



Faculty of Sciences,
Technologie
and Communication

Towards a formal semantics of UML2 classes and protocol state machines for the specification-based testing of embedded systems

Dante Zanarini
November 7, 2007

About the Speaker

- I'm doing a 5-years career in Computer Sciences at the Faculty of Exact Sciences and Engineering , National University of Rosario, Argentina.
- Since August, I'm doing an internship in the SESAME project under supervision of Benoît Ries, until January 31, 2008.

Outline

- 1 Introduction
 - Context
 - Specification based testing
 - Giving Semantics to UML, Questions
- 2 UML2 and CSP-OZ-DC Overview
 - UML Class Diagrams and Protocol State Machines
 - CPS-OZ-DC: Combination of Processes, Data and Time
- 3 From UML to CSP-OZ-DC
 - Illustration: The elevator example
 - (Some) General Translation Rules
- 4 Conclusions

Outline

- 1 Introduction
 - Context
 - Specification based testing
 - Giving Semantics to UML, Questions
- 2 UML2 and CSP-OZ-DC Overview
 - UML Class Diagrams and Protocol State Machines
 - CPS-OZ-DC: Combination of Processes, Data and Time
- 3 From UML to CSP-OZ-DC
 - Illustration: The elevator example
 - (Some) General Translation Rules
- 4 Conclusions

SESAME Project

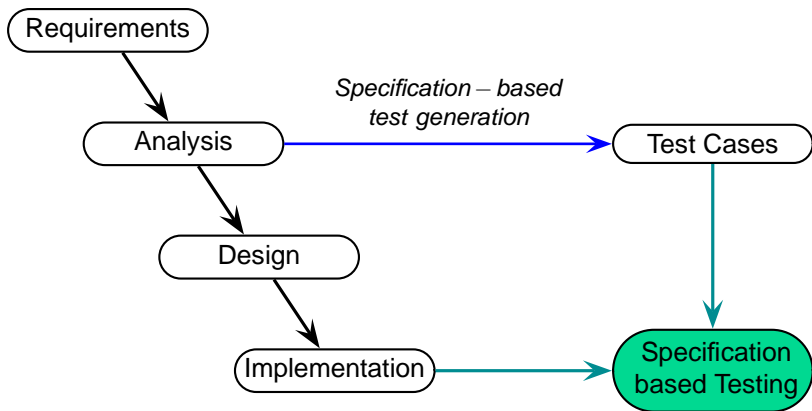
Specification based tE sting of SAfety-critical small-sized eMbedded systEms

- To develop an approach for **specification-based testing** adapted to the needs and the constraints of safety-critical small-sized embedded systems.
- To improve the efficiency of activities performed by test engineers, particularly giving a process to generate test cases **automatically** from analysis specifications [Ries, 2007]



SESAME
DEVELOPMENT METHODOLOGY
FOR EMBEDDED SYSTEMS

Specification Based Testing



Specification-based Testing

The Need for a Formal Specification Language

- In order to generate test cases automatically, we need the system specification expressed in a formal language, with a well defined syntax and semantics.
- In the industrial context of the SESAME project, the system specification is written in UML2
- But, some parts of the syntax and semantics of UML notations are provided in natural language (i.e., is not a formal language).
- To keep UML as our specification language, we must provide a formal semantics for UML modelling constructs.

Specification-based Testing

The Need for a Formal Specification Language

- In order to generate test cases automatically, we need the system specification expressed in a formal language, with a well defined syntax and semantics.
- In the industrial context of the SESAME project, the system specification is written in UML2
- But, some parts of the syntax and semantics of UML notations are provided in natural language (i.e., is not a formal language).
- To keep UML as our specification language, we must provide a formal semantics for UML modelling constructs.

Specification-based Testing

The Need for a Formal Specification Language

- In order to generate test cases automatically, we need the system specification expressed in a formal language, with a well defined syntax and semantics.
- In the industrial context of the SESAME project, the system specification is written in UML2
- But, some parts of the syntax and semantics of UML notations are provided in natural language (i.e., is not a formal language).
- To keep UML as our specification language, we must provide a formal semantics for UML modelling constructs.

Specification-based Testing

The Need for a Formal Specification Language

- In order to generate test cases automatically, we need the system specification expressed in a formal language, with a well defined syntax and semantics.
- In the industrial context of the SESAME project, the system specification is written in UML2
- But, some parts of the syntax and semantics of UML notations are provided in natural language (i.e., is not a formal language).
- To keep UML as our specification language, we must provide a formal semantics for UML modelling constructs.

Giving Semantics to UML

Questions

- In the SESAME project context, what UML modelling constructs are needed? (class diagrams, activity diagrams, behavioural state machines, protocol state machines, all?)
- What kind of semantics we will provide?
 - Operational (In which model: state transitions systems, SECD machines?)
 - Denotational (In which domain: game logic, actor models, partial functions?)

Choosing UML Modelling Constructs

Crucial Aspects in the SESAME Context

- Three crucial aspects are identified in the specification of small-sized embedded systems:
 - Specification of data and operations
 - Reactive behavior specification
 - Expression of timing constraints
- Combined with the abstraction level needed during the the analysis phase, we choose two UML2 modelling constructs for the specification of such aspects:
 - Class Diagram
 - Protocol State Machine

Choosing the Semantic Model

- Semantic model requirements:
 - Allow formal specification of data and operations, reactive behavior and timing constraints
 - Provide a well defined semantics for these concepts , enabling formal reasoning about the generated model
 - Tool support, facilitating the next steps in the project
- The language CSP-OZ-DC fulfills all these requirements, combining three well known formalism to express
 - behavior (CSP),
 - data (ObjectZ)
 - and time constraints (Duration Calculus).
- ▶ The Syspect (<http://syspect.informatik.uni-oldenburg.de>) tool, based on Eclipse, support CSP-OZ-DC specifications.
- ▶ An operational semantics is provided in terms of *phase event automata*

Outline

- 1 Introduction
 - Context
 - Specification based testing
 - Giving Semantics to UML, Questions
- 2 **UML2 and CSP-OZ-DC Overview**
 - UML Class Diagrams and Protocol State Machines
 - CPS-OZ-DC: Combination of Processes, Data and Time
- 3 From UML to CSP-OZ-DC
 - Illustration: The elevator example
 - (Some) General Translation Rules
- 4 Conclusions

UML Class Diagrams

In UML, a class diagram describes the static structure (data) of a system by showing

- the system's classes,
- their attributes and methods,
- and the relationships between the classes

Protocol State Machines (PSM)

A PSM is a labeled transition system, used in UML2 to help defining the usage mode of the operations and events receptions of a class, specifying:

- In which context (under which states and pre conditions) they can be used.
- If there is a protocol order between them
- What result is expected from their use

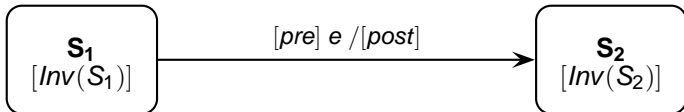
States in Protocol State Machines

- The states of a PSM present an external view of the class that is exposed to its clients.
- A protocol state represents an exposed stable situation of its context class:

When an instance of the class is not processing any operation, users of this instance can always know its state configuration [OMG, 2007].

- This is achieved assigning an invariant $Inv(S)$ to every state S . Then, $Inv(S)$ is the exposed stable situation of S .

Transitions in Protocol State Machines



A protocol transition specifies that

- the associated event can be received for an instance in the origin state under the initial condition (*pre*), and
- at the end of the transition, the destination state will be reached under the final condition (*post*)

CSP-OZ-DC

Combination of Processes, Data and Time

Defined by Jochen Hoenicke in 2006, this language combines in the same formalism three widely used formal languages:

- CSP([Hoare, 1985]): to describe behavioural aspects, such as sequential and concurrent behaviour and synchronous communication
- ObjectZ([Smith, 2000]): to specify complex data operations, in an object oriented style
- Duration Calculus ([Zhou-Hansen, 2004]): to specify timing requirements

CSP-OZ-DC Classes

NAME

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

The *Channel Declarations* part declare the interface of the class.

Every declaration has the form

$$\text{chan } c : [p_1 : T_1; \dots; p_n : T_n], \text{ or}$$

$$\text{local chan } c : [p_1 : T_1; \dots; p_n : T_n],$$

where

- c is the channel name,
- p_1, \dots, p_n is the (possible empty) list of parameter names, and
- T_1, \dots, T_n are the type declarations of the parameters.

CSP-OZ-DC Classes

NAME

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

- The process definition part specifies the behavior of the class using CSP process.
For example:

$$\text{main} \stackrel{c}{=} a \rightarrow P \square b \rightarrow Q$$

$$P \stackrel{c}{=} (e \rightarrow c \rightarrow S \parallel d \rightarrow c \rightarrow Q)$$

$$Q \stackrel{c}{=} a \rightarrow b \rightarrow Q$$

$$S \stackrel{c}{=} \dots$$

- The events used in this part must be declared in the *Channel Declaration* part.

CSP-OZ-DC Classes

NAME

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

The type and constant definitions have the same syntax as global type and constant definitions in Z.

$PIN : \mathbb{N}$

$Prime : \mathbb{P} \mathbb{Z}$

$PIN \leq 9999$

$Color ::= Red \mid Yellow \mid Green$

Their scope is limited to the class in which they are declared.

CSP-OZ-DC Classes

NAME

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

Declarations

Predicate

- The declarations of the state schema are referred to as the state variables and the predicate as the state invariant.
- The state invariant restricts the possible values of not only the state variables but also the constants.
- The state variables and constants collectively define the class attributes.

CSP-OZ-DC Classes

NAME _____

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

- The Init schema gives the initial values of the state variables,

INIT _____

Predicate

- The declaration part is always implicitly the state schema of the current class

CSP-OZ-DC Classes

NAME

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

com_Name

$\Delta(v_1, \dots, v_n)$

Input and Output Variables

Predicate

- *Name* is the name of the channel that is associated with this operation
- v_1, \dots, v_n are the state variables changed by the operation
- The *Predicate* part specifies the pre and post condition

CSP-OZ-DC Classes

NAME

Channel Declarations

Process Definition

Type and Constant

Definitions

State

[Init]

Operations

Duration Calculus

- This part is a list of “Duration Calculus Counterexample Formulae” that restrict the timing behavior of the CSP-OZ-DC Class.
- A counterexample formulae express what timing behavior the system should not have.

Duration Calculus Formulas

Examples

- “If P change from *true* to *false*, then event e must occur in at most three seconds”

$$\neg \diamond \left(\lceil P \rceil \wedge \lceil \neg P \rceil \wedge \ell > 3 \wedge \boxplus e \right)$$

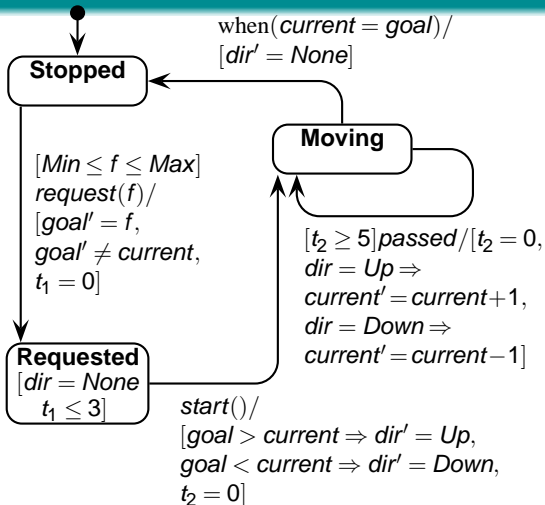
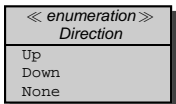
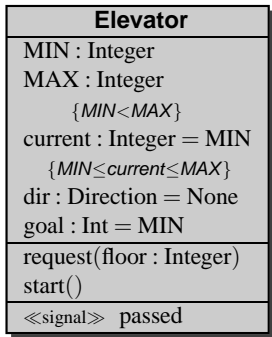
- “Two succesives occurrences of event e are separated with a time spam of at least five seconds”

$$\neg \diamond \left(\downarrow e \wedge \ell < 5 \wedge \downarrow e \right)$$

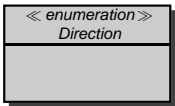
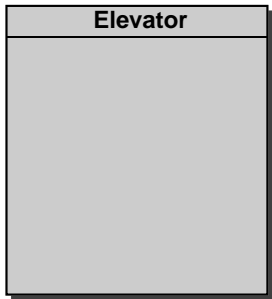
Outline

- 1 Introduction
 - Context
 - Specification based testing
 - Giving Semantics to UML, Questions
- 2 UML2 and CSP-OZ-DC Overview
 - UML Class Diagrams and Protocol State Machines
 - CPS-OZ-DC: Combination of Processes, Data and Time
- 3 From UML to CSP-OZ-DC
 - Illustration: The elevator example
 - (Some) General Translation Rules
- 4 Conclusions

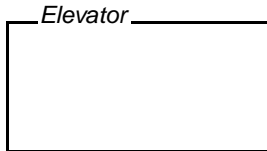
Example



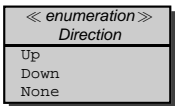
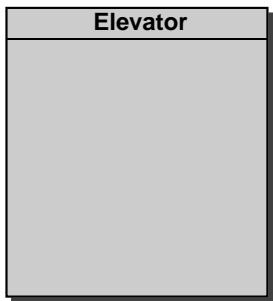
Translating the Class Diagram



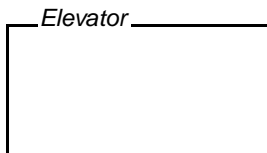
Direction ::=



Translating the Class Diagram



Direction ::= Up | Down | None



Translating the Class Diagram

Elevator

MIN : Integer

MAX : Integer

{MIN < MAX}

current : Integer = MIN

{MIN ≤ current ≤ MAX}

dir : Direction = None

goal : Int = MIN

request(floor : Integer)

start()

«signal» passed

Elevator

chan request : [floor : \mathbb{Z}]; start, passed : []

Min, Max : \mathbb{Z}

Min < Max

current, goal : \mathbb{Z}

dir : Direction

Min ≤ current ≤ Max

com_request

floor? : \mathbb{Z}

INIT

goal = Min

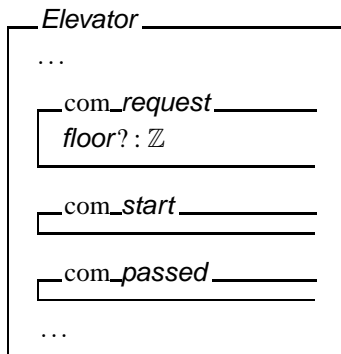
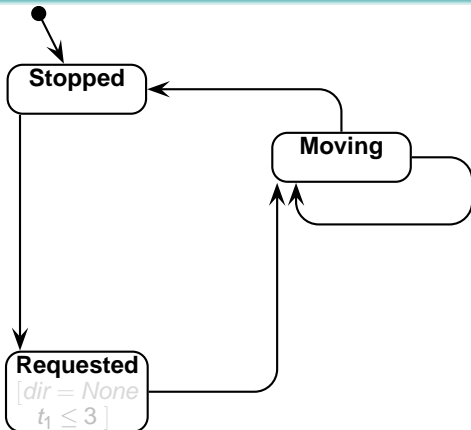
current = Min

dir = None

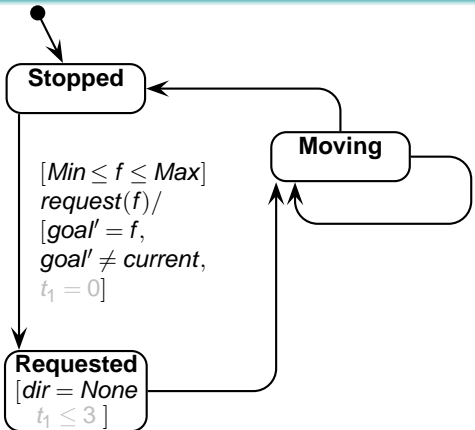
com_start

com_passed

Adding Structure



Adding Structure



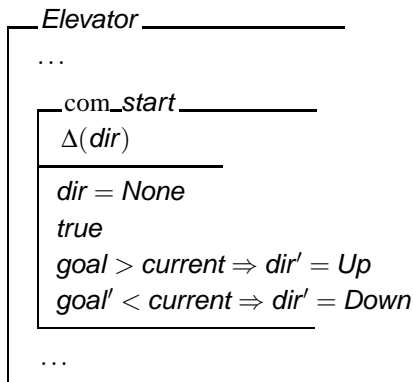
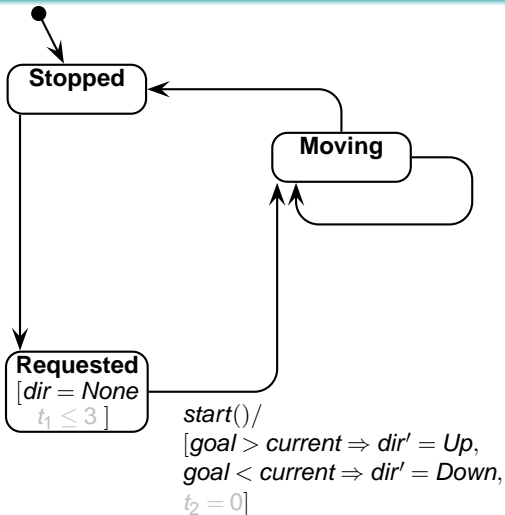
Elevator

...

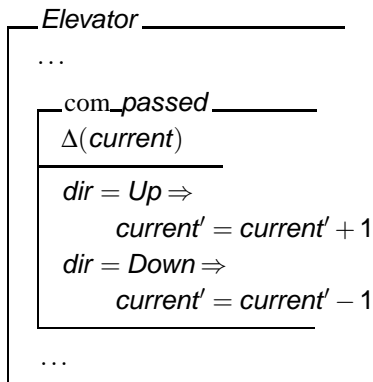
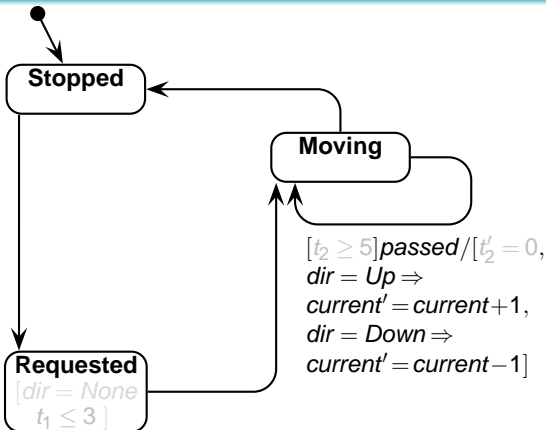
com_request $\Delta(goal, dir)$ $floor? : \mathbb{Z}$ $Min \leq floor? \leq Max$ $goal' = floor?$ $goal' \neq current$ $dir' = None$

...

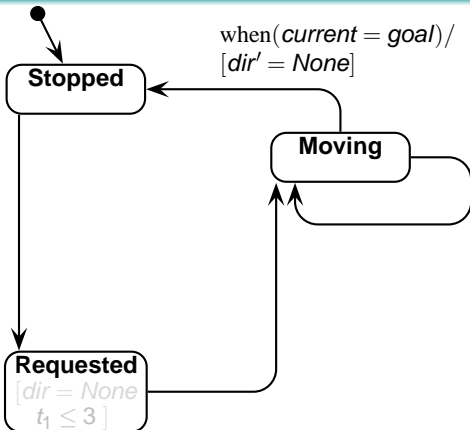
Adding Structure



Adding Structure



Adding Structure



Elevator

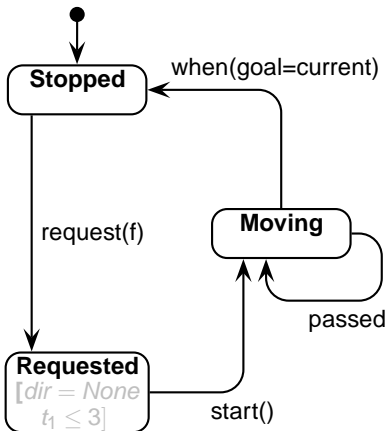
```
...
chan change1 : []
...

com_ change1
Δ(dir)

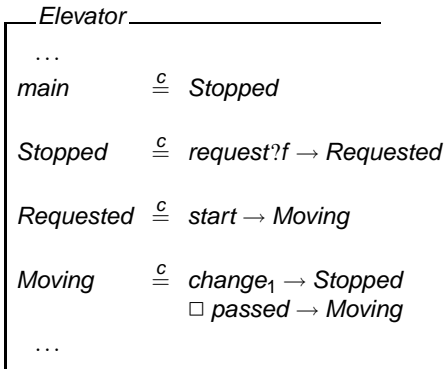
current = goal
dir' = None
...

```

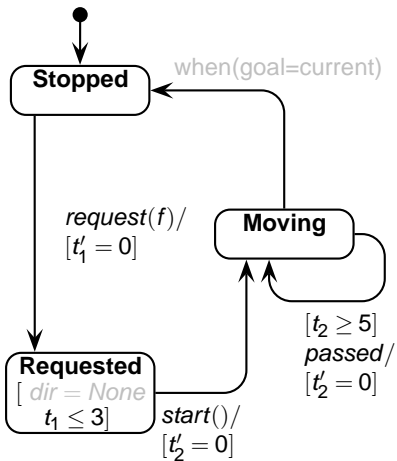
Adding Behavior



Process Definition part:



Adding Real Time Properties



Duration Calculus part:

Elevator _____

...

inRequested : \mathbb{B}

...

$\neg \diamond (\downarrow request \wedge [inRequested] \wedge l > 3)$

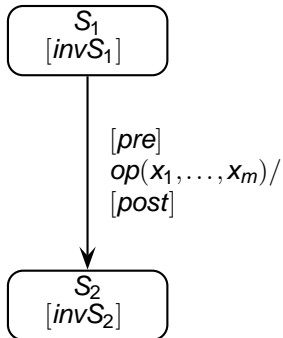
$\neg \diamond (\downarrow start \wedge l < 5 \wedge \downarrow passed)$

$\neg \diamond (\downarrow passed \wedge l < 5 \wedge \downarrow passed)$

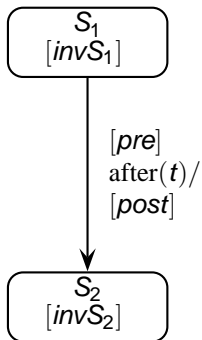
General Translation Rules

- Our semantics aims to be as formal as possible, enabling in the future the construction of a tool support for the translation process.
- We will present here some general translation rules in terms of the UML2 and CSP-OZ-CD concrete syntax,
- The next step of my work will comprise the definition of the translation rules in terms of the abstract syntax of UML2 and CSP-OZ-DC

Call Events


 com_op
 $\Delta(\text{Var}(\text{invS}_2) \cup \text{Var}'(\text{post}))$
 $\text{arg}^?_1, \dots, \text{arg}^?_m$
 $\text{Tr}_Z(\text{invS}_1)$
 $\text{Tr}_Z(\text{pre})[x_1/\text{arg}^?_1, \dots, x_m/\text{arg}^?_m]$
 $\text{Tr}_Z(\text{post})[x_1/\text{arg}^?_1, \dots, x_m/\text{arg}^?_m]$
 $\text{Tr}_Z(\text{invS}_2)[sv_1/sv'_1, \dots, sv_n/sv'_n]$

Time events



Class

...

```
chan relTimeEvent1 : []
```

```
inS1 : ℔
```

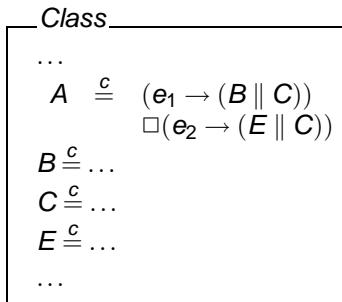
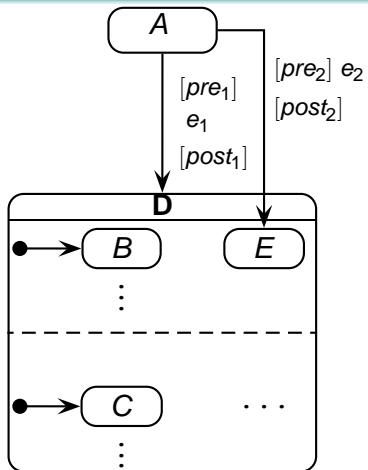
...

```
com_relTimeEvent1
```

Δ list, pre and post conditions as in call events

```
¬◇([inS1] ∧ l > t ∧ ∃ relTimeEvent1)
```

Behavior



Outline

- 1 Introduction
 - Context
 - Specification based testing
 - Giving Semantics to UML, Questions
- 2 UML2 and CSP-OZ-DC Overview
 - UML Class Diagrams and Protocol State Machines
 - CPS-OZ-DC: Combination of Processes, Data and Time
- 3 From UML to CSP-OZ-DC
 - Illustration: The elevator example
 - (Some) General Translation Rules
- 4 Conclusions

Related Work

- [Kim-Carrington, 1999] UML Class diagram translation in terms of Object-Z Classes:
 - We use this translation for the class diagram part
 - No behavioral model
 - No time model

Related Work

- [Ng-Butler, 2003] Semantic for Behavioral States Machines in CSP:
 - No time model
 - No data model
 - Based on behavioral state machines
 - The translation has a non-terminating problem when the state machine has a particular structure.

Related Work

- [Kamuller-Helke, 2000] Formalization of Class Diagrams and Behavioral State Machines in Isabelle/HOL.
 - No time model
 - The behavior and the data model are translated as independent entities, difficulting the reasoning about the complete model.
 - The translation into a theorem prover gives automatically a tool support, where properties of the system can be formally demonstrated.

Conclusions

Summary of my work in the first half of the internship

- State of the art overview
- Survey of UML constructs: UML Class Diagrams and Protocol State Machines
- Survey different languages for the specification of data, behavior and time, in particular CSP-OZ-DC
- Definition of a time model for Protocol State Machines, based on timed-automata [Giese-Burmester, 2003]
- Definition of general transformation rules

Future Work

- Time and Behavior Translation:
 - Add more constructs to the time model for PSM, (like intervals), to simplify the real-time specification part.
 - Define a new transformation function for the CSP part, to reduce the number of process definitions.
- Formalization:
 - Abstract Syntax Specification
 - Formal translation in terms of the abstract syntax

Bibliography

SESAME Project and Related Work



S-K. Kim and D. Carrington. *Formalizing the UML class diagram using Object-Z.*

UML'99, pp. 83-98, Springer Verlag, October 1999.



F. Kammüller and S. Helke. *Mechanical Analysis of UML State Machines and Class Diagrams.*

ECOOP 2000, Cannes, June 2000.



M. Y. Ng, M. Butler Towards Formalizing UML State Diagrams in CSP (SEFM'03), 2003



B. Ries First Annual Report of the BFR 05/075
Technical Report, University of Luxembourg.



H. Giese and S. Burmmester Real-Time Statechart Semantics
Technical Report n tr-ri-03-239, University of Paderborn, Germany

Bibliography

Formal Languages

-  **Object Management Group.** *Unified Modeling Language: Superstructure.*
<http://www.omg.org/cgi-bin/doc?formal/07-02-03>.
-  **Jochen Hoenicke** *Combination of Processes, Data, and Time*
PhD thesis, University of Oldenburg, July 2006.
-  **C. A. R. Hoare.** *Communicating Sequential Processes.*
Prentice-Hall International, 1985
-  **G. Smith** *The Object-Z Specification Language*
Kluwer Academic Publisher, 2000
-  **C. Zhou and M.R. Hansen** *Duration Calculus: A Formal Approach to Real-Time Systems*
Springer-Verlag, 2004

Questions?