Analysis and Implementations of MSC Specifications

Carsten Kern
Lehrstuhl für Informatik II, RWTH Aachen, Germany
kern@informatik.rwth-aachen.de

Abstract. In this paper we consider the formal model of (compositional) Message Sequence Graphs – (C)MSGs for short – which provide a standardized modeling language for specifying communication protocols. We are mainly interested in detecting classes of implementable protocols and therefore define properties on (C)MSGs that ensure their implementability. The implementation of a CMSG in our context means to deploy the model of communicating finite-state machines for constructing automata which recognize (up to synchronization messages) the same language as a given (C)MSG. These syntactic properties can be checked by our tool MSCAN. To our knowledge there is no other tool that provides a protocol designer with a likewise great variety of facilities to analyze (C)MSGs. The complexity of algorithms for testing the membership of a CMSG to a certain property class ranges from PTIME to CO-NP completeness. Because of this discrepancy in complexity we group the property classes into two inclusion hierarchies. The first one is a property hierarchy and the second one is a language hierarchy which identifies equivalent property classes. Using the results of these diagrams eases the verification of protocol characteristics. For the final phase of system design we also introduce an implementation of the very important class of local-choice CMSGs, exhibiting the nice property of deadlock freedom.

1 Introduction

The complexity of today’s software systems is increasing rapidly and so is the need for the employment of formal methods to guarantee their reliability. It is desirable to apply formal methods already in the early stages of system design to avoid costly and extensive reimplementation and redesign. When developing communicating systems, it is a widespread design practice to start with drawing scenarios showing the intended interaction of the system in mind. Message Sequence Charts (MSCs), a modeling language at a high level of abstraction, provide a prominent notion to further this approach. They are widely used in industry, standardized [ITU98,ITU99], and similar to UML’s sequence diagrams [Ara98]. An MSC depicts a single partially ordered execution sequence of a system. Moreover, it defines a collection of processes, which, in its visual representation, are drawn as vertical lines and interpreted as time axes. An arrow from one line to a second corresponds to the communication events of sending and receiving a message. But the MSC standard does not only allow to specify single scenarios. To make MSCs a flexible specification language, it also supports choice, concatenation, and iteration, which gives rise to Message Sequence Graphs (MSGs). To ease a protocol designer’s life our goal was, on the one hand, to provide an analysis tool [MSC05] offering a great variety of properties that may be checked for given CMSGs to ensure important protocol characteristics. In chapter 2 the most important of these properties will be described. On the other hand we built inclusion hierarchies for relating these properties and finding classes of equal expressiveness. These hierarchies will be introduced in chapter 3. As the implementation of a protocol is the final step in a protocol-design cycle we propose an implementation for the very important class of local-choice CMSGs in chapter 4 and close with a short section on the tool’s web page in chapter 5.

2 CMSGs and their properties

In the following we define the most important objects we are going to deal with throughout this paper, namely compositional Message Sequence Charts and Message Sequence Graphs.
**Definition 1 (Compositional Message Sequence Chart).** A compositional Message Sequence Chart $M$ consists of:

- a finite, non-empty set of processes $P$,
- a finite, non-empty set of events $E = \bigcup_{p \in P} E_p = S \uplus R$, occurring on the processes and being divided into send events ($S$) and receive events ($R$),
- a function $t$ labeling the events $t : E \rightarrow \text{Act}$,
- a partial and injective function $m : S \rightarrow R$ matching send events to receive events of the correct type (Note: not every send event needs to have a corresponding receive event and vice versa), and
- a partial order $< \subseteq E \times E$ on the events.

This definition can easily be extended to message contents but for the sake of brevity we omitted them here. We call a CMSC an MSC if $m$ is total and bijective.

CMSCs usually describe a single execution of a system. If we want to specify the system itself we need (compositional) Message Sequence Graphs.

**Definition 2 (Compositional Message Sequence Graph).** A Compositional Message Sequence Graph $G$ consists of:

- a graph $\langle V, R \rangle$ ($V \neq \emptyset$, $R \subseteq V \times V$),
- a non-empty set of start nodes $V^0 \subseteq V$, a set of end nodes $V^f \subseteq V$, and
- a function $\lambda$ that assigns a CMSC to each node of the graph.

If, in the CMSG definition, $\lambda$ maps to the set of MSCs than we call it an MSG.

But using MSGs for protocol design easily leads to problems. There is, for example, no possibility to create an MSG for the famous alternating-bit protocol [Tan03]. Thus we will use CMSGs throughout this paper to avoid such problems.

The following definition will be used in the subsequent subsection for describing global communication behavior of CMSGs.

**Definition 3 (Communication Graph (cf. [Gen05])).** The communication graph of a CMSC $M = \langle P, E, C, t, m, < \rangle$ is defined as the digraph containing a node for each active process (i.e., a process with at least one event) in $M$ and there is an edge from node $p$ to node $q$ whenever there is at least one send event on process $p$ with type $p!q$ and one receive event on process $q$ with type $q?p$.

### 2.1 CMSG-Properties

If we design systems, their underlying protocols usually have to fulfill properties like, for example, “sending a message always expects an acknowledgement of the receiver” also known as the regularity-property. In this section we want to itemize further examples of important characteristics of protocols. We will describe most of them intuitively because formal definitions would be too involved.

- **general case**: The general case has no restrictions regarding communication behavior.
- **globally cooperative**: A CMSG $G$ is called globally cooperative if every CMSC labeling a cycle in $G$ has got a connected communication graph. This property assures that on these cycles every process participating in the communication interacts with all others from time to time. This requirement seems to be essential for allowing synchronization between the autonomous processes.
- **regular**: This property restricts the cycles of $G$ to have strongly connected communication graphs and thus ensures that for each sent message the receipt of an acknowledgement becomes possible.

- **locally cooperative**: This property changes the focus from globally observable behavior to local characteristics checking nodes and pairs of nodes instead of strongly connected graph components.

- **local-choice**: This property assures that in choice nodes, i.e., in nodes which possess more than one direct successor, a group of processes (weak local-choice) or even a single process (strong local-choice) may choose the next node to enter. After the choice has been made this information is passed to the other processes. The local-choice property will play an important role at implementing CMSGs in chapter 4.

- **local**: The local property checks the local-choice property not only for each branching node but for all nodes contained in the graph. As with local-choice we distinguish between the weak and the strong version of locality.

The weak versions of local and local-choice, of course, induce some kind of non-determinism into the protocol because different processes may select different successor nodes for system continuation.

In figure 2, on page 5, we present an overview over the complexity for checking the properties we just introduced.

## 3 Property and Language Hierarchy

### 3.1 Property Hierarchy

Due to the huge complexity gap between the property classes concerned with local characteristics of CMSGs and the ones describing global communication structures, we introduce a property hierarchy which depicts the inclusion relation of our syntactic properties. A diagram like this eases the analysis of systems in the way that if easy-to-check (according to figure 2) properties are disproved for a certain CMSG the designer would not have to check for properties higher in the inclusion hierarchy or vice versa if a harder-to-check property would be verified for the given CMSG all properties in the inclusion relation lying beneath it would automatically hold.

On the left side of figure 1 we see the property hierarchy for the properties from chapter 2.1.

![Property Hierarchy Diagram](image)

**Fig. 1.** CMSG-property and language hierarchy

**Theorem 1** (cf. [Ker05]). *The previously introduced properties obey the strict (i.e., “$\subset$”) inclusion hierarchy depicted on the left side of figure 1.*
3.2 Language Hierarchy

Now we want to detect classes of properties which can be proved to be equivalent. For that purpose we define the language a CMSG is describing.

**Definition 4.** The language $L(G)$ described by a CMSG $G$ is the set of all MSCs representing the accepting paths of $G$. (A path in a CMSG $G$ is called accepting if it starts in a start node of $G$ and ends in one of its end nodes.)

Moreover two CMSGs $G$ and $G'$ are said to be equivalent if they describe the same language, i.e., if $L(G) = L(G')$. Now we can specify the language classes we have proved to be equivalent.

**Definition 5.** Language classes:

- $\theta - \text{locCMSG} := \{L(G) \mid G \text{ is a } \theta \text{ local CMSG}\}$
- $\theta - \text{lcCMSG} := \{L(G) \mid G \text{ is a } \theta \text{ local-choice CMSG}\}$

In the following we present the results we have achieved so far describing the center and the right part of figure 1.

**Theorem 2 ([Ker05]).** Every weak, respectively strong local-choice CMSG $G = (V, R, V^0, V^I, \lambda)$ can be transformed into an equivalent weak, respectively strong local CMSG $G'$ of size $O(|V|)$.

Then it directly follows from theorem 2 and figure 1:

**Corollary 1 ([Ker05]).**

- strong - lcCMSG = strong - locCMSG
- weak - lcCMSG = weak - locCMSG

A second transformation relates weak and strong locality.

**Theorem 3 ([Ker05]).** Every weak local CMSG $G = (V, R, V^0, V^I, \lambda)$ can be transformed into a strong local CMSG of size $O(|V|^2)$.

From theorem 3 and figure 1 it directly follows:

**Corollary 2 ([Ker05]).**

- weak - lcCMSG = strong - locCMSG

Using the transformations from theorems 2 and 3 we are able to construct strong local out of weak local choice CMSGs and hence to eliminate the non-determinism resulting from the weak versions of the properties, namely weak locality and local-choice. This result is important if one wants to implement CMSGs. If a given CMSG fulfills the weak local-choice property without satisfying the strong version, then there is at least one branching node on which two distinct processes may decide on the further progress of the system. If these processes make different choices the implementation may run into a deadlock. Thus it is desirable to eliminate such behavior whenever this is possible. Our transformation algorithms resolve the problem by transforming the weak property versions into the strong ones. In the next section we will concentrate on an implementation for local-choice CMSGs keeping the results from this section in mind.

4 CMSG implementations

The main goal of defining the properties from section 2.1 is to identify classes of implementable (C)MSGs. Implementability in this context means that the CMSG can be transformed into communicating automata which represent the processes in the CMSGs exchanging messages among each other. For this purpose we deploy the formal model of communicating finite-state machines (CFMs).
Definition 6 (Communicating Finite-State Machines). A Communicating Finite-State Machine over a process set $\mathcal{P}$ is defined as: $A = ((A_p)_{p \in \mathcal{P}}, \text{Sync}, F)$.

- For every $p \in \mathcal{P}$, the CFM $A$ possesses a finite automaton $A_p = (S_p, s_p, \rightarrow_p)$ over the finite, non-empty set of actions $\text{Act}_p$ with:
  - a finite, non-empty set $S_p$ of local states,
  - a starting state $s_p \in S_p$ and
  - a transition relation $\rightarrow_p \subseteq S_p \times \text{Act}_p \times \text{Sync} \times S_p$ describing the state changes of $A_p$ (Sync is a finite set of synchronization messages for synchronizing the local automata. It is basically needed for deadlock prevention.)

- and a set of global final states $F \subseteq \prod_{p \in \mathcal{P}} S_p$.

Note 1. The local automata communicate with each other over error-free fifo channels while the CFM changes from a configuration to another by letting one automaton perform a write or read to or from one of its channels. The CFM begins in a starting configuration where all local automata are situated in their start state and all buffers are empty and ends in a final configuration, i.e., a state from $F$ is reached and all buffers are empty again. We call a configuration of a CFM a deadlock if there is no possibility to reach a final configuration.

Having an autonomous automaton for each process results in a problem. Without synchronization, the automata do not know about the other automatas’ choices and thus may run into different branches of the CMSG resulting in a deadlock of the system. Hence, for implementing CMSGs, we have to use the set of synchronization messages $\text{Sync}$.

In general deadlocks result from the discrepancy between the global control of the CMSG switching from node to node and the local autonomous processes not being aware of the graph’s choice. But if we require the local-choice property, a CMSG becomes implementable without deadlocks ([Ker05]).

Theorem 4 ([Ker05]). Let $G$ be a strong local-choice CMSG, then $G$ is implementable without deadlocks.

Combining this result and the results from figure 1 yields the following:

Corollary 3 ([Ker05]). Every (weak and strong) local and local-choice CMSG $G$ is implementable without deadlock.

Finally we want to present an overview of the complexity classes and some implementability results from [GMSZ02], [GKM04] and [Ker05]. We divided the implementability property into normal and deadlock-free implementability.

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
<th>Implementability/ Deadlock Freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>polynomial time</td>
<td>YES/YES</td>
</tr>
<tr>
<td>local-choice</td>
<td>polynomial time</td>
<td>YES/YES</td>
</tr>
<tr>
<td>locally coop.</td>
<td>polynomial time</td>
<td>YES/ ?</td>
</tr>
<tr>
<td>regular</td>
<td>Co-NP complete</td>
<td>YES/NO</td>
</tr>
<tr>
<td>globally coop.</td>
<td>Co-NP complete</td>
<td>YES/NO</td>
</tr>
<tr>
<td>general case</td>
<td>$\geq$ Co-NP complete</td>
<td>NO/NO</td>
</tr>
</tbody>
</table>

Fig. 2. Complexity classes and implementability results (cf. [GMSZ02], [GKM04], [Ker05])

5 Additional Information and Acknowledgement

As we already mentioned, in addition to our theoretical results, we implemented a tool named MSCan which provides possibilities for specifying and analyzing CMSGs. For further information on this project please consult the tool’s web page:

http://www-i2.informatik.rwth-aachen.de/MSCan

I am deeply grateful to Benedikt Bollig for supervising this work.
6 Future Work

As part of my future work, we plan to conduct research on quantitative extensions of Message Sequence Charts (MSC) and Message Sequence Graphs (MSG). We intend to focus in particular on Life Sequence Charts (LSC, cf. figure 3). One of the drawbacks of MSCs and MSGs is that these formalisms do not provide ample means to distinguish between what “must” happen and what “may” happen in a communication systems’ behaviour. This shortcoming hampers the specification of scenarios for realistic system design. LSC diminish these problems by allowing to distinguish between mandatory, optional, and illegal or forbidden behavior which must, may and must not occur, respectively.

The quantitative extensions of Message Sequence Charts include modeling events that appear with a certain probability distribution as well as unreliable channels where messages can get lost with a given likelihood. LSCs do only exhibit possibilities for describing system behavior qualitatively but in this way will be extended to cope with quantitative system aspects. These extensions need not be restricted to the LSC body but might also be integrated into the LSCs head where the preconditions of the chart are constituted. These prerequisites have to be fulfilled to execute the LSC body. We also anticipate an extension of MSGs where we model the different transition probabilities for changing from one node to another, especially in branching nodes.

These extensions need to be formally defined, and must be equipped with a rigid semantics. To that purpose it is planned to investigate extensions of partial-order models (e.g., event structures and pomsets) as well as tree- and trace-based models. Like in the reported work in this paper, elementary properties of such quantitative LSCs/MSCs will be defined and classified. The relation with existing probabilistic extension of statecharts will be studied. The ultimate goal is to come to define a framework in which probabilistic scenarios can be used to synthesize system components in a semi-automated manner.

Fig. 3. small LSC for the antiblock system

References


