

# Dynamic Distributed Systems\*

Towards a Mathematical Model

## Extended Abstract

Manfred Broy

Institut für Informatik, Technische Universität München  
D-80290 München, Germany

**Abstract.** This paper aims at a mathematical foundation of flexible information processing architectures that support the mobility and dynamics of systems by a formal system model.

**Index terms.** Software Engineering, System Models, Formal Methods, Mobility, Dynamic Systems

## 1. Introduction

In this paper we work out a theoretical foundation for dynamic systems architectures. For information processing systems of today and of tomorrow *dynamics* and *mobility* are key issues. We are aiming at a formal model in this paper that allows us to give precise definitions for key notions that arise in this context the *dynamics of nets* and the *dynamics of distributed systems*. We are interested in a precise definition, description and in the mathematical foundation of these notions and, moreover, in modeling the following technical key concepts in system structures:

- a *net* models a distributed system of components interacting in parallel and connected by communication channels,
- a component is called *dynamic*, if it can change its (syntactic) interface consisting of its active input and output channels,
- a net is called *dynamic* if it changes its structure (its set of existing components and its set of channels) during its lifetime, otherwise it is called *static*,

In the following we formalize the notions introduced above. We give a mathematical model that allows us to capture the mentioned aspects.

## 2. Component Models: Interface Models by Streams

A (system) *component* is an active information processing unit that communicates with its environment through a set of input and output channels. This communication takes place in a (discrete) time frame.

---

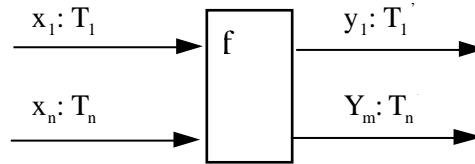
\*) Part of this work was carried out within the Forschungsverbund ForSoft, sponsored by the Bayerische Forschungsförderung.

As a basic model for the behavior of system components we use relations on timed streams<sup>1)</sup>. We model the time flow in systems by a sequence of time intervals. Let  $M$  be a set of elements called messages. Timed streams model communication histories for communication channels by an infinite sequence of finite sequences of messages (for a rigorous treatment see [Broy 95]). Each finite sequence represents the sequence of messages communicated within a particular time interval. On the basis of this simple model we are able to introduce a quite flexible notation that we will use throughout this paper in specifications and for our models.

By  $M^\infty$  we denote the set of infinite sequences of elements of the set  $M$ , which can be represented by functions  $\mathbb{N} \setminus \{0\} \rightarrow M$ , by  $M^*$  we denote the set of finite sequences of elements from  $M$ . Using this notation, by

$$(M^*)^\infty$$

we denote the set of infinite sequences of finite sequences, which can be represented by functions  $\mathbb{N} \setminus \{0\} \rightarrow M^*$ , also called streams of finite sequences of elements of the set  $M$ .



**Fig. 1** Graphical representation of a component as a data flow node with input channels  $x_1, \dots, x_n$  and output channels  $y_1, \dots, y_m$  and their respective types

Let  $I$  be the set of input channels and  $O$  be the set of output channels. Formally a channel is nothing but an identifier. With every channel in the channel set  $I \cup O$  we associate a data type indicating the type of messages sent on that channel. For simplicity, in the following we use uniformly only one set of messages denoted by  $M$  representing the data types for the messages on the channels to keep the mathematics more readable. Our approach, however generalizes to individually typed channels in a straightforward way.

By the pair  $(I, O)$  the *syntactic interface* of a system component is represented. A graphical representation of a component with its syntactic interface and individual channel types is shown in Fig. 1.

We describe the *black box behavior* of a component by an *I/O-function*. It represents a relation between the input streams and the output streams of a component that fulfills certain conditions with respect to their timing. An I/O-function is a set-valued function on valuations of the input channels by timed streams. The function yields a set of histories for the output channels for every input history. This way, an I/O-function is a function

$$F: (I \rightarrow (M^*)^\infty) \rightarrow \wp(O \rightarrow (M^*)^\infty)$$

which fulfills the following *timing property*. The timing property axiomatises the time flow and reads as follows:

<sup>1)</sup> Another option is to use nontimed streams, especially, when dealing with systems where time is not an issue. However, even for these systems it is often convenient to be able to talk about time, especially, when combining such systems with time dependent components.

$$x \downarrow t = z \downarrow t \Rightarrow \{y \downarrow t+1: y \in F(x)\} = \{y \downarrow t+1: y \in F(z)\}$$

For a stream  $s$ ,  $s \downarrow t$  denotes the sequence that is the prefix of the stream  $s$  and contains  $t$  finite sequences. In other words,  $s \downarrow t$  denotes the communication history of  $s$  until time  $t$ . This operation is extended to histories in  $C \rightarrow (M^*)^\infty$  where  $C$  is a set of channels, pointwise. The timing property expresses that the set of possible output histories for the first  $t+1$  time intervals only depends on the input histories for the first  $t$  time histories. In other words, the processing of messages in a component takes at least one tick of time. This way causality between input and output is guaranteed. We call functions with this property *time-guarded* or *strongly causal*.

By  $\text{COM}[I, O]$  we denote the set of all strongly causal I/O-functions with the syntactic interface  $(I, O)$ , that is, with the set of input channels  $I$  and the set of output channels  $O$ . For each  $F \in \text{COM}[I, O]$ ,  $\text{In}(F) = I$  denotes its set of input channels and  $\text{Out}(F) = O$  denotes its set of output channels. By  $\text{COM}$  we denote the set of all strongly causal I/O-functions, also called data flow behaviors.

### 3. A Mathematical Model of Data Flow Nets

We model distributed systems by data flow nets. Let  $K$  be a set of identifiers for components<sup>1)</sup> (represented by data flow nodes) and  $O$  be a set of output channels. A distributed system  $(v, O)$  in the form of a data flow net with the syntactic interface  $(I, O)$  is represented by the mapping

$$v: K \rightarrow \text{COM}$$

that associates with every node labeled by the identifier  $k \in K$  a component behavior (an interface behavior given by an I/O-function)<sup>2)</sup>. In principle, we can think about the component associated with an identifier as its type (or its class). This way we are very close to object orientation.

The set

$$I = (O \cup \{c \in \text{In}(v(k)): k \in K\}) \setminus \{c \in \text{Out}(v(k)): k \in K\}$$

denotes the set of input channels of the net. As a well-formedness condition we require that for all component identifiers  $k, j \in K$  (with  $k \neq j$ ) the sets of output channels of the components  $v(k)$  and  $v(j)$  are disjoint. This is formally expressed by the equation  $\text{Out}(v(k)) \cap \text{Out}(v(j)) = \emptyset$ . In other words, each channel has a uniquely specified component as its source (including the environment as a source). We denote the set of  $K$  (identifiers for the) nodes of the net by

$$\text{Nodes}((v, O))$$

We denote the set of all the channels of the net by  $\text{Chan}((v, O))$  specified by the equation

$$\text{Chan}((v, O)) = O \cup \{c \in \text{In}(v(k)): k \in K\} \cup \{c \in \text{Out}(v(k)): k \in K\}$$

The channels in the set

$$\{c \in \text{Out}(v(k)): k \in K\} \setminus O$$

are called *internal*. Recall that the mapping  $v$  associates with every node its behavior in the form of an I/O-function.

<sup>1)</sup> The fact that we use component identifiers gives components their unique identity over the lifetime of a system. One may think of *object identifiers*.

<sup>2)</sup> We assume that the input and output arcs of each node labeled by an identifier  $k \in N$  are determined exactly by  $\text{In}(v(k))$  and  $\text{Out}(v(k))$  respectively.

A data flow net seen from the outside describes an I/O-function. This I/O-function is called the *black box view* of the distributed system that is described by the data flow net. It defines an abstraction of the distributed system that is represented by the data flow net  $(v, O)$  leading to its black box view by mapping it to a component in  $\text{COM}[I, O]$ . Here  $I$  denotes the set of input channels and  $O$  denotes the set of output channels of the data flow net. This black box view is represented by the I/O-function  $F_{(v, O)} \in \text{COM}[I, O]$  specified by the following formula:

$$F_{(v, O)}(x) = \{y|_O : y|_I = x \wedge \forall k \in K: y|_{\text{Out}(v(k))} \in v(k)(y|_{\text{In}(v(k))}) \}$$

Here we use the notation of function restriction. For a function  $g: D \rightarrow R$  and a set  $T \subseteq D$  we denote by  $g|_T: T \rightarrow R$  the restriction of the function  $g$  to the domain  $T$ .

The formula essentially expresses that the output history of a data flow net is the restriction of a channel evaluation for all the channels of the net that is a fixpoint<sup>1)</sup> for all the net equations to the output channels.

## 4. Dynamic Systems

The dynamics of information processing systems is not so easy to grasp. On one hand universal programmable computer systems show a very dynamic behavior by definition. They gain their flexibility by the fact that they can be programmed to compute any computable task. However, this form of dynamics is rather specific: it requires human interaction by the programmer. In contrast to this we are interested in this paper in dynamic systems with a dynamics that is part of the programmed system behavior.

A crucial question here is to find the appropriate level of abstraction of computing systems to study their dynamics. On the machine level, where only bits and bytes are processed and even the difference between data and instructions disappears it is quite difficult to capture the notion of dynamics of systems. At this level computers process bit streams. Only at higher levels of abstractions dynamics becomes explicit. On very high levels of abstraction, however, in many cases the dynamics may no longer be explicitly visible.

Another crucial issue for an information processing system and its description is the balance between statics and dynamics. In a world without types, for instance, any behavior can be encoded (see  $\pi$ -calculus [Milner 91]). Types introduce restrictions on the system behavior and this way restrict the dynamics of systems. Obviously, it is crucial to find the right balance between static aspects including types and dynamic aspects. In system development, we want to associate a number of properties, structure, and views with a system model. Due to the dynamics of a system, some of these views may change.

A distributed, interactive system is called *dynamic*, if it changes its set of components or channels, its distribution structure, its topology and/or its channel connection structure during its lifetime. This means that it may change step by step

- its set of existing components,
- the locations of its components,
- its set of communication links and its interconnection structure (internal and external channels).

---

<sup>1)</sup> According to the fact that we consider only time-guarded I/O-functions it can be shown that if the I/O-function is deterministic, there is always a fixpoint and the fixpoint is unique. Due to time guardedness, the recursive equations are sufficient to characterize this fixpoint and the idea of a least fixpoint is not needed.

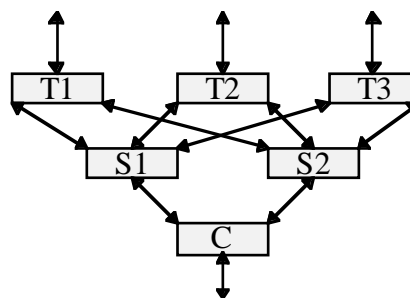
Along these lines we can even speak in the case of black box views of dynamic systems and of dynamic components, if components change their *syntactic interface* over their lifetime. Of course, state transition machines where the set of components and channels is a part of their state easily model dynamic systems. In such models, however, the distribution structure is rather implicit. Moreover, it is difficult to arrive at modular system models that way with clean and simple composition operators. We are, however, interested in models which present and deal with the distribution structure more explicitly.

#### 4.1 Dynamic Nets and Mobility

A dynamic net typically changes its set of components and its channels during its lifetime. Speaking in more general terms, a dynamic system changes its component structure (its architecture) over its lifetime. In principle, these changes are not very difficult to model. If we model a system by a state transition machine we may describe the connection network as a part of the state. By state transitions the state may change and so may the network. This way we may describe state transition steps with radical changes of the system structure. However, for most applications we are not interested in radical changes of the network structure within one step. Rather we are interested in very specific, small, relatively local changes where in one step most of the net structure remains unchanged and only

- one component is added or deleted, or
- one channel is added to or deleted from a component or its source and/or target is changed.

This leads to the idea of evolving networks along the lines of the  $\pi$ -calculus (see [Milner 91], [Milner et al. 92]) or the ambient calculus (see [Cardelli 95]) where the steps of changes are captured by rewriting rules. In fact, in  $\pi$ -calculus the meaning and behavior of dynamic networks is specified in a purely operational way, which does not lead to a clear notion of an interface nor to a denotational model of a dynamic system. We are interested in the following in a denotational model and its modularity as a basis of specification and design techniques.



**Fig. 2** Static Network of Central C, Stations S1, S2 and Mobile Phones T1, T2, T3

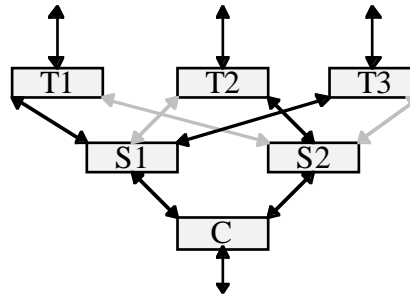
##### 4.1.1 Dynamics of Structure, Interface, and Behavior

We start with an example to illustrate the applications, notions, and goals of mobility and dynamics.

**Example:** Mobile Telephone

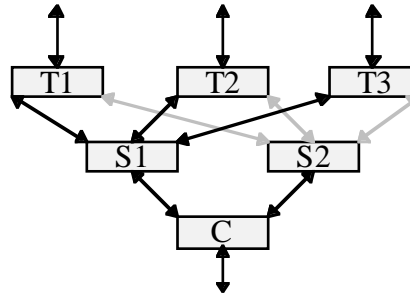
Let us discuss the different options to deal with dynamic systems by a rather simple, well-known example namely that of a mobile telephone system comprising two switching stations, one central telephone exchange, and up to three mobile phones.

As shown in Fig. 2, the system consists of a network which contains as components a telephone central C, two switching stations S1 and S2 and three mobile phones T1, T2, and T3.



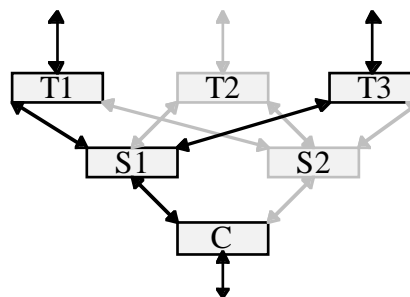
**Fig. 3** Dynamic Network with Three Active Mobile Phones (Inactive Channels in Grey)

In this network the component C is static (its set of active channels never changes) as long as both stations are always active while the components S1, S2, T1, T2, and T3 are dynamic since their set of active channels may change.



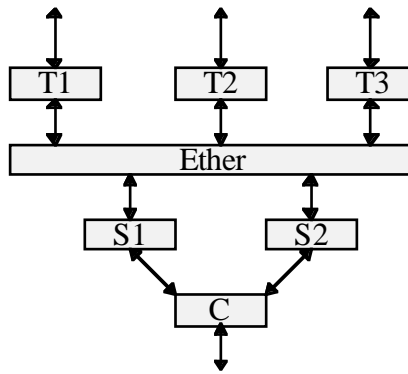
**Fig. 4** Dynamic Network with Three Active Mobile Phones and Changed Connection

We may model the case that mobile phones exchange their stations as illustrated by the Fig. 2 to 5.



**Fig. 5** Dynamic Network with Two Active Phones and One Active Switching Station

Fig. 2 gives the static network of all possible components and all possible connections. These are all the connections and components that may exist over the lifetime of the system. Fig. 3 shows a system configuration where each phone is active and connected to exactly one station. Fig. 4 shows the result of changing the connection for telephone T2 from station S2 to station S1. Fig. 5 shows a configuration where telephone T2 and station S2 are inactive. In this case the central C is a dynamic component, too.



**Fig. 6** Dynamic Network with an Ether Connection

Fig. 6 shows a solution where a transmission component called Ether is introduced that acts as a communication bus. In this case, the system structure is static and no dynamic behavior occurs explicitly since the channel and message switching is done by the component Ether and therefore the net appears static.

Note that there are two options to give a behavior to the component Ether. Either the component Ether is a *merge* component (also called a *multiplexer*) that merges all its input message streams and forwards them to all its output channels (broadcasting) or it is a *switch* that connects certain inputs with certain outputs or it is a combination of a filter and a merger. As a special case Ether can be described as a switching network that has a state indicating exactly which channels are connected. This state reflects the net structure as it is shown in Fig. 3 to 5 explicitly. □

In a network of the characteristics introduced in the example above the structure of the system, the set of existing components and connections of the components may change during the lifetime of the system. In fact, the example shows only two essential aspects of dynamics, the dynamics of the interface (in Fig. 5 the switching station S1 has only two (active) channel connections to mobile phones while in Fig. 4 it has three) and the dynamics of the connections and sets of components of a system. In fact, we are also interested in components that change their functionalities by offering modified or additional services.

To model the dynamics of a system we can describe a sequence of states of a system by snapshots. In each snapshot the system distribution and communication connection structure may change. We believe that it is important to keep track of the individual objects (components) during the lifetime of dynamic systems. This is achieved by a particular notion of *identity* for each of the components. This identity is captured easily with the help of unique component identifiers such as in object orientation or in the Internet by the IP addresses. In dynamic systems we are interested to keep track of the individual components. If we compare two snapshots of the system showing the subnet of active components and channels we obtain two different nets, in general. Which of the components of these nets represent the same computing entity can be determined only by the identifiers associated with the components.

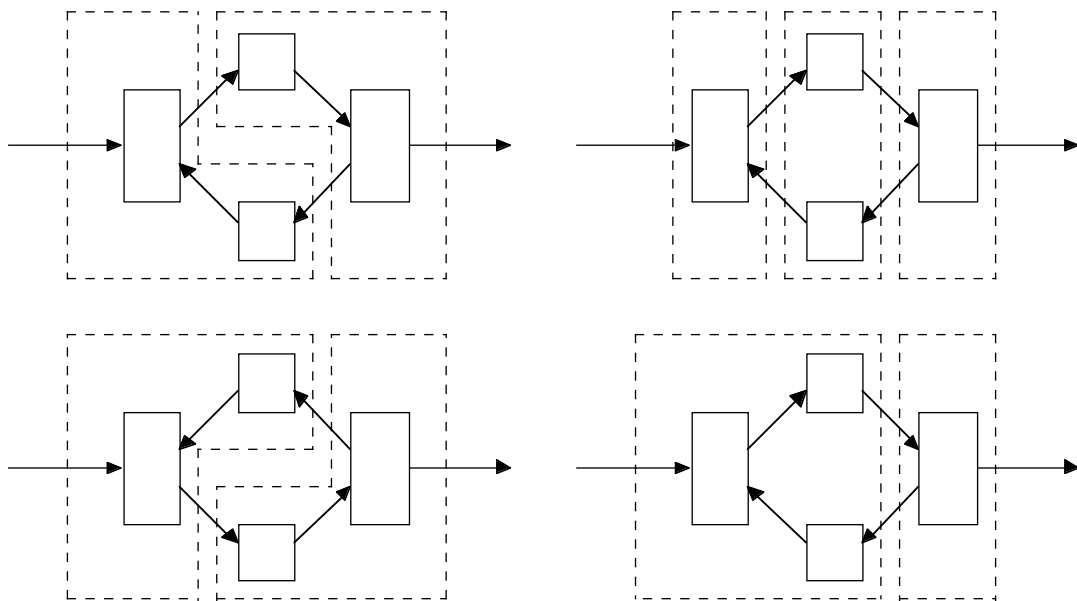
#### 4.1.2 Mobility

Interesting aspects in dynamic systems that we did not mention explicitly in our example so far are the individual steps by which the number of components changes. Examples are the melting of components (making one component from several ones), the cloning of components (creating a second copy of a component), or their split (dividing a component into two). These patterns of

system behaviors can be seen as special cases of system dynamics that are achieved by naming conventions and/or hierarchical networks. We are interested to treat, in addition to the dynamics of systems, a hierarchical partitioning of networks. We explain this idea again by an example showing the dynamic reconfiguration of systems.

**Example: Reconfiguration**

We give a very simple syntactic example for reconfiguration. A reconfiguration collects and encapsulates different sets of components into locations. Each of the locations contains a set of components. These sets are forming again components. As a result we get partitions of a system that may change dynamically.



**Fig. 7** Four Examples of Configuration of a Simple Net

Fig. 7 shows four different configurations of a simple net. Each corresponds to a particular partition of the set of components. We could also study configurations with nondisjoint sets of overlapping subsystems. We can think about steps of reconfiguration of a system where such changes appear. □

The idea of partitions gives us, in particular, one handle to speak about component mobility. By partitions where the elements of the partitions are identified by identifiers called locations we can speak about system state changes where one component moves from one partition to the other.

Configurations define system topologies and their neighborhoods for systems. We may ask in such a system model if two components are in the same set of the partition ("are in the same environment"). This can be used to allow, for instance, additional ways of interactions for components that are in the same set of the partition. Those aspects are modeled in the *ambient calculus* (see [Cardelli 95]).

## 4.2 A Formal Model for Dynamic Systems

To keep our discussion simple we restrict our discussion to deterministic components and systems in the following. An extension to nondeterministic systems is possible along the lines of Focus [Broy 98], too.

Let  $I$  be throughout this section a (possibly infinite) set of input channels and  $O$  be a (possibly infinite) set of output channels.

#### 4.2.1 Dynamic Streams

A dynamic stream represents the history of a channel that may become active and inactive several times through its lifetime. Given a type  $M$  (which in our setting is simply a set of data elements used as messages) we define the set of dynamic streams over the set  $M$  by the set of streams over  $M \cup \{ @, \odot \}$  as follows:  $s$  is a dynamic stream over  $M$  and we write  $s \in M^\circledast$  if the following formula holds:

$$\begin{aligned} \forall t \in \mathbb{N} \setminus \{0\}: & \quad (\neg \alpha(s).t \Rightarrow s.t = \diamond \vee s.t \in \{ @ \} \times M^*) \\ & \quad \wedge \quad (\alpha(s).t \Rightarrow s.t \in M^* \vee s.t \in M^* \times \{ \odot \}) \end{aligned}$$

where the function

$$\alpha : ((M \cup \{ @, \odot \})^*)^\infty \rightarrow (\mathbb{N} \rightarrow \mathbb{B})$$

is an auxiliary function with the meaning  $\alpha(s).t$  expresses that the stream  $s$  is active at the beginning of time interval  $t$ .  $\alpha$  is specified by the equations

$$\begin{aligned} \alpha(s).0 &= \text{false} \\ \alpha(s).t+1 &= (\alpha(s).t \wedge \neg \odot \in s.t) \vee (\neg \alpha(s).t \wedge @ = \text{ft}(s.t+1)) \end{aligned}$$

This definition essentially defines the patterns of activation and deactivation signals and thus expresses the rules of activating and deactivating a channel. Note that a channel can either be activated and deactivated in one time interval of the system.

Streams on dynamic channels may carry activation and deactivation messages. Therefore the type of a channel indicates whether the channel is dynamic. Here we assume for simplicity that the channels are active or inactive always during complete time intervals. So at the beginning of each time interval (say at time  $t$ ) we have a set of active channels that stay active over the entire time interval and the other channels stay inactive throughout the entire time interval<sup>1)</sup>. For simplicity, all dynamic channels are inactive at time 0.

We introduce a type function for sets of channels  $C$ :

$$\text{type}: C \rightarrow T$$

where  $T$  is the set of all types each type in  $T$  is a set. By

$$\vec{C}$$

we denote the set of all channel valuations. A channel valuation  $x \in \vec{C}$  is a mapping

$$x: C \rightarrow (M^*)^\infty$$

where  $M = \cup \{t: t \in T\}$  is the universe of all messages. We require for a channel valuation that

$$x.c \in (t^*)^\infty$$

if  $\text{type}(c) = t$ . Furthermore, if we have  $\{ @, \} \subseteq \text{type}(c)$  then we require that  $x.c \in M^\circledast$  that means that  $x.c$  is a dynamic stream.

---

<sup>1)</sup> Note that only due to our model of time we speak so easily about the "state" of the system w.r.t. the activity of channels.

### 4.2.2 Dynamic Components

A *dynamic component* (deterministic behavior) with the syntactic interface  $(I, O)$  corresponds to an I/O-function

$$F: \vec{I} \rightarrow \vec{O}$$

where some of the channels in  $I \cup O$  are dynamic. The function  $F$  models the input/output behavior of the component. On this basis we are able to define a function

$$\alpha: \{x \downarrow t: t \in \mathbb{N} \wedge x \in I\} \rightarrow \wp(I \cup O)$$

The function  $\alpha$  yields for every initial segment  $x \downarrow t$  of an input history  $x$  at time  $t$  a snapshot that represents the set of active channels. Only if a channel is active communication may take place along it. The only exceptions are the activation messages (see above). The set  $\alpha(x \downarrow 0)$  yields the set of channels that are active initially, which are the static channels.

The set of dynamic components is denoted by DCOM. The set of dynamic components with the syntactic interface  $(I, O)$  is denoted by DCOM  $[I, O]$

Note that we did not make any assumptions about the cardinality of the channel sets for a component. Of course, the set of dynamic channels of a component can be infinite. Whether we are interested in and allow for components with an infinite number of active channels is another question. Certainly, for particular applications components with an unbounded set of active channels are of major interest.

### 4.2.3 Dynamic Nets

Let DCOM be the set of dynamic components. We work with a function that assigns behaviors to component identifiers. A *dynamic net* consists of a (possibly infinite) set of nodes  $K$  (identifiers for components) and a mapping

$$v: K \rightarrow \text{DCOM}$$

where DCOM is the set of dynamic components. Let  $I$  be the set of input channels of the net. To model activation and deactivation of the components in the net, we define a mapping

$$\beta: \{x \downarrow k: k \in \mathbb{N} \wedge x \in \vec{I}\} \rightarrow \wp(K)$$

This mapping indicates which components of the net are active at a given time. The channels that are active in the network are the active channels of the active components.

For simplicity we assume that a component is inactive, if and only if all its channels are inactive. We define the activity function  $\beta$  as follows:

$$\begin{aligned} \beta(x).t = \{k \in K: \exists y \in \vec{C}: & y|_I = x \\ & \wedge \forall k \in K: y|_{\text{Out}(v(k))} \in v(k)(y|_{\text{In}(v(k))}) \\ & \wedge (\exists c \in \text{In}(v(k)) \cup \text{Out}(v(k)): \alpha(y.c \downarrow t))\} \end{aligned}$$

At a first glance for some readers it may look strange to work in our system model with a set consisting of all the components and of all the channels that might become active during the lifetime of a system. However, this is not so difficult and unusual as it may seem at a first glance. In object-oriented systems the set of object identifiers determines the set of potentially active objects. And the syntactic structure of the methods (where in the body of a method a method of another class is called) determines together with the links between the objects the possible connection structure. According to this model at each time there exists a uniquely

defined network of active components and channels. Moreover, we may assume an infinite set of object identifiers for each class. Therefore each class represents an infinite set of components most of which being inactive in a particular state (time) of the system.

Also in other systems it is quite common that the set of potentially active components is determined (and bounded) by a set such as the set of potentially active components. An example is the Internet with its finite number of IP-addresses.

## 5. Conclusions

It is the main goal of this paper to demonstrate that the methods, notations, and concepts used in practice for the modeling and description of digital systems including dynamics can well be scientifically based on the more foundational and theoretical work created so far in computing science. This way we obtain a mathematical basis for powerful system modeling languages as well as software and system engineering methods.

The benefits of such a mathematical foundation are quite obvious. In particular, we obtain this way:

- A deeper understanding of the methods leading to helpful "Gedankenmodelle" based on the mathematical models.
- Better description techniques for dynamic systems.
- The conceptual consistency of description and development methods for developing dynamic and mobile systems.
- The mathematical basis that can help as a guideline for the definition of development methods for dynamic and mobile systems.
- Advanced tool support for specification as well as consistency checking, prototype generation, simulation, and verification of dynamic systems with a firm basis.

Apart from these more direct benefits of a mathematical foundation of software engineering methods, a scientific foundation is badly needed as a step toward a more systematic study of methods for dealing with dynamic and mobile systems. Only if we manage to develop general common criteria to compare the expressive power and quality of software engineering methods we will be able to free our discipline from dogmatic view points and marketing-based judgments and thus prepare the ground for scientifically justified and practically tractable systems and software engineering methods.

## Acknowledgment

The thoughts presented above have benefited greatly from discussions within the SysLab team and the Forsoft I Project AI research group.

## References

[Broy 95]

M. Broy: Advanced Component Interface Specification. In: Takayasu Ito, Akinori Yonezawa (Eds.). Theory and Practice of Parallel Programming, International Workshop TPPP'94, Sendai, Japan, November 7-9, 1994, Proceedings, Lecture Notes in Computer Science 907, Springer 1995

[Broy 98]

M. Broy: Compositional Refinement of Interactive Systems Modelled by Relations. In: W.-P. de Roever, H. Langmaack, A. Pnueli (eds.): Compositionality: The Significant Difference. LNCS State of the Art Survey, Lecture Notes in Computer Science 1536, 1998, 130-149

[Cardelli 95]

R. Cardelli: A Language with Distributed Scope. ACM Trans. Comput. Syst. 8, 1 (Jan.), 27-59. Also appeared in POPL 95.

[Milner 91]]

R. Milner: The polyadic  $\pi$ -calculus: A tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, 1991.

[Milner et al. 92]]

R. Milner, J. Parrow, D. Walker: A calculus of mobile processes. Part i + ii, Information and Computation, 100:1 (1992) 1-40, 41-77