TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Specification of an Elevator Control System

Frank Strobl and Alexander Wisspeintner

TUM-I9906
SFB-Bericht Nr. 342/04/99 A
März 99

TUM–INFO–03-I9906-200/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Specification of an Elevator Control System

An AutoFocus Case Study

Frank Strobl and Alexander Wisspeintner

stroblf@informatik.tu-muenchen.de

wisspein@informatik.tu-muenchen.de

Institut für Informatik
Technische Universität München

Abstract

In this paper we present a case study with AutoFocus, a tool prototype for the development of distributed embedded systems. We develop a controller of an elevator system using different description techniques to illustrate the development process. Furthermore we use the simulation component of AutoFocus, SimCenter, to validate the behavior of the specified system. Using a device independent interface SimCenter can control both external multimedia applications for visualization as well as real hardware for rapid prototyping. We use the AutoFocus specification of the elevator control system to control a Fischertechnik model of an elevator.

Acknowledgement

Many people helped us during the development of this case study. We would especially like to thank Franz Huber for his help as an expert of the AutoFocus CASE tool. He is involved in the AutoFocus project since its beginning.

Special thanks go to Alexander Schmidt and Jan Philipps, who built the Fischertechnik elevator model, supported us in many hardware specific questions and helped us writing this paper.

Finally we want to thank Marc Sihling. He gave us the first introduction to AutoFocus and helped us writing the documentation of this case study.

Contents

1 Introduction

For the last few years the importance of the development of software for embedded systems has been rapidly increasing. More and more of the functionality of embedded systems is realized by software instead of using expensive specialized hardware solutions. This tendency will continue due to the decreasing hardware costs on the micro-controller market.

In the early years of embedded systems, software development was focused on optimizing the program code to cope with the limited hardware resources. Today advanced micro-controllers are available, that remove many memory space and performance restrictions and allow more complex software solutions.

To cope with the complexity of embedded systems software it is important to ensure a clear structure of the views of the description techniques and the development process itself [2]. Both of these aspects should be supported by suitable development tools. In addition, features like verification of correctness and validation of general statements about the developed system are desirable. Tools supporting the development process in the field of software engineering are called CASE-tools (Computer Aided Software Engineering). These tools support and automate procedures, methods, concepts and notations [10].

AutoFocus is a tool prototype to develop distributed embedded systems. Its origin and background is the formal specification framework Focus [1]. It allows the graphical, structured description and modeling of embedded and distributed systems. A simulation component allows the developer to test the specified software system and to build prototypes for the control software.

The description techniques offered by AutoFocus result in a specific software development process. This process will be illustrated using the example of the specification of an elevator control system. This case study is the result of a students project at the Institute for Computer Science at the Technische Universität München.

In Section 2 of this report we introduce the basic description techniques of AutoFocus.

Section 3 contains an informal presentation of the elevator case study. The development of the elevator control system in AutoFocus is explained in Section 4. Section 5 shows how SimCenter can be used for simulating the specification and to drive the external Fischertechnik model. We conclude with Section 6. An Appendix contains some further diagrams and specifications which for reasons of brevity we have omitted from the main text.

2 The Description Techniques of AutoFocus

To get a complete and structured understanding of a system, one has to examine it using different point of views and different levels of abstractions.

In AutoFocus this is supported by four different description techniques:

- System Structure Diagrams (SSDs),
- State Transition Diagrams (STDs),
- Data Type Definitions (DTDs),
- Extended Event Traces (EETs).

Each description technique shows a different view of the system. Through a common mathematical basis the integration of these views leads to a formal specification of the whole system.

The AutoFocus description techniques support hierarchical development of a system. Components or views can be atomic or they can themselves consist of sub-components or substructures. Complex descriptions can be divided into smaller ones. The developer can describe the system in less detail in early development phases. The system description is then more and more refined during further development until the system is described in all details.

The description techniques offered by AutoFocus are described in the rest of this section.

2.1 System Structure Diagrams

An embedded system is modeled in AutoFocus as a network of components that connected with each other via channels. These channels allow message exchanges between the individual components and between the components and the environment of the embedded system. This network is described with AutoFocus System Structure Diagrams (SSDs).

Figure 2-1 shows a SSD. Squared labeled nodes symbolize components. These components are connected with each other via labeled directed edges, called channels. At both ends of an edge are small cycles called ports. Ports are the interfaces between the different hierarchical layers. A connection to a sub-component, super-component or the system environment is made via a port. Ports that are not part of a component in the SSD belong to the interface to the hierarchically higher component or the environment of the system.

Each component has a name and is connected via a set of ports to a set of input and output channels. A channel is defined through its name and the data type of the corre-

sponding signal. SSDs shows the topological view of the system and the signature of each component.

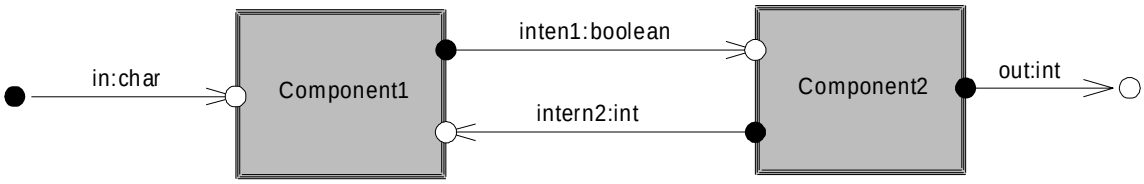

Figure 2-1: System Structure Diagram

Semantically, each channel carries at most one signal at each time point. It is possible to check whether a channel is empty, or whether it carries a specific data value.

Sub-components can be assigned to each component. In this way hierarchical system descriptions can be specified. Moreover, state machine specifications (STDs, see below) can be assigned to each component.

2.2 State Transition Diagrams

State Transition Diagrams (STDs) describe the behavior of a component. STDs are are graphical represenations of extended finite state machines. They consist of states and transitions (Figure 2-2).. States are represented as labeled ellipses; they can be marked to represent start or end states. State changes take place by transitions. Transitions are represented as directed edges between states.

Each transition can have a pre and post condition over local variables, an input expression to read signals from the component's input channels, an output expression to send signals via the component's output channels. In addition, transitions can be labeled.

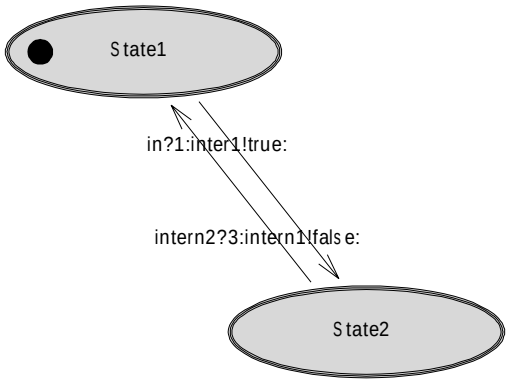

Figure 2-2: State Transition Diagram

Semantically, if the pre condition and the input expression are true, a transition is triggered. In this case the signals described in the output expression are sent via channels. Furthermore the post condition is evaluated. For simulation purposes, post conditions

are just a sequence of assignments to local state variables. More general post conditions cannot be simulated, but can be useful for verification and validation tasks.

Table 2-1 shows the definition of the simple transition with the label “trans1”.

label	pre	input	output	post
trans1	t > 0	ch1?true, ch2?false	ch3!true	t=0

Table 2-1: Example transition

If the local variable t is greater the 0 and the signals “true” and “false” are read from the channels “ch1” and “ch2”, this transition is fired. This results in sending the signal “true” via “ch3”. Furthermore the value 0 is assigned to the variable “t”.

If more than one transition can be triggered (i.e., its precondition and input statement are valid), one of the transitions is non-deterministically chosen. It is up to the developers to specify the state machines to get deterministic behavior of the whole system.

STDs are hierarchical, too: each state can contain a sub-STD. In AutoFocus, the hierarchical structure of hierarchical state machines is semantically mapped to a single flat state machine.

The state machines in the components of an SSD are executed in parallel. AutoFocus uses a global system clock to synchronize the whole system. Each machine can make exactly one transition per clock cycle and all transitions within one clock cycle happen simultaneously.

2.3 Data Type Definitions

It is possible to define new data types in AutoFocus. This is done in a style similar to functional programming languages, such as Gofer. These data types can be used to define local variables within components and the signal types for communication channels.

2.4 Extended Event Traces

In addition to STDs there is a second description technique for behavior in AutoFocus: Extended Event Traces (EETs) [14] describe the communication between different components in a system run. The notation is a variant of the standardized Message Sequence Charts [13]. Each EET is assigned to a SSD and shows the temporal flow of the communication between the components of the SSD and its environment (Figure 2-3).

EETs can be used for different purposes. They can describe elementary functionality at an early development stadium to validate early requirements. Furthermore, the simulation component SimCenter (Section 4.1) can generate EETs automatically during a simulation session. EETs can then be used as a simulation protocol to support the troubleshooting process.

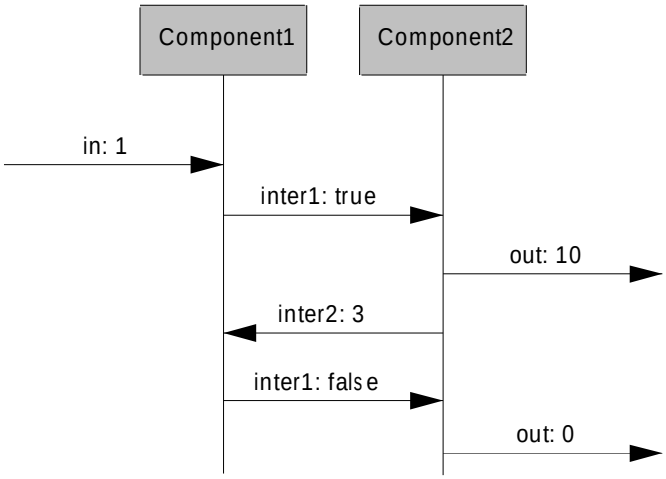

Figure 2-3: Extended Event Trace

3 The Elevator Control System

To illustrate the advantages of AutoFocus and SimCenter, the classic example of an elevator control system has been specified. This specification was part of our students project at the Technische Universität München. An elevator control system has many of the typical aspects of an embedded and distributed system, and it has a number of time or safety critical requirements. This system is sufficient complex to allow to draw some conclusions about realizing more complex projects with AutoFocus.

First we specify the static structure of our elevator control system with AutoFocus. The behavior can be specified in SSDs. During the whole development process parts of the system can be simulated and tested with SimCenter. Additional multimedia front end applications can be used to create a realistic test environment. In this way it is possible to develop a first prototype of the control software. Finally, a hardware interface allows us to test the software system in its real environment. The specified software directly controls an elevator model.

In the rest of this section the individual steps of the design process are described.

3.1 Requirements

The goal of our development of the elevator control system is to control an existing elevator model. This model is built with Fischertechnik components. Consequently this model defines part of the requirements on the control system. The elevator has four floors and each floor has its request button and a control light. A sensor is located at every floor. We can use these sensors to locate the current position of the elevator car. The car itself consists of several parts: A door, which can be opened and closed by a motor. Two sensors inform the control system about the door position. A light sensor can detect objects while the door is closing. The elevator car engine moves the car up or down.

Because of space limitations in the Fischertechnik there are some restrictions. The individual floors do not have their own door in contrast to most real elevator systems. Each floor has only one request button (not two, as is common in modern elevators), and the car itself does not have any request button. However, when there is only one button at each floor, the functionality of these car buttons is often equivalent to that of the corresponding floor button.

Informally, the elevator behavior is defined as follows. If you press the request button at a floor, the request light is switched on. The car moves to this floor within finite time. When the floor is reached, the door opens, and the request lamp is turned off again. The door stays open for some time (10 seconds) to allow passengers to enter or exit the car. After this time, the door closes again. If, while the door closes, the optical sensor is interrupted or the floor's request button is pressed by a user, the door must open again immediately. After a shorter waiting time (5 seconds) the door closes again.

The car is moved by the elevator motor. If the car should stop at a certain floor, the motor is stopped immediately after the reception of a signal from the corresponding floor sensor. The control system should send an additional signal to the motor, if the motor should stop the car at the next floor. The motor uses this signal to reduce the car speed. This enables the motor to stop the car at the exact floor position.

The requests from the individual floors can be served using different strategies. The most important requirement for such a strategy is fairness. Every request must be served in finite time. The strategy used in this specification is described in detail in Section 3.6

3.2 Analysis

Before we specify the elevator control system in AutoFocus, we split the large starting problem into smaller sub problems.

First we look at a real elevator system, and determine its components. These components will be mapped to the elevator control system specification. Figure 3-1 shows the general model of an elevator. The elevator consists of two or more floors, a car and a control system. Each floor has its request button and an elevator door. Every elevator door owns a motor to open and close it. Door sensors are used to determine the door position. Every button has an assigned control light. This light displays a request from the corresponding floor. The car itself consists of several buttons, one car door and a motor to move the car up and down. The component “Elevator Control System” controls the behavior of the whole elevator system. This component decides which request is served next (Section 3.1). The interface between the elevator control system and the other components of the elevator system defines our system border of the embedded system.

In this case study we do not specify a general elevator control system. Rather we develop a special control system for the elevator model described in Section 3.1. This model has four floors, no car buttons, and no floor doors.

3.3 Architectural Design

Now we can make a rough draft of the elevator control system. We use the general model of an elevator system and map it to our Fischertechnik elevator model. Figure 3-2 shows the resulting component structure. This structure will be retained in essence during the whole specification process. We do not make any statements about the communication structure at this phase. This structure will be defined later on using the AutoFocus SSDs.

We refine the elevator control into two components. The new components are the car component “Elevator Control” and the component “Floors”. “Floors” consists of four floor control systems, which observe the request button with its control light and forward requests to the central control system. The component “Elevator Control” consists of three control components Each of these components is assigned to subsystem of the elevator.

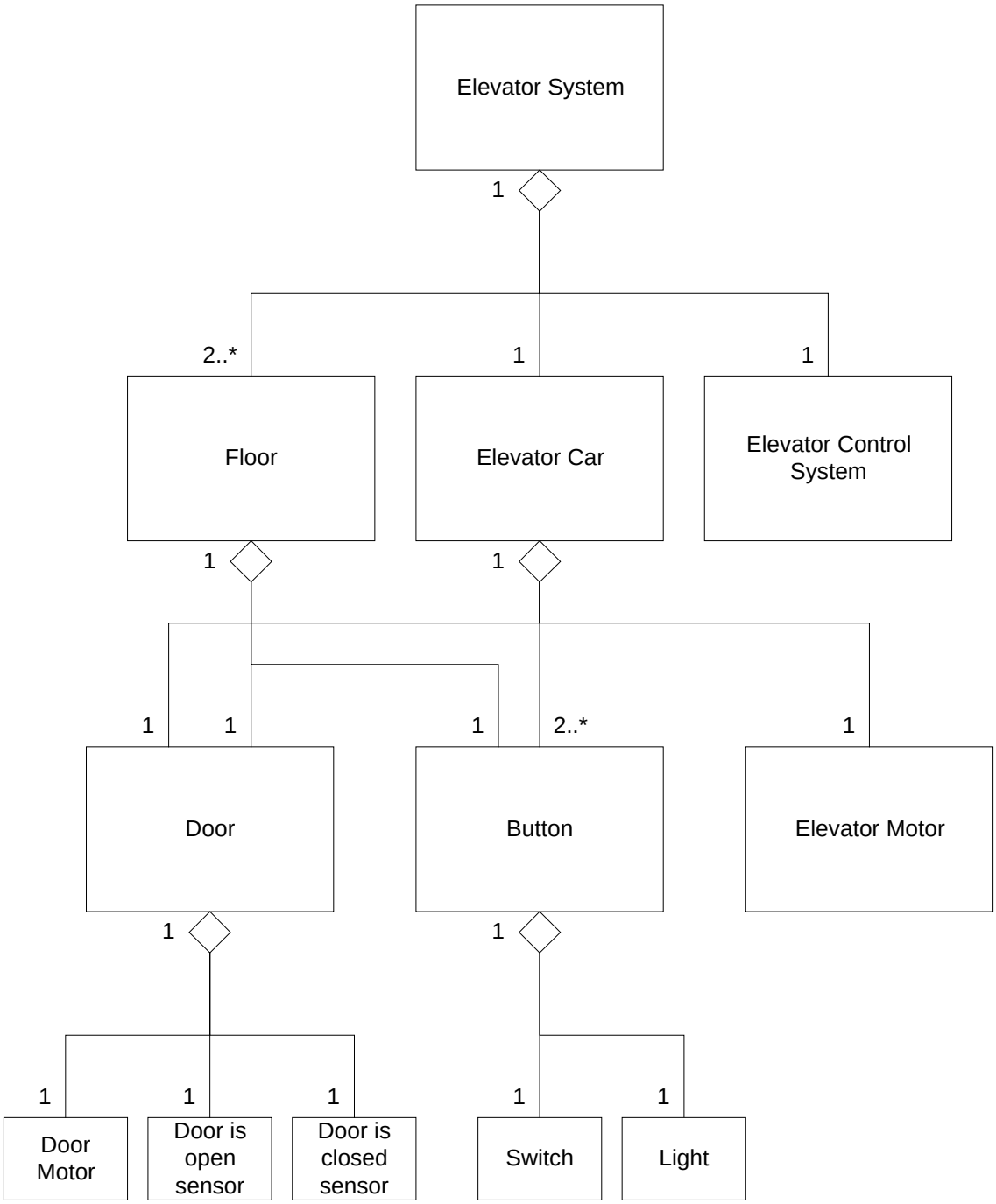

Figure 3-1: General model of an elevator system with one car in UML class diagram notation [10]

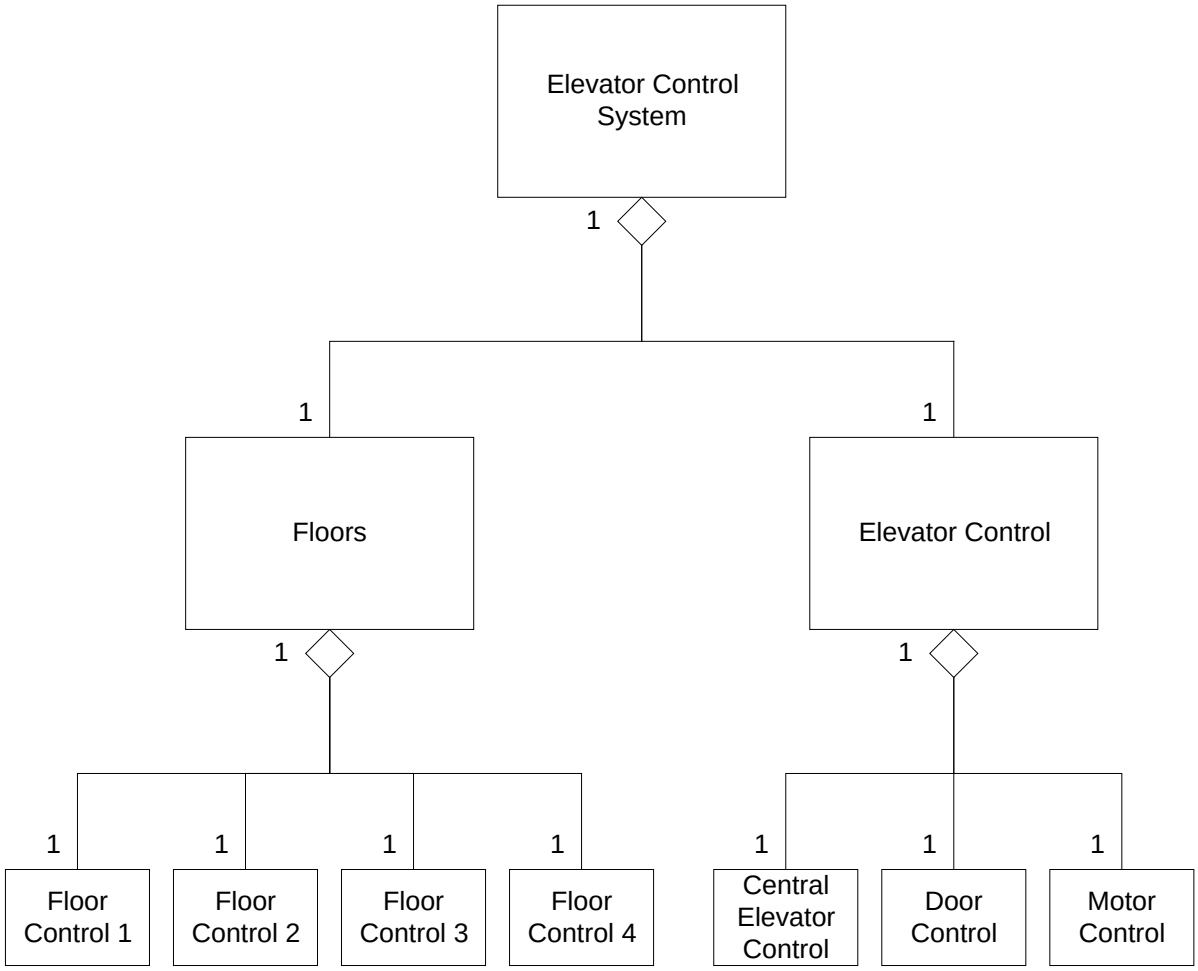

Figure 3-2: Model of the specific elevator control system in UML class diagram notation [10]

The main component of the elevator control system is “Central Elevator Control”. This component realizes the strategy for serving the requests. Furthermore this component controls the other units, “Door Control” and “Motor Control”.

It is actually not necessary to create class diagrams like shown in this section when working with AutoFocus. You can also specify the structure directly in AutoFocus using SSDs. But in SSDs it is not possible to illustrate the different hierarchical layers in a single diagram. A SSD only shows one hierarchical layer of the system. For this reason it is meaningful to make a draft of the component structure before starting with the specification in AutoFocus.

3.4 Creating a new Project in AutoFocus

Given the architectural structure of the system from Section 3.3, we can relize the following design using AutoFocus.

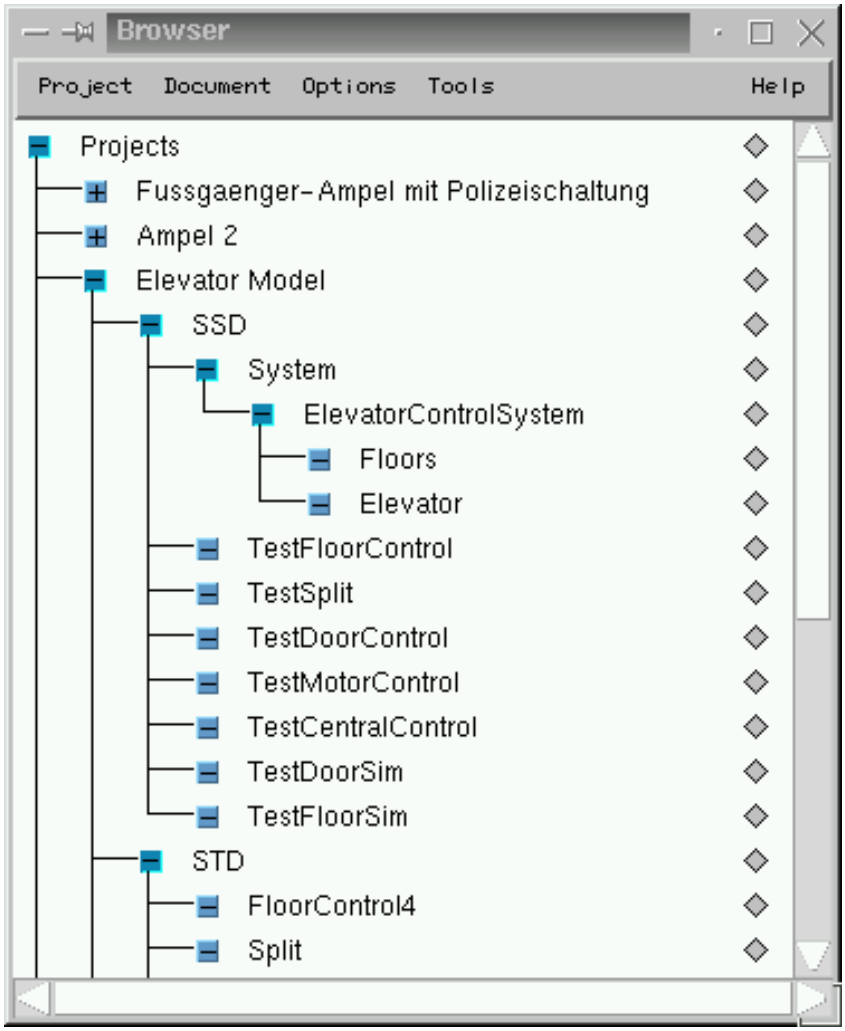

Figure 3-3: Project Browser in AutoFocus

First we create a new project in the project browser. You can manage all documents in this central window. Thanks to a version control system (RCS), it is possible to manage different versions of the specification documents. Figure 3-3 shows a screenshot of the project browser. This screenshot has been made during the development of the elevator control system.

3.5 Interface and Structure Specification

We use System Structure Diagrams (SSDs) to describe the static model in AutoFocus. The static model describes the system components and the communication structure between them. Now we use the model from Section 3.3 and map it to a more detailed structure specification using AutoFocus and SSDs.

3.5.1 SSD „System“

First we design the top view of our elevator control system. The interface between the control system and the environment is defined in this diagram. We adopt the component “Elevator Control System” from Figure 3-2 (analysis phase) and place it into the system SSD (Figure 3-4). Then all signals needed for controlling the elevator are added.

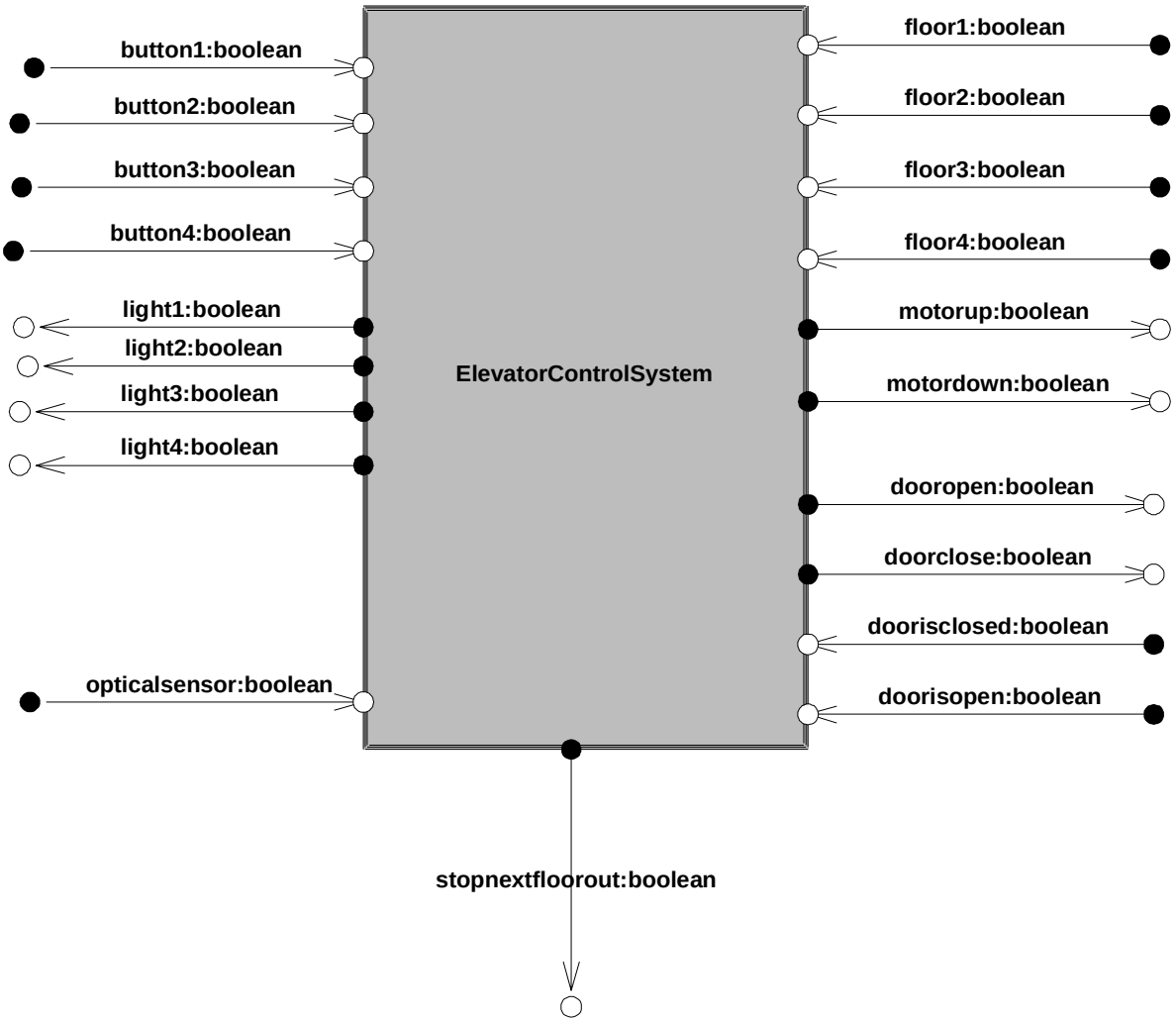

Figure 3-4: SSD „System“

We only use communication channels with the data type “Boolean” in this SSD: A “Boolean” channel can transmit the signals “true” and “false”. This correspond directly to the voltages 0 and 5 Volt used by the hardware model. The channels “button1-4” are connected with the request buttons at the single floors. If the user presses a button, the signal “true” should be transmitted via the corresponding channel. The channels “light1-4” controls the request lamps at the different floors. If a continuous signal “true” is sent via a light channel, the light turns on. A continuous signal “false” turns the request light off. In this connection a continuous signal is a signal that is repeatedly sent every clock

cycle. The floor sensors of the elevator model communicate via the channels “floor1-4” with the elevator control system. They inform the control system about the current position of the elevator car. We use the channels “dooropen” and “doorclose” respectively “doorisopen” and “doorisclosed” to control the elevator door. If the light sensor detects an object between the car door and the door frame, it sends a message to the control system via the channel “lightsensor”. The control system uses the channel “stopnextfloorout” to announce a motor stop event at the next floor. Finally there are the channels “motorup” and “motordown”. We use these channels to control the elevator motor that moves car up and down.

3.5.2 SSD „Elevator Control System“

In the next step we further refine the component “Elevator Control System” of the SSD “System”. It is split into the sub components “Floors” and “Elevator” (Figure 3-5). This division meets the structure of a real elevator system, that consists of a car with an integrated control system and several floors.

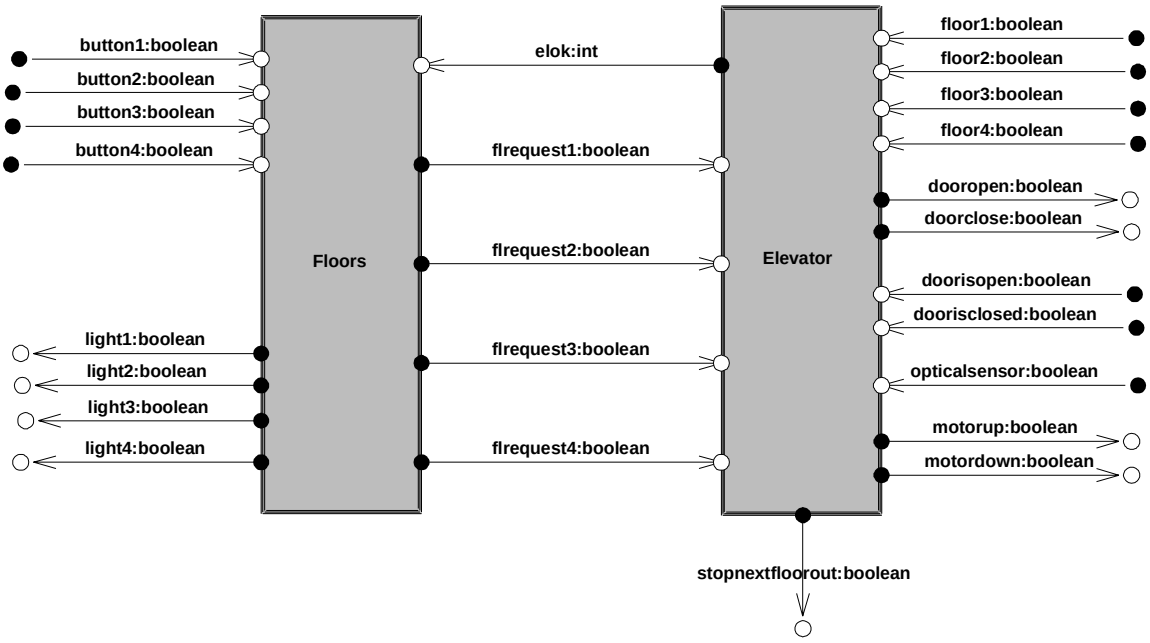

Figure 3-5: SSD „Elevator Control System“

There are several channels in this SSD that are only connected at one side to an other component. These channels communicate with the higher hierarchical layer “System” and define in this way the interface of the component “Elevator Control System”. The channels for component-internal communication between “Floors” and “Elevator” are new. We use the channels “flrequest1-4” to send new requests from the single floors to the central control system. The central control system uses the channel “elok” to clear floor requests after they have been served.

3.5.3 SSD „Floors“

The component “Floors” contains the floor control systems of each floor (Figure 3-6). These single control systems are named “FloorControl1-4”. If the user presses a request button, the signal “true” is sent via the corresponding “button” channel. The floor control system must turn on the request lamp by sending a continuous signal via one of the “light1-4” channels. Furthermore the floor control system sends a message about the new request via one of the “flrequest1-4” channels. We use continuous signals again for this task. The channels “flrquest1-4” acts like a kind of memory storing the current request from the four floors.

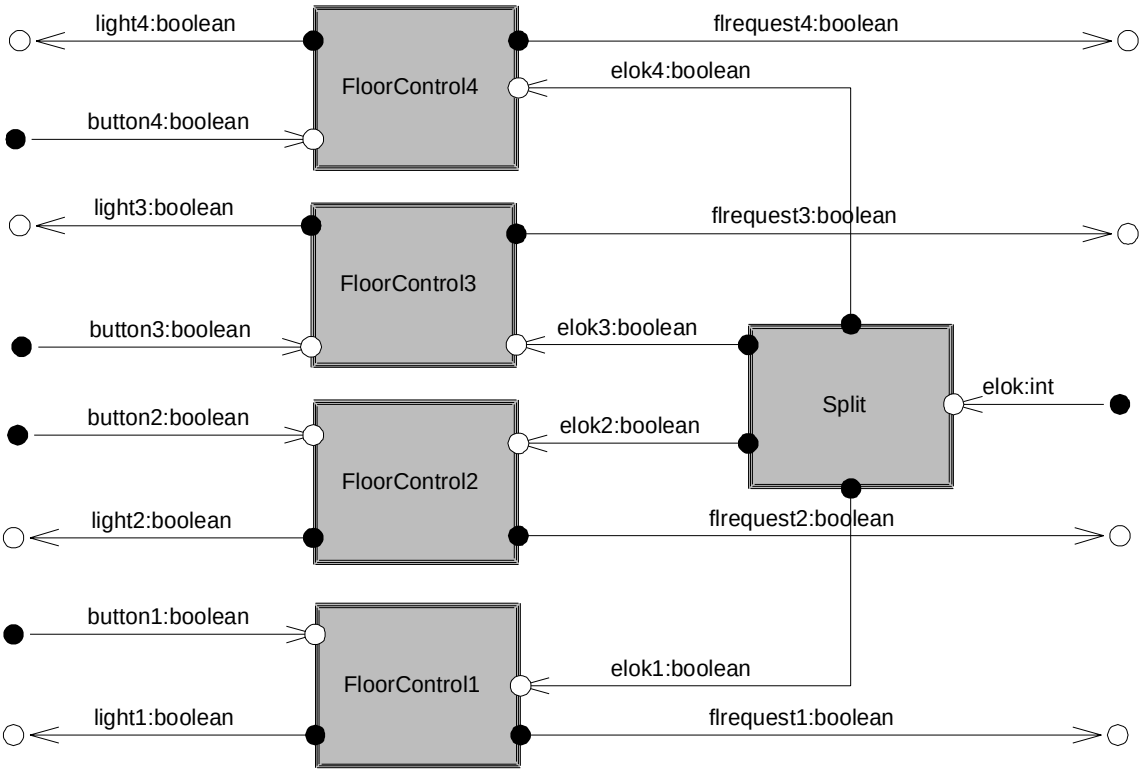

Figure 3-6: SSD „Floors“

When the component “Elevator” sends a floor number via the channel “elok” to the floor control system, the component “Split” forwards this message to the corresponding “FloorControl” component, which then switches off the request lamp and clears the request (by starting to send a continuous “false” via the corresponding “flrequest” channel).

3.5.4 SSD „Elevator“

In the component “Elevator” (Figure 3-7) the real control tasks are realized. It consists of the components “Central Elevator Control”, “Door Control”, “Motor Control” and “Stop Next Continue”.

The central control system can send a signal to the motor control system via the channel “motorcom” that controls the moving direction of the car. „Motor Control“ should generate the signals “motorup“ and “motordown“ for direct motor controlling.

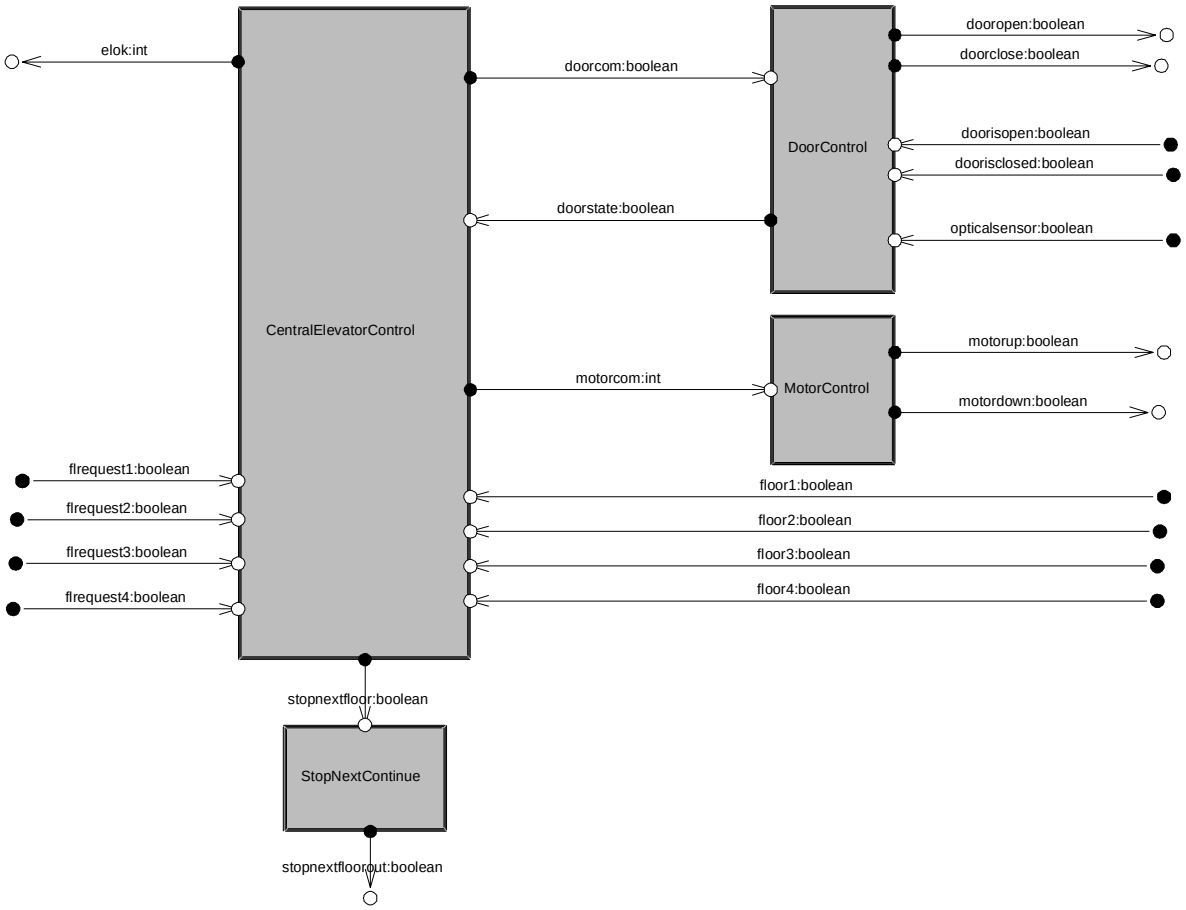

Figure 3-7: SSD „Elevator“

The central control system can instruct the door control system to open the door. This is done by sending a signal via the “doorcom” channel. In the opposite direction the central control system can check if the door is open at the moment via the “doorstate” channel. The component “Stop Next Continue” creates a continuous signal “stopnextfloorcont” out of the signal “stopnextfloor”. This continuous signal is used by the motor as described in Section 3.1.

3.6 Specification of the behavior

After specifying the static components and communication channels, we describe the behavior of the elevator system. We specify state machines by STDs and assign them to the SSD components.

3.6.1 STD "Floor Control"

First we describe the behavior of the components „Floor Control 1-4". They just observe the request buttons and lights and forward requests to the central control station. The STDs only have two states („LightOff" and „LightOn") and their functionality can be compared with a switch. In the following example we describe the state machine of the component „Floor Control 1" (Figure 3-8). The state machines of the components „Floor Control 2-4" are similar.

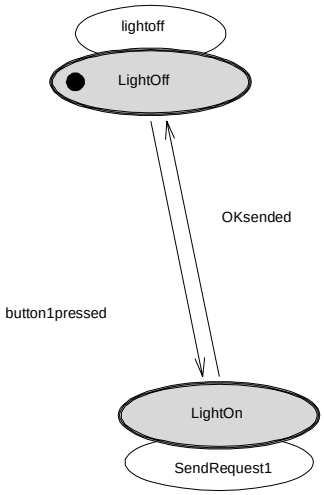

Figure 3-8: STD „Floor Control 1"

Table 3-1 shows all the transitions with their labels, conditions and output in the AutoFocus notation.

label	pre	Input	output	post
button1pressed		Button1?true	light1!true, flrequest1!true	
lightoff		Button1?	flrequest1!false, light1!false	
Oksent		elok1?true	flrequest1!false, light1!false	
Sendrequest1		elok1?	light1!true,flrequest1!true	

Table 3-1: Transitions of „Floor Control 1"

After initialization of the elevator system, „Floor Control 1" enter the state „LightOff", which means that there is no elevator request for the first floor and its request light is off. Until the request button is pressed, the system stays in this state and sends the signal „false" in the transition „lightoff" through the channels „light1" and „flrequest1". When the button is pressed the transition „button1pressed" triggers and the state machine changes to the state „LightOn". Now the signal „true" is sent through the channels „light1" and „flrequest1" in the transition „SendRequest1". The floor light is switched on and an request signal is sent to the central control system. When the request is serviced, i.e. when the elevator car arrived at the floor, the central control system sends the signal „true" and the state machines changes back to the state „LightOff".

3.6.2 STD „Split“

The next step is to specify the behavior of the component „Split“. Together with the components „Floor Control 1-4“ it forms the main component „Floors“. After describing the behavior of „Split“, also the behavior of the component „Floors“ is completely defined.

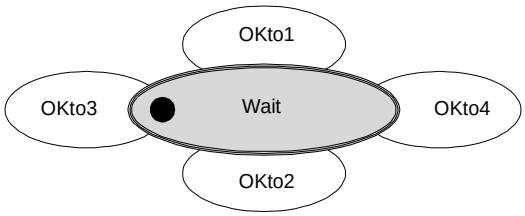

Figure 3-9: STD „Split“

When the central control system has finished working on an request, it sends the signal „elok“ to the floor control. This signal has the type integer and transmits the corresponding floor number of the finished request.

label	pre	input	output	post
OKto1		elok?1	elok1!true	
OKto2		elok?2	elok2!true	
OKto3		elok?3	elok3!true	
OKto4		elok?4	elok4!true	

Table 3-2: Transitions of „Split“

The component „Split” (Figure 3-9, Table 3-2) receives this signal and passes it on to the corresponding floor control. For example if the value of „elok“ is three, the signal „true“ is sent to the component „Floor Control 3“.

3.6.3 STD „Door Control“

After specifying the behavior of the component „Floors“ with all its details, we start describing the behavior of the sub components of the component „Elevator“. The components „Door Control“ and Motor Control“ are relatively simple compared to the central component „Central Elevator Control“. We describe them first.

The component „DoorControl“ controls the car door. The requests to shut or open the door are sent from the central elevator control system. When the elevator car reaches a floor where a request is sent, it stops and the central elevator control sends the signal „true“ through the channel „doorcom“ to the component „Door Control“, to start the service of that floor.

When it receives this signal, „Door Control“ opens the car door, waits until the passengers have entered or left the car and finally shuts the door again. When the door is closed, “Door Control“ sends the signal “true“ through the channel “doorstate“ and the central elevator control system services the next request.

Figure 3-10 shows the state machine of the component “Door Control”. The corresponding transitions are described in Table 3-3. The state “Init” has a special purpose, because it is used to make sure that the car door is closed and so the elevator system is in a well-defined state when the system is started.

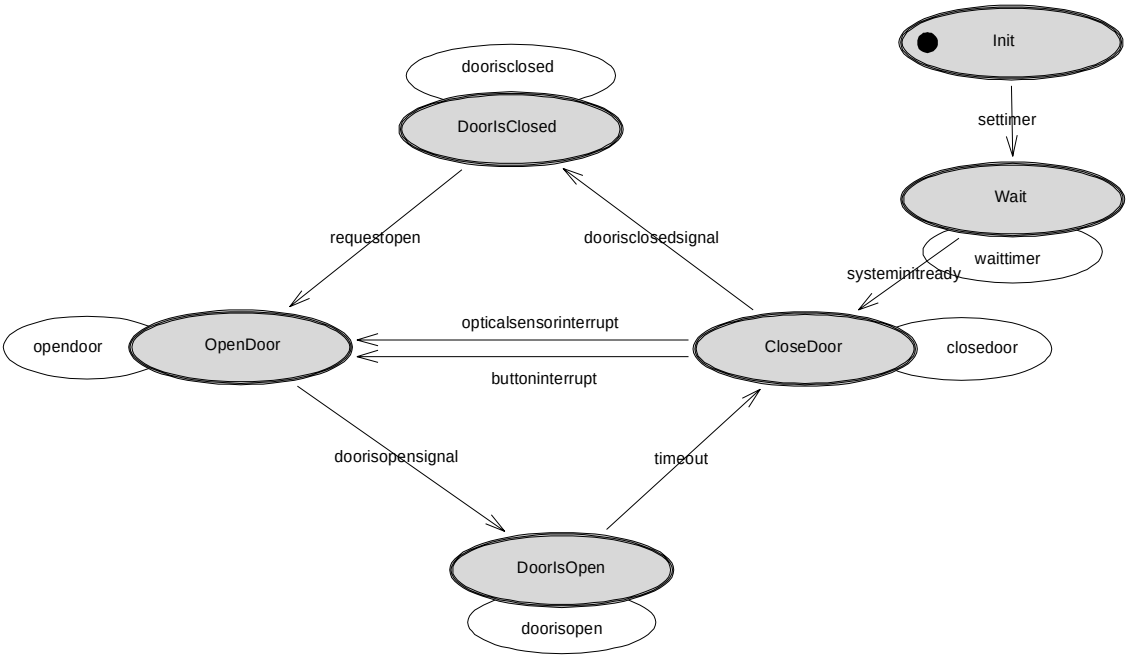

Figure 3-10: STD „Door Control“

label	pre	input	Output	post
buttoninterrupt		doorcom?true	doorstate!false, doorclose!false, dooropen!true	Timer=3
closedoor		doorisclosed?, opticalsensor?,doorcom?	doorclose!true, dooropen!false, doorstate!false	
doorisclosed		doorcom?	doorstate!true, dooropen!false, doorclose!false	
doorisclosedsignal		doorisclosed?true	doorstate!false, dooropen!false, doorclose!false	
doorisopen	Timer>0		doorstate!false, dooropen!false, doorclose!false	Timer=Timer-1
doorisopensignal		doorisopen?true	doorstate!false, dooropen!false, doorclose!false	
opendoor		doorisopen?	dooropen!true, doorclose!false, doorstate!false	
opticalsensorinterrupt		opticalsensor?true	doorstate!false, doorclose!false, dooropen!true	Timer=3
requestopen		doorcom?true	doorstate!false, dooropen!true, doorclose!false	Timer=10
settimer				Timer=5
systeminitready	Timer==0			
timeout	Timer==0		doorclose!true, doorstate!false, dooropen!false	
waittimer	Timer>0			Timer--

Table 3-3: Transitions of „Door Control“

The local variable “Timer” is used as a counter, which ensures that the car door stays open for certain time after it is opened. Within the initialization the counter is also used to guarantee that all simulation components have finished their initialization.

The two transitions “opticalsensorinterrupt” and “buttoninterrupt” implement the requirements to immediately open the door, when either the optical sensor is interrupted (i.e., the signal “true” is sent through the channel “opticalsensor”) or the request button is pressed again while the door is closing (the central control system again sends an signal “doorcom” to the door control). In both cases the state machine changes to the state “OpenDoor” and opens the door again.

3.6.4 STD „Motor Control“

The component “Motor Control” is used as the control system of the car motor. Figure 3-11 shows the corresponding state machine. Table 3-4 shows its transitions.

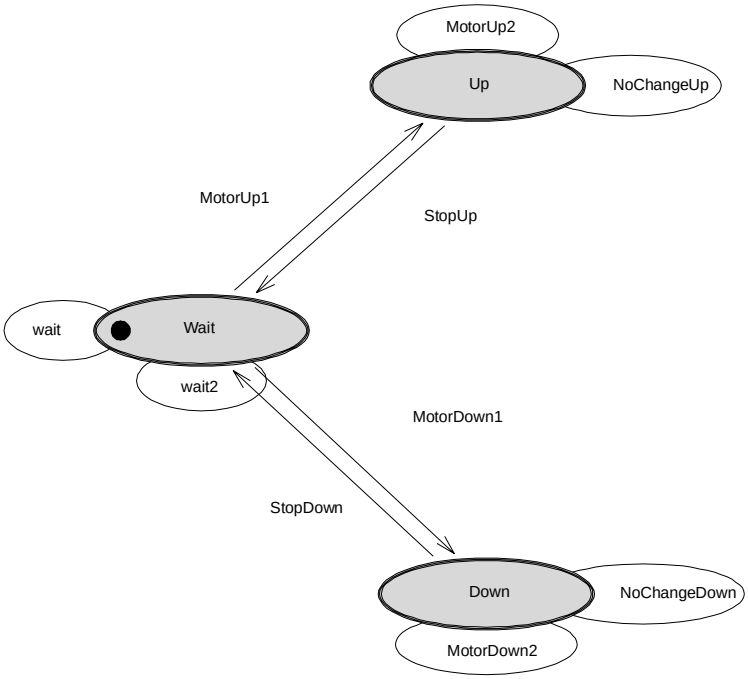

Figure 3-11: STD „Motor Control“

Through the channel “motorcom”, which has the type “integer”, the central elevator control system sends the direction into which the car is to move. The value “1” means that the car has to be moved upwards and “2” means downwards. To stop the motor the number “2” is sent through the channel. The component “Motor Control” transforms this signals into continuous signals to control the motor through the channels “motorup” and “motordown”.

label	pre	input	output	post
motordown1		motorcom?2	motorup!false, motordown!true	
motordown2		motorcom?2	motordown!true, motorup!false	
motorup1		motorcom?1	motorup!true, motordown!false	
motorup2		motorcom?1	motorup!true, motordown!false	
nochangedown		motorcom?	motordown!true, motorup!false	
nochangeup		motorcom?	motorup!true, motordown!false	
stopdown		motorcom?0	motorup!false, motordown!false	
stopup		motorcom?0	motorup!false ,motordown!false	
wait		motorcom?	motorup!false, motordown!false	
wait2		motorcom?0	motorup!false, motordown!false	

Table 3-4: Transitions of „Motor Control“

3.6.5 STD “Central Elevator Control“

The STD “Central Elevator Control” controls the other components. Furthermore it implements the strategy to handle the incoming requests. We use the following strategy, which is relatively easy to implement and used in most real elevator systems. The elevator car has a priority direction. The car is moved in this direction until no more requests come from this direction. Now the priority direction changes and the elevator can work on requests from the other direction.

The corresponding state machine is quite complex and so it uses hierarchy. Figure 3-12 shows the main state machine. The states “Init Elevator” and “Search Request” contain sub-STDs (Figure 3-13, Figure 3-14).

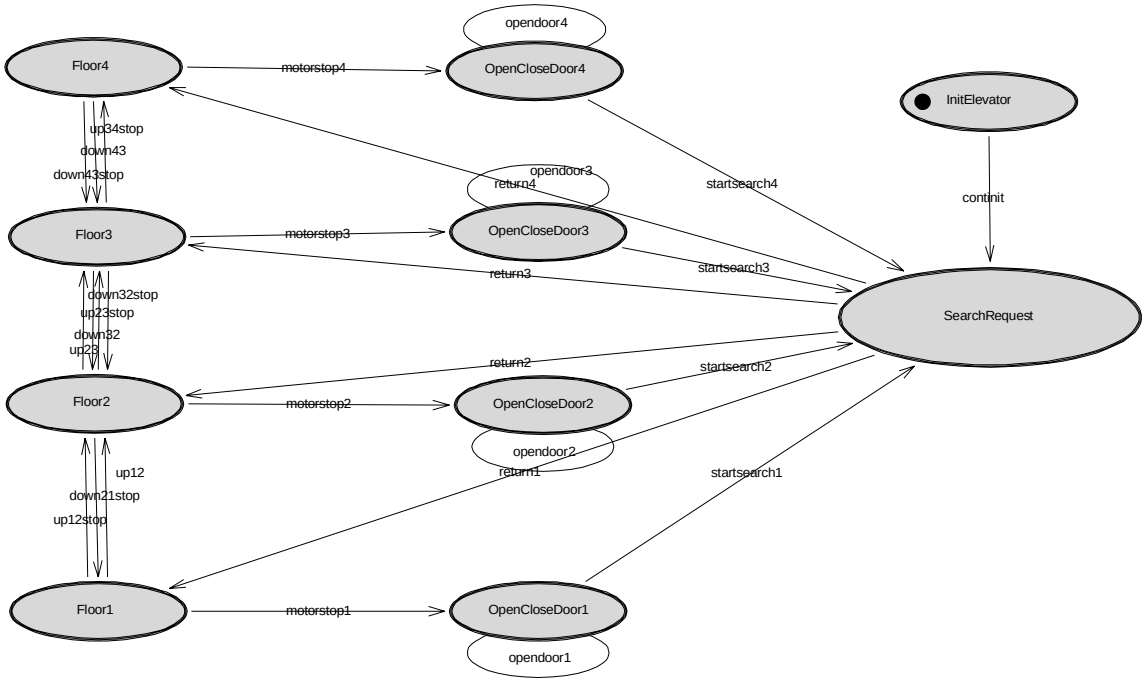

Figure 3-12: STD „Central Elevator Control“

label	pre	input	output	post
continit				stnextfloor=true
down21stop	(direction==2)&& (stnextfloor==false)	doorstate?true, floor2?true	motorcom!2, stopnextfloor!true	currfloor=1; stnextfloor=true
down32	(direction==2)&& (stnextfloor==false)	doorstate?true, floor3?true, flrequest2?false	motorcom!2, stopnextfloor!false	currfloor=2; stnextfloor=false
down32stop	(direction==2)&& (stnextfloor==false)	doorstate?true, flrequest2?true, floor3?true	motorcom!2, stopnextfloor!true	currfloor=2; stnextfloor=true
down43	(direction==2)&& (stnextfloor==false)	doorstate?true, floor4?true, flrequest3?false	motorcom!2, stopnextfloor!false	currfloor=3; stnextfloor=false
down43stop	(direction==2)&& (stnextfloor==false)	doorstate?true, flrequest3?true, floor4?true	motorcom!2, stopnextfloor!true	currfloor=3; stnextfloor=true
motorstop1	stnextfloor==true	flrequest1?true, floor1?true	motorcom!0, elok!1, doorcom!true	
motorstop2	stnextfloor==true	flrequest2?true, floor2?true	motorcom!0, elok!2, doorcom!true	
motorstop3	stnextfloor==true	flrequest3?true, floor3?true	motorcom!0, elok!3, doorcom!true	
motorstop4	stnextfloor==true	flrequest4?true, floor4?true	motorcom!0, elok!4, doorcom!true	
opendoor1		flrequest1?true	doorcom!true, elok!1	
opendoor2		flrequest2?true	doorcom!true, elok!2	
opendoor3		flrequest3?true	doorcom!true, elok!3	
opendoor4		flrequest4?true	doorcom!true, elok!4	
return1	currfloor==1			
return2	currfloor==2			
return3	currfloor==3			
return4	currfloor==4			
startsearch1		doorstate?true, flrequest1?false		
startsearch2		doorstate?true, flrequest2?false		
startsearch3		doorstate?true, flrequest3?false		
startsearch4		doorstate?true, flrequest4?false		
up12	(direction==1)&& (stnextfloor==false)	doorstate?true, floor1?true, flrequest2?false	motorcom!1, stopnextfloor!false	currfloor=2; stnextfloor=false
up12stop	(direction==1)&& (stnextfloor==false)	doorstate?true, floor1?true, flrequest2?true	motorcom!1, stopnextfloor!true	stnextfloor=true; currfloor=2
up23	(direction==1)&& (stnextfloor==false)	doorstate?true, floor2?true, flrequest3?false	motorcom!1, stopnextfloor!false	currfloor=3; stnextfloor=false
up23stop	(direction==1)&& (stnextfloor==false)	doorstate?true, flrequest3?true, floor2?true	motorcom!1, stopnextfloor!true	currfloor=3; stnextfloor=true
up34stop	(direction==1)&& (stnextfloor==false)	doorstate?true, floor3?true	motorcom!1, stopnextfloor!true	currfloor=4; stnextfloor=true

Table 3-5: Transitions of „Central Elevator Control“

In the state “Init Elevator” the whole elevator system is initialized. This ensures that tthe elevator begins its service in a well-defined state: when the car stays at the ground floor. After initialization the state machine changes to the state “Search Request”. In the corresponding state machine (Section 3.6.7) all floors are searched for requests. The local variable “currfloor” stores the number of the current floor. If an request is found the state machine changes to state which corresponds the current state (“Floor 1-4”). Furthermore the variable “direction” shows from which direction the request was sent. The car moves in that direction and in every floor the next floor is inspected whether the elevator car has to be stopped or not. If not the state machine changes to the next floor state using the transitions “upxy” or “downxy” and the floor variable “currfloor” is reset. In the next state the procedure is repeated.

If there is a request on the next floor a signal is sent through the channel “stopnextfloor” and the local variable “stnectfloor” is set to “true”. The state machine changes to the following floor state using the transitions “upxystop” or “downxystop” and the variable “currfloor” is also resetted.

In this state the central elevator system waits until the channels “floor1-4” shows that the elevator car reached the right floor. To stop the motor and to open the door, signals are sent to the components “Motor Control” and “Door Control”. Then the state machine changes to the state “OpenCloseDoor1-4”. While the control system waits until the car door is closed again, the transition “opendoor1-4” can trigger if the request button is pressed in the current state. In this case, an interrupt signal is sent through the channel “doorcom” to the component “Door Control”. When the door is finally closed the state machine changes to the state “Search Request” and starts searching for a new request. Table 3-5 shows the transition in detailed AutoFocus notation.

3.6.6 STD “Init Elevator“

The state machine “Init Elevator” (Figure 3-13 / Table 3-6) is a sub-STD of “Central Elevator Control”. First it waits until the component “Door Control” has finished its initialization, which means that the car door is closed. If the elevator car is already in the first floor, the machine could directly change to state “OK”. Otherwise the motor control is instructed to drive the elevator to the first floor. The transitions “alreadyfloor1” and “arrivedfloor1” are used to assign values to the important local variables.

3.6.7 STD “Search Request“

The state machine “Search request” delivers the next request to the main state machine “Central Elevator Control”. It implements the strategy proposed in Section 3.6.5. Figure 3-14 shows the STD of this state machine. When you entry the state machine, the variable “direction” stores the direction in which the elevator is driving to. This is also the priority direction for the search for the next request.

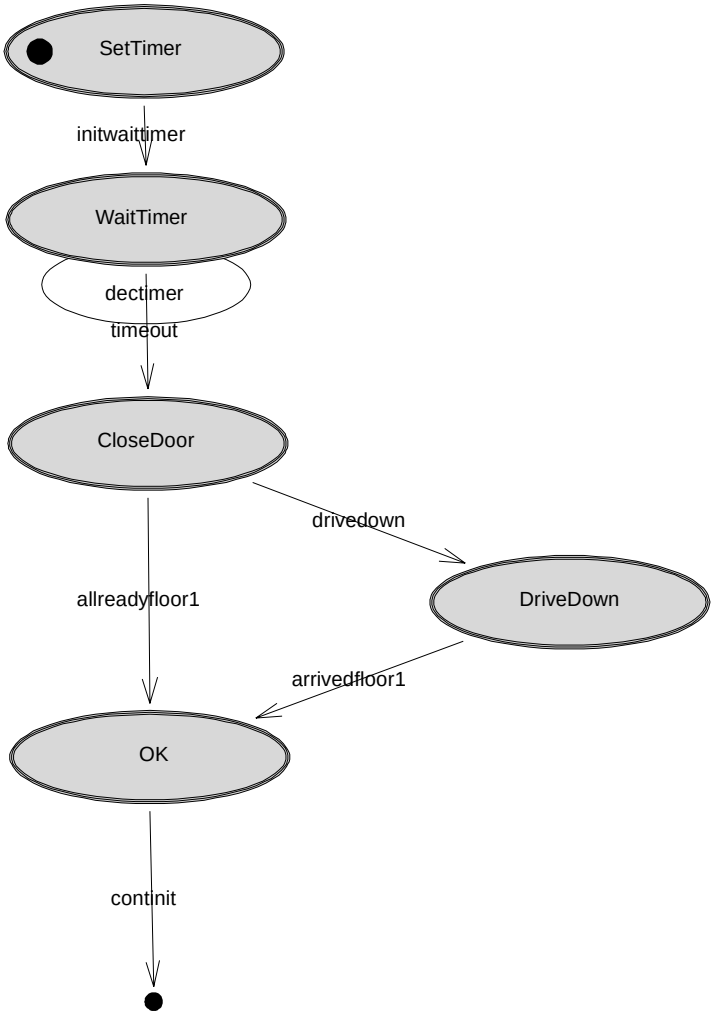

Figure 3-13: STD „Init Elevator“

label	pre	input	output	post
allreadyfloor1		floor1?true, doorstate?true		currfloor=1; direction=1
arrivedfloor1		floor1?true	motorcom!0	currfloor=1; direction=1
continit				stnextfloor=true
dectimer	Timer>0			Timer--
drivedown		doorstate?true, floor1?	motorcom!2	
initwaittimer				Timer=5
timeout	Timer==0			

Table 3-6: Transitions of „Init Elevator“

The search begins in one of the states “FloorX”, depending on the current floor level of the car. The contents of channel “flrequestX” determine whether there is a request for the current floor or not.

If not, the state machine changes to the next floor state in the priority direction. The transitions “search12”, “search23”, “search34” and “search41” are used if the priority direction has the value 1 (up). The transitions “search43“, “search32“, “search21“ and “search14“ are used for the other direction. This is repeated until a request is found.

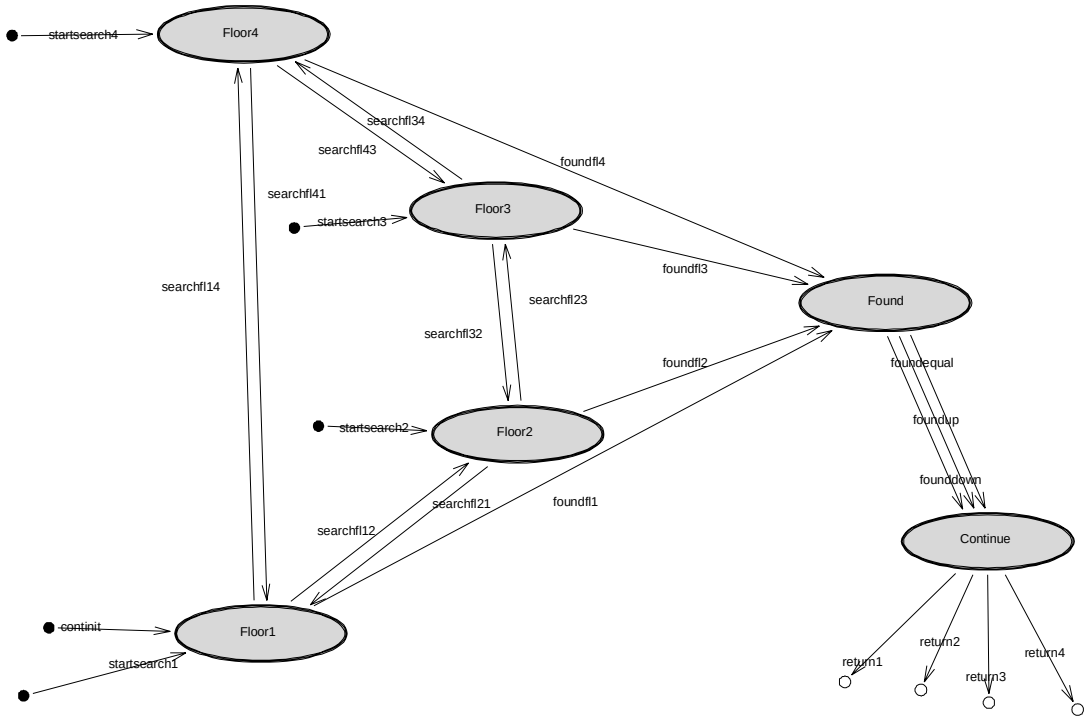

Figure 3-14: STD „Search Request“

If a request is found, on of the “foundflX”-transitions triggers and the number of the floor is stored in the local variable “foundfloor”. In the state “Found” one the transitions “foundequal”, “foundup” and “founddown” triggers, corresponding to the direction where the request has been found. For this the local variable “direction” is set to indicate in which direction the car is to move; this determines the new priority direction, which is used at next call of “Search Request”. The transitions “returnX” changes the state to the current floor state with the help of the local variable “currfloor”.

Table 3-7 shows the transitions of the STD “Search Request”.

label	pre	input	output	post
continit				
founddown	Currfloor>foundfloor			direction=2; stnextfloor=false
foundequal	Currfloor==foundfloor			stnextfloor=true
foundfl1		flrequest1?true		foundfloor=1
foundfl2		flrequest2?true		foundfloor=2
foundfl3		flrequest3?true		foundfloor=3
foundfl4		flrequest4?true		foundfloor=4
foundup	currfloor<foundfloor			direction=1; stnextfloor=false
return1	currfloor==1			
return2	currfloor==2			
return3	currfloor==3			
return4	currfloor==4			
searchfl12	direction==1	flrequest1?false		
searchfl14	direction==2	flrequest1?false		
searchfl21	direction==2	flrequest2?false		
searchfl23	direction==1	flrequest2?false		
searchfl32	direction==2	flrequest3?false		
searchfl34	direction==1	flrequest3?false		
searchfl41	direction==1	flrequest4?false		
searchfl43	direction==2	flrequest4?false		
startsearch1		doorstate?true, flrequest1?false		
startsearch2		doorstate?true, flrequest2?false		
startsearch3		doorstate?true, flrequest3?false		
startsearch4		doorstate?true, flrequest4?false		

Table 3-7: Transitions of „Search Request“

3.6.8 STD “Stop Next Continue“

The component “Stop Next Continue” is used to transform the single signal “true” from the channel “stopnextfloor to a continuos signal “true”. This signal is sent through the channel “stopnextfloorout”.

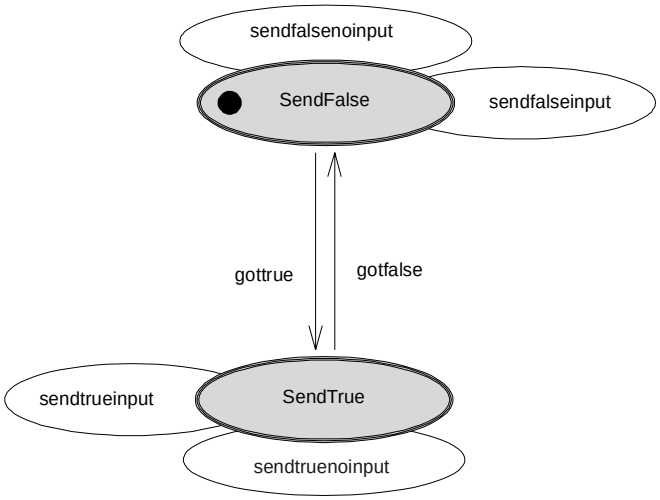

Figure 3-15: STD „Stop Next Continue“

The signal “false” on the channel “stopnextfloor” triggers the transition “gotfalse” and the signal “false” is sent through the channel “stopnextfloorout”.

label	pre	input	output	post
gotfalse		stopnextfloor?false	stopnextfloorout!false	
gottrue		stopnextfloor?true	stopnextfloorout!true	
sendfalseinput		stopnextfloor?false	stopnextfloorout!false	
sendfalsenoinput		stopnextfloor?	stopnextfloorout!false	
sendtrueinput		Stopnextfloor?true	stopnextfloorout!true	
sendtruenoinput		Stopnextfloor?	stopnextfloorout!true	

Table 3-8: Transitions of „Stop Next Continue“

3.7 Consistency of the Specification

As AutoFocus uses different hierarchical views of a system, it is necessary to check the consistency of the views. Therefore AutoFocus offers different consistency checks. You can check for example whether each port is linked with a channel or whether the interface of a component correspond to its substructure. For more flexibility the underlying conditions are not fixed, but can be modified or extended by the user. Consistency conditions are written in a declarative notation, similar to first-order logic (Figure 3-16).

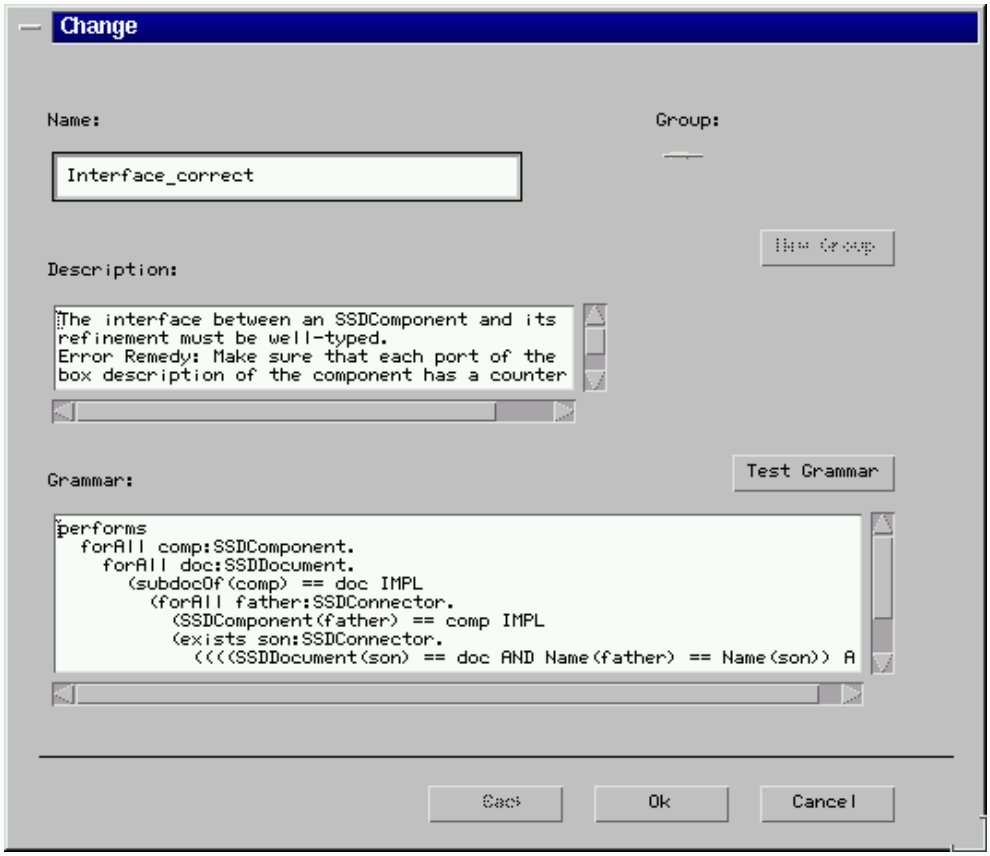

Figure 3-16: Changing the condition of the test „Interface Correct“

During development the specified system is often not consistent. The consistency checks are not automatically done when an STD or an SSD is changed, to allow the developer more freedom in the design. Consistency checks are explicitly started by the developer in the project browser.

For the elevator, we use only predefined consistency checks (Figure 3-17).

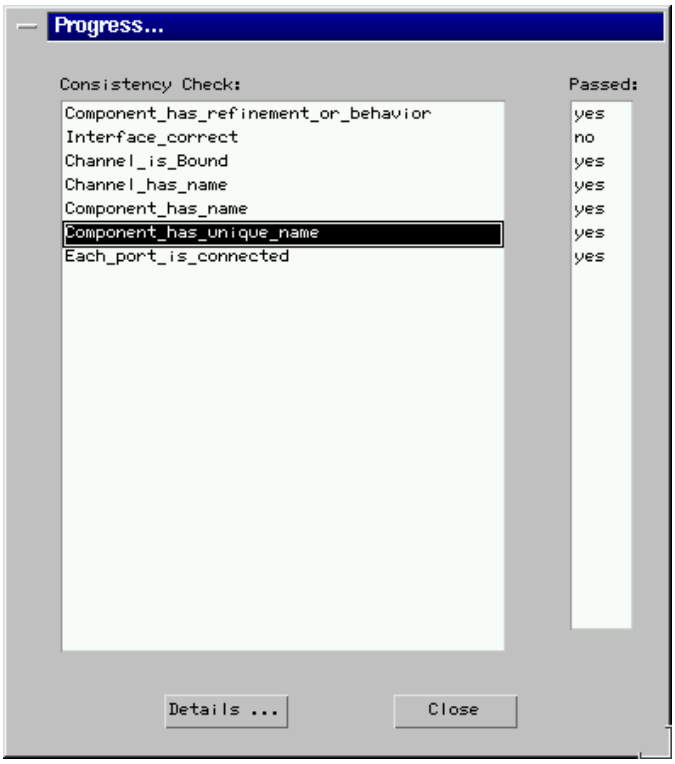

Figure 3-17: Consistency check of the elevator system

4 System Execution with SimCenter

After checking the consistency of the system, the behavior of the elevator system is ready to be tested in a simulation environment. The formal background of AutoFocus allows the simulation of system specifications based on the description techniques introduced in Section 2. The tool SimCenter is the integrated simulation component of AutoFocus. It generates Java code from AutoFocus system specifications. Each component with specified behavior is translated into a Java class [4]. The structure of the original specification is mapped to the Java code. SimCenter uses the generated Java code to execute the system in a special simulation environment. During the simulation, the user can observe different views of the system behavior in special windows, called "animators".

SSD animators show the data flow between the component and the environment. They show the channel contents during the simulation. State machines, described with STD diagrams, can be observed with the STD animators, where the current state of the STD and the transition leading to that state are highlighted.

SimCenter can generate EETs automatically during the simulation. The user can observe these EETs in an EET animator window. Finally, there are so called "Variable Animators" which can be used to examine the values of the local state variables.

In this section, we first describe how SimCenter is used for our elevator specification, and how EETs can be generated to help in finding errors in the specifications. Then we show how SimCenter can be connected to external applications to build custom views of the specified system, for visualization or for prototyping. This is described in detail in Sections 0 - 0.

4.1 Simulation of the elevator with SimCenter

To simulate the whole elevator system, we chose the SSD component "system" in the project browser and start the menu item "Simulation".

Figure 4-1 shows an "SSD Animator" for the elevator component "System" during a simulation run.

In the figure you can see, that there is a request in the fourth floor (channel "light4:true"), and the elevator car is moving up ("motorup:true"). In the same manner it possible to show the sub components of "System" during the simulation.

The window "Environment" (Figure 4-2) is used for the interaction with the environment. All channels from and to the main component and their current signals are displayed. The values on the input channels of the main component can be explicitly changed.

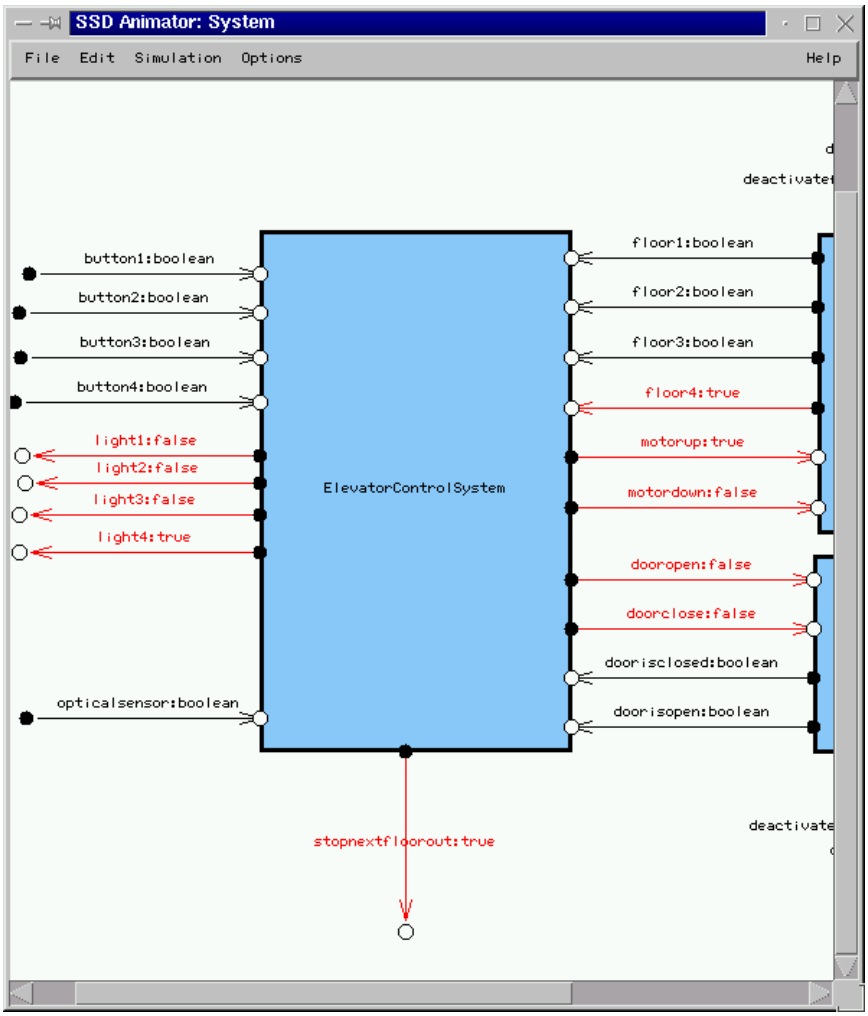

Figure 4-1: SSD Animator of the component „System“

For example if you write the value “true” on the channel “button2” during the simulation, this means that a user of the elevator system pressed the request button in the second floor and the elevator control system will react in the right manner.

4.2 The components “Floor Sim“ and “Door Sim“

During simulation of the elevator system the user effectively has to play the role of the complete elevator environment. To help with this role, we built two new components called “FloorSim” and “DoorSim” and inserted them into the SSD “System” (Figure 4-3). These components simulate the behavior of the Fischertechnik elevator model if it is not connected. Moreover, they provide all the information needed for a graphical front end to visualize the system behavior. The channels “deactivatefloorsim” and “deactivatedoorsim” are used to switch these components off. The channels “deactivatefloorok” and “deactivatedoorok” serves the acknowledgement signal after the deactivation.

Figure 4-2: „Environment“ of the elevator system during the simulation

With deactivated simulation components the signals from the channels “floor1-4in” are directly passed through from the hardware elevator model to the control system.

The signal from the door sensors and motor control unit are handled the same way.

The state machines of the simulation components can be found in the appendix; they are quite complex in spite of their simple functionality.

4.3 Error detection with EETs

The automatically generated Event Traces (EETs) of SimCenter help to find errors in the specification by simulation. EETs mirror the temporal order of the communication between several components. Figure 4-4 shows the communication history within the component “Elevator Control System”.

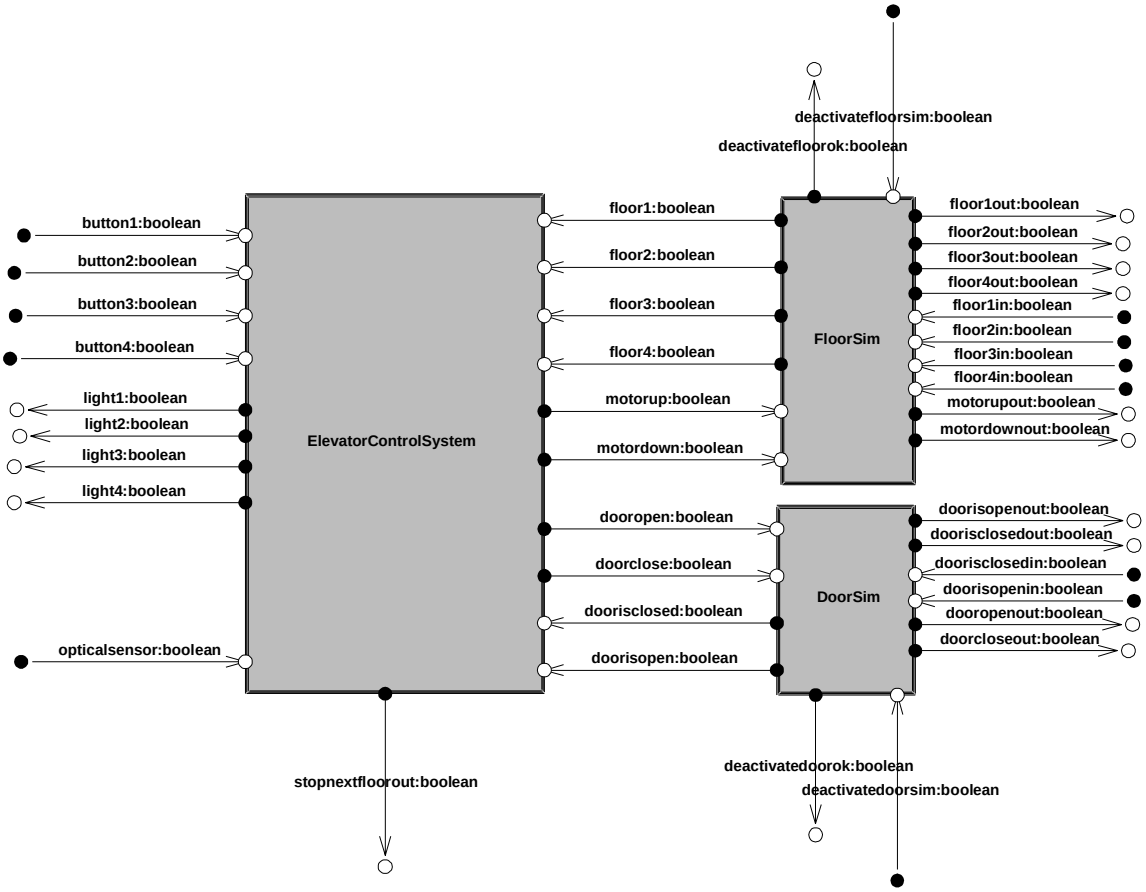

Figure 4-3: SSD „System“ and the simulation components

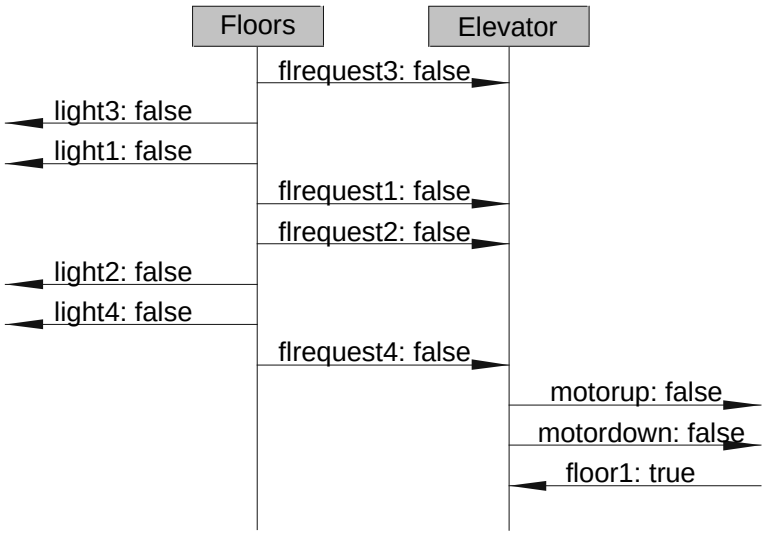

Figure 4-4: EET of the component „Elevator Control System“

Within the diagram, time progresses from top to bottom. The arrows describe the communication direction between the components. An arrow which has no link to a component on one of its ends, represents a communication event with the environment of the system or with a higher level in the SSD hierarchy. For example the signal “flrequest4:false” is sent from “Floors” to “Elevator” just before the signal “floor1:true”.

This kind of description serves as a different view to the dynamical behavior of the system. It is often easier to detect functional errors of the system than in the STD specifications.

The appendix contains a complete test run of the elevator control system.

4.4 Device Dependent Interface

SimCenter offers a special interface to external applications. This interface links the simulated system and its environment. An external application is allowed to write and read all channels which goes from and to the system. Graphical applications can be connected to the simulated system to visualize the system behavior and to allow the developer to show his customers the behavior of the system in a suggestive way.

The interface is divided in a device dependent and a device independent part. The device independent part is realized in Java and communicates with “Remote Method Invocations” (RMIs) with the device dependent interface (DDI). The DDI transform the RMI call into a form which is accessible to the external application. Its implementation depends on the application.

The DDI can communicate with several applications at the same time. For the elevator control system, one application is a multimedia application that provides independent views of the systems; another application is the hardware interface to the Fischertechnik model.

Since currently the DDI’s implementation is based on the Microsoft Windows message system, it is possible to control arbitrary Windows applications. However, the DDI does not includes a strategy to avoid conflicts. It is the developers job to ensure that the applications are well-behaved, and do not write to the same channel of the specified system concurrently.

Every application offers another view to the environment of the system or a part of it. The complete set of applications should be designed in a way that they can be interpreted as one component “Environment”. Then, from the view of the specified system, it communicates just with only one external component.

The easiest solution is to allow only one application to write on system channels. In the example of the elevator system only the hardware model is allowed to write on channels from the environment to the system’s main component.

4.5 Visualization with Formula Graphics

Above we mentioned that the SimCenter can control several front-end applications when simulating AutoFocus specifications. These application were built with the freely-available multimedia tool “Formula Graphics”. Formula Graphics allows to design multimedia presentations (which can interact with the user) in an easy way. For complex presentations the tool offers a script language. We used this language to realize the communication between the DDI of SimCenter and the applications.

We developed two multimedia applications. Figure 4-5 shows the graphical model of a elevator system with four floors. All typical interactions between an elevator passenger and an elevator are possible with this application. The lights on the left side show the current floor of the car. The lights on the right side show whether there is a request in a certain floor or not. The buttons are used to request the car at a floor. The demo button activates a random test run.

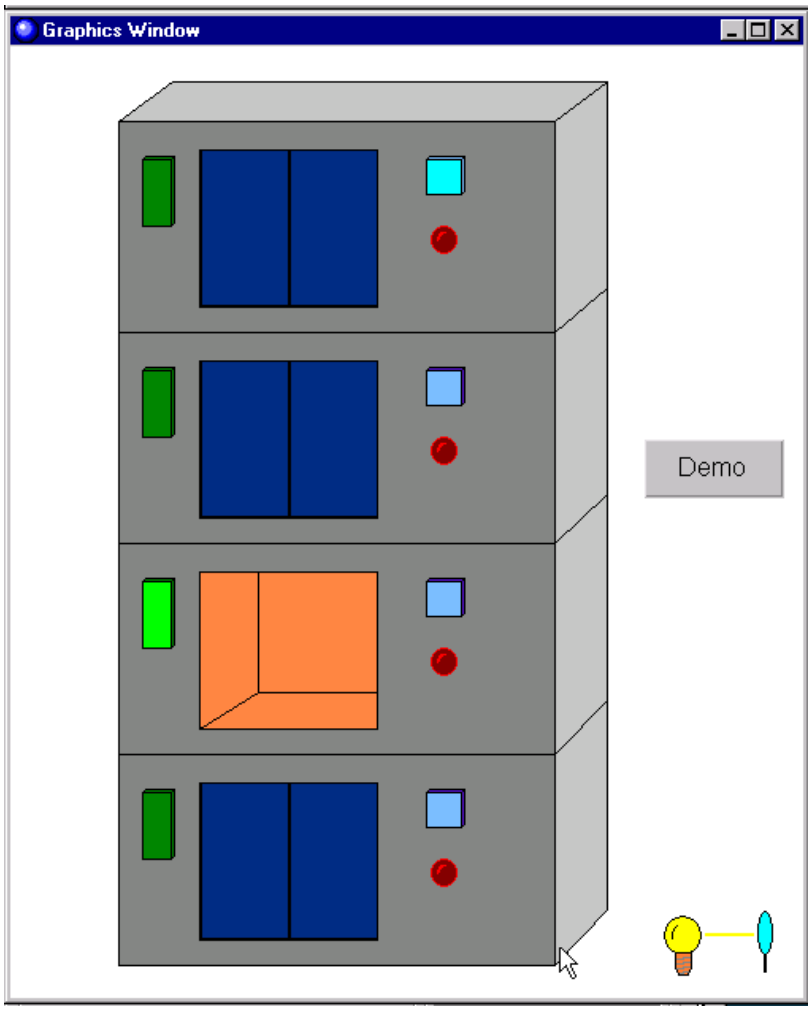

Figure 4-5: Formula Graphics application „Elevator“

Figure 4-6 shows a completely different view on the system. It imitates a service panel, which protocols how often the elevator arrived at a certain floor and how often that floor was requested. Moreover, the total amount of requests and services is shown. This application involved a completely different functionality than the visualization of the elevator system. It just observes a part of the communication and represent the information in another way.

Figure 4-6: Formula Graphics application „Service Panel“

Through of the simultaneous connection of both multimedia applications, SimCenter offers different views of the specified system.

4.6 Prototyping with the Fischertechnik model

A goal of whole the project was it to control a Fischertechnik elevator model with the AutoFocus specification and SimCenter, to test the specified system in a realistic way without implementing it on the target platform. A first prototype of the embedded system is realized in this way.

To control the Fischertechnik elevator we use a simple mutlti I/O card, which is compatible with the 8 Bit ISA bus of a common PC. To control the communication between the I/O card and SimCenter, we developed yet another interface called “Elevator Control” (Figure 4-7).

The program transforms incoming messages from SimCenter into control sequences for the I/O card. In other direction it controls the input ports of the I/O in fixed time intervals and sends the results the SimCenter. Additional it shows the state of all incoming and outgoing channels of the elevator model and allows with some special button the direct control of the hardware.

The program was implemented in the C++ language. Like Formula Graphics, it runs on a common Windows NT system.

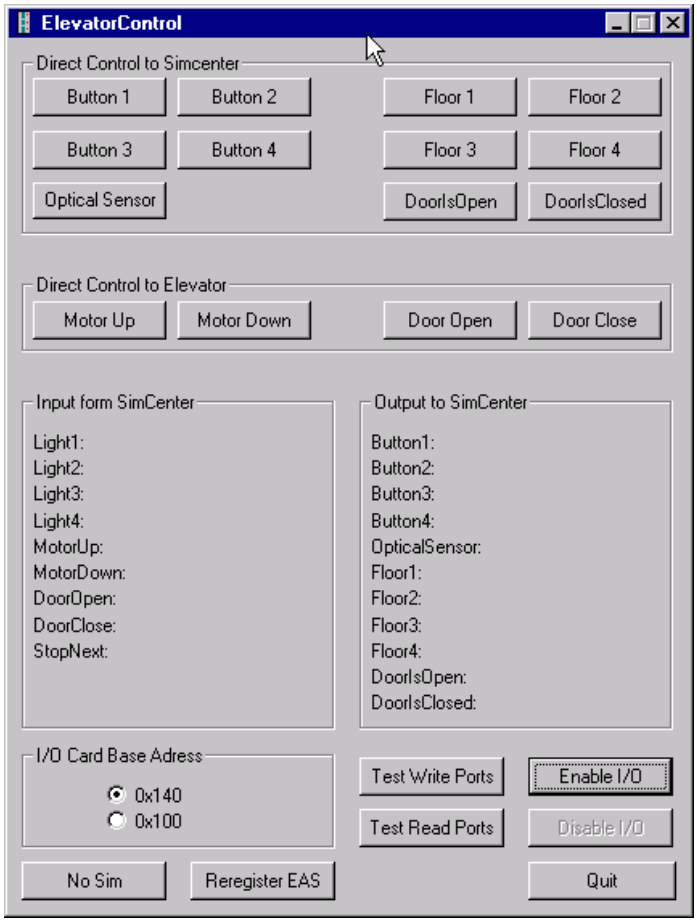

Figure 4-7: The hardware controller „Elevator Control“

5 Conclusions

In this paper we have shown an exemplary development process for an embedded system using AutoFocus and SimCenter.

Using the different description technique of AutoFocus, it is possible to accelerate the software development process. The simulation facilities of SimCenter allows the developer to produce system specifications with less errors. The open architecture of SimCenter offers new possibilities for embedding the system specification in a special test environment; even the direct control of existing hardware is possible AutoFocus and SimCenter open the door to system visualization and rapid prototyping. Both help to validate customer requirements already in the early development phases.

Embedded systems consist of both hardware and software components. It is common to develop the hardware and the software simultaneously. With AutoFocus single components can easily be modified during the development process of the system. Therefore components can be adapted to the needs of the current development phase. You can first describe the environment of the system using software components. When a prototype of the hardware is available, the software component can be replaced by hardware and the specification can be tested in its real environment.

Still testing a specification of an embedded system does not allow to prove that the system fulfills certain requirements. For this purpose the current and future work on AutoFocus includes the integration of model checker techniques and tools as well theorem interactive provers.

Appendix A: Additional Diagrams of the Elevator Control System

A.1 STDs

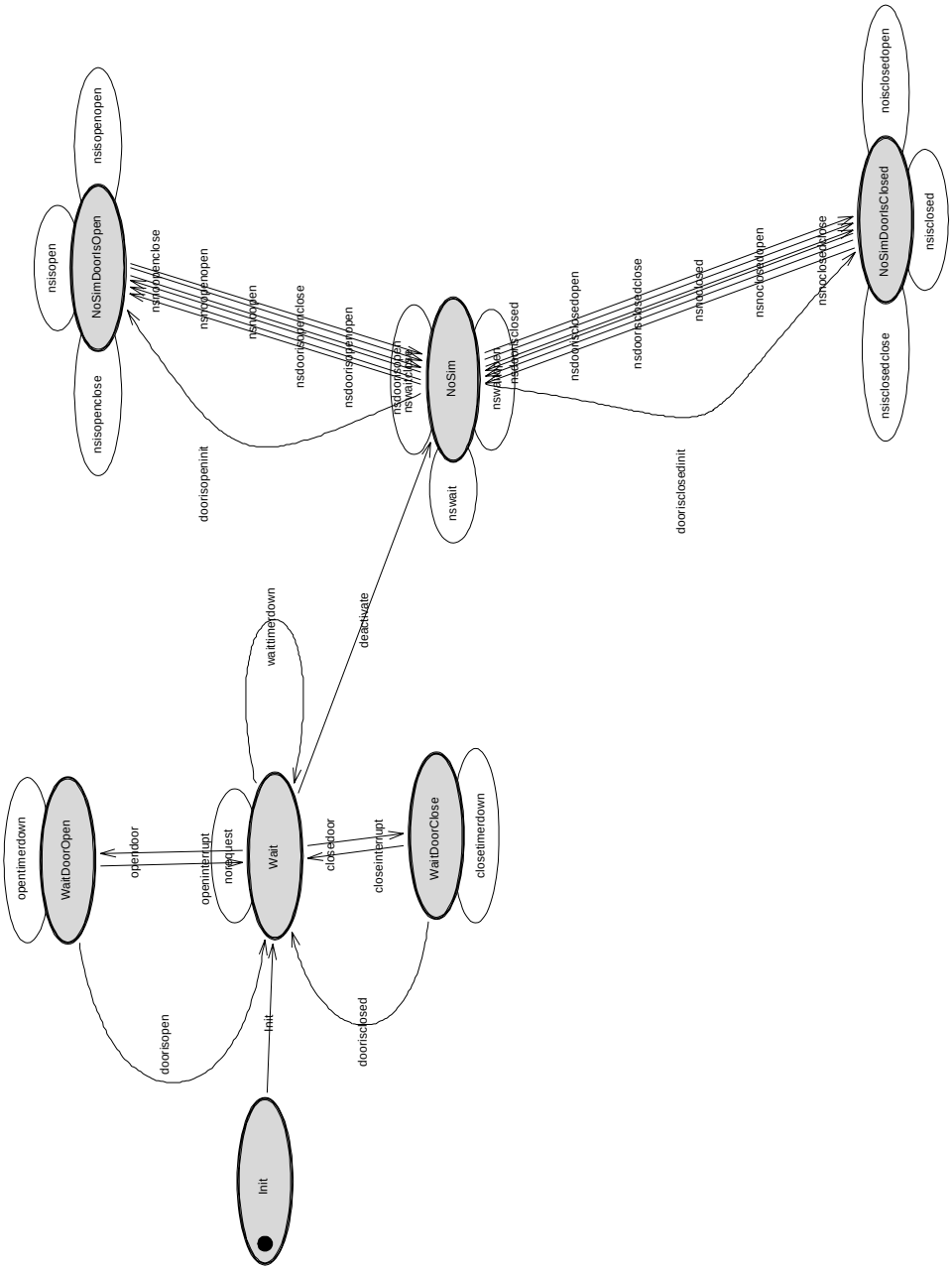

Figure A-1: STD Door Sim

label	pre	input	output	post
closedoor	Timer==0	doorclose?true, deactivatedoorsim?	doorcloseout!true, dooropenout!false	Timer=6
closeinterrupt		doorclose?false	doorcloseout!false, dooropenout!false	Timer=0
closetimerdown	Timer>0	doorclose?true	doorcloseout!true, dooropenout!false	Timer=Timer-1
deactivate		deactivatedoorsim?true	deactivatedoorok!true	
doorisclosed	Timer==0	doorclose?true	doorisclosed!true, doorisclosedout!true, dooropenout!false, doorcloseout!true	Timer=1
doorisclosedinit		doorisclosedin?true, dooropen?, doorclose?		
doorisopen	Timer==0	dooropen?true	doorisopen!true, doorisopenout!true, dooropenout!true, doorcloseout!false	Timer=1
doorisopeninit		doorisopenin?true, dooropen?, doorclose?		
Init			deactivatedoorok!true	Timer=0
noisclosedopen		doorisclosedin?, dooropen?true	doorisclosed!true, doorisclosedout!true, dooropenout!true, doorcloseout!false	
norequest	Timer==0	dooropen?false, doorclose?false, deactivatedoorsim?	dooropenout!false, doorcloseout!false	
nsdoorisclosed		doorisclosedin?true, dooropen?false, doorclose?false	doorisclosed!true, doorisclosedout!true, dooropenout!false, doorcloseout!false	
nsdoorisclosedclose		doorisclosedin?true, doorclose?true	doorisclosed!true, doorisclosedout!true, dooropenout!false, doorcloseout!true	
nsdoorisclosedopen		doorisclosedin?true, dooropen?true	doorisclosed!true, doorisclosedout!true, dooropenout!true, doorcloseout!false	
nsdoorisopen		doorisopenin?true, doorclose?false, dooropen?false	doorisopenout!true, doorisopen!true, doorcloseout!false, dooropenout!false	
nsdoorisopenclose		doorisopenin?true, doorclose?true	doorisopen!true, doorisopenout!true, dooropenout!false, doorcloseout!true	
nsdoorisopenopen		doorisopenin?true, dooropen?true	doorisopen!true, doorisopenout!true, dooropenout!true, doorcloseout!false	
nsisclosed		doorisclosedin?, dooropen?false, doorclose?false	doorisclosed!true, doorisclosedout!true, dooropenout!false, doorcloseout!false	
nsisclosedclose		doorisclosedin?, doorclose?true	doorisclosed!true, doorisclosedout!true, dooropenout!false, doorcloseout!true	
nsisopen		doorisopenin?, dooropen?false, doorclose?false	doorisopen!true, doorisopenout!true, doorcloseout!false, dooropenout!false	
nsisopenclose		doorisopenin?, doorclose?true	doorisopen!true, doorisopenout!true, dooropenout!false,	

			doorcloseout!true	
nsisopenopen		doorisopenin?, dooropen?true	doorisopen!true, doorisopenout!true, dooropenout!true, doorcloseout!false	
nsnoclosed		doorisclosedin?false, dooropen?false, doorclose?false	dooropenout!false, doorcloseout!false	
nsnoclosedclose		doorisclosedin?false, doorclose?true	dooropenout!false, doorcloseout!true	
nsnoclosedopen		doorisclosedin?false, dooropen?true	dooropenout!true, doorcloseout!false	
nsnoopen		doorisopenin?false, dooropen?false, doorclose?false	dooropenout!false, doorcloseout!false	
nsnoopenclose		doorisopenin?false, doorclose?true	dooropenout!false, doorcloseout!true	
nsnoopenopen		doorisopenin?false, dooropen?true	dooropenout!true, doorcloseout!false	
nswait		doorisopenin?, doorisclosedin?, dooropen?false, doorclose?false	dooropenout!false, doorcloseout!false	
nswaitclose		doorisopenin?, doorisclosedin?, doorclose?true	dooropenout!false, doorcloseout!true	
nswaitopen		doorisclosedin?, doorisopenin?, dooropen?true	dooropenout!true, doorcloseout!false	
opendoor	Timer==0	dooropen?true, deactivatedoorsim?	dooropenout!true, doorcloseout!false	Timer=6
openinterrupt		dooropen?false	dooropenout!false, doorcloseout!false	Timer=0
opentimerdown	Timer>0	dooropen?true	dooropenout!true, doorcloseout!false	Timer=Timer-1
waittimerdown	Timer>0	deactivatedoorsim?	doorcloseout!false, dooropenout!false	Timer=Timer-1

Table A-1: Transitions of Door Sim

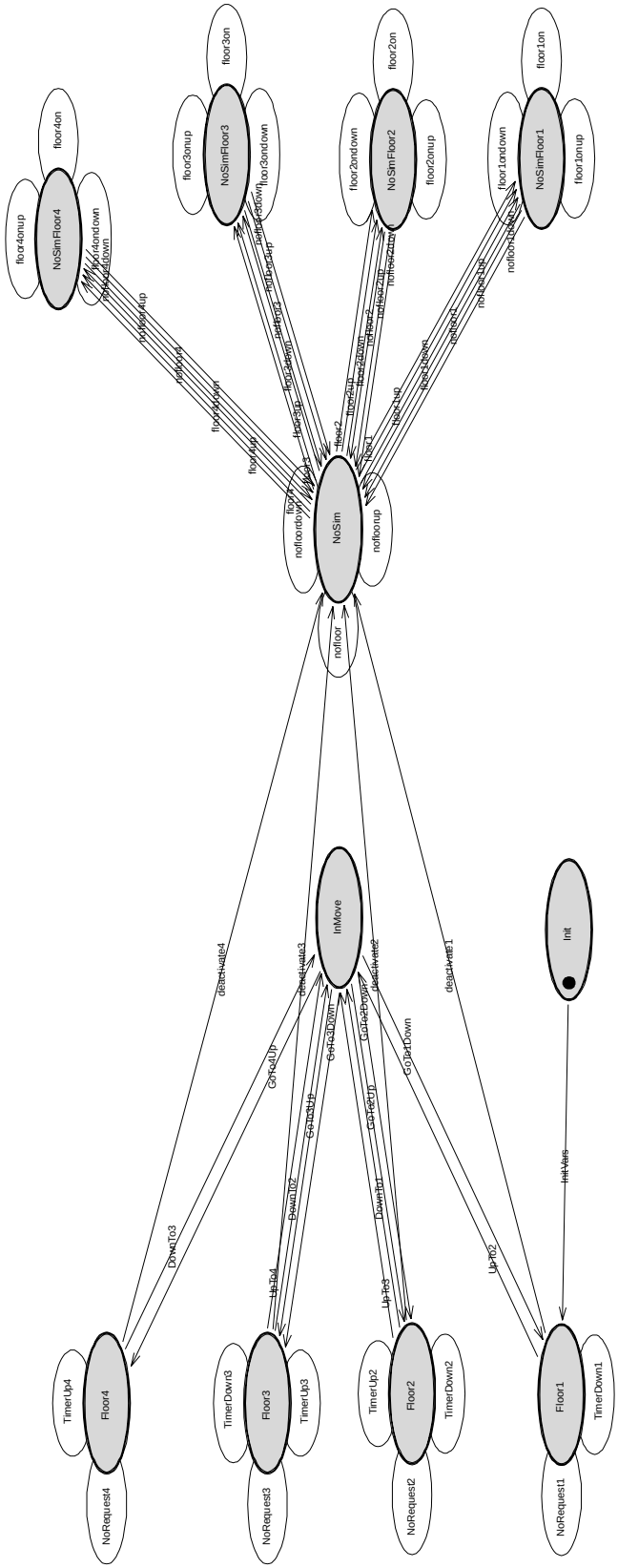

Figure A-2: STD Floor Sim

label	pre	input	output	post
cont	timer==0	deactivatefloorsim?		timer=5
deactivate1		deactivatefloorsim?true	deactivatefloorok!true	
deactivate2		deactivatefloorsim?true	deactivatefloorok!true	
deactivate3		deactivatefloorsim?true	deactivatefloorok!true	
deactivate4		deactivatefloorsim?true	deactivatefloorok!true	
deactivateinit		deactivatefloorsim?true	deactivatefloorok!true	
DownTo1	timer==0	motordown?true, deactivatefloorsim?	motordownout!true, motorupout!false	togo=1
DownTo2	timer==0	motordown?true, deactivatefloorsim?	motordownout!true, motorupout!false	togo=2
DownTo3	timer==0	motordown?true, deactivatefloorsim?	motordownout!true, motorupout!false	togo=3
floor1		floor1in?true, motorup?false, motordown?false	floor1!true, floor1out!true, motorupout!false, motordownout!false	
floor1down		floor1in?true, motordown?true	floor1!true, floor1out!true, motorupout!false, motordownout!true	
floor1on		floor1in?, motorup?false, motordown?false	floor1!true, floor1out!true, motorupout!false, motordownout!false	
floor1ondown		floor1in?, motordown?true	floor1!true, floor1out!true, motorupout!false, motordownout!true	
floor1onup		floor1in?, motorup?true	floor1!true, floor1out!true, motorupout!true, motordownout!false	
floor1up		floor1in?true, motorup?true	floor1!true, floor1out!true, motorupout!true, motordownout!false	
floor2		floor2in?true, motorup?false, motordown?false	floor2!true, floor2out!true, motorupout!false, motordownout!false	
floor2down		floor2in?true, motordown?true	floor2!true, floor2out!true, motorupout!false, motordownout!true	
floor2on		floor2in?, motorup?false, motordown?false	floor2!true, floor2out!true, motorupout!false, motordownout!false	
floor2ondown		floor2in?, motordown?true	floor2!true, floor2out!true, motorupout!false, motordownout!true	
floor2onup		floor2in?, motorup?true	floor2!true, floor2out!true, motorupout!true, motordownout!false	
floor2up		floor2in?true, motorup?true	floor2!true, floor2out!true, motorupout!true, motordownout!false	
floor3		floor3in?true, motorup?false, motordown?false	floor3out!true, floor3!true, motorupout!false, motordownout!false	
floor3down		floor3in?true, motordown?true	floor3!true, floor3out!true, motorupout!false, motordownout!true	
floor3on		floor3in?, motorup?false, motordown?false	floor3!true, floor3out!true, motorupout!false, motordownout!false	
floor3ondown		floor3in?, motordown?true	floor3!true, floor3out!true, motorupout!false, motordownout!true	
floor3onup		floor3in?, motorup?true	floor3!true, floor3out!true, motorupout!true, motordownout!false	
floor3up		floor3in?true, motorup?true	floor3!true, floor3out!true, motorupout!true,	

			motordownout!false	
floor4		floor4in?true, motorup?false, motordown?false	floor4!true, floor4out!true, motorupout!false, motordownout!false	
floor4down		floor4in?true, motordown?true	floor4!true, floor4out!true, motorupout!false, motordownout!true	
floor4on		floor4in?, motorup?false, motordown?false	floor4!true, floor4out!true, motorupout!false, motordownout!false	
floor4ondown		floor4in?, motordown?true	floor4!true, floor4out!true, motorupout!false, motordownout!true	
floor4onup		floor4in?, motorup?true	floor4!true, floor4out!true, motorupout!true, motordownout!false	
floor4up		floor4in?true, motorup?true	floor4!true, floor4out!true, motorupout!true, motordownout!false	
GoTo1Down	togo==1	motordown?true	motordownout!true, motorupout!false	timer=4
GoTo2Down	togo==2	motordown?true	motordownout!true, motorupout!false	timer=4
GoTo2Up	togo==2	motorup?true	motorupout!true, motordownout!false	timer=4
GoTo3Down	togo==3	motordown?true	motordownout!true, motorupout!false	timer=4
GoTo3Up	togo==3	motorup?true	motorupout!true, motordownout!false	timer=4
GoTo4Up	togo==4	motorup?true	motorupout!true, motordownout!false	timer=4
			deactivatefloorok!true	togo=1; timer=5
nofloor		floor1in?, floor2in?, floor3in?, floor4in?, motorup?false, motordown?false	motorupout!false, motordownout!false	
nofloor1		floor1in?false, motorup?false, motordown?false	motorupout!false, motordownout!false	
nofloor1down		floor1in?false, motordown?true	motorupout!false, motordownout!true	
nofloor1up		floor1in?false, motorup?true	motorupout!true, motordownout!false	
nofloor2		floor2in?false, motordown?false, motorup?false	motordownout!false, motorupout!false	
nofloor2down		floor2in?false, motordown?true	motorupout!false, motordownout!true	
nofloor2up		floor2in?false, motorup?true	motorupout!true, motordownout!false	
nofloor3		floor3in?false, motordown?false, motorup?false	motordownout!false, motorupout!false	
nofloor3down		floor3in?false, motordown?true	motorupout!false, motordownout!true	
nofloor3up		floor3in?false, motorup?true	motorupout!true, motordownout!false	
nofloor4		floor4in?false, motorup?false, motordown?false	motorupout!false, motordownout!false	
nofloor4down		floor4in?false, motordown?true	motorupout!false, motordownout!true	
nofloor4up		floor4in?false, motorup?true	motorupout!true, motordownout!false	
nofloordown		floor1in?, floor2in?, floor3in?, floor4in?, motordown?true	motorupout!false, motordownout!true	
nofloorup		floor1in?, floor2in?, floor3in?, floor4in?, motorup?true	motorupout!true, motordownout!false	
NoRequest1		motorup?false, motordown?false, deactivatefloorsim?	floor1!true, floor1out!true, motorupout!false, motordownout!false	timer=0

NoRequest2		motorup?false, motordown?false, deactivatefloorsim?	floor2!true, floor2out!true, motorupout!false, motordownout!false	timer=0
NoRequest3		motorup?false, motordown?false, deactivatefloorsim?	floor3!true, floor3out!true, motorupout!false, motordownout!false	timer=0
NoRequest4		motorup?false, motordown?false, deactivatefloorsim?	floor4!true, floor4out!true, motorupout!false, motordownout!false	timer=0
TimerDown1	timer>0	deactivatefloorsim?, motordown?true	floor1!true, floor1out!true, motorupout!false, motordownout!true	timer--
TimerDown2	timer>0	deactivatefloorsim?, motordown?true	floor2!true, floor2out!true, motordownout!true, motorupout!false	timer--
TimerDown3	timer>0	deactivatefloorsim?, motordown?true	floor3!true, floor3out!true, motordownout!true, motorupout!false	timer--
TimerUp2	timer>0	deactivatefloorsim?, motorup?true	floor2!true, floor2out!true, motorupout!true, motordownout!false	timer--
TimerUp3	timer>0	deactivatefloorsim?, motorup?true	floor3!true, floor3out!true, motorupout!true, motordownout!false	timer--
TimerUp4	timer>0	deactivatefloorsim?, motorup?true	floor4!true, floor4out!true, motordownout!false, motorupout!true	timer--
UpTo2	timer==0	motorup?true, deactivatefloorsim?	motorupout!true, motordownout!false	togo=2
UpTo3	timer==0	motorup?true, deactivatefloorsim?	motorupout!true, motordownout!false	togo=3
UpTo4	timer==0	motorup?true, deactivatefloorsim?	motorupout!true, motordownout!false	togo=4
Wait	timer>0	deactivatefloorsim?		timer--

Table A-2: Transitions of Floor Sim

A.2 EETs

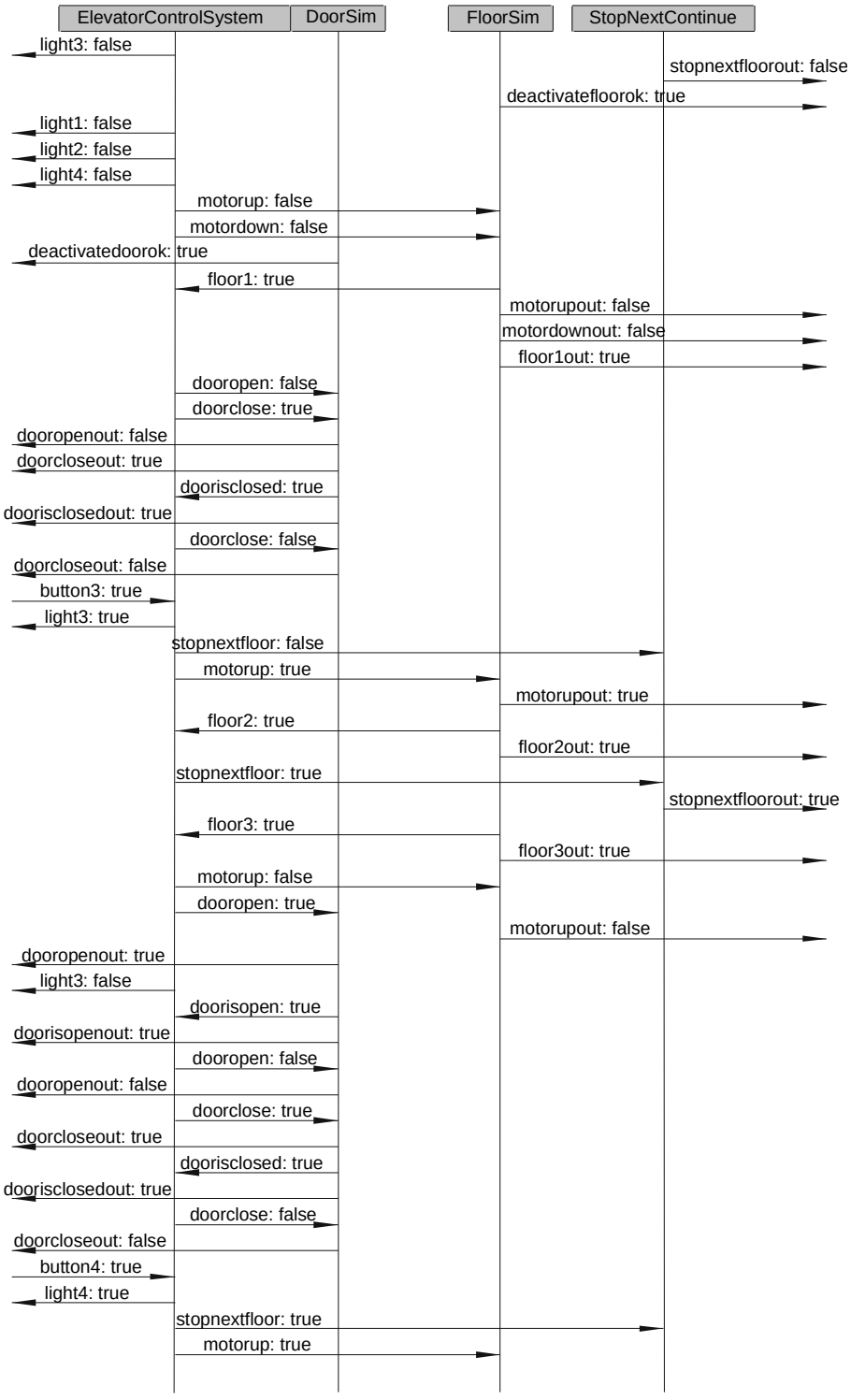

Figure A-3: EET System

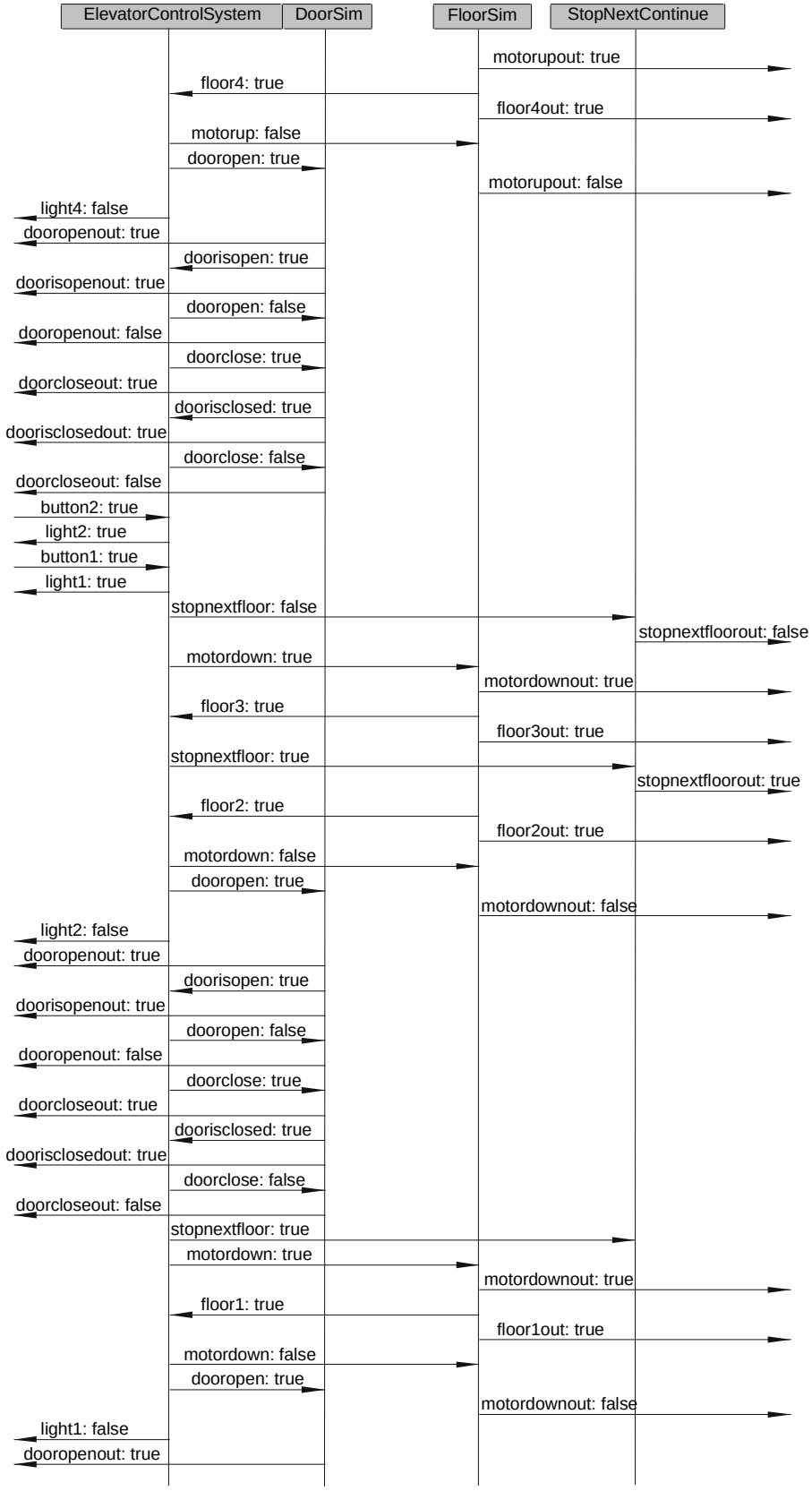

Figure A-4: EET System (continuation)

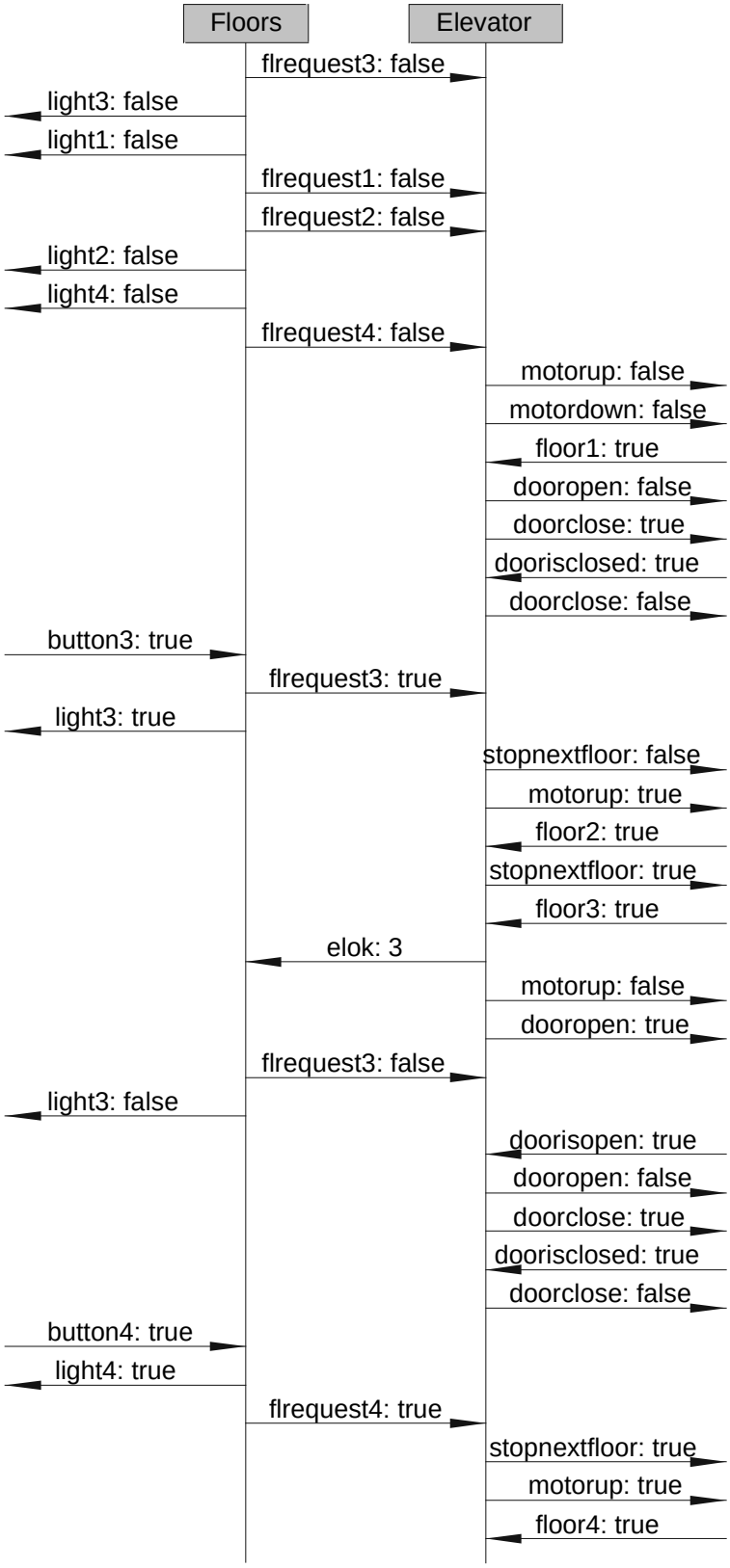

Figure A-5: EET Elevator Control System

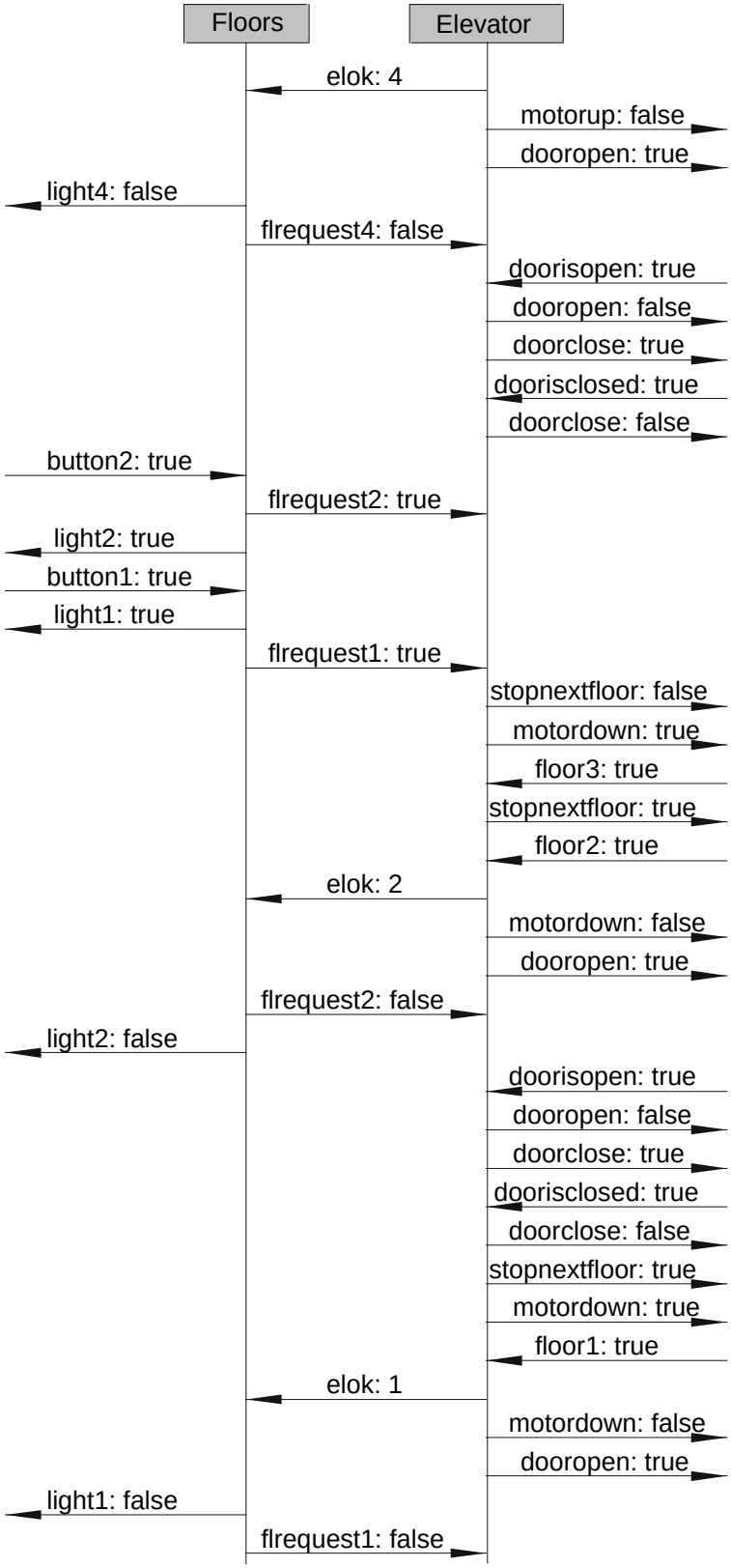

Figure A-6: EET Elevator Control System (continuation)

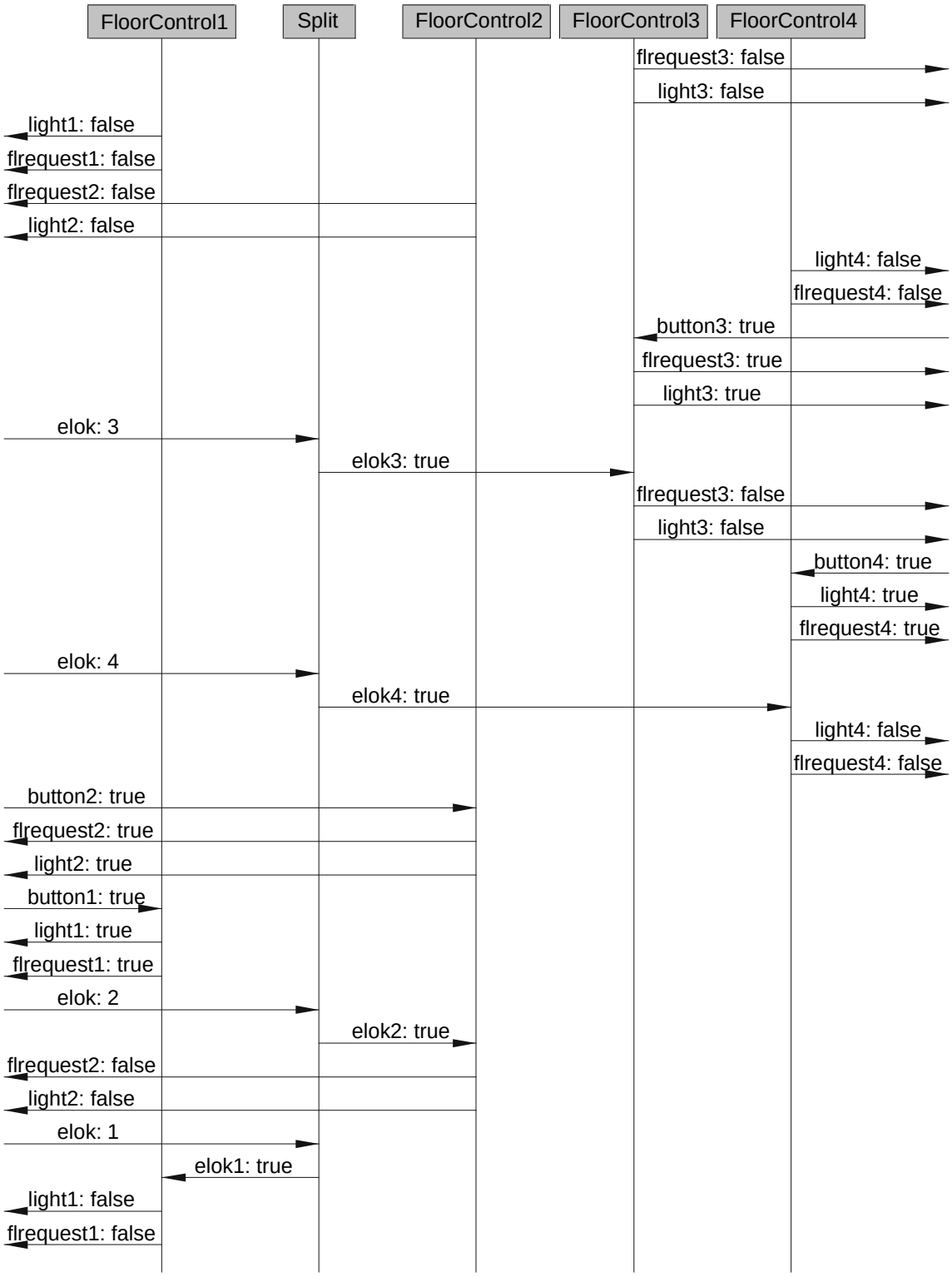

Figure A-7: EET Floors

Figure A-8: EET Elevator

Figure A-9: EET Elevator (continuation)

References

[1] M. Broy, F. Dederichs, C. Denhofer, M.Fuchs, T.F. Grilzner and R. Weber,
The design of distributed systems – an introduction to FOCUS,
Tech. Rep. TUM-I9203,
Technische Universität München, Institut für Informatik, 1992

[2] Franz Huber, Sascha Molterer, Bernhard Schätz, Oscar Slotosch and Alexander Vilbig,
Traffic Lights – An AutoFocus Case Study,
Technische Universität München, Institut für Informatik, 1997

[3] Joon-Sung Hong,
Object-Oriented Analysis and Design Method for Concurrent and Realtime Systems,
CORE-TR96-001, 1996

[4] Franz Huber and Bernhard Schätz,
Rapid Prototyping with AutoFocus,
in Formale Beschreibungstechniken für verteilte Systeme, Seite 343-352,
A. Wolisz, I. Schieferdecker and A. Rennoch,
GMD Verlag, 1997

[5] Franz Huber, Bernhard Schätz and Geralf Einert,
Consistent Graphical Specifikation of Distributed Systems,
in FME 97, LNCS 1313, Seite 122-141, Peter Lucas John Fitzgerald, Cliff B. Jones, Springer Verlag, 1997

[6] Radu Grosu, Cornel Klein, Bernhard Rumpe and Manfred Broy,
State Transition Diagramms,
Tech. Rep. TUM-I9630,
Technische Universität München, 1996

[7] Franz Huber, Bernhard Schätz, Alexander Schmidt, Katharina Spies,
AutoFocus – A Tool for Distributed Systems,
Institut für Informatik, Technische Universität München, 1996

[8] Franz Huber, Bernhard Schätz, Katharina Spies,
AutoFocus – Ein Werkzeugkonzept zur Beschreibung verteilter Systeme,
Institut für Informatik, Technische Universität München, 1996

[9] Manfred Broy, Eva Geisberger, Radu Grosu, Franz Huber, Bernhard Rumpe, Bernhard Schätz, Alexander Schmidt, Oscar Slotosch, Katharina Spies,
Das AutoFocus Bilderbuch – Eine anwenderorientierte Beschreibung,
Institut für Informatik, Technische Universität München, 1996

[10] Helmut Balzert,
Lehrbuch der Software-Technik – Software-Entwicklung,
Spektrum akademischer Verlag,
ISBN 3-8274-0042-2, 1996

[11] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides,
Design Pattern – Elements of Reusable Object-Oriented Software,
Addison-Wesley Verlag,
ISBN 0-201-63361-2, 1995

[12] UML Summary, UML Semantics, UML Notation Guide,
Rational Software Cooperation, 1997

[13] International Telecommunication Union,
Recommendation Z. 120, Message Sequence Chart (MSC),
ITU, 1993

[14] M.Broy, H. Hußmann and B. Schätz,
Graphical Development of Consistent System Descriptions
Proceedings of FME'96 Formal Methods Europe, 1996

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids

342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law

342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space

342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems

342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter

342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology

342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies

342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness

342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design

342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System

342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming

342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks

342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs

342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations

342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications

342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C

Reihe A

342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication

342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control

342/22/95 A Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes

342/23/95 A Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation

342/24/95 A M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems

342/01/96 A Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries

342/02/96 A Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

342/03/96 A Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes

342/04/96 A Thomas Huckle: Efficient Computation of Sparse Approximate Inverses

342/05/96 A Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification

342/06/96 A Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components

342/07/96 A Richard Mayr: Some Results on Basic Parallel Processes

342/08/96 A Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht

342/09/96 A P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme

342/10/96 A Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFS-Lib – A File System for Parallel Programming Environments

342/11/96 A Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions

342/12/96 A Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete

342/13/96 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks

342/14/96 A Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project

342/15/96 A Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus

342/16/96 A Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

Reihe A

342/17/96 A Richard Mayr: Model Checking PA-Processes

342/18/96 A Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States

342/01/97 A Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsf”ahigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken

342/02/97 A Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations

342/03/97 A Bernhard Mitschang: Technologie f”ur Parallele Datenbanken - Bericht zum Workshop

342/04/97 A nicht erschienen

342/05/97 A Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik

342/06/97 A Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump

342/07/97 A Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming

342/08/97 A Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement

342/09/97 A Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic

342/10/97 A Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus

342/11/97 A Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity

342/12/97 A Christian B. Czech: Architektur und Konzept des Dycos-Kerns

342/13/97 A Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow

342/14/97 A Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level

342/15/97 A Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen

342/16/97 A Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung

342/17/97 A Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing

342/18/97 A Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

342/19/97 A Frank Wallner: Model Checking LTL Using Net Unfoldings

342/20/97 A Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –

342/21/97 A Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving

Reihe A

342/22/97 A T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)

342/23/97 A Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP

342/24/97 A Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II

342/25/97 A Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS

342/26/97 A Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration

342/27/97 A Max Breitling: Formalizing and Verifying TimeWarp with FOCUS

342/28/97 A Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)

342/29/97 A Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems

342/01/98 A A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): "'Anwendungsbezogene Lastverteilung"', ALV'98

342/02/98 A Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS

342/03/98 A Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten

342/04/98 A Stefan Bischof, Ernst-W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions

342/05/98 A St. Bischof, R. Ebner, Th. Erlebach: Load Balancing for Problems with Good Bisectors and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results

342/06/98 A Giannis Bozas, Susanne Kober: Logging and Crash Recovery in Shared-Disk Database Systems

342/07/98 A Markus Pizka: Distributed Virtual Address Space Management in the MoDiS-OS

342/08/98 A Niels Reimer: Strategien für ein verteiltes Last- und Ressourcenmanagement

342/09/98 A Javier Esparza, Editor: Proceedings of INFINITY'98

342/10/98 A Richard Mayr: Lossy Counter Machines

342/11/98 A Thomas Huckle: Matrix Multilevel Methods and Preconditioning

342/12/98 A Thomas Huckle: Approximate Sparsity Patterns for the Inverse of a Matrix and Preconditioning

342/13/98 A Antonin Kucera, Richard Mayr: Weak Bisimilarity with Infinite-State Systems can be Decided in Polynomial Time

342/01/99 A Antonin Kucera, Richard Mayr: Simulation Preorder on Simple Process Algebras

342/02/99 A Johann Schumann, Max Breitling: Formalisierung und Beweis einer Verfeinerung aus FOCUS mit automatischen Theorembeweisern – Fallstudie –

Reihe A

342/03/99 A M. Bader, M. Schimper, Chr. Zenger: Hierarchical Bases for the Indefinite Helmholtz Equation

342/04/99 A Frank Strobl, Alexander Wisspeintner: Specification of an Elevator Control System

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B Wolfgang Reisig: Petri Nets and Algebraic Specifications

342/2/90 B Jörg Desel: On Abstraction of Nets

342/3/90 B Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems

342/4/90 B Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme

342/1/91 B Barbara Paech1: Concurrency as a Modality

342/2/91 B Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox - Anwenderbeschreibung

342/3/91 B Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen

342/4/91 B Werner Pohlmann: A Limitation of Distributed Simulation Methods

342/5/91 B Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis

342/6/91 B Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme

342/7/91 B W. Reisig: Concurrent Temporal Logic

342/1/92 B Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support
Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support

342/2/92 B Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen

342/1/93 B Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung

342/2/93 B Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick

342/1/94 B Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS