23

A stream-based mathematical model for distributed information processing systems - the SysLab system model - [1]

Cornel Klein, Bernhard Rumpe, Manfred Broy
Fakultät für Informatik, Technische Universität München
80290 Munich, Germany, http://www4.informatik.tu-muenchen.de/

Abstract

In the SysLab-project, we develop a software engineering method based on a thorough mathematical foundation. The SysLab system model serves as an abstract mathematical model for information systems and their components. It is used to formalize the semantics of all used description techniques, such as object diagrams, state automata, sequence charts or data-flow diagrams. The system model hence is the key to the semantic integration of the method. Based on the requirements for such a reference model, we define the system model including its different views and their relationships.

Keywords

Semantic Model, Refinement, Decomposition, Streams, Data-flow, Software Development Method

1 INTRODUCTION

Methods for systems and software development, like OMT (Rumbaugh, Blaha, Premerlani, Eddy and Lorensen 1992) and Fusion (Coleman, Arnold, Bodoff, Dollin, Gilchrist, Hayes and Jeremaes 1994), model a system at different abstraction levels and under different views. Within the process of modeling they provide description techniques like entity-/relationship-diagrams and their object-oriented extensions, state automata, sequence charts or data-flow diagrams. A critical point of existing commercial methods is the imprecision of the semantic description. The definition of the description techniques as well as the relationships between different description techniques of a method is usually only given informally. A lot of problems during the use of the methods exist, which are caused by the ambiguous and vague interpretation of the semantics of the used modeling concepts:

[1]This paper originated in the SysLab project, which is supported by the DFG under the Leibnizprogramme and by Siemens-Nixdorf.

A full version of the paper is available through the SysLab-Homepage at http://www4.informatik.tu-muenchen.de/proj/syslab/. It contains the formalization of the system model, which is omitted here in favor of a careful explanation of the motivation.

E. Najm et al. (eds.), *Formal Methods for Open Object-based Distributed Systems*

- the communication between the persons involved in the project is more difficult, because of ambiguities arising from informal semantic descriptions;
- it is impossible to define formal relationships between different description levels and to define rules to transfer information between two description levels;
- a solid basis for tool support is missing;
- even in one description level issues concerning "consistency" and "completeness" can only be tackled informally.

The SysLab-project aims at the development of a practicable method for system- and software development that is scientifically founded and does not show the above-mentioned disadvantages due to the lack of a semantic foundation. Moreover, in SysLab a prototype of a tool system is created. The formalization does not end in itself, but it provides the semantic basis for the check for consistency of the concepts, both with respect to the method and the tool. The semantic foundation is achieved by the usage of a uniform mathematical system model for SysLab. This abstract mathematical model of information processing systems serves for relating to it all description techniques used in SysLab, such as object diagrams, state diagrams, data-flow diagrams, etc., and all transformation rules for the transformation of documents. Each document, such as an object diagram, is regarded as a proposition over the mathematical system model.

It is the aim of this paper to provide a common basis for all people involved in the SysLab-project concerning the notion of a system and the definition of the semantic of the various description techniques. Therefore, the system model has to cover all phases and all description techniques of the SysLab method, and it may not be restricted to a certain class of information processing systems, such as commercial information systems. From that results the requirement to develop a system model which is as *general as possible.*

On the other hand, it should be easy to define a semantics based on the system model for the description techniques to be developed. This leads to the requirement that the system model has to be tailored for the description techniques we are aiming at. This means for instance that the model has to support the dynamic creation and deletion of components ("objects").

The basic assumption with respect to the structure of information processing systems is that such systems are hierarchically and modularly constructed from a number of components, which may interact in parallel and which can be viewed as information processing systems themselves. In this case, we call the system a *distributed system.* Distribution here means *spatial distribution* as well as *logical distribution* of functionality across components. However, there are systems which are not parallelized or distributed any further. Such *basic components* can be modeled using state automata with input and output. The repeated decomposition of a system into subsystems yields a hierarchical system, the structure of which can be viewed as a tree with distributed systems on the inner nodes and with basic components on the leaves.

We are interested in a system model in which each kind of interaction is expressible. In our opinion, each kind of interaction can be viewed as the *exchange of a*

message between the interacting components. Thus components can be modeled as having *input ports* to receive messages from their environment, and *output ports* to send messages to their environment. The ports constitute the *interface* of a component, they provide the only possibility for the interaction between a component and its environment. The *behavior* of such a component is the relationship between the sequences of messages on its inputs ports and the sequences of messages on its output ports. Systems and their components encapsulate data as well as processes. *Encapsulation of data* means that the state is not directly visible to the environment, but can only be accessed using explicit communication. *Encapsulation of process* means that the exchange of a message does not imply the exchange of control, and that therefore each component is a process of its own.

Exchange of messages between the components of a system is *asynchronous*. This means that a message can be sent independently of the actual readiness of the receiver to receive the message. The requirement for asynchronous communication results from experience in the project FOCUS (Broy, Dederichs, Dendorfer, Fuchs, Gritzner and Weber 1993): Asynchronous system models provide the most abstract system model for systems with message exchange. They can precisely, easily and intuitively be modeled using stream processing functions, for which a multitude of tractable specification techniques for untimed as well as for timed systems exist (Grosu and Stølen 1995, Broy et al. 1993). Moreover, for stream processing functions a powerful theory for compositional refinement has been developed. By using an asynchronous system model, in contrast to process algebraic approaches like the π-calculus (Milner 1991) or CCS (Milner 1989), we do not have to tackle synchronization issues. To take into account synchronization aspects is in our opinion an issue which is irrelevant in the early phases of system development. However, synchronization can easily be encoded in our model, for instance by using an appropriate protocol.

If possible, the system model should not impose any constraints concerning the *addressing of messages*. One possibility for the addressing is that the input- and output ports are statically connected through *channels*. Alternatively, it is also possible in our model to address messages using *identifiers*, as they are used in the context of object-oriented programming languages. Moreover, in defining the semantics of object-oriented programming languages we cannot assume that the set of components is static, but we have to allow for the dynamic generation of components. These requirements lead to two concepts for communication. The first uses ports and the second uses identifiers. The system model has to be prepared for both communication concepts, where one of them or a combination of both may be chosen if the system model is applied.

To allow for the consideration of systems in which quantitative time is relevant, the system model has to provide an explicit notion of time which goes beyond the causality relation formalized by the monotonicity requirement for stream processing functions (Broy et al. 1993). We assume that a *discrete time*, which is obtained by partitioning the time scale into equidistant time intervals, is sufficient for the purpose of SYSLAB.

This paper is organized as follows: In the next section the black-box view of systems is presented. This is done by describing the mathematical structure of streams, by presenting stream processing functions as a model of interactive systems, and by introducing identifiers for components. In section we introduce two glass-box views, the system as a *basic component* and the system as a *distributed system.* In section 4 we give a conclusion by comparing the presented system model with the requirements stated in this section.

2 BLACK-BOX VIEW

An information processing system is an entity interacting with its environment by the exchange of messages. The interface between the system and the environment can be modeled by so called *ports*, which are often also called *channels*, over which data flow. We distinguish between *input ports* and *output ports.* A graphical representation of a component is given in figure 1, left. We assume that all port names are contained in the set P of *port names.*

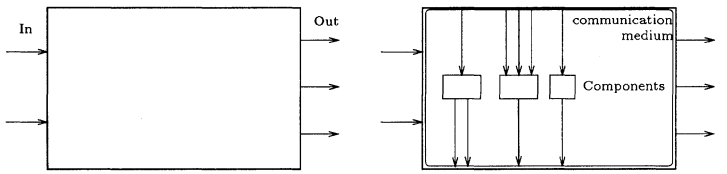

Figure 1 Black-box view and glass-box view of a system

At runtime, a system receives messages on its input ports and sends messages on its output ports according to its behavior. In the remainder of this section, we introduce *streams* as a model for the communication history of ports, *stream processing functions* as a model of interactive systems and *identifiers* of components for the addressing of messages in our system model.

The behavior of a system is modeled by its system runs, which describe the relationship between the messages arriving on the input ports of the system and the messages sent on the output ports of the system. We assume that for each run the events on a port are totally ordered, which means that for two different events always one causally and temporarily precedes the other. This allows to model the communication history on a port by a stream of messages.

In addition to the total order of events our system model also provides an explicit notion of time. Given a set M of messages a *timed stream* is an infinite sequence of messages over the set $M \cup \{\surd\}$, containing infinitely many copies of the time tick $\surd$. We assume that time proceeds in equidistant time intervals, and model the proceeding of time by the special time signal $\surd \notin M$, called *tick.* The requirement for infinitely many copies of $\surd$ models the fact that time never ends and that we consider only infinite communication histories.

Given the data-type of timed streams, the *behavior* of a system is modeled by a *stream processing function* mapping streams on the input ports of the system to streams on the output ports of the system.

However, not every function with this functionality represents an adequate model of an information processing system: In reality, it is impossible that at any point of time the output depends on future input. Therefore, we postulate that stream processing functions are *pulse driven*, which means that the output of a component up to to time j is only determined by the input up to time j.

We are interested in systems that allow to address a message by the *identifier* of the receiver, like this is in general done in object-oriented programming languages. We use a countable set ID of identifiers for this purpose. Every identifier names exactly one component in the system and every component has exactly one identifier. However every component id may have several input and output ports. We denote them by In_{id} and Out_{id}. We require the sets of portnames of different components to be disjoint. This requirement does not restrict the power of our system model, but simplifies the definitions in the sequel, because now every portname is uniquely attached to one component.

3 GLASS-BOX VIEWS

As already mentioned in the beginning, regarding the internal construction, we distinguish between basic components and distributed systems that are decomposed into a nonempty set of components.

The set of identifiers ID can therefore be divided into the disjoint sets of identifiers for basic components ID_b and of identifiers for distributed components ID_s.

Basic components are systems that are not composed of other components. They can be modeled by stream processing functions or by state-machines with input and output. Mathematical models for basic components are for example state-transition-systems (Broy, Dederichs, Dendorfer and Weber 1991) or I/O-automata (Lynch and Stark 1989). Especially concurrent timed port automata (Grosu and Rumpe 1995) are suited to describe basic components with several input and output ports in a timed environment.

A description of basic components by state-machines is suitable whenever concrete assumptions about the structure of the internal state of the component are made. If a description-technique only considers the black-box behavior of a component, we will not explicitly construct state-machines, but instead we will use a characterization of the behavior just by stream processing functions.

Besides being a basic component, a component can internally be decomposed into a set of subsystems called *components*. In this case we speak of a *distributed system*. As already mentioned, distribution in this case means spatial distribution as well as logical distribution. The set of identifiers of the components of a distributed system id is denoted by $Parts(id)$.

By repeated decomposition of a system we get a hierarchy of systems and subsystems. Function $Parts$ therefore characterizes a tree with a special identifier $RootSystem \in ID$ as root of this tree. By this arrangement of all components in

a component hierarchy, the superior components as well as the parts of a component are uniquely determined. The set of identifiers together with function *Parts* is used to define this hierarchical structure of systems, while the set of portnames determines communication channels.

We now examine the relationship between the behavior of a distributed system $id \in ID_s$ and the behaviors of its components. Figure 1 on the right shows a diagram of a distributed system. A distributed system consists of its components *Parts*(id) and a *communication medium*, which transmits the messages from the sender to the correct port of the receiver. The communication medium acts like a "membrane" between the inner and the environment of a component. In the following, we characterize the message flow through this membrane by relating the input and the output message streams of this membrane.

The communication medium has a complex signature, the *message origins* $Origins_{id}$ and the *message destinations* $Destinations_{id}$. The message origins consist of the input ports of system id and of the output ports of the components of id. Conversely the message destinations consist of the output ports of id and of the input ports of the components of id.

For description purposes, we assume that every message carries the addresses of its origin and destination in itself. We therefore do not allow message broadcasting, but require that every message carries the information that identifies a unique destination. We model this by two functions

$$origin_{id} : M \rightarrow Origins_{id}$$
$$destination_{id} : M \rightarrow Destinations_{id},$$

that describe the origin and the destination port of a message depending on the system id through which the message actually flows. The two functions $origin_{id}$ and $destination_{id}$ define the connection structure between the components of a distributed system. If we have an object-oriented system, messages carry their destination identifier and $destination_{id}$ solely depends on this identifier. If we have hard-wired systems, such as hardware systems, function $destination_{id}$ may only depend on function $origin_{id}$, where it is required that messages with the same origin have the same destination.

We require that the following properties of the message flow hold within the system model:

- For each input port of the system and for each output port of a component the order of messages sent to a certain destination has to be maintained. This requirement enforces a linear ordering of messages within every connection.
- The contents of messages may not be modified. Messages cannot be duplicated or lost. No new messages are generated by the communication medium.

Some kinds of systems exhibit connection structures where these requirements for message transmission are not valid. These systems can easily be encoded within our system model if we use special transmitter components exhibiting the behavior of such a connection structure.

We do not require our communication medium to be free of *delay*, since we do not impose any requirement on the time difference between the sending and the receiving of a message besides the requirement that this time is finite.

We are now able to specify a communication medium that distributes messages according to the above requirements by relating origin and destination streams of the communication medium.

1. Origin and destination streams restricted to the input resp. output ports of system *id* exhibit the behavior of system *id*.
2. Input and output ports of every component have to exhibit message streams according to the behavior of the component.
3. Every destination stream contains exactly the messages for this destination port.

These rules can easily be formalized such that a fully formal system model is provided.

4 DISCUSSION AND CONCLUSION

In this paper a so-called system model has been presented as an abstract mathematical model for information processing systems. Because the model is based on FOCUS (Broy et al. 1993), a mathematical modeling and development technique for distributed systems, a multitude of refinement and verification techniques for the system model exists. The model allows for the formal foundation and semantic integration of a large class of description- and programming techniques. The applicability ranges from analysis, specification and design documents to programs in (distributed) object-oriented programming languages. An explicit notion of time makes the model also well-suited for real-time and hardware systems. The flexibility of the system model is to a large extent possible due to the underspecification of the communication medium which allows for a large number of different applications.

However, it is lengthy and to some extent intricate to model systems directly within this system model. Instead we propose a coherent set of description techniques, that do not only exhibit a formal syntax, but also a formal semantics based on the system model. This is done within the SYSLAB project, for which the system model is a vital part.

References

Broy, M., Dederichs, F., Dendorfer, C. and Weber, R. (1991). Characterizing the behaviour of reactive systems by trace sets, *Technical Report SFB 342/2/91 A, TUM-I9102*, Technische Universität München, Institut für Informatik.

Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. and Weber, R. (1993). The Design of Distributed Systems - An Introduction to FOCUS,

Technical Report SFB 342/2/92 A, Technische Universität München, Institut für Informatik.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P. (1994). *Object-Oriented Development - The Fusion Method*, Prentice Hall, Inc., Englewood Cliffs, New Jersey.

Grosu, R. and Rumpe, B. (1995). Concurrent timed port automata, *Technical Report TUM-I 9533*, Technische Universität München.

Grosu, R. and Stølen, K. (1995). A Denotational Model for Mobile Point-to-Point Dataflow Networks, *Technical Report TUM-I 9527*, Technische Universität München.

Lynch, N. and Stark, E. (1989). A proof of the kahn principle for input/output automata, *Information and Computation* **82**: 81–92.

Milner, R. (1989). *Communication and Concurrency*, Prentice Hall, Inc., Englewood Cliffs, New Jersey.

Milner, R. (1991). The polyadic π-calculus: A tutorial, *Technical Report ECS-LFCS-91-180*, Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburg EH9 3JZ.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1992). *Object-Oriented Modeling and Design*, Prentice Hall, Inc., Englewood Cliffs, New Jersey.

Prof. Dr. Manfred Broy is full professor of computing science at the Technical University of Munich. His research interests are software and systems engineering comprising both theoretical and practical aspects. This includes system models, specification and refinement of system components, specification techniques, development methods and verification. He is leading a research group working in a number of industrial projects that try to apply mathematically based techniques and to combine practical approaches to software engineering with mathematical rigor. Professor Broy is a member of the European Academy of Sciences. In 1994 he received the Leibniz Award by the Deutsche Forschungs Gemeinschaft.

Cornel Klein studied computer science at the Technical University of Munich. Since 1993 he is a member of the research group of Prof. Broy. His research interests include the formal foundation of practical software engineering methods, prototyping based on formal specifications, and componentware technologies.

Bernhard Rumpe studied mathematics and computer science at the Technical University of Munich. Since 1992 he is a member of the research group of Prof. Broy. His research interests include the formal foundation of distributed object oriented software engineering methods, especially incorporating behavioral specifications into the software engineering process.