

# A Little Synopsis on Streams, Stream Processing Functions, and State-Based Stream Processing

Jan Oliver Ringert and Bernhard Rumpe

Software Engineering

RWTH Aachen University, Germany

<http://www.se-rwth.de/>

**Abstract** Specification of interactive distributed systems has been a challenge for decades. We present an overview of the specification techniques for these systems based on dataflow networks and stream processing. It covers models of streams and specification of stream processing systems that are related to and based on the development method FOCUS invented by Manfred Broy and his group. We introduce a basic set of manipulator operations for streams, stream bundles, stream processing functions, and give a summary of related state-based specification techniques. Furthermore we sketch an overview of implementations for the FOCUS framework. These range from formalizations using interactive proof assistants and model checkers to the modeling IDE AutoFocus.

**Key words:** FOCUS, stream processing, automata, semantics

**J. O. Ringert and B. Rumpe. Jan Oliver Ringert, et al.: A little synopsis on streams, stream processing functions, ....** *Int J Software Informatics*, 2011, 5(1-2 (2011), Part I): 29–53. <http://www.ijsi.org/1673-7288/5/i75.htm>

*“Software development is a difficult and complex engineering task. It would be very surprising if such a task could be carried out properly without a proper theoretical framework.”, Manfred Broy [Bro03, Bro05]*

## 1 Introduction

The specification of large and safety critical hard- and software systems is an ongoing challenges that has been addressed over the last decades in computer science and software/hardware systems engineering. Specific challenges arise from the design, specification and verification of reactive systems [HP85, BS01] and real-time systems [MS97, Bro97, Lee09]. Reliable distributed software has now become one of the big challenges of software development: Not only traditional embedded systems, such as plant construction, vehicle, aircraft and train construction but nowadays also reliable internet services, many-cores, scientific and cloud-computing are heavily distributed reactive systems without shared memory, but communication via networks of messages.

Stream processing models have been used to approach these challenges and displayed some stories of success in the engineering of software systems as well as hardware design and verification. An early survey on the advances of stream processing of

---

This work is sponsored by DFG GK/1298 AlgoSyn.

Corresponding author: Bernhard Rumpe, Email: [rumpe@se-rwth.de](mailto:rumpe@se-rwth.de), <http://www.se-rwth.de/>

Received 2010-10-13; Accepted 2011-01-03; Final revised version 2011-01-17

the last century is given in [Ste97]. Manfred Broy and his group are major players in the exploration of possible semantic models as well as specification and verification techniques for the modeled systems. As these techniques evolve over time, this special issue is a great opportunity to give a little synopsis, of what forms of streams, operators and specification techniques have been defined and proven useful.

The history of stream processing systems has its roots in Kahn networks [Kah74, KM77], which describe reactive systems as networks of concurrent stream processors using fixed point semantics. Streams are possibly infinite sequences of data messages and can be used to define the behavior of systems modeled as dataflow networks. They are used to observe the interaction between components constituting the network. These traces are built of discrete messages or continuous data exchanged on channels between components. Traces are formalized mathematically in a number of theories, which can be of diverse nature as described in Sect. 2.

Distributed reactive systems are composed of entities described as actors, coroutines, processes, stream processing functions, agents or simply components [Kah74, KM77, Bur75, BDD<sup>+</sup>92, Rum96]. In different stages of a system development these are formalized using a number of specification methods (see Sect. 3).

In detailed design reactive systems can be modeled using states, transitions, input and output operations as known from statecharts [HP85, Har87, OMG07]. This methodology has been used, e.g., in [Bro95, GR95] and extended to  $I/O^\omega$  automata with fully formalized semantics in [Rum96]. The semantics of  $I/O^\omega$  automata is given by a translation to sets of stream processing functions. An overview over state- and transition-based modeling of dataflow networks is given in Sect. 4 and related ideas of suitable models and tools are presented.

Modeling systems as entities with input consumption and output production has motivated the term of stream processing functions [Bur75, Bro81] that describe a component's dataflow reaction to given input sequences. Extensions of this idea together with elements of system development processes such as creating specifications and successive refinement steps have lead to the FOCUS framework [BDD<sup>+</sup>92, BS01]. This framework is introduced in Sect. 3 and an overview of its tool support is given in Sect. 5.

## 2 Streams and Stream Processing

A stream describes the communication history on a channel connecting entities of a dataflow network. Such a history describes the observation of communication between two agents, interaction of software components, events in distributed pieces of hardware, etc. Examples of special types of dataflow networks are among others:

- sensors, control units, and actuators in automobiles exchanging data values and control signals,
- real-time software controlling actions of actuators depending on sensors' data,
- interaction between objects via message passing in object oriented software systems or
- messages transmitted between web services in cloud computing applications.

The history on a directed channel connecting two components in a network is modeled as a stream of messages. The general notion of a stream is a possibly infinite

Topology	Kind of Stream	Basic Form	Application Domain
discrete	event stream	$M^\omega$	software systems
discrete	time-sync. stream	$\mathbb{N} \rightarrow M$	time synchronous systems
discrete	timed event stream	$M^\omega := M^\omega$	timed specification <sup>1</sup>
discrete	time slice stream	$\mathbb{N} \rightarrow M^*$	time pulsed specification
dense	hybrid streams	$\mathbb{R}_+ \rightarrow M$	hybrid hardware systems
dense	signal set streams	$\mathbb{R}_+ \rightarrow \wp(M)$	discrete/hybrid systems
super dense	super dense streams	$\mathbb{R}_+ \rightarrow M^*$	hybrid systems

**Table 1.** Overview of kinds of streams

list of elements of some domain  $M$ . This domain  $M$  can according to the specified network be an abstraction of:

- event signals, e.g., messages on a bus,
- continuous values measured by sensors and discrete event signals,
- simple messages for signaling or method invocations or
- complex data structures passed between software services.

The notation of streams as finite and infinite sequences of messages of domain  $M$  is  $M^*$  and  $M^\infty$ .  $M^\omega$  is used to denote both finite and infinite sequences. Despite this general model of data histories, more detailed representations of streams are possible and necessary for specific modeling and specification tasks (see Table 1).

Whereas discrete streams model systems with event occurrences at discrete points in time, hybrid streams can have dense domains and even model continuous valued behavior as needed to express, e.g., analogous hardware and hybrid systems.

Another dimension to distinguish between stream system models is their way to express timing of events or values [SRS99]. The kinds of streams in Table 1 typically have further constraints regarding timing or the continuity of values on certain intervals, discussed below.

While each of these kinds of streams has its advantages, relationships such as embedding, refinement and abstraction help to map between these kinds. Some refinement and abstraction relations between stream kinds are presented in [Bro97]. This variety of streams is necessary, as the more a stream distinguishes the more effects can be expressed, but the more involved a specification of such streams typically is. The merge anomaly [Bro88] is a famous example.

### 2.1 Untimed Streams

The most generic model  $M^\omega$  describes untimed discrete event streams. It however cannot talk about timing and has some problems to distinguish unfinished specifications (prefixes) of communication histories and histories where nothing will happen anymore. However, it is relatively easy to use for specification.

An algebraically sound and useful set of specification operators on streams  $M^\omega$  is given in Table 2. The consolidated version shown in Table 2 is syntactically inspired by functional languages such as Haskell and its definitions are given, e.g., in [GR07,

<sup>1</sup>  $M_x$  is a shorthand for  $M \cup \{x\}$ .

Let  $s, s' \in M^\omega, m \in M, n \in \mathbb{N}_\infty, A \in (M \rightarrow \mathbb{B})$ :

Notation	Signature	Functionality
$\langle \rangle$ or $\varepsilon$	$\text{sempy} : M^\omega$	empty stream
$m:s$	$\text{scons} : M \times M^\omega \rightarrow M^\omega$	append first element
$s \hat{\ } s'$	$\text{sconc} : M^\omega \times M^\omega \rightarrow M^\omega$	concatenation of streams
$s \sqsubseteq s'$	$\text{spref} : M^\omega \times M^\omega \rightarrow \mathbb{B}$	prefix relation
$\#s \in \mathbb{N}_\infty$	$\text{slen} : M^\omega \rightarrow \mathbb{N}_\infty$	length of stream
	$\text{shd} : M^\omega \rightarrow M$	first element of stream
	$\text{srt} : M^\omega \rightarrow M^\omega$	stream without first element
$s.n$	$\text{snth} : \mathbb{N} \times M^\omega \rightarrow M$	$n^{\text{th}}$ element of stream
$s _n$	$\text{stake} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	prefix of length $n$
	$\text{sdrop} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	remove first $n$ elements
$m^n$	$\text{sntimesm} : \mathbb{N}_\infty \times M \rightarrow M^\omega$	message iterated $n$ times
$s^n$	$\text{sntimes} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	stream iterated $n$ times
$f * s$	$\text{smap} : (M \rightarrow M) \times M^\omega \rightarrow M^\omega$	elementwise function application
	$\text{site} : (M \rightarrow M) \times M \rightarrow M^\omega$	infinite iteration of function
$A \textcircled{S} s$	$\text{sfilter} : (M \rightarrow \mathbb{B}) \times M^\omega \rightarrow M^\omega$	filtering function
	$\text{stakewhile} : (M \rightarrow \mathbb{B}) \times M^\omega \rightarrow M^\omega$	prefix where predicate holds
	$\text{sdropwhile} : (M \rightarrow \mathbb{B}) \times M^\omega \rightarrow M^\omega$	drop prefix while predicate holds
$\alpha.s$	$\text{srcdups} : M^\omega \rightarrow M^\omega$	remove duplicates

**Table 2.** Operations on discrete event streams

GGR06], but not all of them are realizable (see Sect. 3.1). Quite a range of laws apply, some more important but rather obvious ones are given in Table 3.

## 2.2 Discrete Timed Streams

We distinguish three kinds of discrete event streams with timing information: time synchronous, time slice and timed event streams.

A function  $\mathbb{N} \rightarrow M$  is typically used to model time synchronous streams, where there exists exactly one event at every discrete point of time. While  $\mathbb{N} \rightarrow M$  is technically isomorphic to the subset of infinite streams from  $M^\omega$ , its semantic interpretation differs, as we assume a pulse driven system with one message per pulse, like in electronic circuits. A pseudo message  $\perp$  can be used to model time slices, where no real message is present:  $\mathbb{N} \rightarrow M_\perp$ . Causality as well as timing issues can be modeled easily with  $\mathbb{N} \rightarrow M$ . However, instant feedback in loops must be prevented and makes a delay necessary in each loop. This delay is also necessary in single-component-loops, which implies less elegant specifications. For practical purposes the set of incomplete observations  $[1..n] \rightarrow M$  should be allowed, e.g., when specifying inductively. This is then isomorphic to  $M^\omega$  (modulo different interpretations in the real world).

Timing can alternatively be introduced into discrete event streams by adding a pseudo message  $\checkmark$  (tick) denoting the end of a time interval. The tick separates equidistant intervals that may carry finitely many messages. These messages are still ordered, but the exact time of appearance within the time slice is left open. As a constraint, streams are only well formed when infinite observations contain infinitely many ticks:  $\forall s \in M^\infty : \{\checkmark\} \textcircled{S} s = \checkmark^\infty$ .

Let  $s, s', s'' \in M^\omega, m \in M, n \in \mathbb{N}, p \in (M \rightarrow \mathbb{B})$ :

Property	Info
$\langle \rangle : s = s$	neutral empty stream
$(s \hat{\ } s') \hat{\ } s'' = s \hat{\ } (s' \hat{\ } s'')$	associativity of concatenation
$\#(s \hat{\ } s') = \#s + \#s'$	length over concatenation
$s \sqsubseteq s' \Leftrightarrow \exists s'' : s \hat{\ } s'' = s'$	prefix operator
$\text{shd}(m:s) = m$	first element
$\text{srt}(m:s) = s$	rest element
$s^n = s \hat{\ } s \hat{\ } s \hat{\ } \dots \hat{\ } s$	stream iterated $n$ times
$(s \hat{\ } s') _{\#s} = s$	prefix
$\text{sdrop}(\#s, s \hat{\ } s') = s'$	remove first $\#s$ elements ( $s$ is finite)
$s.n = \text{shd}(\text{sdrop}(n-1, s))$	access to $n^{\text{th}}$ element
$\text{sntimes}(\infty, s) = s \hat{\ } \text{sntimes}(\infty, s)$	stream infinite iteration
$\text{siterate}(f, m) = \langle m, f(m), f(f(m)), f^3(m), \dots \rangle$	function infinite iteration
$p \textcircled{\&S} \langle \rangle = \langle \rangle$	filtering function
$p(m) \Rightarrow p \textcircled{\&S}(m:s) = m:(p \textcircled{\&S}s)$	
$\neg p(m) \Rightarrow p \textcircled{\&S}(m:s) = p \textcircled{\&S}s$	
$\text{stakewhile}(p, \langle \rangle) = \langle \rangle$	prefix where predicate holds
$p(m) \Rightarrow \text{stakewhile}(p, m:s) = m:(\text{stakewhile}(p, s))$	
$\neg p(m) \Rightarrow \text{stakewhile}(p, m:s) = \langle \rangle$	
$\text{stakewhile}(p, s) \hat{\ } \text{sdropwhile}(p, s) = s$	stakewhile + sdropwhile
$\alpha.\langle \rangle = \langle \rangle$	remove duplicates
$\alpha.(m:s) = m:(\alpha.(\lambda x.x \neq m) \textcircled{\&S}s)$	

**Table 3.** Some properties of stream operators

Timed event streams do not represent continuous time but rather group messages that occur in the same interval. The granularity of time, i.e., length of intervals is not part of the stream itself but can for example be given by a system's context. There are also mechanisms to map between interface granularity, e.g., through application of [Bro93] techniques.

Time synchronous streams  $(\mathbb{N} \rightarrow M)$  carry exactly one message per time unit. Thus kind  $\mathbb{N} \rightarrow M$  is embedded in  $M^\omega$  by restricting the latter to exactly one message per interval:  $\#s = \infty \wedge (\text{even}(n) \Leftrightarrow s.n = \checkmark)$ .

As  $M^\omega$  has some interesting properties, we consolidate a set of operations in Table 4. Please note that operations from Table 2 also apply, furthermore alternate versions with special treatment of ticks exist (listed in the lower part of Table 4). These versions are often helpful in writing specifications and some of their properties are shown in Table 5.

As a third kind, we introduce time slice streams  $\mathbb{N} \rightarrow M^*$ , which look very similar to timed event streams  $M^\omega$ . Actually their subsets of infinite streams are isomorphic, as both can contain arbitrary finite sequences of messages in a time slice. The operation `slice2event` maps between the two:  $\text{slice2event} : (\mathbb{N} \rightarrow M^*) \rightarrow M^\omega$ :

$$\text{slice2event}(s) = s' \Rightarrow \forall n \in \mathbb{N} : s(n) = s'._t n$$

Both forms of streams differ strongly in their inductive construction, one based on

Let  $s, s' \in M^\omega, m \in M, n \in \mathbb{N}_\infty, p \in (M \rightarrow \mathbb{B})$ :

Notation	Signature	Functionality
$\checkmark$	$\checkmark : M$	tick denoting end of time interval
$s \hat{\ }_t s'$	$\text{tscons} : M^* \times M^\omega \rightarrow M^\omega$	prefix first time interval
$\#_t s$	$\text{tsconc} : M^* \times M^\omega \rightarrow M^\omega$	concatenation with finite timed prefix
	$\text{tslen} : M^\omega \rightarrow \mathbb{N}_\infty$	number of time intervals in stream
	$\text{tshd} : M^\omega \rightarrow M^*$	first time interval
	$\text{tsrt} : M^\omega \rightarrow M^\omega$	stream without first time interval
$s_{.t}n$	$\text{tsnth} : \mathbb{N} \times M^\omega \rightarrow M^*$	$n^{\text{th}}$ time interval
$s \downarrow_n$	$\text{tstake} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	first $n$ time intervals
$s \uparrow_n$	$\text{tsdrop} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	drop first $n$ time intervals
$\diamond s$	$\text{tsabs} : M^\omega \rightarrow M^\omega$	abstraction to untimed stream
	$\text{delay} : M^\omega \rightarrow M^\omega$	prefixing with one empty time interval
$\#_{\checkmark} s$	$\text{tlen} : M^\omega \rightarrow \mathbb{N}_\infty$	number of data messages in stream (not $\checkmark$ )
	$\text{thd} : M^\omega \rightarrow M$	first data message on stream
	$\text{trt} : M^\omega \rightarrow M^\omega$	stream starting after first data message
$s_{.\checkmark}n$	$\text{tnth} : \mathbb{N} \times M^\omega \rightarrow M$	$n^{\text{th}}$ data message
	$\text{ttake} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	first $n$ non-empty time intervals
	$\text{tdrop} : \mathbb{N}_\infty \times M^\omega \rightarrow M^\omega$	stream after $n$ non-empty time intervals
$f *_{\checkmark} s$	$\text{tmap} : (M \rightarrow M) \times M^\omega \rightarrow M^\omega$	message wise function application
$p \textcircled{\text{S}}_{\checkmark} s$	$\text{tfilter} : (M \rightarrow \mathbb{B}) \times M^\omega \rightarrow M^\omega$	message wise filtering function
$\alpha_{\checkmark}.s$	$\text{trcdups} : M^\omega \rightarrow M^\omega$	remove duplicate data messages

**Table 4.** Operations for timed streams (lower part explicitly treats  $\checkmark$ )

events, the other on time slices. So, e.g., streams  $\langle 1, 2 \rangle$  and  $\langle 1, 2, \checkmark \rangle$  can only be distinguished in the event-based case  $M^\omega$ .  $M^\omega$  describes processing of individual events (with a tick as special event), whereas time sliced streams  $\mathbb{N} \rightarrow M^*$  have to describe handling of complete slices. This difference also induces different forms of processing of messages in an implementation. Event-based systems handle events on arrival, while time slice-based systems typically collect inputs and run on equidistant interrupts (representing intervals). The former are more efficient, the latter more amenable for deterministic distributed behavior modeling. It is noteworthy, that time sliced streams  $\mathbb{N} \rightarrow M^*$  are isomorphic to time synchronous streams  $\mathbb{N} \rightarrow O$ , when using sequences of messages as single new message:  $O = M^*$ .

For specifications it may also be helpful to abstract from time using  $\text{tsabs} : M^\omega \rightarrow M^\omega$  (short  $\diamond$ ) defined by  $\diamond s = M \textcircled{\text{S}} s$ .

### 2.3 Dense Timed Streams

A further refinement of discrete time is dense time where timing information is not restricted to abstract time intervals of arbitrary length but uses  $\mathbb{R}_+$  as time axis [MS96, Bro97]. The dense time model is also used in timed automata and related approaches to describe continuous and hybrid systems [AD90, AR02]. Dense streams can be represented as a function  $\mathbb{R}_+ \rightarrow M$  assigning a message to each point in time. This timing model is, e.g., used in [GSB98] for the specification of real-time systems.

Signal set streams with structure  $\mathbb{R}_+ \rightarrow \wp(M)$  are streams to model the indepen-

Let  $s \in M^\omega$ ,  $m \in M \setminus \{\checkmark\}$ ,  $n \in \mathbb{N}_\infty$ ,  $p \in (M \rightarrow \mathbb{B})$ :

Property	Info
$\text{delay}(s) = \checkmark : s$	delay stream
$\text{tlen}(s) = \#(\diamond s)$	length abstracting from timing
$\text{thd}(\checkmark : s) = \text{thd}(s)$	first data message of timed stream
$\text{thd}(m:s) = m$	
$\text{tnth}(s) = \text{snth}(\diamond s)$	$n^{\text{th}}$ data message
$\text{tmap}(f, \checkmark : s) = \checkmark : \text{tmap}(f, s)$	function application ignoring $\checkmark$
$\text{tmap}(f, m:s) = f(m) : \text{tmap}(f, s)$	
$p \otimes_{\checkmark} (\checkmark : s) = \checkmark : (p \otimes_{\checkmark} s)$	filtering data messages
$p(m) \Rightarrow p \otimes_{\checkmark} (m:s) = m : (p \otimes_{\checkmark} s)$	
$\neg p(m) \Rightarrow p \otimes_{\checkmark} (m:s) = p \otimes_{\checkmark} s$	
$\alpha_{\checkmark} . (\checkmark : s) = \checkmark : (\alpha_{\checkmark} . (s))$	remove duplicate data messages
$\alpha_{\checkmark} . (m:s) = m : (\alpha_{\checkmark} . (\lambda x. x \neq m) \otimes_{\checkmark} s)$	

**Table 5.** Some properties of operators for timed streams treating  $\checkmark$  explicitly

dent occurrence of events in time intervals [SRS99]. Signal sets have to be present for an interval  $I \subseteq \mathbb{R}_+$  and model the occurrence of signals in discrete hardware structures [Sch98]. A mapping of signal set streams to super dense event streams is possible by introducing messages  $m_s$  and  $m_e$  to denote start and end of a signal  $m \in M$ .

Super-dense time [MMP91] allows to observe a sequence of signals at each point in dense time. This model is strictly more expressive than dense time and used, e.g., for the specification of hardware systems and semantics of hybrid systems [LML06, MP93].

Although a discrete timed model  $M^\omega$  is strictly less expressive, we can transfer a super-dense (or dense) time model to a discrete if the dense timed specification does not allow zeno behavior (infinitely many messages in a finite interval) [AL91, LML06]. The abstraction  $\text{denseAbs} : (\mathbb{R}_+ \rightarrow M^*) \rightarrow (\mathbb{N} \rightarrow M^*)$  is well-defined by

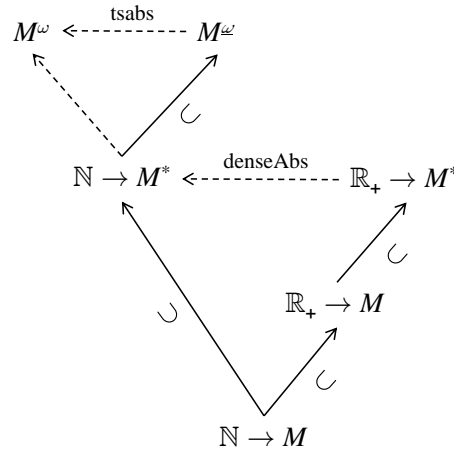
$$(\text{denseAbs } s)(n) = \bigwedge_{r \in [n-1..n]} s(r)$$

There are further approaches to encode time into streams. In the component coordination and composition framework Reo [Arb04] timed streams are modeled as a pair of streams. One stream carries data and the other one timing information [AR02]. The semantics of timed automata [AD90, AD94] are given as timed traces/streams in the dense time model as a pair of two sequences  $(\rho, \tau)$  of data and timestamps.

An overview in Fig. 1 shows the discussed stream kinds and their most interesting embedding and abstraction relations. A solid arrow depicts embedding while a dashed arrow denotes a possible abstraction. Using  $\text{timeAbs} : M^\omega \rightarrow M^\omega$  ( $\diamond$ ) is always a valid abstraction while  $\text{denseAbs} : \mathbb{R}_+ \rightarrow \wp(M)$  only works if zeno is absent.

### 3 Specification of Stream Processing Systems

While  $M^\omega$  can also describe traces of events [Die95], we get strictly more expressiveness when distinguishing input and output. There are several approaches that focus on stream processing as fundamental blocks of their system model [Ste97], while by far



**Fig. 1.** Abstraction and inclusion relations of stream kinds

Component with input  $(I_1^\omega \times \dots \times I_n^\omega) = \vec{I}$  and output  $(O_1^\omega \times \dots \times O_m^\omega) = \vec{O}$ :

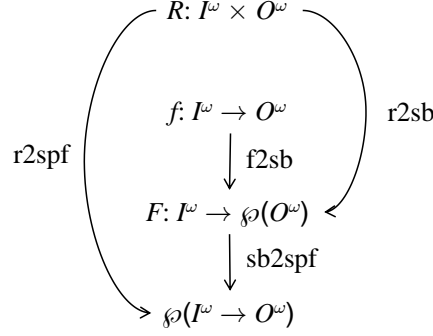
Concept	Body	Description
relational	$R : \vec{I} \times \vec{O}$	Specification as relation of observable I/O behavior, [Stø96, Lot96]
set-based function	$f : \vec{I} \rightarrow \wp(\vec{O})$	Specifying nondeterminism by allowing multiple output histories (using set-based functions)
set of functions	$\mathbb{F} : \wp(\vec{I} \rightarrow \vec{O})$	Definition of functions describing deterministic I/O behavior, [Bro95, BRSS97]
A/G	$A, G : \vec{I} \rightarrow \vec{O} \rightarrow \mathbb{B}$	Based on assumption $A$ the system guarantees that $G$ holds, [FP95, Bro95, SM97, Bro05, Spi07]
state-based	$\delta : \vec{I} \times S \rightarrow \vec{O} \times S$	I/O depending on state – see Sect. 4 and, e.g., [GR95, GKRB96, Rum96, BS01]

**Table 6.** Intuition of specification styles in FOCUS

the most elaborate method of the recent decades for the development of distributed systems based on stream processing is Manfred Broy’s FOCUS [BDD<sup>+</sup>92, BS01].

Denotational semantics have been investigated to describe deterministic and non-deterministic systems [Kah74, Bro81, KP85, BA81, Abr83], where only some tried to handle component specifications as first class entities. For example [Kah74] uses history functions mapping input streams to output streams. But its natural extension for non-deterministic processes to simply relate input streams to a set of possible output streams turned out to be inadequate [Kel78, BA81], e.g., for non-deterministic merge with feedback or the merge anomaly where components with equal history relations do not behave alike when embedded in a network.

FOCUS provides several specification styles: relational style, assumption/guarantee style, equational/functional style and graphical style, which subsumes state transition



**Fig. 2.** Transformations of relational and functional specifications

diagrams, tables and composition diagrams [BS01]. These specification styles correspond to underlying concepts denoted in Table 6. Again, the more detailed behavior can be distinguished, the less anomalies exist, but the more complicated the mechanisms to deal with become. The most fine grained model of components are sets of stream processing functions (SPF)  $\wp(\vec{I} \rightarrow \vec{O})$ . While inspired by Kahn process networks among others the merge anomaly is avoided using sets of SPF to describe non-deterministic behavior [Bro88]. Set-based functions  $\vec{I} \rightarrow \wp(\vec{O})$  are strictly weaker, still distinguishing enough to avoid anomalies, but have a problem to ensure realizability (see below). Relations  $\vec{I} \times \vec{O}$  are again strictly weaker and exhibit too many problems with composition.

As a single function is a deterministic description of behavior and can directly be understood as realization, all other styles allow non-deterministic and underspecified descriptions. Realizability deals with the question, whether such a specification has any possible implementations. While this is trivial to answer for sets of SPF, realizability needs more caution with set-based functions and relations.

Fortunately there are transformations between the styles as shown in Fig. 2 using:

$$\begin{aligned}
 \text{r2spf}(R) &= \{f : I^\omega \rightarrow O^\omega \mid \forall i \in I^\omega : R(i, f(i))\} \\
 \text{f2sb}(f) &= F : I^\omega \rightarrow \wp(O^\omega) \text{ where } \forall i \in I^\omega : F(i) = \{f(i)\} \\
 \text{r2sb}(R) &= F : I^\omega \rightarrow \wp(O^\omega) \text{ where } \forall i \in I^\omega : R(i, F(i)) \\
 \text{sb2spf}(F) &= \{f : I^\omega \rightarrow O^\omega \mid \forall i \in I^\omega : f(i) \in F(i)\}
 \end{aligned}$$

Please note that these translations between specification styles are not taking care of realizability. Properties of realizable stream processing functions follow in the next two subsections.

### 3.1 Stream Processing Functions - Untimed Case

An untimed stream processing function  $f : I^\omega \rightarrow O^\omega$  maps an input stream to a corresponding output stream. However, when receiving new input, it may not *undo* messages that have been emitted (monotonicity):

$$\forall x, y \in I^\omega : x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$$

Let  $s \in M^\omega, c, c' \in C, C' \subseteq C, b, b' \in B^\Omega \supset B^\Phi$ :

Notation	Signature	Functionality
$c \in C$	$C$	channel names
$B^\Phi$	$B^\Phi \subseteq C \rightarrow M^*$	finite stream bundles
$B^\Sigma$	$B^\Sigma \subseteq C \rightarrow M^\infty$	infinite stream bundles
$B^\Omega$	$B^\Omega \subseteq C \rightarrow M^\omega$	stream bundles $B^\Phi, B^\Sigma \subset B^\Omega$
	$\text{bdom} : B^\Omega \rightarrow \wp(C)$	channel names in bundle
$b \cup b'$	$\text{bunion} : B^\Omega \times B^\Omega \rightarrow B^\Omega$	merge of bundles
$b \setminus c$	$\text{bremch} : B^\Omega \times C \rightarrow B^\Omega$	remove channel
$b[c \mapsto s]$	$\text{bsetch} : B^\Omega \times C \times M^\omega \rightarrow B^\Omega$	add or replace channel
$b_c$	$\text{bgetch} : B^\Omega \times C \rightarrow M^\omega$	retrieve channel content
$b[c \rightarrow c']$	$\text{brenamech} : B^\Omega \times C \times C \rightarrow B^\Omega$	rename channel
$b \doteq b'$	$\text{beq} : B^\Omega \times B^\Omega \rightarrow \mathbb{B}$	equality on common channels
$b \stackrel{C'}{=} b'$	$\text{beqch} : B^\Omega \times B^\Omega \times \wp(C) \rightarrow \mathbb{B}$	equality on specific channels

**Table 7.** Operations on stream bundles

In addition it may not emit any message based on knowledge that nothing happens anymore (continuity):

$$\forall \text{chain } x_n \subset I^\omega : f(\sqcup x_n) = \sqcup f(x_n)$$

This definition of continuity is based on the important property that  $(I^\omega, \sqsubseteq)$  and  $(O^\omega, \sqsubseteq)$  are complete partial orders (for details see [Kah74, Bro86]). Continuity and monotonicity are necessary for a function to be implementable (realizable). Table 2 of Sect. 2.2 shows a number of realizable functions, such as  $. : .$ ,  $\text{shd}$ ,  $\text{srt}$ ,  $s.n$ ,  $s|_n$ ,  $\text{sdrop}$ ,  $m^n$ ,  $s^n$ ,  $\text{smap}$ ,  $\text{siterate}$ ,  $A\textcircled{s}$ ,  $\text{stakewhile}$ ,  $\text{sdropwhile}$ ,  $\alpha.s$ . However,  $\#s$  and  $s \sqsubseteq s'$  are not realizable (due to infiniteness) and concatenation is realizable only in its second argument.

The above formulated continuity allows behavior observable on infinite streams to be approximated from finite input histories. This property guarantees the existence of a least fixed point when describing compositions such as feedback loops [Kah74, Bro88, BDD<sup>+</sup>92, Den95]. Most importantly, the above formulated continuity of SPF is preserved under function composition [Bro86] allowing to conveniently develop new functions.

For general component models, SPFs need to be expanded to several input and output streams typically of different message types and are thus of the form  $f : I_1 \times \dots \times I_n \rightarrow O_1 \times \dots \times O_m$ . Such  $n$ -to- $m$  functions can be rewritten in a closed form using stream bundles  $B^\Omega$  [Fuc94, Rum96]. Assuming a given set of channel names  $C$  and a typing function  $\text{ctype} : C \rightarrow \wp(M)$ , a bundle  $b \in B^\Omega$  is a partial function  $C \rightarrow M^\omega$  mapping channel names  $c \in C$  to their corresponding streams with the typing constraint that each stream must have its correct type ( $b_c \in \text{ctype}(c)^\omega$ ). Some operations on stream bundles are shown in Table 7. Stream bundles are only partial functions since not every channel name has a corresponding stream in a bundle. A special equality on shared channels of two bundles is defined by:

$$b \doteq b' \Leftrightarrow \forall c \in \text{bdom}(b) \cap \text{bdom}(b') : b_c = b'_c$$

Let  $f, g \in SPF$ ,  $c_i, c_o \in C$ ,  $C' \subseteq C$ :

Notation	Signature	Functionality
$f \in SPF$	$SPF = B^\Omega \rightarrow B^\Omega$ $\text{spfdom} : SPF \rightarrow \wp(C)$ $\text{spfrange} : SPF \rightarrow \wp(C)$	stream processing functions domain of an SPF range of an SPF
$f \otimes g$	$\text{spfcamp} : SPF \times SPF \rightarrow SPF$	composition of SPFs
$f _{C'}$	$\text{spfrestrict} : SPF \times \wp(C) \rightarrow SPF$	range restriction of SPF (hiding)
$f \uparrow_{c_i}^{c_o}$	$\text{spflift} : (I^\omega \rightarrow O^\omega) \times C \times C \rightarrow SPF$	lifting of an SPF to bundle

**Table 8.** Operations on SPF

Property	Info
$\text{spfrange}(f _C) = \text{spfrange}(f) \cap C$	range restriction
$\forall b \in B^\Omega : (f _C)(b) \doteq f(b)$	
$b \doteq b' \Leftrightarrow \forall c \in \text{bdom}(b) \cap \text{bdom}(b') : b_c = b'_c$	equal on common channels
$b \stackrel{C'}{\doteq} b' \Leftrightarrow \forall c \in C' : b_c = b'_c$	equal on selected channels
$\forall b \in B^\Omega : (f \uparrow_{c_i}^{c_o})(b)_{c_o} = f(b_{c_i})$	lifted SPF
$\text{spfrange}(f \otimes g) = \text{spfrange}(f) \cup \text{spfrange}(g)$	$\otimes$ not hiding output
$\text{spfdom}(f \otimes g) = (\text{spfdom}(f) \cup \text{spfdom}(g)) \setminus \text{spfrange}(f \otimes g)$	$\otimes$ hiding local inputs
$\forall b \in B^\Omega : \text{bdom}(b) = \text{spfdom}(f) \cup \text{spfdom}(g) \Rightarrow$ $f(b) \doteq (f \otimes g)(b) \doteq g(b)$	composition

**Table 9.** Properties of  $SPF$  operators

Bundles can be merged, and channels can be removed from a bundle's domain or added as a new channel pointing to a given stream. Also many functions like the head and rest operator for streams and others like map and filter can be extended pointwise to stream bundles.

The domain of stream processing functions  $SPF$  is defined as the set of all monotonic and continuous functions  $f : B^\Omega \rightarrow B^\Omega$ . The defined input and output channels of  $f$  are  $\text{spfdom}(f)$ ,  $\text{spfrange}(f) \subseteq C$ . As discussed, continuous operations from Table 2 can be lifted to domain  $SPF$ , e.g., using operators like  $\text{spflift}$  (Table 8). Furthermore function composition straightforwardly extends to  $SPF$  and the most important operator on  $SPF$  is bundle-based composition  $f \otimes g$  for  $f, g \in SPF$ , which composes all shared input and output channels (Table 8 and 9). Bundle composition can be extended to finite sets of components and if hidden channels are not reused several times it is associative and commutative. Moreover, it comprises classic operators, like sequential composition  $g \succ f$ , where only  $f$  feeds into  $g$ , parallel composition  $f \parallel g$  where no channels coincide and feedback  $\mu f$ , where the output of a component is fed back to its own input.

As a major result  $f \otimes g$  is compositional, thus allowing to define decomposed glass box definitions of components as in the FOCUS graphical composite style. Even hierarchical decomposition is possible. The overall system is realizable, when each of its components is and the composed behavior can be derived from the behavior of its subcomponents using  $\otimes$ .

### 3.2 Stream Processing Functions - Timed Case

The mechanism introduced for the untimed case can uniformly be adapted to the timed case. In particular the bundle mechanism can be adapted to each kind of streams. It is even possible to mix those kinds within one specification by adapting the typing function  $c_{type}$  appropriately.

Timed functions however need to obey timing appropriately to be realizable. In the case of  $f : I^\infty \rightarrow O^\infty$  realizability is a causality restriction [BDDW91, Bro95, BS01]. Let us denote timed bundles with  $B^\Omega$  and timed SPF with  $\overline{SPF}$ . A component cannot predict the future and thus all its behavior describing functions  $f \in \overline{SPF}$  have to be weakly causal:

$$\forall b_1, b_2 \in B^\Omega, t \in \mathbb{N} : C_i := \text{spfdom}(f), C_o := \text{spfrange}(f) :$$

$$b_1 \downarrow_t \stackrel{C_i}{=} b_2 \downarrow_t \Rightarrow f(b_1) \downarrow_t \stackrel{C_o}{=} f(b_2) \downarrow_t$$

This means output can only rely on input that already has arrived. But weakly causal functions may react instantly, which leads to problems when a feedback loop has no delay. Thus weak causality is not necessarily preserved in compositions. A slightly stronger constraint is strong causality, which is preserved:

$$\forall b_1, b_2 \in B^\Omega, t \in \mathbb{N} : C_i := \text{spfdom}(f), C_o := \text{spfrange}(f) :$$

$$b_1 \downarrow_t \stackrel{C_i}{=} b_2 \downarrow_t \Rightarrow f(b_1) \downarrow_{t+1} \stackrel{C_o}{=} f(b_2) \downarrow_{t+1}$$

If a composition (like  $\otimes$  above) has one strong causal function in each feedback loop then it is well defined. Please note that weak causality replaces monotonicity and continuity of untimed SPF.

### 3.3 Component Specifications with Sets of Stream Processing Functions

A single SPF  $f \in SPF$  resp.  $f \in \overline{SPF}$  can by nature be regarded as a deterministic implementation. Specification of systems however intrinsically is interested in allowing underspecification. A component specification therefore describes a set of SPF where all functions are of the same signature. Underspecification is removed by allowing all possibly non-deterministic behavior. We define  $PSPF \subseteq \wp(SPF)$  as the semantics domain of specifications. Semantics of a specification  $S$  is denoted  $\llbracket S \rrbracket \subseteq PSPF$ .

Any specification  $S$ , e.g., in relational or A/G-style, is realizable, exactly when its semantics is a nonempty set of SPF, i.e.,  $\llbracket S \rrbracket \neq \emptyset$ . Certain specification styles, such as the use of state machines in Section 4.1, are realizable by construction. For others, such as the relational or set-based function style this is sometimes tricky to prove.

In the timed case, we call a specification (strongly/weakly) realizable if at least one (strongly/weakly) causal SPF exists that conforms to the I/O relation induced by the specification [BS01].

On the domain of  $PSPF$  we can use set operations as well as elementwise application of  $SPF$  operations. This gives us two major techniques that any good specification technique needs and Manfred Broy's FOCUS delivers: semantically sound hierarchical decomposition with  $\otimes$  and behavioral refinement based on set inclusion  $\subseteq$  that are compatible:

$$\llbracket S \rrbracket \subseteq \llbracket S' \rrbracket \Rightarrow (\llbracket S \rrbracket \otimes \llbracket T \rrbracket) \subseteq (\llbracket S' \rrbracket \otimes \llbracket T \rrbracket)$$

This means that refinement is transitive, refinement of components leads to refinement

of the overall system and thus allows us to refine components independently of their hierarchic embedding.

Additional operators allow to adapt interfaces, e.g., for different abstraction levels. The given specification  $S$  is refined via an interface refinement to  $S'$  if and only if  $\llbracket S \rrbracket \supseteq (\llbracket D \rrbracket \otimes \llbracket S' \rrbracket \otimes \llbracket U \rrbracket)$  with sequential composition of suitable interface mapping specifications  $D$  and  $U$ . Even conditional refinement is possible [Bro93].

It is also very helpful that these refinement techniques work in the same form for any underlying kind of streams. Furthermore, the semantic refinement relation can be adapted to various specification styles by definition of refinement rules on the syntax of the specification. This is for example done in [PR94, Rum96, RK96] for several types of automata and in [PR97, PR99] for pipes and filters dataflow networks.

#### 4 Automata and State-Based Specification

State-based modeling of the behavior of reactive systems has proven useful and practical to describe systems in a succinct and comprehensive way [HP85, Har87, Rum97, BS01]. Reactive systems have to respond to messages and events generally depending on their history. States comprise an abstraction of this history relevant to the component to react properly. A transition allows to model such a reaction, defining source and destination states, an event to occur as trigger and the performed action. In a finite representation, guard conditions, more complex event descriptions, and actions as well as state invariants may be used. State machines are a rather successful and widely adapted modeling and specification technique. The representation of interactive state transition systems ranges from simple Mealy automata with pure I/O transitions to UML state charts [Har87, OMG07] with guarded transitions, hierarchy, concurrency and communication concepts as well as related forms of automata inspired by these languages [GR95, Rum96, BS01, GGR06].

As discussed in Table 6 of Sect. 3 FOCUS provides a state-based specification of the form  $(I^\omega \times S \rightarrow O^\omega \times S)$ . One way to integrate this with the functional style is to start from an SPF specification by grouping equivalent behaviors (so called “continuation functions”) and regard those as states [BDD<sup>+</sup>92, Den95]. This reflects that a component’s state comprises the history of all its messages received and produced so far and thus determines the future behavior (modulo nondeterminism). However, this implicit I/O history state is often rather complex to describe in a functional implementation. We thus describe a direct approach that uses states and transitions descriptions for specification purposes [Rum96, BS01]. We go into detail for the state- and transition-based modeling and specification approach in the next section and concentrate on the basic form of state machines (without transition guards, state invariants etc.), but allow, e.g., an infinite state space. A general concept of state transition diagrams is presented in [GKRB96] with graphical and textual syntax.

##### 4.1 Automata for Stream Processing - Untimed Case

The notion of automata to describe interactive distributed systems based on the FOCUS calculus is investigated in [Rum96]. A component is modeled as an  $I/O^\omega$  automaton, where in contrast to I/O-automata [LT89, dAH01] each transition accepts one input message and produces a sequence of output messages. An  $I/O^\omega$  automaton is a tuple  $(S, M_{in}, M_{out}, \delta, I)$  with a non-empty set of possibly infinite states  $S$ , in- and

output alphabet  $M_{in}$  and  $M_{out}$ , a state transition function  $\delta \subseteq S \times M_{in} \times S \times M_{out}^\omega$  and initial state and output pairs  $I \subseteq S \times M_{out}^\omega$ . A transition with infinite output is called final since no other action is visible after it. Final transitions typically resemble non-terminating actions. A restriction to finite transitions is possible and resembles that technology enforces all actions to terminate.

A denotational semantics for an I/O $^\omega$  automaton is given by a mapping to a set of stream processing functions [Rum96] as shown in Def. 4.1 for the case of a total transition function  $\delta: \forall s \in S, i \in M_{in}. \exists o \in M_{out}^\omega, t \in S. (s, i, t, o) \in \delta$ .

**Definition 4.1 (Semantics of total I/O $^\omega$  automata)** *The set of SPF realized by the total automaton  $(S, M_{in}, M_{out}, \delta, I)$  is*

$$\begin{aligned} \llbracket (S, M_{in}, M_{out}, \delta, I) \rrbracket &= \{g \in M_{in}^\omega \rightarrow M_{out}^\omega \mid \\ &\exists h \in \llbracket (S, M_{in}, M_{out}, \delta, I) \rrbracket^p, (s_i, out_i) \in I. \\ &\forall in. g(in) = out_i \hat{\ } h(s_i, in)\} \end{aligned}$$

where  $\llbracket \cdot \rrbracket^p$  is the unique largest solution of the following recursive definition:

$$\begin{aligned} \llbracket (S, M_{in}, M_{out}, \delta, I) \rrbracket^p &= \{h \in S \rightarrow M_{in}^\omega \rightarrow M_{out}^\omega \mid (\forall s. h(s, \langle \rangle) = \langle \rangle) \wedge \\ &\forall m \in M_{in}, s \in S. \exists t \in S, out \in M_{out}^\omega. (s, m, t, out) \in \delta \wedge \\ &\exists h' \in \llbracket (S, M_{in}, M_{out}, \delta, I) \rrbracket^p. \\ &\forall in \in M_{in}^\omega. h(s, m:in) = out \hat{\ } h'(t, in)\} \end{aligned}$$

While the first part of the definition handles the selection of a start state and initial output, the second defines a predicate  $\llbracket \cdot \rrbracket^p$  recursively over the transition relation. This recursive definition is well formed and does have a unique largest solution [Rum96]. The core idea of predicate  $\llbracket \cdot \rrbracket^p$  is to unfold the transition relation one step, by defining the state parametrized function  $h$  through selection of a transition  $(s, m, t, out) \in \delta$  and a continuation function  $h'$  that handles the rest. Taking the largest solution resembles that nondeterminism is interpreted as underspecification. Please note that the ability to respond with infinite output motivates the notion of non-terminating execution and neither destination state nor further input contribute to the components behavior anymore.

[Rum96] shows that for any nonempty set  $S$  of SPF there is a corresponding I/O $^\omega$  automaton  $A$  with  $\llbracket A \rrbracket = S$ . This demonstrates the equivalence of expressiveness for both approaches, but only works, because we do not restrict the state space to be finite.

If transition function  $\delta$  is furthermore deterministic ( $\forall s \in S, i \in M_{in}. \exists_1 o \in M_{out}^\omega, t \in S. (s, i, t, o) \in \delta$ ) then  $\llbracket A \rrbracket$  is also deterministic, i.e., exactly one SPF. A simplified definition of  $\llbracket \cdot \rrbracket$  is then possible [Rum96].

As an extension, partiality of the transition relation is interpreted as underspecification, where arbitrary behavior is possible. Thus a partial automaton can via non-deterministic chaos completion be mapped to a total automaton.

I/O $^\omega$  automata are quite an amenable style of specification with quite a nice concrete representation ([Rum96]). Due to their more expressive transitions they also allow to describe many systems with finite state spaces where, e.g., Lynch's I/O-automata already have an infinite state space. However, they (still) lack of a sound

Refinement rules for  $(S, M_{in}, M_{out}, \delta, Init) \rightsquigarrow (S', M'_{in}, M'_{out}, \delta', Init')$ 

Change	Semantics	Condition & Description
$Init' \subseteq Init$	refined	removing initial non-determinism
$\delta' \subseteq \delta$	refined	removing non-deterministic transitions (with same input in same state); constraint: only for reduction of nondeterminism
$\delta' \supseteq \delta$	refined	add transitions: removing partiality of accepted input; constraint: not allowed to introduce alternatives to existing transitions
$S' \subseteq S, \delta' \subseteq \delta$	preserved	removing states not reachable with any finite or infinite transition sequence
$S' \supseteq S$	preserved	adding states
$S \rightsquigarrow S'$	preserved	$S$ replaced by $S'$ with a total, surjective relation that respects $\delta'$ from $S$ to $S'$ (adapting $\delta'$ and $Init'$ )
$Init \rightsquigarrow Init'$	preserved	changing initial state where initial output is infinite
$\delta \rightsquigarrow \delta'$	preserved	changing destination state where output is infinite
$M_{in} \subseteq M'_{in}$	preserved	extending input alphabet: semantics preserved for inputs of $M_{in}^\omega$
$S' = S_\perp, \delta \subseteq \delta'$	preserved	chaos complete: adding error state $\perp$ , making transition relation total using target state $\perp$ , and allowing any output
$\delta \supseteq \delta'$	preserved	compactify: transforming transitions with infinite output to self-loops

**Table 10.** Operations preserving or refining semantics of automata

composition, that somehow retains the structure of the composed automata, such that formal treatment using verification tools like Isabelle [NPW02] on the modeling level is not very comfortable. A composition through mapping to *PSPF* and use of the function composition is not very satisfying. Furthermore, the defined form of automata does not deal with several input channels. Such an extension however needs some additional assumptions, like e.g. timing. On the other hand, we do have a very useful set of refinement techniques at hand (see [Rum96] and Table 10) that is compatible with refinement in the semantics domain. Furthermore,  $I/O^\omega$  automata represent some operational flavor, as each transition that has a reachable source state can actually be taken and thus contributes to realizations. This is not necessarily the case with relational specifications. As another advantage of  $I/O^\omega$  automata specifications, properties like monotonicity or continuity and causality hold by construction.

Expressive power of  $I/O^\omega$  automata is, e.g., shown in [RK96, KPR97, BCR07] where they are used to describe object behavior, regarding calls and returns as messages. This leads to an object refinement calculus as studied in [PR94]. The refinement techniques from [Rum96] allow to model inheritance and subtyping and are quite similar to [HK02] in handling nondeterminism, but take an opposite view on interpreting underspecification. While [HK02] follows a constructive approach, [Rum96] favors a specification approach by allowing arbitrary reaction to message occurrences not defined in the transition relation.

A bisimulation relation between  $I/O^\omega$  automata ensures equality of the resulting semantics and a simulation relation results in a semantics refinement. As discussed in [Rum96] this leads to a number of possible operations on  $I/O^\omega$  automata listed in Table 10. Modifications from Table 10 that preserve semantics can be used for restructuring the syntactical representation of an automaton and apply in both directions. The operation chaos complete is used to create total automata from partial ones to also define their semantics using Def. 4.1 and Def. 4.2.

#### 4.2 Automata for Stream Processing - Timed Case

In Section 3.2 we have refined the domain of untimed SPF to event based timing by introducing  $\checkmark$  as special message to denote progress of time. In the same way, we can now use  $\checkmark$  as special message in  $I/O^\omega$  automata. We speak of event driven timed  $I_\checkmark/O_\checkmark^\omega$  automata, when for  $\delta$  holds  $(s, \checkmark, t, o) \in \delta \Rightarrow \text{shd}(o) = \checkmark$  and no transition has infinite output. With these restrictions we ensure that the semantics of  $I_\checkmark/O_\checkmark^\omega$  automata is weakly causal and thus describes a nonempty set of well-formed timed SPF  $\wp(M_{in}^\omega \rightarrow M_{out}^\omega)$ . This means that time progress in input is directly mapped to the output, but the remainder of the reaction is free to handle time progress (e.g. as a counter or a time out). Please note that a reasonable refinement would be to allow transitions to emit one single  $\checkmark$  exactly when a  $\checkmark$  is processed, because emission of additional  $\checkmark$ s means introducing delay. This is how strong causality is ensured:  $(s, out) \in Init \Rightarrow \#(\{\checkmark\} \otimes out) \geq 1$ .

Therefore, the theory for  $I/O^\omega$  automata can easily be transferred to event driven timed  $I_\checkmark/O_\checkmark^\omega$  automata, where the transitions are interpreted as event processing transitions and  $\checkmark$  again is just a special form of event. Furthermore, refinement techniques for  $I/O^\omega$  automata either apply directly or can slightly be adapted (e.g. when new transitions are introduced or chaos completion is used, we see timing restrictions). In contrast to other timed automata [AD94] event driven timed  $I_\checkmark/O_\checkmark^\omega$  automata have no need for explicit timers but model time directly instead.

Like with SPF, there are alternative ways to use automata for timed specifications. Interpreting a transition as a description of behavior in one time slice leads to time pulsed automata  $(S, I, O, \delta, Init)^{ta}$ , where the transition relation  $\delta \subseteq S \times I^\Phi \times S \times O^\Phi$ , initial states  $Init \subseteq S \times O^\Phi$  and  $B^\Phi$  denotes a single slice of a stream bundle  $B^\Omega$ .

Such a pulse timed automaton consumes one time slice of finitely many input messages ( $i \in I^\Phi$ ) in every transition step and produces a time slice with finitely many output messages ( $o \in O^\Phi$ ) and is therefore closely related to the time slice based SPF domain. In contrast to untimed  $I/O^\omega$  automata, this approach can handle multiple input channels in parallel and is thus defined on stream bundles. The meaning of time pulsed automata is defined by mapping them to the  $\overline{SPF}$  domain in Def. 4.2.

**Definition 4.2 (Semantics of time pulsed total  $I/O^*$  automata)** *The behavior realizing stream processing functions of the automaton  $(S, I, O, \delta, Init)^{ta}$  is*

$$\begin{aligned} \llbracket (S, I, O, \delta, Init) \rrbracket^{ta} &= \{g \in I^\Sigma \rightarrow O^\Sigma \mid \\ &\exists h \in \llbracket (S, I, O, \delta, Init) \rrbracket^{tap}, (s_i, out_i) \in Init. \\ &\forall in. g(in) = out_i \hat{\checkmark} \hat{h}(s_i, in)\} \end{aligned}$$

where  $\llbracket \cdot \rrbracket^{tap}$  is the unique largest solution of the following recursive definition:

$$\begin{aligned} \llbracket (S, I, O, \delta, Init) \rrbracket^{tap} = & \{h \in S \rightarrow I^\Sigma \rightarrow O^\Sigma \mid \\ & \forall m \in I^\Phi, s \in S. \exists t \in S, out \in O^\Phi. (s, m, t, out) \in \delta \wedge \\ & \exists h' \in \llbracket (S, I, O, \delta, Init) \rrbracket^{tap}. \\ & \forall in \in I^\Phi. h(s, m \hat{\vee} \hat{\vee} in) = out \hat{\vee} \hat{\vee} h'(t, in)\} \end{aligned}$$

As in Def. 4.1 the first part of the definition deals with the selection of an initial state and an initial output. The  $\hat{\vee}$  used (here representing a  $\checkmark$  on all channels of the concatenated bundles) ensures strong causality of all resulting functions. This also means that the output of a transition is emitted exactly one step after the input is being consumed. The second part again is a recursive set-based definition with a unique largest solution. The definition is in fact structurally equivalent to Def. 4.1, when regarding  $I^\Phi$  as set of messages  $M_{in}$ .

Refinement and other operations from I/O $^\omega$  automata can again be easily adapted to pulsed automata. In addition there is an important composition technique available that uses the fact that the pulse synchronizes these automata [GR95]. This composition uses a cross product of the states of composed automata. This technique unfortunately does not work for event driven timed I $^\checkmark$ /O $^\checkmark$  automata, as the concept of state there denotes states of the object at the end of processing events, while in pulsed timed automata state describes a situation at a specific point of time and computation may still be in progress. This is also the reason why a mapping between event driven timed automata and pulsed timed automata is intrinsically complex, although both are equally expressive. Without any additional assumptions, mapping event driven automata to pulsed automata leads to an explosion of transitions, whereas a mapping of pulse driven automata to event driven automata leads to an explosion of states.

The above discussed automata are, e.g., used in [KPR97] to model features of complex systems and their interaction. There the notion of adding a feature  $F$  is a refinement of a system  $S \xrightarrow{F} S'$ . The notion of conflicting features corresponds to the notion of consistency and independence of applying those refinements.

Further variants of automata are, e.g., HyCharts [GSB98] for the specification of hybrid systems, based on the theory of dense streams [Bro97, MS97]. The architecture of component networks is graphically represented in HyACharts, which are dataflow graphs describing composition of hybrid components in a hierarchic and modular way. HySCharts are hybrid state charts to describe the combinational and analog part of a hybrid component. The combinational part describes discrete state changes while the analog part is specified by differential (in)equations.

The AutoFocus tool [HSS96, HS97, BHS99, Wil06] for the specification and prototyping of distributed systems allows the behavioral specification of components using state transition diagrams. These can be edited in a graphical representation and translated to executable simulation code. Currently the timing model supported by AutoFocus are time synchronous streams and strongly causal behavior of components.

## 5 Tool Support for Stream Processing Specifications

A good theory can only be put to practice, if there are practical tools supporting it. Therefore, several implementations of aspects of the FOCUS methodology have been

created. These differ in their purpose and underlying technology. While some approaches offer interactive proof support for tasks that often repeat in similar ways during development (see Sect. 5.1), others offer a tool infrastructure that covers amenable modeling languages, partially as well as fully automated checks and code generation from artifacts (see Sect. 5.2).

### 5.1 Interactive Theorem Proving

One of the first approaches to formalizing semantics of Kahn networks [Kah74], which also inspired FOCUS to a great extent, using theorem prover support is [DS89]. The authors interpret component networks as functional programs, which they verify using an own implementation of the  $\mu$ -calculus in the theorem prover Isabelle [NPW02]. The implementation is, e.g., illustrated on a machine-checked verification of the correctness of the alternating bit protocol.

There have been other approaches to use Isabelle for an implementation of the FOCUS framework. One of the first embeddings of FOCUS in Isabelle/HOLCF has been done by Schätz and Spies [SS95]. They present a theory for the specification of agents (components) with functional and predicative specifications of behavior. Timing of streams is not taken into account. A translation of network descriptions given in ANDL [BDD<sup>+</sup>92, SS95] syntax to HOLCF is also shown. This work focuses mainly on a syntactic translation and does not discuss further proof support.

Later works focus on the implementation of stream processing functions linked to FOCUS in terms of methodological usage of the framework. A work by Sandner and Müller [SM97] is on proving refinements based on the Assumption/Commitment style [SDW93]. It also covers rules for structural refinement (sequential and parallel composition, introduction of feedback loops and refinement of specifications). The approach is evaluated on a case study of a production cell.

Several implementations of stream processing functions are based on the logic HOLCF [Slo97, MNvOS99] with adds concepts of domain theory to Isabelle's implementation of higher order logic HOL. General refinement rules for interactive systems with an application to FOCUS are studied by Slotosch [Slo97] creating Isabelle/HOLCF.

A less interactive approach of using formal methods on FOCUS specifications is evaluated in [SB99]. Specification refinement is checked using two automated proof tools: the first order logic theorem prover SETHEO [LSBB92] and the model checker SMV [BCM<sup>+</sup>92]. Because of the limited expressiveness of first order logic the reasoning task on stream processing functions is translated to induction cases by hand that can then be automatically solved using SETHEO. Reasoning over finite domains is carried out using SMV.

More recent works [GR06, GR07] are focused on the implementation of the FOCUS streams system model in Isabelle/HOLCF. The theory ALICE formalizes discrete finite and infinite streams as well as infinite timed (event) streams. Further work focuses on the specification of stream processing functions and stream bundle processing functions to represent components and component networks using Isabelle/ALICE [GR07].

Another recent contribution to a formalization of FOCUS in Isabelle has been made by Spichkova [Spi07]. System specifications can either be translated manually or developed directly in Isabelle/HOL. The methodology proposed follows the FOCUS

idea of specification and continuous refinement until the desired level of detail is reached. The Isabelle/HOL implementation covers timed streams and proposes also ways to handle time synchronous streams. For the description of larger systems sheaves of channels are introduced to specify arbitrary but finite numbers of components. A scheme for translating specifications to Isabelle/HOL with manual simplifications like removing mutual recursion is proposed.

The behavioral specification of automotive embedded systems is investigated in [Tra09]. Part of the semantics of the IDE AutoFocus are verified in Isabelle/HOL. A framework for temporal and other declarative specifications of functional properties is developed. The work aims at supporting the development process from design phase to an executable specification. As an industrial case study an adaptive cruise control system is formalized.

## 5.2 *AutoFocus*

The distributed systems specification tool AutoFocus [HSS96, BHS99, Wil06] is based on the FOCUS method and offers graphical representations of modeling and specification artifacts. Supported specification documents for artifacts are: system structure diagrams (SSD), data type definitions (DTD), state transitions diagrams (STD), and extended event traces (EET). SSDs describe the logical architecture of the developed system as component dataflow networks. STDs represent automata where each transition consumes and produces one message on each incoming and outgoing port and interaction is time synchronous. EETs are similar to UML sequence charts [OMG07] and used for exemplary interaction of components.

AutoFocus allows Java code generation [HS97] and simulation. Consistency between exemplary runs from EET and STD specifications as well as behavioral refinements of specifications can be checked using the SMV [BCM<sup>+</sup>92] or  $\mu$ -cke [Bie97] model checkers directly from AutoFocus [HSE97]. Multiple case studies have been conducted, which include: a traffic lights case study [HMS<sup>+</sup>98], a home shopping system [Hin98], a tamagotchi [KvKPS99], and the specification of an electronic purse in e-commerce [JW01]. A prominent application of AutoFocus is in the area of automotive software within the AutoMoDe (Automotive Model-based Development) project [BRS05, ZBF<sup>+</sup>05, BBR<sup>+</sup>07] and related quality assurance tasks [JRT08, Tra09].

While the AutoFocus tool suite is a very enhanced and innovative tooling infrastructure, we see quite a number of open practical issues both on the verification side as well as on the model management side. While the theory is rather well elaborated, its practical use is still limited, as comfortable tools are not easy to develop. This, however, is a generally visible situation, as it remains a considerable effort to develop convenient tooling infrastructure for defining, analyzing and managing models and specifications, even though they are based on a sound and stable theory.

## 6 Conclusion

Specification and modeling of interactive distributed systems is an ongoing challenge. We have presented some background of approaches based on dataflow networks and stream processing. Streams can be of many kinds ranging from discrete untimed streams to super dense timed streams up to continuous functions. On examples we

have shown that some of these kinds of streams although of different structure and intuition can be related by abstraction and refinement or even be embedded one into the other.

A larger part of the FOCUS method uses stream processing functions to describe behavior. We have introduced a basic set of manipulator operations for streams, stream bundles, stream processing functions, state-based functions (automata) and finally specifications. We believe that this set of concepts together with their operators, e.g., for composition and refinement, is suited well for the implementation of a proof theory in Isabelle/HOL and HOLCF.

There are quite a number of implementations of assisting and automating parts of the FOCUS method in the context of formal verification. As stated, a good theory needs a practical tooling infrastructure to become amenable. Tooling however is generally a big issue, why the software engineering community is still waiting for broadly used, comfortable modeling tools that allow to specify, analyze and manage models (beyond mere code generation). The most elaborated stream-based tool currently is AutoFocus.

In summary, the FOCUS theory and method is well elaborated, has a variety of theoretical sidelines for several domains as well as many applications in case studies and industrial evaluations. This can to a large extent be credited to its intellectual father Manfred Broy, his many own research papers on this theme and his great supervision to numerous PhD members of his group. Which is reason enough, for the following.

## 7 Laudation for Manfred Broy (by the second author)

Manfred is legendary. I did have the chance to be a member of his group almost from his come-back to Munich in 1989 to my leave in 2003, so I think I know him a little. The first time I met Manfred, I was still a student. I worked on my diploma thesis using the newly explored Spectrum approach on algebraic specification, with loose semantics, quite an elaborated type system and some other nice extras. It was a great project with lots of interesting results. At least in those days, there was a rumor that one had to be able to play Schaffkopf, a pretty complex Bavarian game, to be hired in his group. But I made it and while we still play occasionally, this rule seems to have been relaxed a while ago.

Manfred has a vision: Streams are the optimal technology to model the world. We have explored their abilities and how the various versions of streams fit together in the FOCUS approach, that Manfred invented and together with many of his members brought to a handsome, well explored and sound theory. While there are still many things to explore, e.g., how to bring streams to the cloud or to the sky and how to model ad hoc, mobile embedded systems best, this theory is well settled and has a major impact among the formal community.

Manfred has a second vision: let the FOCUS theory and method become practical. For this purpose, he started tool development projects, AutoFocus 1, 2 and 3 being the most prominent. These tooling infrastructures are on their way, but as we now know, development of comfortable tools is generally very labor intensive. So Manfred and his group as well as former members of his group still work hard to elaborate this second vision. That is why, Manfred and his group in parallel also inject FOCUS ideas into other commercial tooling infrastructures, such as Automotive or UML tools. For

that purpose he is quite active in publishing, giving presentations around the world and acquiring projects.

Manfred has a very integrative ability to bring people together and let them explore new, innovative and typically brilliant ideas, while at the same time he inspires people with ideas towards his visions. That is why many research results come from his group and why many of these results quite nicely fit together or complement each other.

He and his group have a lot of strong links to the international and national community, which allow them to further explore and discuss the practicality of research issues – let them be in formal methods, their application, or methodical issues of Software Engineering in general.

Manfred is successful in almost anything that he starts. Let it be innovative projects, development of a new standard method for Germany’s software development projects, or deep theoretical issues. And he is a very successful Schaffkopf player too.

So we wish him a seamless stream of practical and research questions to solve, disseminate and teach to his students, but most of all we wish his vision on the use of the stream-based approach, known as FOCUS to become widely used.

**Acknowledgments** We thank Christian Langauer and Shahar Maoz for proofreading a draft of this paper.

## References

- [Abr83] Samson Abramsky. On Semantic Foundations for Applicative Multiprogramming. In *Proceedings of the 10th Colloquium on Automata, Languages and Programming*, pages 1–14, London, UK, 1983. Springer-Verlag.
- [AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 322–335, 1990.
- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [AL91] Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. In *REX Workshop*, pages 1–27, 1991.
- [AR02] Farhad Arbab and Jan J. M. M. Rutten. A coinductive calculus of component connectors. In *WADT*, pages 34–55, 2002.
- [Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.
- [BA81] J. Dean Brock and William B. Ackerman. Scenarios: A model of non-determinate computation. In *Proceedings of the International Colloquium on Formalization of Programming Concepts*, pages 252–259, London, UK, 1981. Springer-Verlag.
- [BBR<sup>+</sup>07] Andreas Bauer, Manfred Broy, Jan Romberg, Bernhard Schätz, Peter Braun, Ulrich Freund, Nuria Mata, Robert Sandner, Pierre Mai, and Dirk Ziegenbein. Das automode-projekt. *Computer Science - Research and Development*, 22(1):45–57, 2007.
- [BCM<sup>+</sup>92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking:  $10^{20}$  States and beyond. *Information and Computation*, 98(2):142 – 170, 1992.
- [BCR07] Manfred Broy, María Victoria Cengarle, and Bernhard Rumpe. Semantics of UML – Towards a System Model for UML: The State Machine Model. Technical Report TUM-I0711, Institut für Informatik, Technische Universität München, February 2007.

- [BDD<sup>+</sup>92] Manfred Broy, Frank Dederich, Claus Dendorfer, Max Fuchs, Thomas Gritzner, and Rainer Weber. The Design of Distributed Systems - An Introduction to FOCUS. Technical report, TUM-I9202, SFB-Bericht Nr. 342/2-2/92 A, 1992.
- [BDDW91] Manfred Broy, Frank Dederichs, Claus Dendorfer, and Rainer Weber. Characterizing the Behaviour of Reactive Systems by Trace Sets. SFB-Bericht 324/2/91, Technische Universität München, jan 1991.
- [BHS99] Manfred Broy, Franz Huber, and Bernhard Schätz. AutoFocus – Ein Werkzeug-prototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, 14(3):121–134, 1999.
- [Bie97] Armin Biere.  $\mu$ cke - Efficient  $\mu$ -Calculus Model Checking. In *CAV*, pages 468–471, 1997.
- [Bro81] Manfred Broy. A Fixed Point Approach to Applicative Multiprogramming. In *Lecture notes for the International Summer School on Theoretical Foundations of Programming Methodology*, 1981.
- [Bro86] Manfred Broy. A theory for nondeterminism, parallelism, communication, and concurrency. *Theoretical Computer Science*, 45:1 – 61, 1986.
- [Bro88] Manfred Broy. Nondeterministic data flow programs: How to avoid the merge anomaly. *Science of Computer Programming*, 10(1):65 – 85, 1988.
- [Bro93] Manfred Broy. *(Inter-)Action Refinement: The Easy Way*, volume 118 of *Series F: Computer and System Sciences*. Springer NATO ASI Series, 1993.
- [Bro95] Manfred Broy. Mathematical system models as a basis of software engineering. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 292–306. Springer Berlin / Heidelberg, 1995.
- [Bro97] Manfred Broy. Refinement of time. *Transformation-Based Reactive Systems Development*, Volume 1231/1997:44–63, 1997.
- [Bro03] Manfred Broy. Multi-view Modeling of Software Systems. In *Formal Methods at the Crossroads: From Panacea to Foundational Support (LNCS 2757)*, 2003.
- [Bro05] Manfred Broy. *Service-oriented Systems Engineering: Specification and Design of Services and Layered Architectures–The Janus-Approach*, pages 47–81. Springer, 2005.
- [BRS05] Andreas Bauer, Jan Romberg, and Bernhard Schätz. Integrierte Entwicklung von Automotive-Software mit AutoFocus. *Informatik - Forschung und Entwicklung*, 19:194–205, 2005. 10.1007/s00450-005-0187-7.
- [BRSS97] Manfred Broy, Franz Regensburger, Bernhard Schätz, and Katharina Spies. The Steamboiler Specification - A Case Study in Focus. Technical Report TUM-I9714, Technische Universität München, 1997.
- [BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement*. Springer Verlag Heidelberg, 2001.
- [Bur75] William H. Burge. Stream processing functions. *IBM journal of research and development*, 19(1):12–25, 1975.
- [dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, 2001.
- [Den95] Claus Dendorfer. *Methodik funktionaler Systementwicklung*. PhD thesis, Technische Universität München, 1995.
- [Die95] Volker Diekert. *The Book of Traces*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.
- [DS89] Peter Dijkstra and Herbert P. Sander. A Functional Programming Approach to the Specification and Verification of Concurrent Systems. *Formal Asp. Comput.*, 1(4):303–319, 1989.
- [FP95] Max Fuchs and Jan Philipps. Formal Development of a Production Cell in Focus – A Case Study. *Formal Development of Reactive Systems*, (LNCS 891), 1995.
- [Fuc94] Maximilian Fuchs. *Technologieabhängigkeit von Spezifikationen digitaler Hardware*. PhD thesis, Technische Universität München, 1994.
- [GGR06] Boris Gajanovic, Hans Grönniger, and Bernhard Rumpe. *From MDD Concepts*

- to Experiments and Illustrations*, chapter Model Driven Testing of Time Sensitive Distributed Systems, pages 131–148. ISTE Ltd., 2006.
- [GKRB96] Radu Grosu, Cornel Klein, Bernhard Rumpe, and Manfred Broy. State Transition Diagrams. Technical Report TUM-I9630, Technische Universität München, 1996.
- [GR95] Radu Grosu and Bernhard Rumpe. Concurrent timed port automata. Technical Report TUM-I9533, Technische Universität München, 1995.
- [GR06] Boris Gajanovic and Bernhard Rumpe. Isabelle/HOL-Umsetzung strombasierter Definitionen zur Verifikation von verteilten, asynchron kommunizierenden Systemen. Informatik-Bericht 2006-03, Technische Universität Braunschweig, Carl-Friedrich-Gauss-Fakultät für Mathematik und Informatik, 2006.
- [GR07] Borislav Gajanovic and Bernhard Rumpe. Alice: An advanced logic for interactive component engineering. In *4th International Verification Workshop (Verify'07)*, Bremen, 2007.
- [GSB98] Radu Grosu, Thomas Stauner, and Manfred Broy. A Modular Visual Model for Hybrid Systems. In Anders Ravn and Hans Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1486 of *Lecture Notes in Computer Science*, pages 471–471. Springer Berlin / Heidelberg, 1998.
- [Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
- [Hin98] Ursula Hinkel. Home Shopping - Die Spezifikation einer Kommunikationsanwendung in Focus. Technical Report TUM-I9808, Technische Universität München, 1998.
- [HK02] David Harel and Orna Kupferman. On object systems and behavioral inheritance. *IEEE Trans. Softw. Eng.*, 28(9):889–903, 2002.
- [HMS<sup>+</sup>98] Franz Huber, Sascha Molterer, Bernhard Schätz, Oscar Slotosch, and Alexander Vilbig. Traffic Lights - An AutoFocus Case Study. pages 282 – 294, 1998.
- [HP85] David Harel and Amir Pnueli. *On the development of reactive systems*, pages 477–498. Springer-Verlag New York, Inc., New York, NY, USA, 1985.
- [HS97] Franz Huber and Bernhard Schätz. Rapid Prototyping with AutoFocus. In A. Wolisz, I. Schieferdecker, and A. Rennoch, editors, *Formale Beschreibungstechniken für verteilte Systeme, GI/ITG Fachgespräch*, pages 343 – 352. GMD Verlag (St. Augustin), 1997.
- [HSE97] Franz Huber, Bernhard Schätz, and Geralf Einert. Consistent graphical specification of distributed systems. *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, pages 122–141, 1997.
- [HSS96] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. Autofocus — a tool for distributed systems specification. In Bengt Jonsson and Joachim Parrow, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *Lecture Notes in Computer Science*, pages 467–470. Springer Berlin / Heidelberg, 1996.
- [JRT08] Jan Jürjens, Daniel Reiss, and David Trachtenherz. Model-Based Quality Assurance of Automotive Software. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings*, volume 5301 of *LNCS*, pages 858–873. Springer, 2008.
- [JW01] Jan Jürjens and Guido Wimmel. Security Modelling for Electronic Commerce: The Common Electronic Purse Specifications. In Beat Schmid, Katarina Stanoevska-Slabeva, and Volker Tschammer, editors, *Towards the E-Society. Proceedings of 1st IFIP International Conference on E-Commerce, E-Business and E-Government*, pages 489 – 506. Kluwer Academic Publishers, 2001.
- [Kah74] Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475, New York, NY, 1974. North-Holland.
- [Kel78] R.M. Keller. Denotational models for parallel programs with indeterminate op-

- erators. *Proc. Formal Description of Programming Concepts*, pages 337–366, 1978.
- [KM77] Gilles Kahn and David B. MacQueen. Coroutines and networks of parallel processes. *Proceedings of IFIP Congress*, 77(7):993–998, 1977.
- [KP85] Robert M. Keller and Prakash Panangaden. Semantics of networks containing indeterminate operators. In *Seminar on Concurrency, Carnegie-Mellon University*, pages 479–496, London, UK, 1985. Springer-Verlag.
- [KPR97] Cornel Klein, Christian Prehofer, and Bernhard Rumpe. Feature specification and refinement with state transition diagrams. In Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors, *Feature Interactions in Telecommunications Networks IV, June 17-19, 1997, Montréal, Canada*, pages 284–297. IOS Press, 1997.
- [KvKPS99] Erik Kamsties, Antje von Knethen, Jan Philipps, and Bernhard Schätz. Eine vergleichende fallstudie von acht case-werkzeugen für formale und semi-formale beschreibungstechniken. In *FBT*, pages 103–112, 1999.
- [Lee09] Edward A. Lee. Computing needs time. *Communications of the ACM*, 52(5):70–79, May 2009.
- [LML06] Xiaojun Liu, Eleftherios Matsikoudis, and Edward A. Lee. Modeling timed concurrent systems. In *CONCUR*, pages 1–15, 2006.
- [Lot96] Volkmar Lotz. Threat scenarios as a means to formally develop secure systems. pages 242–265, 1996.
- [LSBB92] Reinhold Letz, Johann Schumann, Stefan Bayerl, and Wolfgang Bibel. Setheo: A high-performance theorem prover. In *Journal of Automated Reasoning (JAR)*, volume 8, pages 183–212. Springer, 1992.
- [LT89] Nancy A. Lynch and Mark R. Tuttle. An Introduction to Input/Output Automata. *CWI Quarterly*, 2:219 – 246, 1989.
- [MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *REX Workshop*, pages 447–484, 1991.
- [MNvOS99] Olaf Müller, Tobias Nipkow, David von Oheimb, and Oscar Slotosch. Holcf = hol + lcf. *Journal of Functional Programming*, 9(02):191–223, 1999.
- [MP93] Zohar Manna and Amir Pnueli. Verifying Hybrid Systems. In Robert Grossman, Anil Nerode, Anders Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 4–35. Springer Berlin / Heidelberg, 1993.
- [MS96] Olaf Müller and Peter Scholz. Specification of Real-Time and Hybrid Systems in FOCUS. Technical Report TUM-I9627, Technische Universität München, 1996.
- [MS97] Olaf Müller and Peter Scholz. Functional specification of real-time and hybrid systems. In *HART '97: Proceedings of the International Workshop on Hybrid and Real-Time Systems*, pages 273–285, London, UK, 1997. Springer-Verlag.
- [NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer, 2002.
- [OMG07] Object Management Group. Unified Modeling Language: Superstructure Version 2.1.2 (07-11-02), 2007. <http://www.omg.org/docs/formal/07-11-02.pdf>.
- [PR94] Barbara Paech and Bernhard Rumpe. A new Concept of Refinement used for Behaviour Modelling with Automata. In *FME'94, Formal Methods Europe, Symposium '94*, LNCS 873. Springer-Verlag, Berlin, October 1994.
- [PR97] Jan Philipps and Bernhard Rumpe. Refinement of Information Flow Architectures. In *Proceedings of Formal Engineering Methods*, 1997.
- [PR99] Jan Philipps and Bernhard Rumpe. Refinement of Pipe And Filter Architectures. In *FM'99, LNCS 1708*, pages 96–115, 1999.
- [RK96] Bernhard Rumpe and Cornel Klein. Automata Describing Object Behavior. In *Object-Oriented Behavioral Specifications*, pages 265–286, 1996.
- [Rum96] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. Doktorarbeit, Technische Universität München, 1996.
- [Rum97] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter*

- Systeme. In *Ausgezeichnete Informatikdissertationen 1997*, pages 118–134. Teubner Stuttgart, 1997.
- [SB99] Johann Schumann and Max Breitling. Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern - Fallstudie -. Technical Report TUM-I9904, Technische Universität München, 1999.
- [Sch98] Peter Scholz. *Design of reactive systems and their distributed implementation with statecharts*. PhD thesis, Technische Universität München, 1998.
- [SDW93] Ketil Stølen, Frank Dederichs, and Rainer Weber. Assumption/Commitment Rules for Networks of Asynchronously Communicating Agents. Technical Report TUM-I9303, Technische Universität München, 1993.
- [Slo97] Oscar Slotosch. *Refinements in HOLCF: Implementation of interactive systems*. PhD thesis, Technische Universität München, 1997.
- [SM97] Robert Sandner and Olaf Müller. Theorem prover support for the refinement of stream processing functions. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 351–365, 1997.
- [Spi07] Maria Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, Technische Universität München, 2007.
- [SRS99] Thomas Stauner, Bernhard Rumpe, and Peter Scholz. Hybrid System Model. Technical Report TUM-I9903, Technische Universität München, 1999.
- [SS95] Bernhard Schätz and Katharina Spies. Formale Syntax zur logischen Kernsprache der FOCUS-Entwicklungsmethodik. Technical Report TUM-I9529, Technische Universität München, 1995.
- [Ste97] Robert Stephens. A survey of stream processing. *Acta Informatica*, 34(7):491–541, 1997.
- [Stø96] Ketil Stølen. Using relations on streams to solve the RPC-memory specification problem. *Formal Systems Specification*, pages 477–520, 1996.
- [Tra09] David Trachtenherz. *Eigenschaftsorientierte Beschreibung der logischen Architektur eingebetteter Systeme*. PhD thesis, Institut für Informatik, Technische Universität München, 2009.
- [Wil06] Doris Wild. *AutoFocus 2 - Das Bilderbuch*. Technische Universität München, 2006.
- [ZBF<sup>+</sup>05] Dirk Ziegenbein, Peter Braun, Ulrich Freund, Andreas Bauer, Jan Romberg, and Bernhard Schätz. AutoMoDe - Model-Based Development of Automotive Software. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 171–177, Washington, DC, USA, 2005. IEEE Computer Society.