A Model of Dynamic Systems

Manfred Broy

Institut für Informatik, Technische Universität München
80290 München Germany
broy@in.tum.de
http://wwwbroy.informatik.tu-muenchen.de

Abstract. We introduce a model describing discrete dynamic distributed systems. These are systems where their set of connections to the systems in their context captured by their syntactic interfaces as well as the set of their subsystems, and their set of internal connections in their architectures between their subsystems change dynamically over time. To provide such a model we generalize the static system model of Focus (cf. [8]) in terms of their system interfaces and their interface behavior, their system architectures, and their system models in terms of state machines to model dynamic systems. We deal with concepts of causality, composition, abstraction, and system specification for dynamic systems. We analyze properties of dynamic systems and discuss how well the model captures general notions of system dynamics. Finally, we introduce the concept of system classes and their instantiation, which introduces an additional concept of dynamicity.

Keywords: Dynamic Systems, Mobility, Instantiation

1 Introduction: Dynamic Systems

FOCUS (see [8]) is an approach to system modeling where the interfaces of systems and their behaviors are described in terms of streams of interactions exchanged via their input and output channels. Architectures of systems are modeled by sets of subsystems with their interface behavior and their mutual connections by channels. Implementations of systems are described in terms of state machines. FOCUS follows the idea of distributed concurrent data flow.

Systems operate in a timeframe described by a sequence of time intervals called time slots. A time slot behavior of a system is given by a syntactic interface consisting of a set of input channels I and a set of output channels O and a mapping that maps valuations of the input channels to sets of valuations of the output channels. A syntactic interface is defined by a set of input channels and a set of output channels together with their types. In FOCUS this syntactic interface does not change over time. This notion of a syntactic interface is static in FOCUS.

Behaviors of systems then are mappings that associate with every time interval, where time intervals are represented by the natural numbers, a time slot behavior. In a behavior we deal with a set of channels. This is the set of all channels that occur in

S. Bensalem et al. (Eds.): FPS 2014 (Sifakis Festschrift), LNCS 8415, pp. 39–53, 2014.

one time slot behavior. These sets of channels can be infinite, in principle. We assume, however, that in dynamic systems in each time slot the set of channels is finite. For simplicity we do not allow channels occurring both as input and output channels for one component.

An input channel is called *static* for a system if the channel occurs in that system as an input channel in every time slot. In analogy, an output channel is called *static,* if it occurs as an output channel in every time slot.

Composition is defined in FOCUS by composition of the interface behaviors in the time slots where we require that whenever in a composition there is a feedback loop, then the output channel in that time slot is strongly causal. Otherwise composition with feedback could not be properly defined. A simple way to achieve that is to require that output channels are always strongly causal or that feedback does only occur for strongly causal output channels.

In FOCUS, system interfaces are static in the sense that the sets of input and output channels are invariant over the lifetime of a system. Therefore the syntactic interface in FOCUS is static.

In FOCUS besides the notion of an interface, state machine, and architectures are worked out as elements of system models. Both refer to interfaces and both are static in the sense that they support static interfaces. As a consequence, interfaces of architectures are static and thus the architectures are static in the sense that they only support static architectures where the connections between their subsystems do not change over the lifetime of the systems.

In the following we introduce a new, simple, but powerful model for dynamic systems, which are systems where their number of channels (the syntactic interface), and their connections between the subsystems are not necessarily static but may change over time. Basically, we do that by assigning a possibly modified syntactic interface to systems for every time slot of the system.

Causality and strong causality of behaviors is defined as usual.

In the following we introduce such a model for dynamic systems. We first introduce the basics for modeling interactive systems such as streams and histories. Then we introduce the concept of a dynamic interface behavior and finally the idea of dynamics for specifications, state machines, and architectures as well as classes of systems introducing the concept of instantiation.

2 The Dynamic System Model

In this section we introduce the dynamic system model.

2.1 System Propaedeutic

Our approach uses a specific notion of discrete system with the following characteristics and principles.

- A discrete *system* has a well-defined boundary (its “scope”) that determines its *interface*.

  - Everything outside the system boundary is called the system's *environment*. Those parts of the environment that are relevant for the system are called the system's *context*. Actors in the context that interact with the system such as users, neighbored systems, or sensor and actuators connected to the physical environment are called its *operational context*.
  - By a system's interface it is indicated by which steps the system interacts with its operational context. The syntactic interface defines the set of actions that can be performed in interaction with a system over its boundary. In our case, syntactic interfaces are defined by the set of input and output channels together with their types. The input channels and the types determine the input actions for a system while the output channels and their types determine the output actions for a system.
  - We distinguish between *syntactic interface*, also called *static interface*, which describes the set of input and output actions, which can take place over the system boundary, and the *interface behavior* (also called *dynamical interface*), which describes the system's *functionality* in terms of the input and output actions; the interface behavior is captured by the causal relationship between streams of actions captured in input and output *histories*. This way we define a logical behavior as well as a probabilistic behavior for systems.
  - The logical interface behavior of systems is described by logical expressions, called *interface assertions*, by *state machines,* or it can be further decomposed into *architectures*.

- A system has an *internal structure* and some internal *behavior* ("glass box view"). This structure is described by
  - its state space with state transitions and/or
  - its decomposition in sub-systems forming its architecture in case the system is decomposed into a number of subsystems, which interact and also provide the interaction with the system's operational context. The state machine and the architecture associated with a system are called its state view and its structural or architectural view, respectively.
- Complementary, the behaviors of systems can be described by sets of *traces*, which are sets of scenarios of input and output behavior of systems. We distinguish between finite and infinite scenarios.
- Moreover, systems operate in real time. In our case we use discrete time, which seems, in particular, adequate for discrete systems.
- Systems interact with their operational context and sub-systems operate concurrently within the system's architectures.

This gives a highly abstract and at the same time quite comprehensive model of systems. This model is formalized in the following by one specific modeling theory.

This system propaedeutic leads to following notions of a dynamic system

- *dynamic interface behavior*: the syntactic interface changes over time
- *dynamic architecture*: the syntactic interface changes over time
- *dynamic state space* and state transition

In the following we introduce a very compact model that addresses all three aspects of dynamicity.

2.2 Interface Behavior Model

The key concepts for modeling system behavior are streams and communication channels to represent interaction and connections.

Channels and Histories. For the alternative model for dynamic systems we use the following sets

M	universe of all data elements such as messages or values of state attributes,
TYPE	the set of (data) types (each type T in TYPE is a subset of M),
C	the set of typed channels.

The type of the channels in the channel set C is determined by a channel type assignment:

$$type: C \to TYPE$$

By [type(c)] we denote the set of elements associated with type(c). We assume that the types of channels are static, which means they do not change over the lifetime of a channel.

An *interaction pattern* for the set C of typed channels is given by a partial function

$$p: C \to M^*$$

where $Dom(p) \subseteq C$ denotes the set of channels c for which p(c) is defined; furthermore we assume $p(c) \in [type(c)^*]$. An interaction pattern represents the sequences of messages exchanged over the channel set Dom(C) within a time slot.

The set of interaction patterns is denoted by $C^{\rightarrow}$ which is defined by the equation

$$C^{\rightarrow} = \{p: C \to M^*: p \text{ partial } \}$$

We say that the channel c is *active* in pattern p if $c \in Dom(p)$.

Note the subtle difference between the case where $p(c) = \langle\rangle$, which represents the situation where channel c is active but no messages are transmitted, and $c \notin Dom(p)$, which represents the case where channel c is not active and thus there is no specified communication via channel c in that time slot. Then c is actually at that time slot considered not being present as a channel.

A *dynamic channel valuation* (also called a *dynamic history*) for a set C of channels is given by a function:

$$x: \mathbb{N}\backslash\{0\} \to C^{\rightarrow}$$

It specifies for each time slot an interaction pattern. We denote the set of all dynamic valuations of the channels in C by

$$C^{\Rightarrow}$$

We say that channel c is *present* (or *active*) in the history $x \subseteq C^{\Rightarrow}$ at time t if $c \in \mathrm{Dom}(x.t)$. For $x \in C^{\Rightarrow}$ we write x.t.c with time $t \in \mathbb{N}\backslash\{0\}$ and channel $c \in C$ to denote the sequence (x(t))(c) transmitted on channel c in history x in the time slot t provided $c \in \mathrm{Dom}(x.t)$; furthermore we write x(c) for the partial mapping $\mathbb{N}\backslash\{0\} \to$ [type(c)*] where (x(c))(t) is equal to (x(t))(c).

Given the dynamic history $x \in C^{\Rightarrow}$ we denote for given time slot $t \in \mathbb{N}$ by $x{\downarrow}t$ the restriction of the history x to the time slots {1, …, t}:

$$x{\downarrow}t : \{1, \ldots, t\} \to C^{\rightarrow}$$

where

$$(x{\downarrow}t).(k) = x(k) \Leftarrow 1 \leq k \leq t$$

$x{\downarrow}t$ is also called the finite "partial" history of history x till time slot t. A history $x \in C^{\Rightarrow}$ is called *static* if for all times t, t' $\in \mathbb{N}\backslash\{0\}$ Dom(x(t)) = Dom(x(t')). A history $x \in C^{\Rightarrow}$ is called *dynamically finite* if Dom(x(t)) is finite for all $t \in \mathbb{N}\backslash\{0\}$.

Merging Histories. The parallel composition of dynamic channel histories is specified as follows. Given two sets X, Y of channels with consistent types and two dynamic histories $x \in X^{\Rightarrow}$, $y \in Y^{\Rightarrow}$ we define the set of dynamic histories $x \oplus y \subseteq (X \cup Y)^{\Rightarrow}$ composed from histories x and y by the equation

$$\begin{aligned} x \oplus y = \{z \in (X \cup Y)^{\Rightarrow} : \forall\, t \in \mathbb{N}\backslash\{0\} &: \mathrm{Dom}(z.t) = \mathrm{Dom}(x.t) \cup \mathrm{Dom}(y.t) \\ \wedge\ \forall\, c \in (X \cup Y) &: c \in \mathrm{Dom}(z.t) \Rightarrow \\ & (z.t.c = x.t.c \wedge c \notin \mathrm{Dom}(y.t)) \\ \vee\ & (z.t.c = y.t.c \wedge c \notin \mathrm{Dom}(x.t)) \\ \vee\ & (z.t.c \in \mathrm{merge}(x.t.c, y.t.c) \wedge \\ & \qquad c \in \mathrm{Dom}(x.t) \cap \mathrm{Dom}(y.t)))\} \end{aligned}$$

The set-valued function merge yields the set of all interleavings of two finite sequences. It is easily specified as follows:

$$\mathrm{merge}(s_1, s_2) = \{s \in M^* : \exists\, e \in \{1, 2\}^* : \mathrm{proj}(s, e, 1) = s_1 \wedge \mathrm{proj}(s, e, 2) = s_2\}$$

where

$$\mathrm{proj} : M^* \times \mathbb{N}^* \times \mathbb{N} \to M^*$$

is specified by the following equations:

$$\begin{aligned} &\mathrm{proj}(\langle m \rangle \hat{}\, s, \langle k \rangle \hat{}\, e, k) = \langle m \rangle \hat{}\, \mathrm{proj}(s, e, k), \\ &\mathrm{proj}(\langle m \rangle \hat{}\, s, \langle k \rangle \hat{}\, e, k') = \mathrm{proj}(s, e, k) \Leftarrow k \neq k' \\ &\mathrm{proj}(\langle\rangle, s, k) = \mathrm{proj}(s, \langle\rangle, k) = \langle\rangle \end{aligned}$$

As a result $x \oplus y$ denotes the history in which or each channel $c \in (X \cup Y)$ the sequence $x \oplus y.t.c$ is the result of merging the sequences x.t.c and y.t.c provided both are defined, otherwise it is equal to x.t.c or to y.t.c depending on which of these is defined and undefined if both are undefined. For $C' \subseteq C$ and $x \in C^{\Rightarrow}$ we denote by $x|C'$ the

restriction of history x to channels in C' which is the history $z \in C^{\Rightarrow}$ where z.t.c = x.t.c for $c \in C'$ and $t \in \mathbb{N}\backslash\{0\}$ holds.

Interface Behavior. Given two sets I and O of typed channels, we denote the syntactic interface of a dynamic system by (I▸O). A nondeterministic (or under-specified) component behavior (let I and O be sets of typed channels) is represented by the function

$$F: I^{\Rightarrow} \to \wp(O^{\Rightarrow})$$

This function models the behavior of a dynamic component. At every time $t' \in \mathbb{N}\backslash\{0\}$ for given input history x the set Dom(x(t)) denotes the set of channels active at time t as input channels and for $y \in F(x)$ Dom(y(t)) denotes the channels active at time t as output channels.

Given input history $x \in I^{\Rightarrow}$ in each time slot t a sub-interface (I'▸O') with $I' \subseteq I$ and $O' \subseteq O$ of channels is active. It is specified by the sets

$$I' = \mathrm{Dom}(x.t)$$
$$O' = \mathrm{Dom}((y) \text{ where } y \in F(x).(t)$$

(I'▸O') is called the *active syntactic interface at time slot* t for input x and output y. If for all t and all input histories that are static the syntactic interfaces are identical we call the system *static*.

Definition. Causal Interface Behavior
For a mapping

$$F: I^{\Rightarrow} \to \wp(O^{\Rightarrow})$$

we define the set

$$\mathrm{dom}(F) = \{x: F(x) \neq \emptyset\}$$

called the *domain* of F. F is called *total*, if $\mathrm{dom}(F) = I^{\Rightarrow}$, otherwise F is called *partial*.

The mapping F is called *causal*, if (for all $t \in \mathbb{N}$ and all input histories x, $z \in I^{\Rightarrow}$):

$$x, z \in \mathrm{dom}(F) \wedge x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t: y \in F(x)\} = \{y{\downarrow}t: y \in F(z)\}$$

F is called *strongly causal*, if (for all $t \in \mathbb{N}$ and all input histories x, $z \in I^{\Rightarrow}$):

$$x, z \in \mathrm{dom}(F) \wedge x{\downarrow}t = z{\downarrow}t \Rightarrow \{y{\downarrow}t+1: y \in F(x)\} = \{y{\downarrow}t+1: y \in F(z)\} \quad \square$$

Causality (for an extended discussion see [8]) indicates a consistent time flow between input and output histories in the following sense: in a causal mapping input messages received at time t may influence future output only after time t; this output is given by messages communicated via output channels at times $\geq$ t (in the case of strong causality at times > t, which indicates that there is a delay of at least one time step before input has any effect on output).

Definition. I/O-Behavior
A causal mapping $F: I^{\Rightarrow} \to \wp(O^{\Rightarrow})$ is called a *dynamic I/O-behavior*. By DIF[I▸O] we denote the set of all (total and partial) dynamic interface behaviors called I/O-behaviors with syntactic interface (I▸O) and by DIF the set of all I/O-behaviors. $\square$

Interface behaviors model system functionality. For systems we assume that their interface behavior is total. Behaviors F may be deterministic (in this case, the set F(x) of output histories has at most one element for each input history x) or nondeterministic (which allows us to model under-specification). For simplicity we assume throughout the paper that the sets of input and output channels of components are disjoint.

Hiding Channels. Given $F \in [I \blacktriangleright O]$ we hide a channel $c \in I \cup O$ leading to a system with Interface behavior $F \backslash c \in [I' \blacktriangleright O']$ where $I' = I \backslash \{c\}$ and $O' = O \backslash \{c\}$ where for $x' \in I'^{\Rightarrow}$ we define the behavior of $F \backslash c$ by

$(F \backslash c)(x') = F(x) | O'$ where $x \in I^{\Rightarrow}$ is defined as follows

$x.t.c = \langle\rangle$ for all $t \in \mathbb{N}$ and $x(c') = x'(c')$ for $c' \in I'$

Hiding means that we close the channel c to the outside world. F' cannot receive input on c nor produce output on c.

2.3 Composition of Dynamic Systems

In this section we describe the composition of systems in terms of their interface behavior. We show how to calculate the interface behavior of a composed system from the interface behaviors of its components.

The composition of two systems

$$F_1: I_1^{\Rightarrow} \rightarrow \wp(O_1^{\Rightarrow}) \text{ and } F_2: I_2^{\Rightarrow} \rightarrow \wp(O_2^{\Rightarrow})$$

yields an I/O-behavior $(F_1 \otimes F_2)$ with syntactic interface $(I \blacktriangleright O)$ where $O = O_1 \cup O_2$ and $I = (I_1 \cup I_2) \backslash (O_1 \cup O_2)$ that are strongly causal is specified by the following formula:

$$(F_1 \otimes F_2).x = \{y | (O_1 \cup O_2): \exists\, y_1, y_2: y = x \oplus y_1 \oplus y_2 \wedge y_1 \in F_1(y|I_1) \wedge y_2 \in F_2(y|I_2)\}$$

where $x \in ((I_1 \cup I_2) \backslash (O_1 \cup O_2))^{\Rightarrow}$, $y \in (O_1 \cup O_2 \cup I_1 \cup I_2)^{\Rightarrow}$. $(F_1 \otimes F_2)$ is strongly causal again.

The composition of systems is commutative:

$$F_1 \otimes F_2 = F_2 \otimes F_1$$

as well as associative:

$$(F_1 \otimes F_2) \otimes F_3 = F_1 \otimes (F_2 \otimes F_3)$$

The proof of these equations is straightforward.

This way we get a model for dynamic systems, which is more general than the static FOCUS model. It is concise and surprisingly better adapted to describing the dynamics of systems. The model is a direct extension of the FOCUS model of static systems.

2.4 Dynamic State Machines by State Transition Functions

A state space over a given space set V of typed attributes is a set of mappings

$$\sigma: V \rightarrow D$$

where D is the universe of all data and for all attributes $v \in V$ the value $\sigma(v)$ is of the type associated with attribute v. A dynamic state space over a set V of typed attributes is the set of partial mappings

$$\sigma : V \to D$$

where for all attributes in $Dom(\sigma)$ the value $\sigma(v)$ is of the type associated with attribute v.

Given V we denote by $\Sigma(V)$ the dynamic state space over V.

State machines with input and output describe system implementations in terms of states and state transitions. A state machine is defined by a state space and a state transition function.

Definition. Dynamic State Machine with Syntactic Interface ($I \blacktriangleright O$)
Given a set V of typed attributes, a state machine (Δ, Λ) with input and output according to the syntactic interface ($I \blacktriangleright O$) consists of a set $\Lambda \subseteq \Sigma(V)$ of initial states as well as of a nondeterministic state transition function

$$\Delta: (\Sigma(V) \times I^{\rightarrow}) \to \wp(\Sigma(V) \times O^{\rightarrow}) \qquad \square$$

For each state $\sigma \in \Sigma(V)$ and each valuation $a \in I^{\rightarrow}$ of the input channels in I by sequences of input messages every pair $(\sigma', b) \in \Delta(\sigma, a)$ defines a successor state σ' and a valuation $b \in O^{\rightarrow}$ of the output channels consisting of the sequences produced by the state transition in one time slot. In every step of the dynamic state machine the structure of the state space and the sets of active input and output channels may change – more precisely the set of active attributes, the set of active input, and the set of active output channels may change.

(Δ, Λ) is a *Mealy machine* with possibly infinite state space. If in every transition $(\sigma', b) \in \Delta(\sigma, a)$ the output b depends on the state σ only but never on the current input a, we speak of a *Moore machine*.

2.5 Dynamic Architectures

In this section, we describe how to form dynamic architectures by composing dynamic sub-systems, called the dynamic components of the architecture. Architectures are concepts to structure systems. Architectures contain precise descriptions for systems in terms of their sub-systems and how the composition of their sub-systems takes place. In other words, architectures are described by the sets of systems in their set of components together with mappings from input channels to output channels that describe internal communication. Architectures form a data flow network.

In the following we assume that each system used in an architecture as a component is identified by a unique identifier k. Let K be the set of identifiers for the components of an architecture.

Definition. Interpreted Architecture
An interpreted architecture (K, ξ, ψ) for set K of component names associates a syntactic dynamic interface $\xi(k) = (I_k \blacktriangleright O_k)$ with every component identifier $k \in K$ and dynamic interface behavior $\psi(k) \in \mathbb{IF}[I_k \blacktriangleright O_k]$, with every component identifier $k \in K$.

$\square$

An architecture can be specified by a syntactic architecture and an interface specification for each of its components.

The interface behavior of an architecture A = (K, ξ, ψ) is given by a mapping

$$F_A : \vec{I} \rightarrow \wp(\vec{O})$$

where (here all channels in I and O can be used both as internal channels and as external channels)

$$I = \cup \{I_k : k \in K\} \qquad O = \cup \{O_k : k \in K\}$$

and the interface behavior F_A of A can be calculated from the interface behaviors of the components:

$$F_A = \otimes \psi(k)$$

In this construction we do not hide "internal" channels, which are channels that lead inside the architecture from one sub-system to another one. This means that we can observe in behavior $F_A(x)$ the communication on internal channels at the system interface since internal channels are also output channels. Of course, we may also introduce more abstract concepts of composition hiding internal channels.

Given an input history x for an architecture we get a static architecture in every time slot. It is defined by a directed graph with the set K of components as its nodes. A channel c defines a connection at time slot t from component k to component k' if

$$c \in Dom((y|O_k).t)$$

and

$$c \in Dom((\psi(k)(y|I_{k'})).t)$$

where

$$y = F_A(x)$$

This way we characterize temporary connections. Given input $x \in \vec{I}$ and output $y \in \vec{O}$, where t is the set of all channels, let $Out_t(k)$ be the set of all channels that are active and lead from component k at time t and $In_t(k)$ be the set of all channels that are active and lead to component k at time t.

By construction a channel is, in general, a multi-connector. At each time slot it connects a set of components that have this channel as input channel with a set of components that have this channel as output channel. As defined components cannot have a channel both as input and output channel.

As a result a channel has in each step a number of active participants

- senders, that issue output to the channels but do not read input from the channel
- receivers, that consume messages as input from the channel

There are several ways the concept of a channel can be used. In a restricted application of the concept of a channel, a channel connects at each time exactly two subsystems, one sender and one receiver.

Given an input history $x \in I^{\rightarrow}$ for the architecture and some output we denote for $t \in \mathbb{N}$ by $Out_{t\cdot}(k)$ the set of active channels in $[(x \oplus y)|I_K](t)$; these are the channels active in the architecture at time t. A component $k \in K$ is said to *have a past* at time t (otherwise it is called *unborn* at time t) if

$$(\exists\, t' \in \mathbb{N} : t' \leq t \wedge \exists\, c: c \in Out_t(k) \cup In_t(k))$$

A component $k \in K$ is said to *have a future* at time t (otherwise it is called *dead* at time t) if

$$(\exists\, t' \in \mathbb{N} : t \leq t' \wedge \exists\, c: c \in Out_t(k) \cup In_{t\cdot}(k))$$

A component $k \in K$ is called *present* at time t if it has a past and a future (otherwise it is called *inactive* at time t), i.e. if

$$(\exists\, t' \in \mathbb{N} : t' \leq t \wedge \exists\, c: c \in Out_t(k) \cup In_{t\cdot}(k))$$
$$\wedge\, (\exists\, t' \in \mathbb{N} : t \leq t' \wedge \exists\, c: c \in Out_t(k) \cup In_t(k))$$

In other words, component k is involved in communications before and after time slot t.

In each time slot an architecture forms a directed graph, which we call its data flow graph. It consists of all components that are present and all their temporarily active channels. Note that in the graph we may find components that are not connected to any channel. They still may compute (in terms of internal state transitions) and may only later get connected to other components. This graph may change over time. This way a dynamic architecture is modeled. If all components are static, the architecture is static.

2.6 System Interface Behavior: Specification by Interface Assertions

The interface behavior of a system can be specified in a descriptive logical style using interface assertions.

Definition. Interface Assertion
Given a syntactic interface (I▶O) with a set I of typed input channels and a set O of typed output channels, an interface assertion is a logical formula with channel identifiers in I and O as free logical variables denoting streams of the respective types. ❑

We specify the behavior F_S for a system with name S with syntactic interface (I▶O) and an interface assertion P by a specification scheme:

S

in	I
out	O
P	

P is called interface assertion.

The scheme specifies the set of all strongly causal interface behaviors F_S of the system with name S which fulfill the formula

$$\forall x \in I^{\Rightarrow}, y \in O^{\Rightarrow}: y \in F_S(x) \Rightarrow P(x, y)$$

where P(x, y) results from P by replacing all channels c occurring in assertion P by streams x(c) or y(c), respectively.

It is more convenient to replace x(c) or y(c) simply by the channel identifier c, where we use the convention that for channels c that are both input and output channels we use c' for y(c) to distinguish x(c) from y(c).

Note that causality properties are implicitly assumed for each specification this way. If an interface behavior F_s that fulfills the interface assertion does not exist then the specification is called inconsistent.

In FOCUS interface assertions are formulas in predicate logic where channels denote streams. Since both channels and sub-systems may be inactive at certain time slots, we need special notations expressing that fact in interface assertions.

The interface assertions for dynamic systems may become more sophisticated due to the fact, that channels may be active or not. We specify a proposition

$$c@t$$

that yields true if channel c is active at time slot t. If we write c.t = … with a defined expression at the right hand side, this allows us to conclude c@t.

In addition, we allow communicating channels as messages (as in π–calculus, see [28]).

A typical example of a simple specification with type C = {c1, c2, c3} reads as follows (by c#s we donate the number of occurrences of c in sequence s)

S

in cha: C, x: Nat
out c1: Nat, c2: Nat, c3: Nat
∀ t ∈ IN:
∀ c ∈ C:
even(c#(cha↓t)) ⇔ c@(t+1)
c@t+1 ⇒ c.t+1 = x.t

This is an example of a component with three dynamic output channels that forward the input from channel x provided they are active. They get activated by sending their channel id on channel cha and deactivated by sending it once more.

3 Discussion

What we obtain by the introduced concepts is a model that models dynamic systems, however, for the price that syntactic interfaces are no longer static. In each time slot a dynamic system may feature a different syntactic interface.

In FOCUS generally composition of two specifications with interface assertions Q_1 and Q_2 is simply given by the formula

$$Q_1 \wedge Q_2$$

due to the requirement that in Focus their sets of output channels are disjoint. As long as this holds in every time slot, the same formula can be applied for dynamic systems.

Otherwise – and in our case, where we share output channels, a more sophisticated formula for composition is needed, where we have to distinguish between the output of Q_1 and Q_2 on shared channels and their merge. Let c be such a shared channel. We get the interface assertion in the case c is the only shared channel by the assertion (assume that channels c_1 and c_2 are not free in Q_1 or Q_2)

$$\exists\, c_1, c_2\colon (\exists\, c\colon Q_1 \wedge c_1 = c) \wedge (\exists\, c\colon Q_2 \wedge c_2 = c) \wedge c = c_1 \oplus c_2$$

which is equivalent to

$$\exists\, c_1, c_2\colon Q_1[c_1/c] \wedge Q_2[c_2/c] \wedge c = c_1 \oplus c_2$$

The generalization to a set of shared output channels is straightforward.

4 Parameterized Interfaces and Systems

Following the idea of [8] to consider indexed sets of channels and systems, which are called sheaves. By then we generalize the concept of channel and system identifier to families ("sheaves") by introducing indexes. This is an idea following concepts of object orientation where we replace object identifiers by index values and objects by system interface behavior.

Let K be an arbitrary set of index values (a simple choice would be $K \subseteq IN$); then an indexed channel is a channel name c of the type

$$c\colon [K]\ \mathrm{Data}$$

where Data is the channel type and K is a set of indices.

Actually then the indexed channel c describes a set of channels $\{c[k]\colon k \in K\}$. Such an indexed channel may be part of a signature. The same way we introduce indexed system names

$$F\colon [K]\ [I \blacktriangleright O]$$

Then by F we get a set of systems

$$\{F[k]\colon k \in K\}$$

with interfaces $[I[k] \blacktriangleright O[k]]$ for F[k] where each channel in I or O carries the same index k we may see F[K] as a family of systems with a family of input channels.

By this we can describe large networks with a huge number of components and channels and with many instances of the some behavior.

Based on the concept of indexed components we can go a further step into "object orientation" in the sense that we allow for component creation and instantiation. To do so we introduce a creator component for a parameterized set F: [k] $[I \blacktriangleright O]$ of components (where we assume that K is an infinite set of component identifiers). The instantiation component has two channels, an input channel on which requests are received and an output channel on which identifiers are send which then also are attributed to the created instance identifiers. If it receives a message to create a new component it creates one and returns its identifier on its output channel.

A way to avoid the problem of making sure that individual identifiers are received consistently in return to creation messages is to assume universal output channels for every component for creating new instances and individual input channels for receiving the identifiers of created instances. This leads to networks where every subsystem has an identifier and a number of standard channels indexed by that identifier.

5 Related Work and Alternative Models

Of course, the mathematical models we have introduced are not the only way to construct models and theories for dynamic mobile systems. Much work has been carried out towards the investigation of such models and theories. Only some of that is to be mentioned briefly in the following and related to our model.

Pioneering work goes back to Robin Milner in his work on the π-calculus (see [27], [28]) introducing a process algebra for the dynamics of channel connections. π-calculus captures the dynamics of systems by operational semantics in terms of rules that manipulate "process terms" representing systems. In the π-calculus channels can be passed as messages (which can be done in our model, too) and then used as channels by the respective receiver. This idea is captured by the rules of the structured operational semantics of π-calculus that are quite intuitive. However, the rules of π-calculus do not provide a denotational model but only operational models for dynamic systems. A denotational understanding, however, is essential for the constructions of techniques for the specification of dynamic systems.

Besides theoretical approaches to models of mobility and dynamics these ideas are widely used in practice in object-oriented programming languages, however, their time, distribution, communication are not explicitly addressed. In our model we easily capture object orientation by considering each object as a component (see [6] and [18, 19]). There are a number of approaches to give more specific treatments of the dynamics of object orientation (see [1], [21], [30]). Besides this, there is a lot of practical work in the area of object-oriented concepts to program dynamic systems (see also design patterns).

Another topic aims at concepts to formalize mobility and the idea of scopes and residences. This is modeled by scopes of bindings and information access in the ambient calculus (see [11]). The ambient calculus covers a special aspect of dynamic systems that is most relevant for distributed data basis and information systems. Such ideas could be added to our model by channel hiding.

Let us shortly discuss ambient calculus. What do we model by the ambient calculus? The ambient calculus introduces the following conceptual categories and terms to denote them:

- process,
- capability,
- ambient: an ambient n [P] is a named process P with name n.

A process can be composed to form composed processing units describing systems called *ambients*. Ambients include the following notions

- *scope*: location or space in which an identifier is valid. This is achieved by restriction (like Δ x.P),
- *parallel composition*: composition of ambients,
- *replication*: simple form of iteration or recursion on processes,
- *location*: named location or scope in which a process (composed or not) executes.

What can a process do when executing capabilities? It may carry out activities such as:

- *entering* ambient scopes, leaving ambient scopes and opening scopes. Note: given an ambient

  n[P]

  if in P a sub-process executes the capability "**in** m" (by which P becomes P') then n[P'] moves into the scope of ambient m[Q] (which results in an ambient inside which processes Q and n[P'] are running in parallel),
- *exiting*: exiting is inverse to entering,
- *opening*: when in the process P | m:[Q] a sub-process executes in P the capability open m by which P becomes P' then the name m and its scope disappears and the original process becomes P' | Q.

Finally an ambient can communicate. This is done by a synchronous input and output action. This is expressed by the rewrite rule:

$$(x).P \mid \langle M \rangle \rightarrow P[M/x]$$

Note that both names and capabilities may occur as input and output. Note, moreover, that the naming of processes is essential in ambient calculus. The semantic of ambient calculus is given in an operational style.

The ambient calculus is one, in fact, very sophisticated model of dynamic systems. It captures certain aspects of dynamics quite explicitly and leaves others implicit. It is a more operational model of dynamics than our approach since its semantics is captured by rules of structured operational semantics. It considers additional aspects such as scopes, locality (and thus also the idea of mobility) and restricted access to local data that is not considered in our presented work.

6 Summary and Outlook

We have introduced a denotational model for dynamic systems. In contrast to π-calculus and ambient calculus, which are based on operational models, our emphasis is on a denotational modular model and the notion of interface that can be used as a basis for modeling, specification, and verification.

Future work for the presented approach is needed in the following directions

- Introduction of more pragmatic description and modeling concepts in terms of diagrams and tables
- Application of the concepts to representative application examples

- Generalization to models and concepts that are typical for dynamic systems such as interoperability and connectivity

Dynamic systems are typical for a large class of application systems we find today. Therefore their adequate modeling becomes more and more important. Denotational models provide here a major contribution.

Acknowledgements. I am grateful for stimulating discussions with Ingolf Krüger.

References

1. Agha, G., Mason, I.A., Smith, S.F., Talcott, C.L.: Towards a Theory of Actor Computation. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 565–579. Springer, Heidelberg (1992)
2. Broy, M.: Towards a Mathematical Concept of a Component and its Use. First Components' User Conference, Munich (1996); Revised version in: Software - Concepts and Tools 18, 137–148 (1997)
3. Broy, M., Stølen, K.: Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement. Springer (2001)
4. Cardelli, L.: A Language with Distributed Scope. ACM Trans. Comput. Syst. 8(1), 27–59 (January); ALso appeared in POPL 1995
5. Grosu, R., Stølen, K.: A Model for Mobile Point-to-Point Data Flow Networks without Channel Sharing. In: Wirsing, M., Nivat, M. (eds.) AMAST 1996. LNCS, vol. 1101, pp. 505–519. Springer, Heidelberg (1996)
6. Grosu, R., Stølen, K.: A Denotational Model for Mobile Many-to-Many Data Flow Networks. Technical Report TUM-I9622, Technische Universität München (1996)
7. Haridi, S., van Roy, P., Smolka, G.: An Overview of the Design of Distributed Oz. In: The 2nd International Symposium on Parallel Symbolic Computation (PASCO 1997). ACM, New York (1997)
8. Milner, R.: The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh (1991)
9. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. Part i + ii. Information and Computation 100(1), 1–40, 41–77 (1992)
10. van Roy, P., Haridi, S., Brand, P., Smolka, G., Mehl, M., Scheidhauer, R.: Mobile Objects in Distributed Oz. ACM Toplas 19(5), 805–852 (1997)