveness for the specification of both partial and complete interaction behavior, the existence of effective refinement notions supporting systematic development steps, as well as the existence of transformations from MSCs to individual component specifications as important prerequisites for systematic forward engineering with MSCs.

Clearly, a lot of work for driving MSCs closer towards being an adequate description technique within the forward engineering process remains to be done. The thorough treatment of data specifications and states, data parameters within MSCs, and broadcasting communication are only few examples supporting this claim.

Despite the apparent limitations of current MSC notations and their supporting methods we expect the obvious advantages of MSCs – being a description technique for global system behavior as opposed to automata capturing local component behavior – to outweigh the deficits quickly.

Acknowledgments I am grateful to Katharina Spies for reading and commenting on a draft version of this text.

References

- [AEY00] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. In *22nd International Conference on Software Engineering*, pages 304–313, 2000.
- [BK98] Manfred Broy and Ingolf Krüger. Interaction Interfaces – Towards a scientific foundation of a methodological usage of Message Sequence Charts. In J. Staples, M. G. Hinchey, and Shaoying Liu, editors, *Formal Engineering Methods (ICFEM’98)*, pages 2–15. IEEE Computer Society, 1998.
- [HK99] David Harel and Hillel Kugler. Synthesizing Object Systems from LSC Specifications. (submitted), August 1999.
- [HP98] David Harel and Michal Politi. *Modeling Reactive Systems with Statecharts. The STATE-MATE approach*. McGraw-Hill, 1998.
- [IT96] ITU-TS. Recommendation Z.120 : Message Sequence Chart (MSC). Geneva, 1996.
- [IT98] ITU-TS. Recommendation Z.120 : Annex B. Geneva, 1998.
- [KGSB99] Ingolf Krüger, Radu Grosu, Peter Scholz, and Manfred Broy. From MSCs to Statecharts. In *DIPES’98*. Kluwer, 1999.
- [KM93] Kai Koskimies and Erkki Mäkinen. Inferring State Machines From Trace Diagrams. Technical Report A-1993-3, University of Tampere. Department of Computer Science, July 1993.
- [KM96] Kai Koskimies and Hanspeter Mössenböck. Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs. In *Proc. 18th IEEE International Conference on Software Engineering (ICSE’96)*. IEEE Computer Society Press, 1996.

- [KMST96] Kai Koskimies, Tatu Männistö, Tarja Systä, and Jyrki Tuomi. On the Role of Scenarios in Object-Oriented Software Design. Technical Report A-1996-1, University of Tampere. Department of Computer Science, January 1996.
- [Krü99] Ingolf Krüger. Towards the Methodical Usage of Message Sequence Charts. In Katharina Spies and Bernhard Schätz, editors, *Formale Beschreibungstechniken für verteilte Systeme. FBT'99*, 9. GI/ITG Fachgespräch, pages 123–134. Herbert Utz Verlag, June 1999.
- [Krü00] Ingolf Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000. available online via <http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2000/krueger.html>.
- [KSTM98] Kai Koskimies, Tarja Systä, Jyrki Tuomi, and Tatu Männistö. Automated Support for Modeling OO Software. *IEEE Software*, pages 87 – 94, January–February 1998.
- [MZ99] Nikolai Mansurov and D. Zhukov. Automatic Synthesis of SDL models in Use Case Methodology. In *9th SDL Forum, Montreal, Canada*. Elsevier Science Publishers B.V. (North-Holland), 1999.
- [RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [SGW94] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling*. Wiley, 1994.