Semantics of Broadcast MSCs

Ingolf Krüger, Wolfgang Prenninger and Robert Sandner [1]

Technical University of Munich, D-80290 Munich, Germany
email: {kruegeri | prenning | sandnerr}@in.tum.de

Abstract. Message Sequence Charts have proven to be a useful modeling technique especially within the requirements analysis phase of software development. However, MSCs do not support the concept of broadcast communication, which is frequently used in technical applications. In this paper, we present an extension to MSCs for the modeling of broadcast interaction scenarios. Based on the mathematical framework of timed streams we also introduce a semantics for broadcast MSCs. We thoroughly discuss methodological benefits and semantic properties of this approach, consider alternative solutions, and address its scaleability with respect to complex real-time systems applications.

1. Introduction

Message Sequence Charts and similar graphical notations for component interaction have proven useful as a modeling aid for distributed system specifications. Their scope, however, is limited to point-to-point (p2p) message exchange. This significantly limits the applicability of MSCs in domains such as avionics, wireless, and automotive systems, where broadcasting and multicasting are the fundamental communication paradigms. In this paper, we propose a graphical syntax together with a formal, mathematical semantics for broadcasting scenarios. This results in an extension of the MSCs' reach to serve as an intuitive, yet formally and methodologically founded description technique for interaction behavior also for broad- and multicasting systems. The semantics definition we give is based on the mathematical framework FOCUS [BS01], which was successfully employed already in [Krü00] to present a semantics for MSCs without broadcasting. A key benefit of FOCUS is the flexible notion of timed streams it is based on. Timed streams allow us to model, in parallel, interaction-, and state-based aspects of system-behavior. This is a prerequisite for relating scenarios specified by MSCs to other artifacts of system development, such as state-automata for individual components. Moreover, as shown in [BS01], timed streams directly support powerful refinement operations, and are thus an adequate basis for systematic development methodologies.

The syntax of MSCs: MSCs capture the communication or collaboration among a set of components. Typically, an MSC consists of a set of axes, each labeled with the name of a component. An axis represents a certain segment of the behavior displayed by its corresponding component. Arrows in MSCs denote communication. An arrow starts at the axis of the sender or initiator of the communication; the axis at which the head of the arrow ends designates the communication's recipient or destination. Intuitively, the order in which the arrows occur (from top to bottom) within an MSC defines possible sequences of interactions among the depicted components.

[1] Our research was supported by the DFG within the priority program "SoftSpez" (SPP 1064) under project name *InTime*.
Correspondence and offprint requests to: I. Krüger, W. Prenninger and R. Sandner

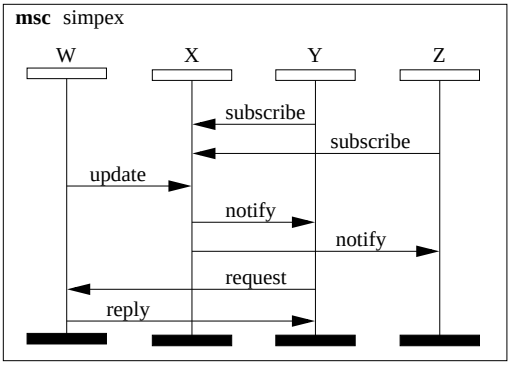


Fig. 1. Example MSC

The MSC of Figure 1, for instance, depicts a certain section of the communication among the four components W, X, Y, and Z within an imaginary distributed system. Time advances from the top to the bottom of the figure; this induces a temporal order on the depicted messages. Intuitively, Figure 1 captures a situation where Y and Z, in turn, send the message *subscribe* to X. Then, X receives message *update* from W. Subsequently, X sends message *notify* to Y and Z (in that order). Upon receipt of message *notify*, component Y sends message *request* to W, and receives message *reply* in return.

Broadcasting vs. the communication model of MSCs: The MSC notation as introduced above aims at p2p communication, where for every message there is precisely one sender and one receiver; this motivates using directed arrows as the representation of message exchange. The focus on p2p communication stems from the origins of MSCs: they were invented for asynchronous communication protocols in the telecommunication domain.

In technical application domains, however, other forms of communication play an equally, if not more important role. Broadcasting, for instance, is the central communication paradigm in practical embedded system applications, as found in the automotive, avionics and wireless communications domains. The increasing complexity and interconnection of such systems calls for a careful identification and documentation of their communication patterns. Adequate support for methodological modeling of interaction patterns is available only for p2p communication so far.

In this paper, we address this challenge by providing notational and semantic extensions to MSC-96[IT96], leading to a formally founded description technique for interaction patterns in broadcasting-based systems. Our semantics incorporates broadcasting into the p2p communication model of MSCs. This foundation allows us to relate Broadcast MSCs to other architectural models of the system under development, placing an emphasis on the important aspect of communication structures and behavior. This is of use both for methodological steps such as deriving structure and behavior specifications from MSCs, as well as for verifying an implementation of the system against the specified scenarios.

Integrated, Architecture-Centric Development with Broadcast MSCs – based on UML-RT: The simple example described above already allows us to illustrate one of the strengths of MSCs. They contain information on the distribution structure, as well as on the interaction behavior of the system under consideration. This combination helps to make explicit the coordination aspect of system behavior, beyond the local scope of individual components. MSCs, such as the one in Figure 1, show one particular interaction pattern (or *scenario*) among the depicted components.

MSCs represent projections of the complete system behavior on (part of) a certain task or service. The projection contains the relevant components together with their relevant behavior for the task under consideration. Instead of having to study the complete behavior descriptions of several individual components simultaneously to get an overview of what happens when the system executes the task, we can zoom in directly on the particularly interesting segment of each participating component's behavior by means of an appropriately chosen MSC.

Based on the foundation presented in this paper, MSCs play an important role in a seamlessly integrated development process we have developed in [KPS01a, KPS02, KPSB]; this process uses the description

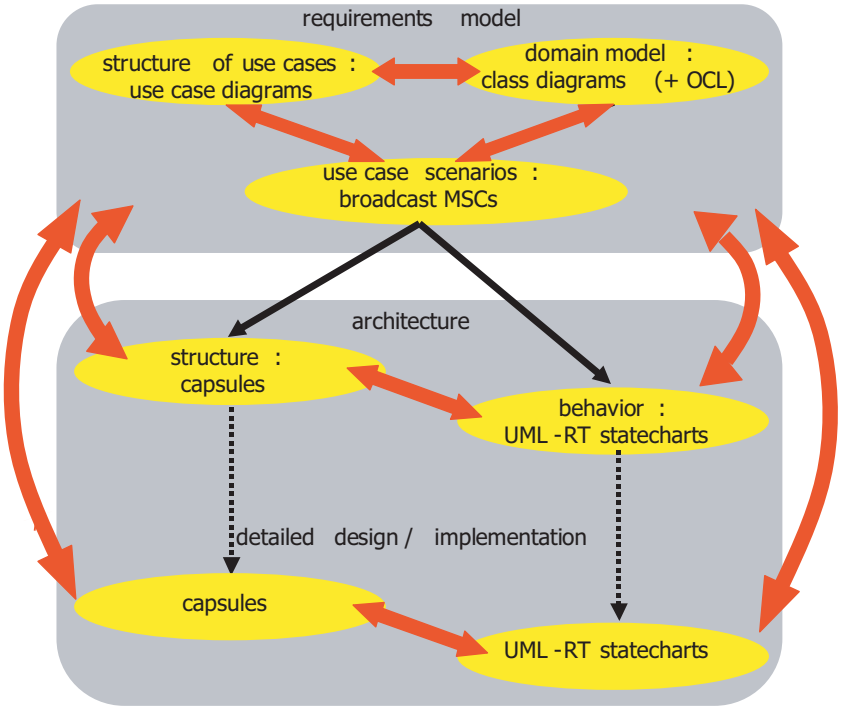


Fig. 2. UML-RT techniques in the development process

techniques of UML-RT [SR98, Lyo98] to represent system structure and behavior. The starting point is a requirements model of the system under development. Following well-established methods for object-oriented analysis, this phase is driven by the inception and elaboration of use cases in our approach. The resulting requirements model is specified using four diagrams: *class diagrams*, enriched by the UML's *object constraint language* (OCL), describe the essential conceptual entities of the application domain; this *domain model* captures key dependencies – such as the existence of a communication path – among central system components. Corresponding *use case diagrams* illustrate the participation of system components and their environment in the major *features* or services offered by the system. Broadcast MSCs are the central description technique within requirements analysis: They are used to identify "active" entities within the domain model, which become components in the sense of UML-RT later in the development process. Moreover, Broadcast MSCs contain the interaction patterns occurring among the active entities while they are involved in a certain use case.

These interaction patterns make a significant amount of design knowledge readily available; certain architectural aspects are directly influenced by the way components interact: component distribution, communication paths, and the underlying communication paradigms, i.e. p2p communication and broadcasting, to name just a few examples. Having this information at our disposal enables us to derive prototypes for component structures and behaviors, including protocols for both p2p and broadcast communication, using synthesis and transformation algorithms presented in [KGSB99, KPS01a, KPS02, KPSB]. The resulting component structures and behaviors specified by UML-RT's *capsule diagrams* and *statechart diagrams* can be refined systematically in subsequent development steps. This allows us to perform the development process described so far in a truly iterative, top-down fashion. An overview on the role of the description techniques in the overall development process is given in Figure 2.

Outline: We illustrate the syntax of Broadcast MSCs by means of an example in Section 2. In Section 3, we introduce the underlying concept of timed streams and define a formal semantics for Broadcast MSCs. Section 4 contains a discussion of methodological benefits and semantics properties of our approach and also reflects alternative solutions. Concluding remarks and an outlook on future work are given in Section 5.

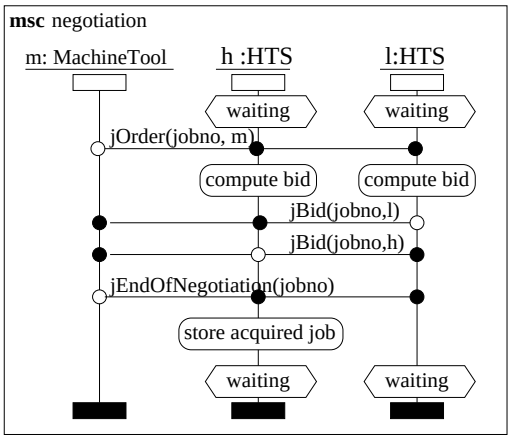


Fig. 3. scenario for order negotiation

2. Example: Broadcasting within an Autonomous Transport System

In order to illustrate our proposed semantics for Broadcast-MSCs we capture interaction patterns of an autonomous transport system within a production plant. The purpose of this system is to ensure that workpieces are transferred from their present location to another where the next production step is then carried out. Whenever a machine tool is free it requests to obtain a workpiece, which is then delivered by an autonomous vehicle (termed "holonic transport system", or "HTS" for short). Machine tools and HTSs use broadcasting to negotiate the delivery of a workpiece: a machine tool broadcasts its requests to all HTSs; the HTSs, in turn, broadcast their offer (an estimate on how long it takes them to satisfy the request). Finally, the machine tool broadcasts which HTS has "won the deal". We use a small part of the full, complex autonomous transport system to illustrate our semantics of broadcast MSCs. Thus, we only consider scenarios consisting of one machine tool and two HTSs. For a more complete model of the system using broadcast MSCs and description techniques of UML-RT we refer the reader to [KPS02, KPS01a, KPSB].

In the following, we extend the syntax of ITU MSC 96 [IT96] to introduce notation for broadcast communication and apply this extension in oder to identify some interaction patterns of the transport system. Figure 3 shows a possible scenario for the negotiation of a transport task.

Just as in regular MSCs, we use labeled, vertical axes to represent part of the behavior of the corresponding components within a Broadcast MSC. By means of labeled horizontal arrows we indicate p2p communication. Labeled hexagons denote control states of a component. A labeled, rounded box denotes a reference to another MSC containing the actual interaction pattern to be substituted in that place – similar to a "macro expansion". This is used to keep interaction patterns manageable in size.

Broadcast communication is modeled by a communication line without arrow head. An outlined circle marks the originator of the message and filled circles mark the corresponding receivers. This allows us to model broadcast as well as multicast communication.

In Figure 3, a machine tool announces an order using broadcast communication. Each HTS calculates its own "price", i.e. how long it would take for this HTS to satisfy the request. This calculation is captured in more detail within another MSC, referred to as `compute bid` in this example; the content of `compute bid` is not shown here for reasons of brevity. In our example scenario, two HTSs announce a bid for the order and finally, after the machine tool ends the negotiation, HTS h has won the deal. After the negotiation, the HTS components return to their initial state – relative to this scenario. Figure 4 shows a combination of broadcast and binary communication which occurs during the execution of a transport: When the HTS arrives at a machine tool to pick up a workpiece, it sends a request to the machine tool, which responds by a release message. Finally, the HTS announces the picking up of the workpiece by means of a broadcast message.

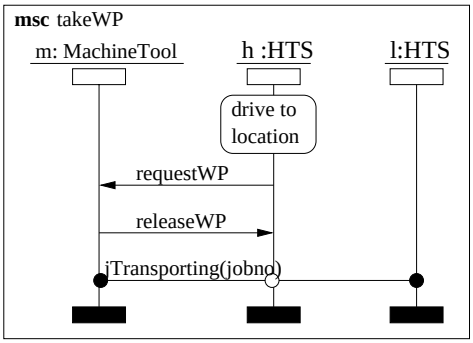


Fig. 4. scenario for picking up a workpiece

3. Semantics for Broadcast MSCs

With the semantics definition given in this section we target, in particular, distributed, reactive systems with static structure. We pay special attention to providing a semantics definition that allows us to to express the interaction- *and* state-oriented behavior in parallel. This is a prerequisite for a seamless integration of these two complementary architectural aspects; it paves the way for a powerful transformation procedure turning MSCs into state-automata (cf. [Krü00]). Here, we embed the semantics of broadcasting into the model developed in [Krü00]. Along the way we introduce the notation and concepts we need to describe the model.

3.1. System Model

To prepare the actual semantics definition, we first introduce the structural and behavioral model (the *system model*) we work with.

3.1.1. Notational Conventions

We start with a few notational conventions. By $\mathbb{B}$ and $\mathbb{N}$ we denote the set of booleans (the constants are true and false) and natural numbers (including 0), respectively. To denote function application we often use an infix dot (".") instead of parentheses to increase readability of our formulae. For $Q \in \{\forall, \exists\}$ and predicates r and p we write $\langle Q x : r.x : p.x \rangle$ to denote the respective quantification over all $p.x$ for which x satisfies the quantification range $r.x$. If the range is understood from the context, we omit it from the quantifying formula. As another form of reduced notation we integrate simple ranges into the specification of the quantified variable; as an example, we sometimes write $\langle \forall x \in \mathbb{N} :: \dots \rangle$ instead of $\langle \forall x : x \in \mathbb{N} : \dots \rangle$. $\mathcal{P}(X)$ denotes the powerset of any set X. Given sets $Y_1, Y_2, \dots$ we define for tuples $y = (y_1, y_2, \dots) \in Y_1 \times Y_2 \times \dots$ the projection onto the i-th element of the tuple as $\pi_i.y \stackrel{\text{def}}{=} y_i$ for $i \geq 1$. For the closed interval between $m \in \mathbb{N}_\infty$ and $n \in \mathbb{N}_\infty$ we write $[m, n]$; if $m > n$ then $[m, n] \stackrel{\text{def}}{=} \emptyset$. We define $\mathbb{N}_\infty \stackrel{\text{def}}{=} \mathbb{N} \cup \{\infty\}$.

The mathematical model which serves as the basis for our notion of system behavior is that of *streams*. Streams and predicates or functions on streams are an extremely powerful specification mechanism for distributed, interactive systems (cf. [BS01, Bro99, Möl99, BS01, Ste97, Rum96]). It serves particularly well for property-oriented component specifications, as well as for the definition of refinement notions and for the verification of corresponding refinement relationships between specifications (cf. [BDD+92, Rum96, Kle98, Sch98]). Here, we give a concise overview of the major concepts and notations in form of a "quick reference" table (cf. tab. 1); in this table X denotes a set of messages, $x, x' \in X^\infty$ are streams over X, $x_1, \dots, x_n \in X$ are some messages and $n, m \in \mathbb{N}$ are suitable natural numbers. For a thorough introduction to the topic, we refer the reader to [BS01, Bro99, Möl99, Ste97].

We lift the operators introduced in Table 1 to finite and infinite tuples and sets of streams by interpreting them in a pointwise and elementwise fashion, respectively. Given, for instance, the stream tuple $x : [1, m] \to X^\omega$, with $x = (x_1, \dots, x_m)$ for $m \in \mathbb{N}$, we denote by $x.n$ the tuple $(x_1.n, \dots, x_m.n)$, if $n \in \mathbb{N}$.

Below, we use streams to model the behavior of components – including both the communication between

Table 1. "Quick reference" for the stream notation used here

Notation	Informal Meaning
X^*	set of finite sequences over set X
X^∞	set of infinite sequences over set X
X^ω	set of streams over set X, whereby $X^\omega \stackrel{\text{def}}{=} X^* \cup X^\infty$
$x.n$	nth element of stream x
$\|x\|$	length of stream x
$x{\downarrow}n$	prefix of length n of stream x
$x{\uparrow}n$	stream obtained from x by removing the first n elements
$x^\frown x'$	concatenation of streams x and x'
$\langle x_1, ..., x_n \rangle$	finite stream consisting, in this order, of the elements x_1 through x_n
$y \in x$	y appears as an element in the finite stream x
$\langle\rangle$	the empty stream
x_1^n	the stream consisting of n consecutive copies of element x_1
$Y©x$	stream obtained from x by dropping all elements not contained in Y
$x\|_{[m,n]}$	stream obtained from x by considering only elements m through n

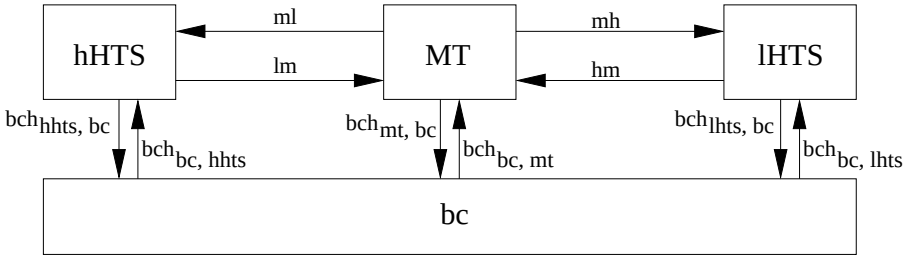


Fig. 5. Simple SSD that defines the sets P and C

components, and the states assumed by these components – over time. To stress this intuition we introduce the name *timed streams* for infinite streams (time does not halt) whose elements at position $t \in \mathbb{N}$ represent the messages transmitted or the states assumed at time t. Based on this intuition we identify tuples over timed streams with streams over tuples, and call both *timed stream tuples*. For instance, we identify $(X \times Y)^\infty$ with $X^\infty \times Y^\infty$ for sets X and Y. Moreover, for finite index sets X, and arbitrary sets Y we identify elements of the domains $X \to Y^\infty$ and $(X \to Y)^\infty$. This is a technical convention that gives us a convenient way of converting streams of functions into functions, whose ranges are streams, and vice versa. If, as an example, we have $z \in (X \to Y)^\infty$, and $x \in X$, then we allow ourselves to write $z.x$ to obtain z's projection onto x. Similarly, if we have $z \in (X \to Y^\infty)$ and $t \in \mathbb{N}$, then we consider $z.x.t$ and $z.t.x$ as synonyms.

3.1.2. System Structure

Structurally, a system consists of a set P of components, objects, or processes[2], and a set C of directed channels. Channels connect components that communicate with one another; they also connect components with the environment. To integrate broadcasting in our model we include a component $bc \in P$; it forwards every incoming message to all components but the originator of the message. Intuitively, bc models a broadcasting communication bus. For this purpose, we introduce channels $bch_{p,bc}$ and $bch_{bc,p} \in C$, which connect the components $p \in P \setminus \{bc\}$ to the broadcast component bc. With every $p \in P$ we associate a unique set of states, i.e. a component state space, S_p. We define the state space of the system as $S \stackrel{\text{def}}{=} \Pi_{p \in P} S_p$. For simplicity, we represent messages by the set M of message identifiers.

Figure 5 shows a system structure diagram (SSDs), describing the sets P and C in graphical notation; it defines $P = \{hHTS, lHTS, MT, bc\}$ and $C = \{ml, lm, mh, hm, bch_{hhts,bc}, bch_{bc,hhts}, bch_{lhts,bc}, bch_{bc,lhts}, bch_{mt,bc}, bch_{bc,mt}\}$.

Each channel $ch \in C$ is directed from its source to its destination component. To distinguish multiple

[2] In the remainder of this document, we use the terms components, objects, and processes interchangeably.

Table 2. Structural elements of the system model

Entity	Meaning
P	set of system components
C	set of directed channels
S_p	state of component $p \in P$
S	system state ($S \stackrel{\text{def}}{=} \Pi_{p \in P} S_p$)
M	set of message identifiers

channels with identical source and destination components we associate a name (an element from the set CN of channel names) with every channel. Thus, we treat each element $ch \in C$ as a triple $ch = (cn, cs, cd) \in CN \times P \times P$, where cn, cs, and cd denote the channel name, the source component, and the destination component of channel ch, respectively. We use the functions $chn : C \to CN$, $src : C \to P$, and $dst : C \to P$ to project a channel on its name, source, and destination component, respectively; hence, we have $ch = (cn, cs, cd) \Rightarrow chn.ch = cn \wedge src.ch = cs \wedge dst.ch = cd$. We assume that channel names are unique within C, and, where no confusion can arise, identify a channel with its name.

The systems we consider here are fixed in the sense that neither set P, nor set C changes over time. Table 2 summarizes these structural elements.

3.1.3. System Behavior

Now we turn to the dynamic aspects of the system model. We assume that the system components communicate among each other and with the environment by exchanging messages over channels. We assume further that a discrete global clock drives the system. We model this clock by the set $\mathbb{N}$ of natural numbers. Intuitively, at time $t \in \mathbb{N}$ every component determines its output based on the messages it has received until time $t - 1$, and on its current state. It then writes the output to the corresponding output channels and changes state. The delay of at least one time unit models the processing time between an input and the output it triggers; more precisely, the delay establishes a strict causality between an output and its triggering input (cf. [Bro99, BK98]).

Formally, with every channel $c \in C$ we associate the histories obtained from collecting all messages sent along c in the order of their occurrence. Our basic assumption here is that communication happens asynchronously: the sender of a message does not have to wait for the latter's receipt by the destination component. This allows us to model channel histories by means of streams.

We define $\tilde{C} \stackrel{\text{def}}{=} C \to M^*$ as a channel valuation that assigns a sequence of messages to each channel; we obtain the timed stream tuple $\tilde{C}^\infty$ as an infinite valuation of all channels. This models that at each point in time a component can send multiple messages on a single channel.

With timed streams over message sequences we have a model for the communication among components over time. Similarly we can define a succession of system states over time as an element of set S^∞.

With these preliminaries in place, we can now define the semantics of a system with channel set C, state space S, and message set M as an element of $\mathcal{P}((\tilde{C} \times S)^\infty)$. Any element (φ_1, φ_2) of a system's semantics consists of a valuation of the system's channels ($\varphi_1 \in \tilde{C}^\infty$) and a description of the system state over time ($\varphi_2 \in S^\infty$). The existence of more than one element in the semantics of a system indicates nondeterminism.

Table 3 lists the semantic entities for modeling system behavior.

In summary, our system model consists of two parts: the system's static structure and its behavior. The set of channels, the set of component states, and the set of messages determine the system's structure. The channel valuations, i.e. the occurrences of messages on channels over time, and the sequences of states determine the system's behavior.

In the following sections we will use the system model we have defined here as the basis for defining the semantics of Broadcast MSCs.

3.2. Simplified Textual Syntax

To simplify the semantics definition for Broadcast MSCs we first introduce their abstract textual syntax. In the syntax definition we use an extended Backus-Naur Form (BNF) (cf. [Wir86, ASU88]).

Table 3. Behavioral elements of the system model

Entity	Meaning
$\tilde{C}$	channel valuation at a particular time point ($\tilde{C} \stackrel{\text{def}}{=} C \to M^*$)
$\tilde{C}^\infty$	overall channel history
S^∞	state history
$(\tilde{C} \times S)^\infty$	combined channel and state history
$\mathcal{P}((\tilde{C} \times S)^\infty)$	semantics domain for system behaviors

MSC Documents: Typically, documents containing MSC specifications consist of more than one MSC; we use the syntactic category ⟨MSCDOC⟩ to represent sequences of MSC definitions.

⟨MSCDOC⟩ ::= {⟨MSCDEF⟩}*

MSC Definitions: An MSC definition, represented by the syntactic category ⟨MSCDEF⟩, associates an interaction description (as defined by ⟨MSC⟩) to an MSC name (as defined by ⟨MSCNAME⟩, which represents a text string). The MSC name becomes important in combination with references (cf. reference `compute bid` in Figure 3); there it acts as a placeholder for the interaction description to which the name relates:

⟨MSCDEF⟩ ::= **msc** ⟨MSCNAME⟩ = ⟨MSC⟩

MSC Terms: The syntactic category ⟨MSC⟩ represents an interaction description. It captures the syntactic correspondences between the graphical notation's arrows, broadcast lines, conditions, inline expressions, and references. In addition, it provides the syntax for expressing arbitrary interactions.

⟨MSC⟩ ::= **empty**
|| **any**
|| ⟨MSG⟩
|| ⟨BCMSG⟩
|| ⟨GMSC⟩
|| ⟨MSC⟩ ; ⟨MSC⟩
|| ⟨GMSC⟩ | ⟨GMSC⟩
|| →⟨MSCNAME⟩

For simplicity, we call elements of ⟨MSC⟩ "MSC terms" or "MSCs" for short. The intuitive interpretations of the syntactic elements in ⟨MSC⟩ is as follows: **empty** and **any** represent the absence of, and any form of interaction, respectively. ⟨MSG⟩ denotes a message specification for p2p communication; it consists of a channel name (defined by ⟨CHNAME⟩, a text string) and a message header (represented by ⟨MSGH⟩).

⟨MSG⟩ ::= ⟨CHNAME⟩▷⟨MSGH⟩

⟨BCMSG⟩ denotes a message specification for broadcast communication. It also consists of a channel name and a message header. ⟨BCHNAME⟩ denotes one of the channel names which connects a component $p \in P \setminus \{bc\}$ to the component bc. Thereby the sender of a broadcast message is determined implicitly.

⟨BCMSG⟩ ::= ⟨BCHNAME⟩ ⊙ ⟨MSGH⟩

⟨GMSC⟩ represents guarded MSCs; they consist of a guard specification (defined by the nonterminal ⟨GUARD⟩, a text string) and the guarded interaction description. Guards can be thought of as constraints at the state-space of the system under development. In particular, we employ guarded MSCs for modeling control states within Broadcast MSCs (cf. condition symbol `waiting` in Figure 3).

⟨GMSC⟩ ::= ⟨GUARD⟩:⟨MSC⟩

Operators ; and | denote the sequencing of and the alternative between the operand interaction descriptions, respectively. → represents MSC referencing. A reference to an MSC X is semantically equivalent

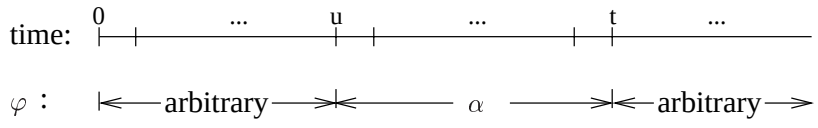


Fig. 6. MSC α constrains system behavior φ only over time interval $[u, t]$

to the interaction sequence represented by X. We omit the obvious definitions of the syntactic categories ⟨MSCNAME⟩, ⟨CHNAME⟩, ⟨MSGH⟩ and ⟨BCHNAME⟩ ; we assume all of them to define text strings.

Example: As an example for the representation of Broadcast MSCs in the syntax introduced above we consider again the scenario depicted in Figure 4. The scenario is a sequence of an MSC reference, two message specifications for p2p communication, and a message specification for broadcast communication. Thus, in our syntax the scenario is expressed as follows:
msc takeWP = →drive to location ; $hm \triangleright requestWP$; $mh \triangleright releaseWP$; $bch_{hhts,bc} \odot jTransporting(jobno)$

3.3. Denotational Semantics

In this section we introduce the formal, denotational semantics for Broadcast MSCs. Intuitively, we associate with a given MSC a set of channel and state valuations, i.e. a set of system behaviors according to the system model we have introduced in Section 3.1. Put another way, we interpret an MSC as a constraint at the possible behaviors of the system under consideration. More precisely, with every $\alpha \in \langle \text{MSC} \rangle$ and every $u \in \mathbb{N}_\infty$ we associate a set $[\![\alpha]\!]_u \in \mathcal{P}((\tilde{C} \times S)^\infty \times \mathbb{N}_\infty)$; any element of $[\![\alpha]\!]_u$ is a pair of the form $(\varphi, t) \in (\tilde{C} \times S)^\infty \times \mathbb{N}_\infty$. The first constituent, φ, of such a pair describes an infinite system behavior. u and the pair's second constituent, t, describe the time interval within which α constrains the system's behavior. Intuitively, u corresponds to the "starting time" of the behavior represented by the MSC; t indicates the time point when this behavior has finished. Hence, outside the time interval specified by u and t the MSC α makes no statement whatsoever about the interactions and state changes happening in the system (cf. Figure 6).

Our motivation for using u as a parameter of the semantics, and for returning t as a "result" parameter in the semantics definition is twofold:

1. we view an individual MSC as a representation of (part of) a system execution. Without further information – MSCs do not contain explicit absolute timing information – we cannot say in advance at what time during the system execution the behavior modeled by an MSC starts; this explains parameter u. Similarly, because of the lack of explicit timing information (say, between two message occurrences), we cannot say in advance at what time the behavior modeled by an MSC is over; this explains parameter t.
2. the use of the lower time-bound u as a parameter eases the semantics definition; sequential composition (see below) demonstrates this benefit.

Definition 1 (Behavior "Beyond Infinity") To model that we cannot observe (or constrain) system behavior "beyond infinity" we define that for all $\varphi \in (\tilde{C} \times S)^\infty$, $\alpha \in \langle \text{MSC} \rangle$, and $t \in \mathbb{N}_\infty$ the following two predicates hold:

$$(\varphi, t) \in [\![\alpha]\!]_\infty$$
$$(\varphi{\uparrow}\infty, t) \in [\![\alpha]\!]_\infty \tag{1}$$

□

The behavior modeled by an MSC is independent of the time point at which the behavior starts, i.e. for all $\varphi \in (\tilde{C} \times S)^\infty$, $u, n \in \mathbb{N}$, $t \in \mathbb{N}_\infty$, and $\alpha \in \langle \text{MSC} \rangle$ we have:

$$(\varphi, t + n) \in [\![\alpha]\!]_{u+n} \equiv (\varphi{\uparrow}n, t) \in [\![\alpha]\!]_u \tag{2}$$

This means that our model is independent of absolute time.

We assume given a relation $MSCR \subseteq \langle \text{MSCNAME} \rangle \times \langle \text{MSC} \rangle$, which associates MSC names with their interaction descriptions. We expect $MSCR$ to be the result of parsing all of a given MSC document's MSC

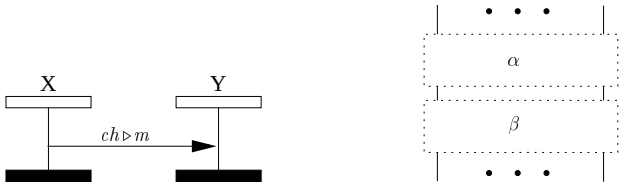


Fig. 7. Single message exchange and sequential MSC composition

definitions. For every MSC definition **msc** $X = \alpha$ in the MSC document we assume the existence of an entry (X, α) in *MSCR*. For simplicity we require the MSC term associated with an MSC name via *MSCR* to be unique.

In the following paragraphs we define the MSC semantics by means of structural induction over the grammar from Section 3.2. Where appropriate we show the graphical notation corresponding to an element of the textual syntax.

Empty MSC For any time $u \in \mathbb{N}_\infty$ **empty** describes arbitrary system behavior that starts and ends at time u. Formally, we define the semantics of **empty** as follows:

$$[\![\mathbf{empty}]\!]_u \stackrel{\text{def}}{=} \{(\varphi, u) : \varphi \in (\tilde{C} \times S)^\infty\}$$

Arbitrary Interactions MSC **any** describes completely arbitrary system behavior; there is neither a constraint on the allowed interactions and state changes, nor a bound on the time until the system displays arbitrary behavior:

$$[\![\mathbf{any}]\!]_u \stackrel{\text{def}}{=} \{(\varphi, t) \in (\tilde{C} \times S)^\infty \times \mathbb{N}_\infty : t \geq u\}$$

any subsumes all possible behavior, i.e. for all $\alpha \in \langle \text{MSC} \rangle$ we have:

$$[\![\alpha]\!]_u \subseteq [\![\mathbf{any}]\!]_u$$

any has no direct graphical representation; we use it to resolve unbound MSC references (see below).

Single Message An MSC that represents the occurrence of message m on channel ch (cf. Figure 7 (a)) constrains the system behavior until the minimum time such that this occurrence has happened:

$$[\![ch \triangleright m]\!]_u \stackrel{\text{def}}{=} \{(\varphi, t) \in (\tilde{C} \times S)^\infty \times \mathbb{N} : t = \min\{v : v > u \wedge m \in \pi_1(\varphi).v.ch\}\}$$

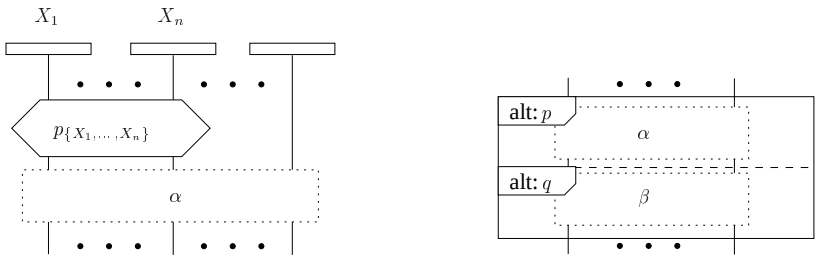
Because we disallow pairs (φ, ∞) in $[\![ch \triangleright m]\!]_u$ we require the message to occur eventually (within finite time). This corresponds with the typical intuition we associate with MSCs: the depicted messages do occur within finite time.

We add the channel identifier explicitly to the label of a message arrow in the graphical representation; this is useful in situations where a component has more than one communication path to another component. The channel names used in message specifications, and the channel names appearing in an SSD (such as the one shown in Figure 5) must be consistent, i.e. a message can occur only on a channel between two components if such a channel exists in the corresponding SSD.

Sequential Composition The semantics of the semicolon operator is sequential composition: given two MSCs α and β the MSC $\alpha\,;\,\beta$ denotes that we can separate each system behavior in a prefix and a suffix such that α describes the prefix and β describes the suffix (cf. Figure 7 (b)):

$$[\![\alpha\,;\,\beta]\!]_u \stackrel{\text{def}}{=} \{(\varphi, t) \in (\tilde{C} \times S)^\infty \times \mathbb{N}_\infty : \langle \exists t' \in \mathbb{N}_\infty :: (\varphi, t') \in [\![\alpha]\!]_u \ \wedge \ (\varphi, t) \in [\![\beta]\!]_{t'} \rangle\}$$

Guarded MSC Let $K \subseteq P$ be a set of instance identifiers. By p_K we denote a predicate over the state spaces of the instances in K. Let $[\![p_K]\!] \in \mathcal{P}(S)$ denote the set of states in which p_K holds. Then we define



(a) (b)

Fig. 8. Guarded MSC and alternative

the semantics of the guarded MSC $p_K : \alpha$ as the set of behaviors whose state projection fulfills p_K at time u, and whose interactions proceed as described by MSC α:

$$[\![p_K : \alpha]\!]_u \stackrel{\text{def}}{=} \{(\varphi, t) \in [\![\alpha]\!]_u : \pi_2(\varphi).u \in [\![p_K]\!]\}$$

We require p_K to hold only at instant u. This allows arbitrary state changes from time u on. In particular, at no other point within the time interval covered by α can we assume that p_K still holds.

We can conjoin multiple guards of an MSC into a single one, i.e. for all $p, q \in \langle \text{GUARD} \rangle$ and $\alpha \in \langle \text{MSC} \rangle$ we have:

$$p : (q : \alpha) \equiv_u (p \wedge q) : \alpha$$

Moreover, the boolean constants true and false hold in all and none of the system's executions, respectively, i.e. for all $\alpha \in \langle \text{MSC} \rangle$ we have:

$$\begin{array}{rcl} [\![\text{true} : \alpha]\!]_u & = & [\![\alpha]\!]_u \\ [\![\text{false} : \alpha]\!]_u & = & \emptyset \end{array}$$

As Figure 8 (a) shows, where $K = \{X_1, \ldots, X_n\}$ for some $n \in \mathbb{N}$, we use the condition symbol from MSC-96 to represent guards in MSCs. Put another way, we have assigned a meaning to conditions by treating them as guards. In the semantics for MSC-96 the meaning of conditions is void (cf. [IT96]).

Alternative An alternative denotes the union of the semantics of its two operand MSCs. The operands must be guarded MSCs; the disjunction of their guards must yield true. Thus, for $\alpha = p : \alpha'$ and $\beta = q : \beta'$ for $\alpha', \beta' \in \langle \text{MSC} \rangle$ and $p, q \in \langle \text{GUARD} \rangle$ with $p \vee q \equiv \text{true}$ we define:

$$[\![\alpha \mid \beta]\!]_u \stackrel{\text{def}}{=} [\![\alpha]\!]_u \cup [\![\beta]\!]_u$$

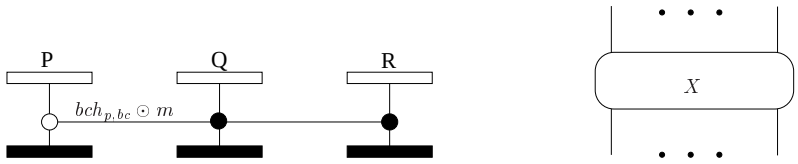
For guards p and q with $p \wedge q \equiv \text{true}$ the alternative expresses a nondeterministic choice.

MSC-96 provides no means for guiding the choice among alternatives; this corresponds to setting both p and q to true in our definition. In the graphical representation of alternatives (cf. Figure 8 (b)) we add the guarding predicates after a colon to the keyword **alt** in the respective compartment of the alternative box.

Single Broadcast Message The semantics of sending a broadcast message is as follows: the message is sent to the broadcast component bc. Thereafter, the broadcast component sends the message to all remaining components – in no particular order.

$$\begin{array}{rl} [\![bch_{p,bc} \odot m]\!]_u \stackrel{\text{def}}{=} & \{(\varphi, t) \in (\tilde{C} \times S)^\infty \times \mathbb{N} : \\ & \langle \exists t' \in \mathbb{N} :: (\varphi, t') \in [\![bch_{p,bc} \triangleright m]\!]_u \rangle \\ & \wedge \langle \forall p' \in P \setminus \{p, bc\} :: \langle \exists t'' \in \mathbb{N} :: (\varphi, t'') \in [\![bch_{bc,p'} \triangleright m]\!]_{t'} \rangle \rangle \\ & \wedge t = \max\{t'' \in \mathbb{N} : \langle \exists q \in P \setminus \{p, bc\} :: (\varphi, t'') \in [\![bch_{bc,q} \triangleright m]\!]_{t'} \rangle\} \end{array}$$

The motivation for adding the channel identifier explicitly to the label of broadcast line in the graphical representation is twofold (cf. Figure 9 (a)):



(a) (b)

Fig. 9. MSC broadcast and reference

1. The sender of the message is coded textually in addition to the graphical notation. Thereby the complete information of the graphical syntax is recorded textually.
2. The channel identifier will be useful when we want to extend our notation and semantics for multicast communication. Then we can easily distinguish different sets of components according to their respective communication "neighborhoods".

References If an MSC named X exists in the given MSC document, i.e. there exists a pair $(X, \alpha) \in \mathit{MSCR}$ for some $\alpha \in \langle \text{MSC} \rangle$, then the semantics of a reference to X equals the semantics of α. Otherwise, i.e. if no adequate MSC definition exists, we associate the meaning of **any** with the reference (cf. Figure 9 (b)):

$$[\![\rightarrow X]\!]_u \stackrel{\text{def}}{=} \begin{cases} [\![\alpha]\!]_u & \text{if } (X, \alpha) \in \mathit{MSCR} \\ [\![\textbf{any}]\!]_u & \text{else} \end{cases}$$

To identify **any** with an unbound reference has the advantage that we can understand the binding of references as a form of refinement. This is an important asset to have in view of the seamless integration of Broadcast MSCs into the overall development process we have in mind (cf. Section1).

4. Discussion

In the previous section, we have given a semantics for broadcast MSCs, based on the very general mathematical framework of FOCUS. In the following, we discuss the adequacy of this semantic approach, address alternative solutions, and examine scalability issues.

Why this semantic model? A large number of suggestions for semantics definitions for MSCs are available in the literature. [IT98] defines the semantics for MSC-96 in a process-algebraic setting. In [Ren99] one of the contributors to [IT98] gives a thorough and accessible introduction to the history, syntax and semantics of MSC-96. In [GRG93, MR94, MR96, Leu95, AHP96, Ber97, KW98, DH99, MP00], to name just a few examples, the authors define and discuss various semantics of MSC dialects. The underlying semantic models include Petri nets, partial order models, process-algebraic terms, variants of state machines, and timed rewriting logic. Their focus is mainly the assignment of a semantics to individual MSCs, the discussion of different communication primitives, the detection of inconsistencies within MSC specifications and corresponding decidability issues, and the exploration of extensions to the standard MSC syntax and semantics. What is *not* in the center of concern of most of these approaches is the integration of MSCs into the development process for distributed systems. The notion of behavioral refinement, for instance, enabling a systematic top-down design approach for MSCs, is not thoroughly addressed in the mentioned literature.

Our goal is to employ Broadcasting MSCs seamlessly across development phases – during requirements capture as well as during implementation and verification. Therefore, we have chosen the flexible semantic framework of timed streams. This model enables us to map directly from the I/O-oriented graphical MSC notation to corresponding message sequences and I/O-relations on the semantic level. In addition, this semantic model captures interaction- and state-oriented behavior specifications concisely within a single mathematical framework.

Our MSC semantics is highly nondeterministic; even for a single message we leave open when exactly it gets transmitted. All that matters is the causal ordering between the depicted messages to capture the interaction pattern of interest. Basing our semantics on sets of streams and state valuations has the distinct advantage that we get an intuitive notion of refinement on Broadcast MSCs for free: we can say that MSC

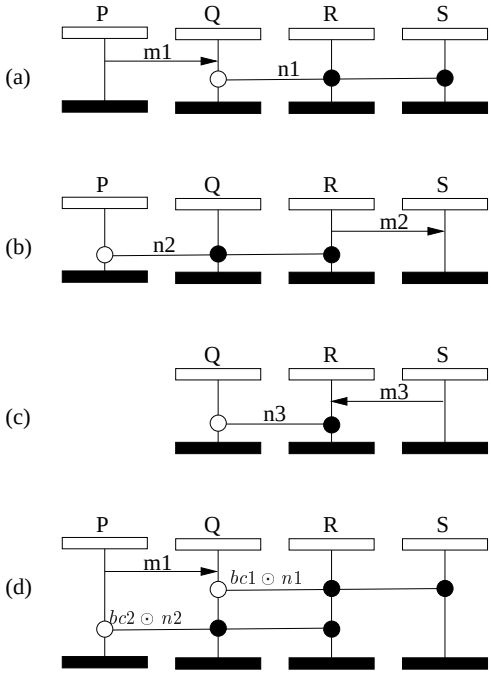


Fig. 10. Some multicast scenarios

α refines MSC β if $[\![\alpha]\!]_u \subseteq [\![\beta]\!]_u$. We refer the reader to [Krü00] for a description of corresponding syntactic refinement steps for MSCs that we can import directly for Broadcast MSCs.

Furthermore, our semantic model directly supports the transition from MSCs interpreted as scenarios (as we do here) to complete specifications for individual components; to that end we eliminate many of the nondeterministic choices in the semantics (as is described in detail in [BK98, Krü00] and still maintain a refinement relationship between the resulting components and the MSCs from which they are derived.

Introduction of a broadcast component at the semantic level When we defined a representation of the concept of broadcasting within our stream-based semantics, we faced a number of choices. We could have, for instance, simply introduced broadcasting by connecting every component with every other component without an explicit broadcast component. However, this has several disadvantages. For complex scenarios, this would lead to an explosion of the number of channels. Furthermore, structural changes, such as adding a new component involved in some broadcast communication, require scattered changes within the semantic model. Whereas these drawbacks are invisible to the modeler of a system, they become quite relevant if we employ broadcast scenarios in the course of verification: they hinder, for example, the reuse of proofs in the case of changes or refinements of models.

A second choice for representing broadcasting would be to extend the underlying logic by a notion for a broadcast medium. Logics which can express broadcasting typically model communication based on the concept of events or actions instead of dedicated messages from one component to another (cf., for instance, [Lam94]). However, unlike streams, which model channels in our approach, such a concept significantly complicates the definition of clear component interfaces. This, in turn, complicates reasoning about a system in terms of the composition of its components. Thus, this approach would suffer from a serious lack of modularity.

A further benefit of the introduction of a broadcast component at the semantic level is that this directly corresponds with the treatment of broadcasting in implementation models: in [KPS02], we present a design pattern which enables us to integrate broadcasting with the hierarchic decomposition of broadcast components into subcomponents, and which scales up to complex structures with multiple broadcast mediums.

Modeling complex scenarios with multiple "broadcast" mediums/multicasting Until now, we have only considered "pure" broadcast scenarios, where a set of components communicates by means of broadcasting in a sense that every component receives every broadcast message in the system. We could also think of more complex systems. For example, consider the MSCs in Figure 10 (a) and (b). In scenario (a), component Q sends a message to all components except P, and P emits a message which does not reach S. Scenario (b) shows P being involved in a multicast. If both scenarios describe interaction patterns within the same system, we could interpret them straightforwardly as communication over two different broadcast mediums, one connecting Q, R, and S, and the second connecting P, Q, and R. However, this may lead to undesired consequences if we consider the scenario in Figure 10 (c): Q sends a message to R, and P does not appear in the MSC. Following the interpretation above, we have a third communication medium, connecting Q and R. However, we could also have intended that we didn't know the role of P at the time we specified the MSC but we want to refine MSC in later development steps. In this case, both interpretations are conflicting.

To reconcile both views in the modeling of complex systems with multiple broadcast mediums, we suggest to add a channel name to specification of a broadcast message, as shown in Figure 10 (d). This notation makes the existence of multiple mediums explicit and allows also for the refinement of scenarios in the sense of Figure 10 (c). This extension can be used immediately in our semantic model: a broadcast channel defines the set of components involved in the interaction. We only need to insert this set into our semantics definition for broadcast messages. For example, at the semantic level the channel name $bc2$ in Figure 10 (d) is mapped to the broadcast component $BC2$ and induces the set $Comp(bc2)$, consisting of P, Q, R, and $BC2$. Thus, the semantics for the message $n2$ is defined as below:

$$\begin{aligned}[\![bch_{P,BC2} \odot n2]\!]_u \stackrel{\text{def}}{=} \{(\varphi,t) \in (\tilde{C} \times S)^\infty \times \mathbb{N} :\\ \langle \exists t' \in \mathbb{N} :: (\varphi,t') \in [\![bch_{P,BC2} \triangleright n2]\!]_u \rangle\\ \wedge \langle \forall p' \in Comp(bc2) \setminus \{P, BC2\} :: \langle \exists t'' \in \mathbb{N} :: (\varphi,t'') \in [\![bch_{BC2,p'} \triangleright n2]\!]_{t'} \rangle \rangle\\ \wedge t = \max\{t'' \in \mathbb{N} : \langle \exists q \in Comp(bc2) \setminus \{P, BC2\} :: (\varphi,t'') \in [\![bch_{bc2,q} \triangleright n2]\!]_{t'} \rangle\}\end{aligned}$$

Broadcasting in timed models Although we have not considered time delay in the specification of broadcast messages from the modelers point of view so far, execution time is an essential parameter of our semantics for MSCs. In our semantics, broadcast messages are represented by a non-atomic construct (consisting of a set of independently sent messages) at the semantic level. From the point of view of real-time modeling, the question whether this representation leads to - undesired - differences in timed behavior is important. As we have stated above, our semantic model is highly nondeterministic with respect to the timing of messages. For example, in the semantics definition of a single message

$$[\![ch \triangleright m]\!]_u \stackrel{\text{def}}{=} \{(\varphi,t) \in (\tilde{C} \times S)^\infty \times \mathbb{N} : t = \min\{v : v > u \wedge m \in \pi_1(\varphi).v.ch\}\}$$

the time at which the message actually occurs on channel ch (parameter t) can be freely chosen. The only restrictions are that the message appears after finite time[3] and later than time instant u. Whether it appears 5 seconds or $0,5$ milliseconds after u is left unspecified. Therefore, both sending of a single message and of a broadcast message, which involves two communication steps at the semantic level, can last equally long.

This model directly supports the specification of timing requirements: standard MSC syntax can be used for that by referencing the points where messages are emitted and read, both for single and broadcast messages. The integration of these mechanisms in our semantic model is straightforward: Time constraints would yield additional constraints on the parameter t, possibly w.r.t. u. Identifying and integrating an adequate set of timing constraints in Broadcast MSCs is subject to future development of our approach.

5. Conclusion and Outlook

In this paper, we have presented an extension to message sequence charts, which enables modeling of broadcast communication in interaction scenarios in a compact, precise and methodologically founded way. Our

[3] This requirement is expressed implicitly: The time instant of every single message in a stream is finite.

extension integrates seamlessly into the "standard" MSC syntax; p2p communication, broadcast, and multicast communication can be combined as desired.

Based on the mathematical framework FOCUS, we have defined a semantics for MSCs, which covers both standard MSCS, and our extensions for broadcasting. This approach has multiple advantages: timed streams provide a powerful notion of refinement. Together with the relaxed time model we have chosen (see Section 4), they enable us to provide a strong guidance for modelers by constructive refinement rules. These rules support the consistent, stepwise development of scenario specifications (for details on refinement rules, see [Krü00]). The flexibility of our semantic model also allows us to relate broadcast scenarios with other artifacts of a development process in a precise, clearly defined manner. This makes broadcast MSCs an integral part of a systematic development process.

Based on a carefully chosen subset of the extensive syntax for standard MSCs [IT96], extended by a notation for broadcast communication, Broadcast MSCs reconcile two important objectives: they scale up well to the specification of complex interaction scenarios, but not at the expense of added complexity of the semantics. Important extensions can be integrated easily into our approach: systems containing multiple broadcast mediums can be modeled by adding channel names. This is supported directly by our semantics. Additional constraints, such as timing information, can be specified by standard MSC syntax. Time is already an integral part of our semantics; the incorporation of timing constraints is an important aspect of future development of our approach.

Broadcast MSCs provide a powerful specification mechanism, which is not only capable for documenting requirements but is also an excellent starting point for the actual system design. In [KPS01a, KPS01b] and [KGSB99], we present algorithms, which allow us to derive a prototypic system architecture from broadcast MSCs in a fully automatic manner. Based on these algorithms and the semantic foundation presented in this paper, Broadcast MSCs are an essential building block of a development methodology for time critical embedded systems being currently developed in the project *InTime*.

Acknowledgments: The authors are grateful to Manfred Broy for encouraging and insightful comments, to Bernd Finkbeiner for valuable suggestions regarding the semantics of broadcasting and MSCs in general, and to the anonymous referees for their constructive remarks.

References

[AHP96] Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An Analyzer for Message Sequence Charts. *Software — Concepts and Tools*, 17:70 – 77, 1996.

[ASU88] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilerbau*, volume I. Addison-Wesley, 1988.

[BDD+92] Manfred Broy, Frank Dederichs, Claus Dendorfer, Max F uchs, Thomas F. Gritzner, and Rainer Weber. The Design of Distributed Systems. An Introduction to FOCU S – Revised Version –. Technical Report TUM-I9202-2, 1992.

[Ber97] Dorothea Beringer. *Modelling Global Behaviour With Scenarios In Object-Oriented Analysis.* PhD thesis, École Polytechnique Fédérale de Lausanne, 1997.

[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, Shaoying Liu Hinchey, and Shaoying Liu, editors, *Formal Engineering Methods (ICFEM'98)*, pages 2–15. IEEE Computer Society, 1998.

[Bro99] Manfred Broy. A Logical Basis for Modular Systems Engineering. In Manfred Broy and Ralf Steinbrüggen, editors, *Calculational System Design*, volume 173 of *NATO Science Series F*, pages 101–130. IOS Press, 1999.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces and Refinement.* Springer New York, 2001. ISBN 0-387-95073-7.

[DH99] Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. In *FMOODS'99 IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems*, 1999.

[GRG93] Peter Graubmann, Ekkart Rudolph, and Jens Grabowski. Towards a Petri net based semantics definition for Message Sequence Charts. In O. Færgemand and A. Sarma, editors, *SDL'93 – Using Objects*, Proceedings of the Sixth SDL Forum, pages 179–190, 1993.

[IT96] ITU-T. *ITU-T Recommendation Z.120 – Message Sequence Chart (MSC 96).* ITU-T, 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 Franz J. Rammig, editor, *Distributed and Parallel Embedded Systems*, pages 61–71. Kluwer Academic Publishers, 1999.

[Kle98] Cornel Klein. *Anforderungsspezifikation durch Transitionssysteme und Szen arien.* PhD thesis, 1998. (in German).

[KPS01a] I. Krüger, W. Prenninger, and R. Sandner. Architectural Design of a Broadcasting System using UML-RT. In Andreas Schürr, editor, *OMER-2 Workshop des Arbeitskreises GROOM der GI Fachgruppe 2.1.9 Objektorientierte Software-Entwicklung, 9.-12. Mai 2001, Herrsching*, 2001.

[KPS01b] I. Krüger, W. Prenninger, and R. Sandner. Deriving Architectural Prototypes for a Broadcasting System using UML-RT. In Philippe Kruchten, editor, *1st ICSE Workshop on Describing Software Architecture with UML, 15. Mai 2001, Toronto*, 2001.

[KPS02] Ingolf Krüger, Wolfgang Prenninger, and Robert Sandner. Development of an autonomous transport system using UML-RT. Technical report, Technische Universität München, to appear, 2002.

[KPSB] Ingolf Krüger, Wolfgang Prenninger, Robert Sandner, and Manfred Broy. From Scenarios to Hierarchical Broadcasting Software Architectures using UML-RT. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*. (to appear 2002).

[Krü00] Ingolf Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[KW98] Piotr Kosiuczenko and Martin Wirsing. Towards an Integration of Message Sequence Charts and Timed Maude. In M.M. Tanik, J. Tauka, K. Itoh, M. Goedicke, W. Rossak, H. Ehrig, and H. Kurfess, editors, *3rd World Conference on Integrated Design and Process Technology, IDPT'98*, pages 88–95, Berlin, Austin: Society for Design and Process Science, 1998. (revised version to appear in Journal of IDPT, 2000).

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[Leu95] Stefan Leue. *Methods and Semantics for Telecommunications Systems Engineering*. PhD thesis, Universität Bern, 1995.

[Lyo98] A. Lyons. UML for Real-Time Overview. *Objectime Ltd.*, April 1998. `http://www.objectime.com/otl/technical/umlrt.html`.

[Möl99] Bernhard Möller. Algebraic Structures for Program Calculation. In Manfred Broy and Ralf Steinbrüggen, editors, *Calculational System Design*, volume 173 of *NATO Science Series F*, pages 25–97. IOS Press, 1999.

[MP00] Anca Muscholl and Doron Peled. Analyzing Message Sequence Charts. In *Proc. of SAM'00*, 2000.

[MR94] Sjouke Mauw and Michel Adriaan Reniers. An Algebraic Semantics of Basic Message Sequence Charts. *The Computer Journal*, 37(4), 1994.

[MR96] Sjouke Mauw and Michel Adriaan Reniers. Refinement in Interworkings. In U. Montanari and V. Sassone, editors, *CONCUR'96*, volume 1119 of *LNCS*, pages 671–686. Springer, 1996.

[Ren99] Michel Adriaan Reniers. *Message Sequence Chart. Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1999.

[Rum96] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter S ysteme*. PhD thesis, TU München, 1996. (in German).

[Sch98] Peter Scholz. *Design of Reactive Systems and their Distributed Implementati on with Statecharts*. PhD thesis, 1998.

[SR98] B. Selic and J. Rumbaugh. Using UML for modeling complex real-time systems. `http://www.objectime.com/otl/technical`, April 1998.

[Ste97] Robert Stephens. A Survey of Stream Processing. *Acta Informatica*, 34:491–541, 1997.

[Wir86] Niklaus Wirth. *Compilerbau*. Teubner, 1986.