Rhapsody in C++

Jan Blumenthal

Universität Rostock

Seminar „Technische Informatik“

08.02.2000

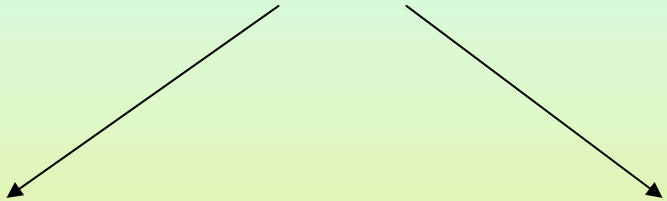
Gliederung

- Einleitung
- Features
- Beispiele
  - Hello World
  - Elevator
- Highlights
- Fazit

Rhapsody in C++

- Spezifikation von Software
- Objektorientierter Entwurf
- Automatische Godegenerierung
- Anwendung in eingebetteten Systemen
- Echtzeitfähige Software

Systeme zur Softwaresynthese



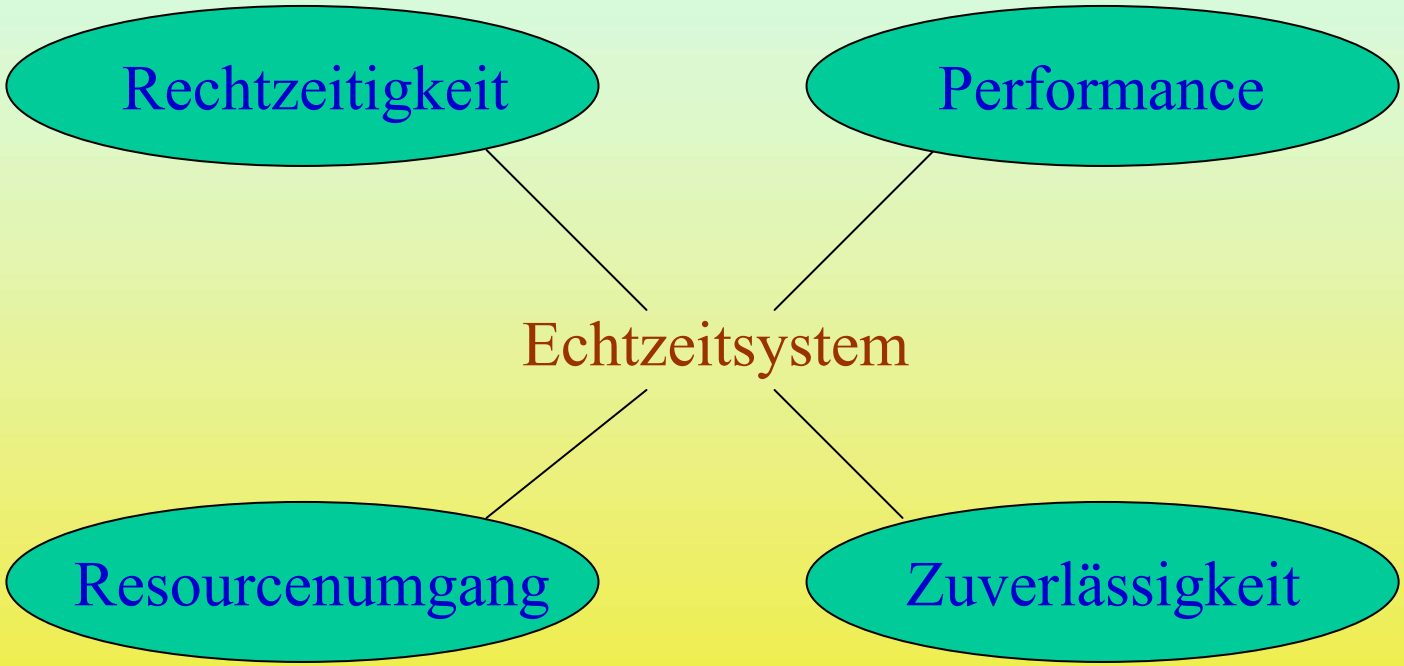
Nicht Echtzeitfähig

- STP
- Rational Rose
- Together

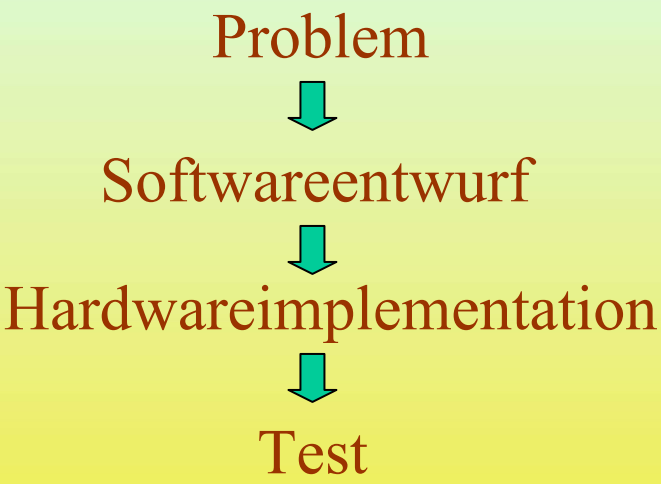
Echtzeitfähig

- Rhapsody
- Room

Echtzeit - Was heißt das?



Herkömmliche Softwareentwicklung



Nachteile:

- Software wird unübersichtlich
- Keine wiederverwendbare Software
- Hardwareabhängig
- Fehleranfällig, Fehlersuche schwierig
- Debugging/Monitoring nahezu unmöglich
- Programmdownload auf Zielhardware
- Echtzeitverhalten fraglich

Ansatz: Softwaresynthese

Problem

Objektorientierte Analyse

OOP-Entwurf

Codegenerierung

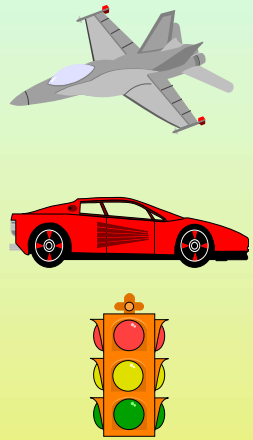
Test

Vorteile:

+ Objektorientiert
+ Automatische Codeerzeugung
+ Übersichtlicher Entwurf
+ Keine direkte Hardwareanbindung

Frage: Wie erfolgt die Hardwareanbindung?

Rapid Prototyping



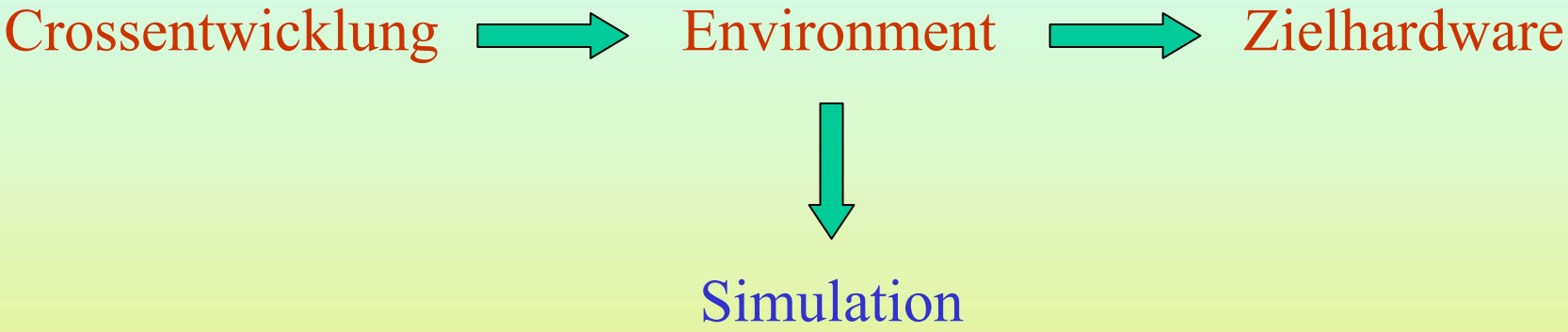
VxWorks

Solaris

PSOS PPC/X86

Crossentwicklung → Environment → Zielhardware

Debugging



Vorteile:

+ Simulation der Hardware
+ Einfaches Debugging
+ Schnelle Fehlersuche
+ Zeitersparnis
+ Gewohnte Entwicklungsumgebung

Was kann Rhapsody ?

+ Objektorientierter Entwurf
+ Codegenerierung
+ Abbildung auf verschiedene Betriebssysteme
+ Einfache Fehlersuche
+ Debugging
+ Echtzeitfähig

Microsoft
Borland
Solaris 2
VxWorks
PSOS x86/PPC

- Nur C/C++ unterstützt
- Betriebssystem auf Zielhardware nötig

Echtzeit mit Rhapsody

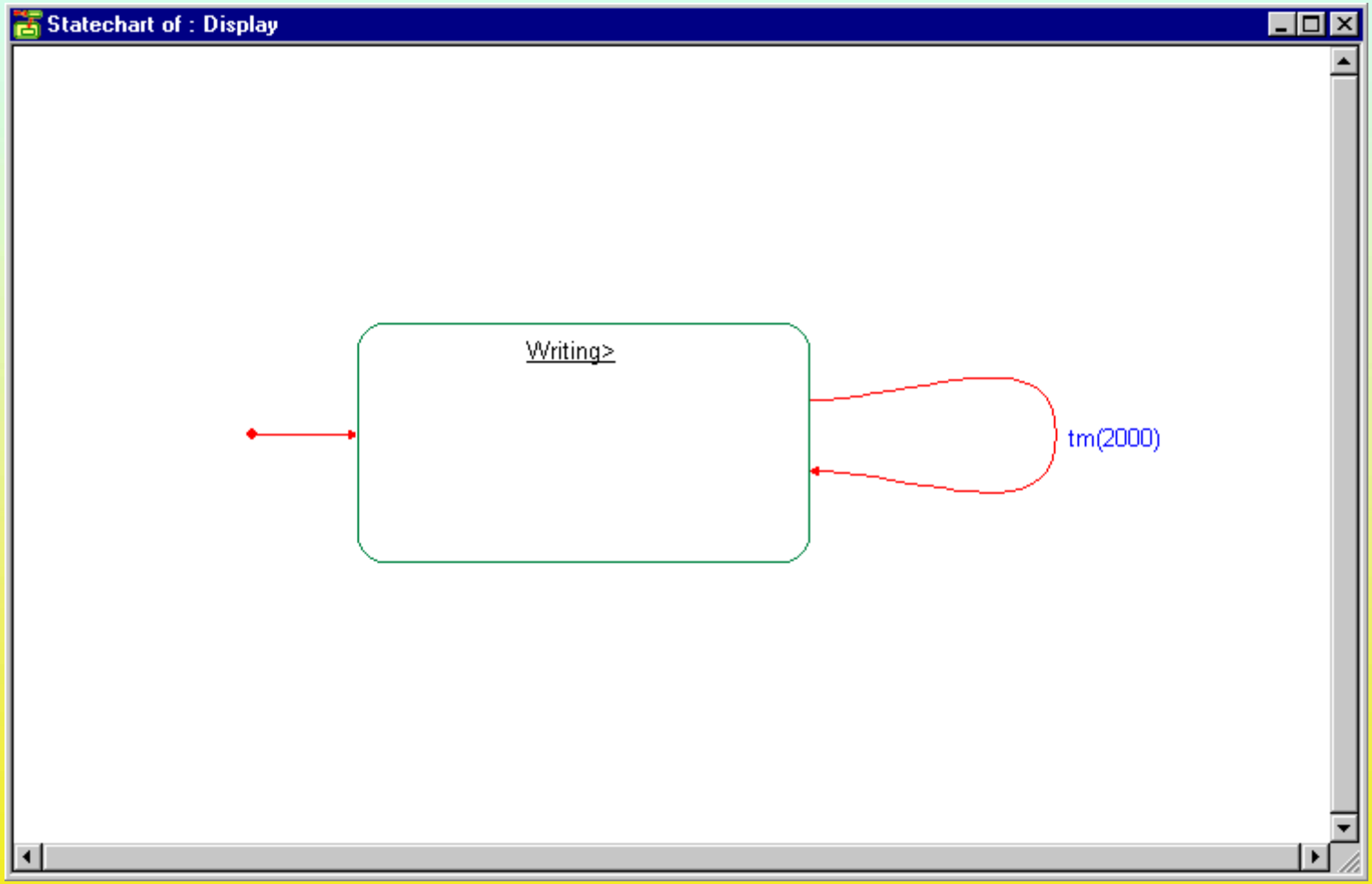
- OXF-Framework
  - Klassenbibliothek zur Resourcenverwaltung
- Statische Architekturen
- Lokale Speicher
- Sequenz Diagramme

Hello World

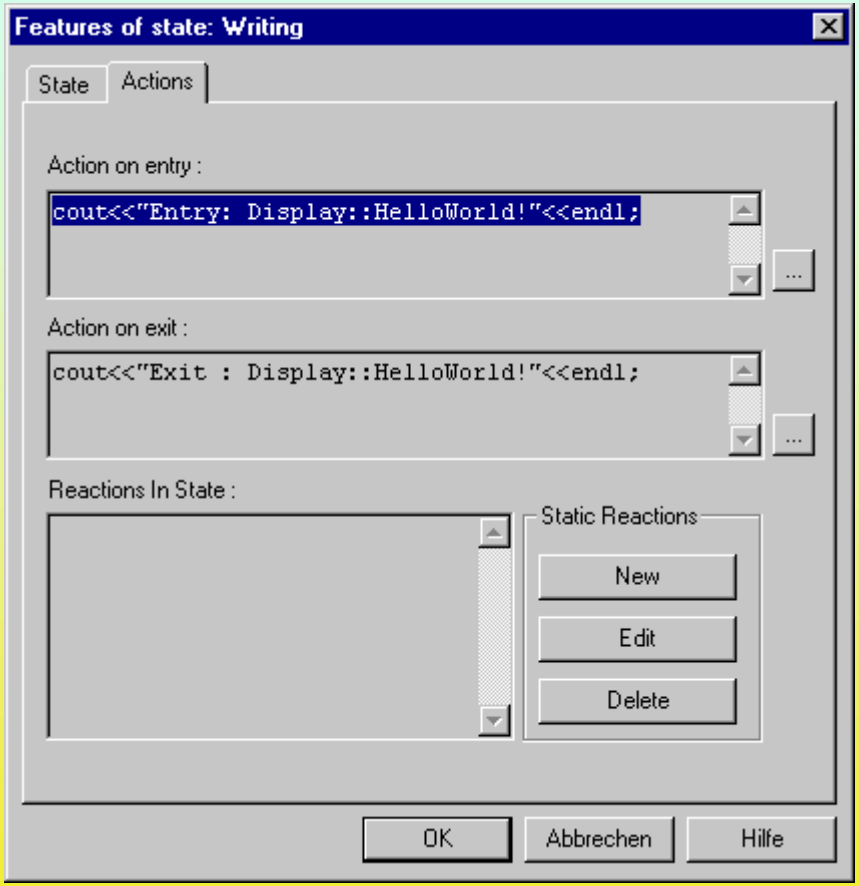
Aufgabe: „Hello World“ alle 2 sec ausgeben.

- Class Display
  - Memberfkt Display::Writing();
  - Statechart
- Object Model Diagram
- Component-Configuration

Statechart: Class Display



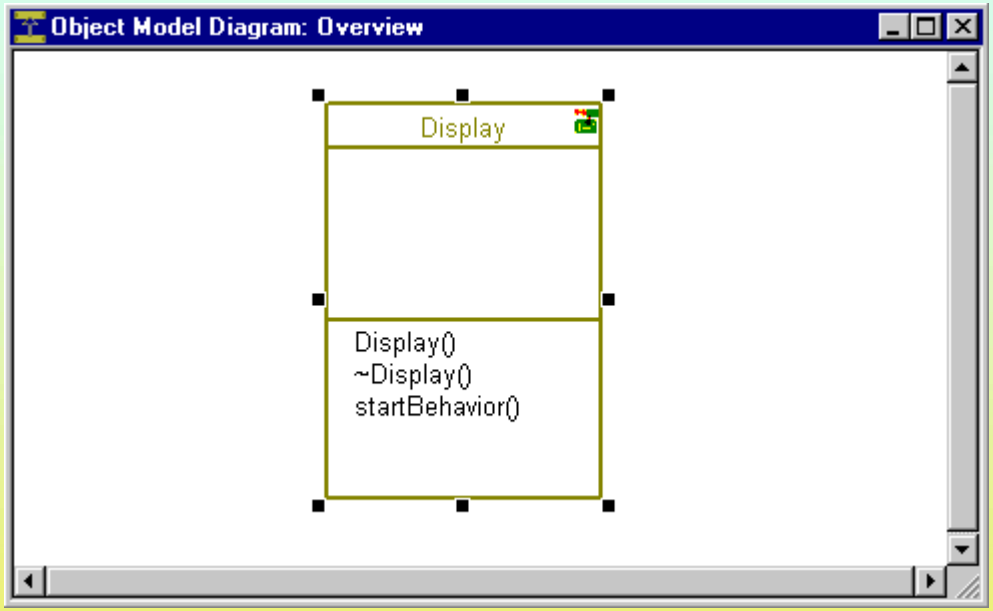
State: Writing



Stateentry

Stateexit

Object Model Diagram



- Statische Struktur des Systems
- Beziehungen zwischen Klassen

Component Configuration

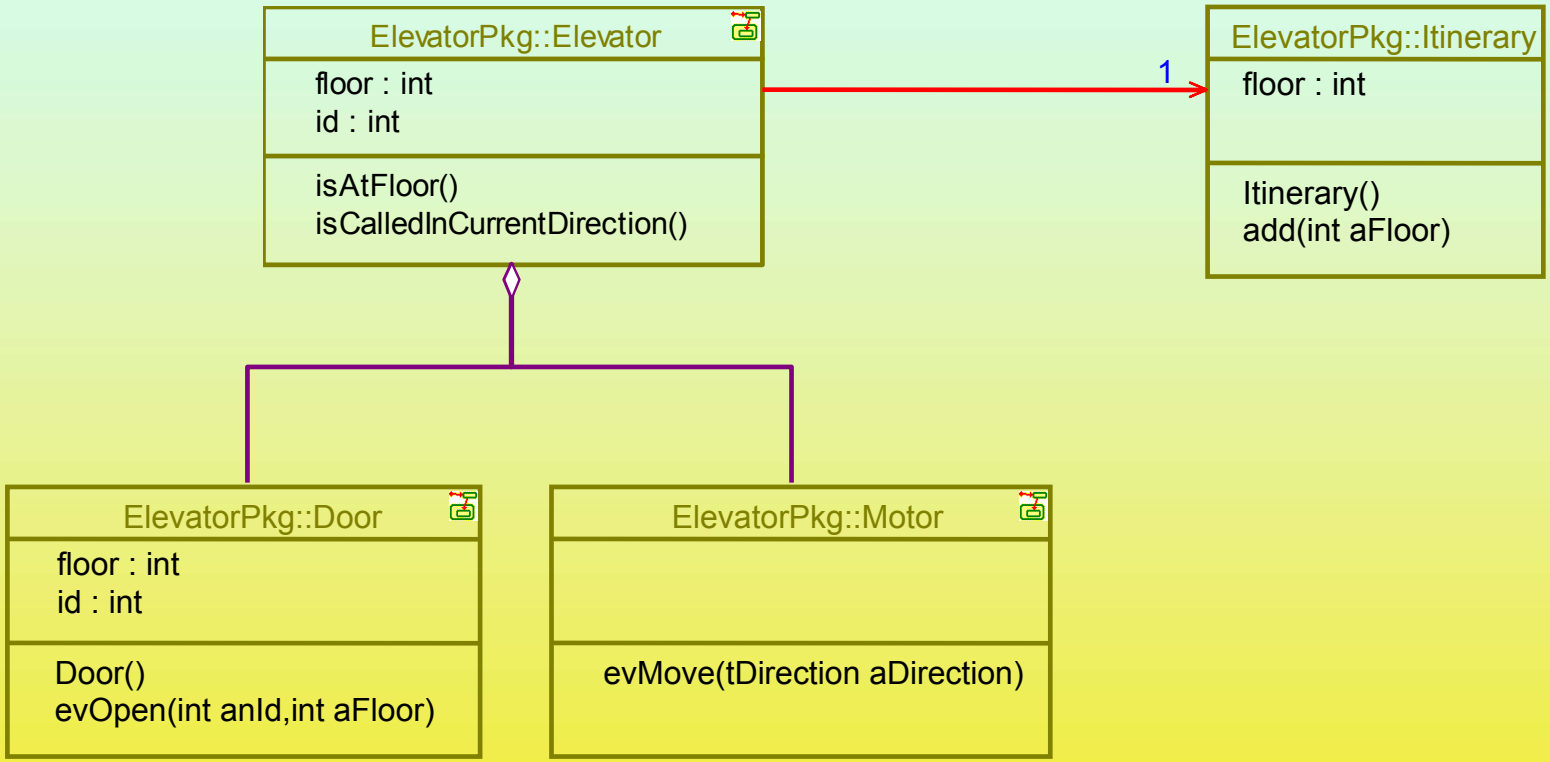
Physikalisches Subsystem:

- Festlegung der Initialinstanzen
- Wahl des Environments
- Compiliert als Library bzw. Programm
- Debuggingmode

HelloWorld:

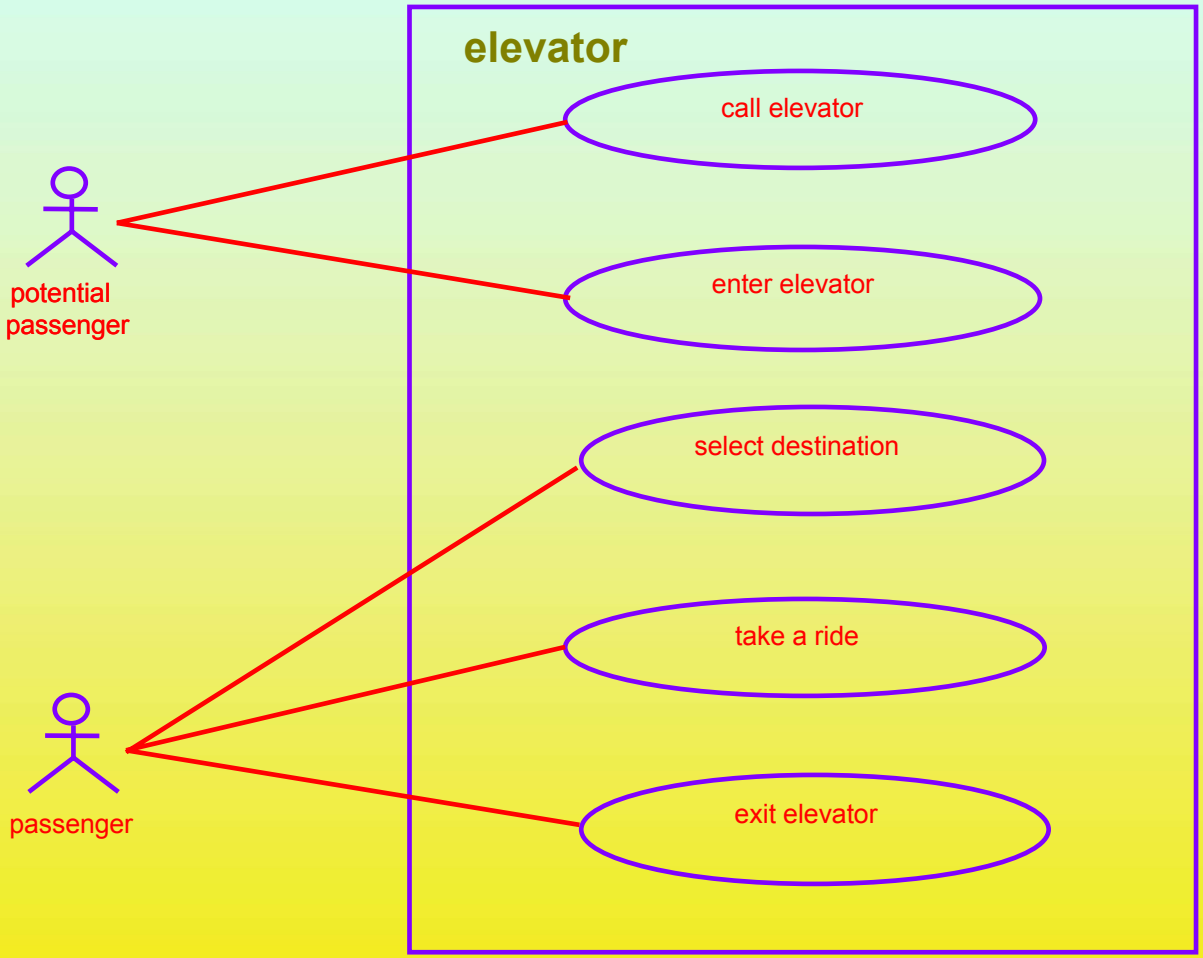
- Initialinstanz: Display
- Environment: Microsoft
- Debuggingmodus: Animation

The Elevator



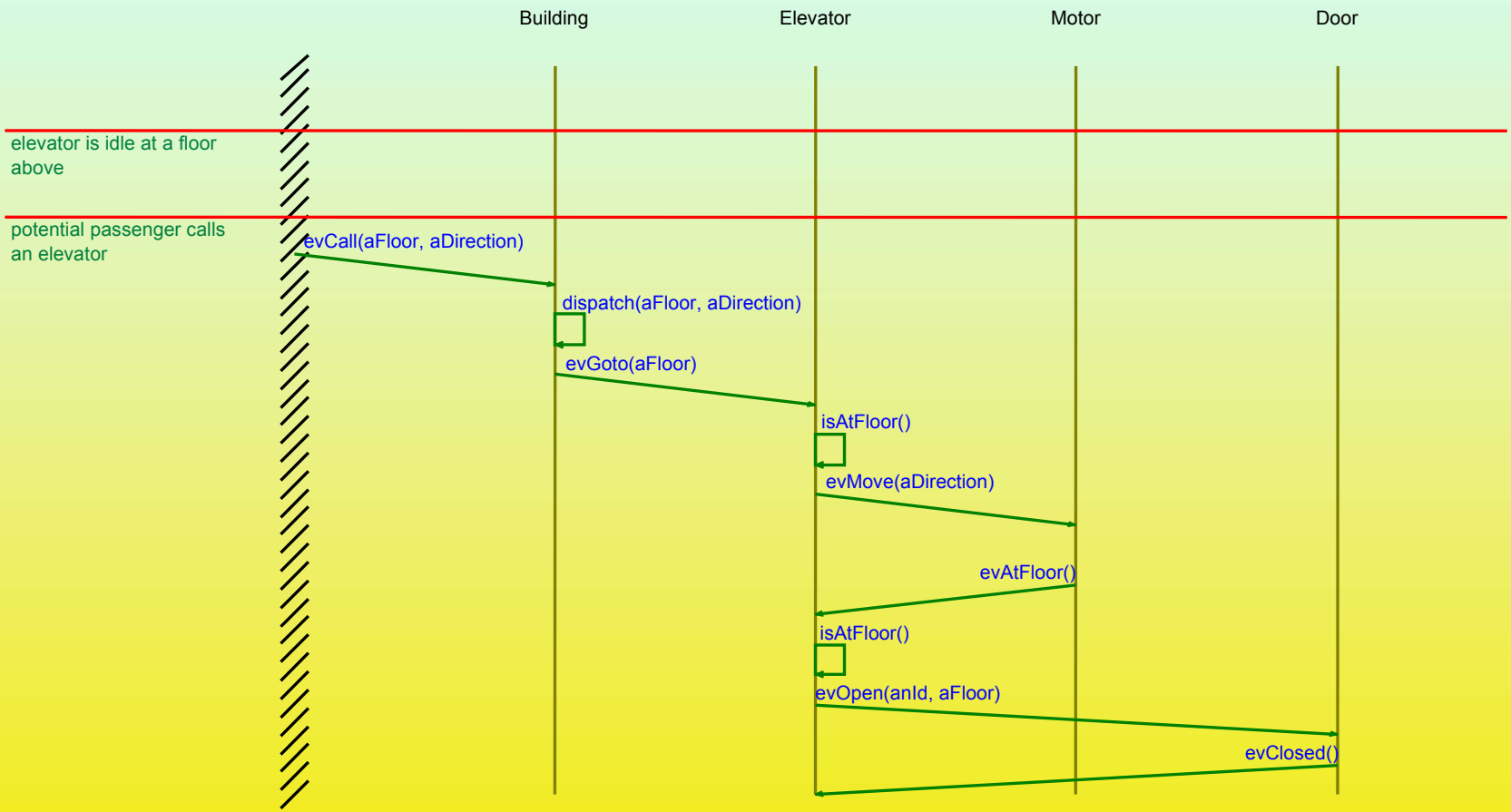
Use Cases

Interaktionen zwischen System und externen Akteuren



Sequenz-Diagramm

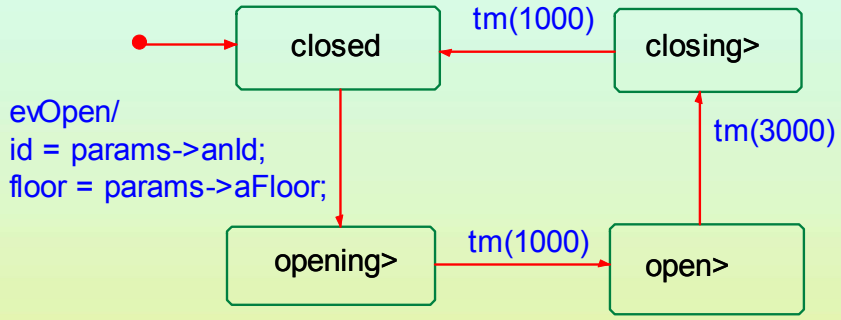
Definiert möglichen Ablauf eines Use-Cases



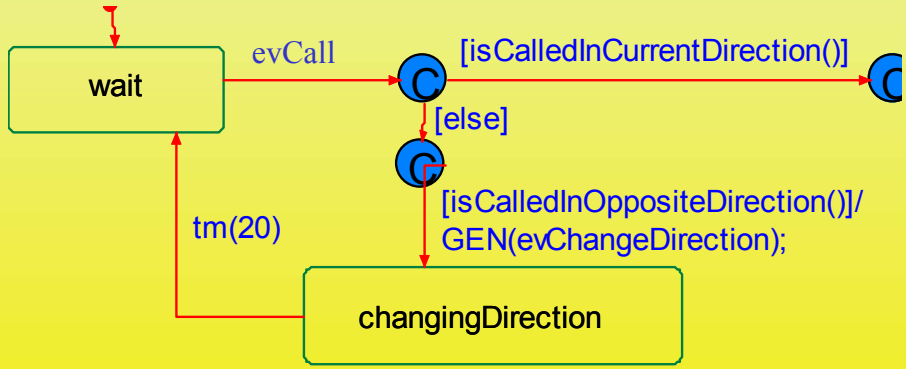
Statechart: Door

- Darstellung aller Zustände der Klasse
- Übergänge durch Events oder Zeitvorgaben
- Verzweigte Übergänge

Statechart: Door

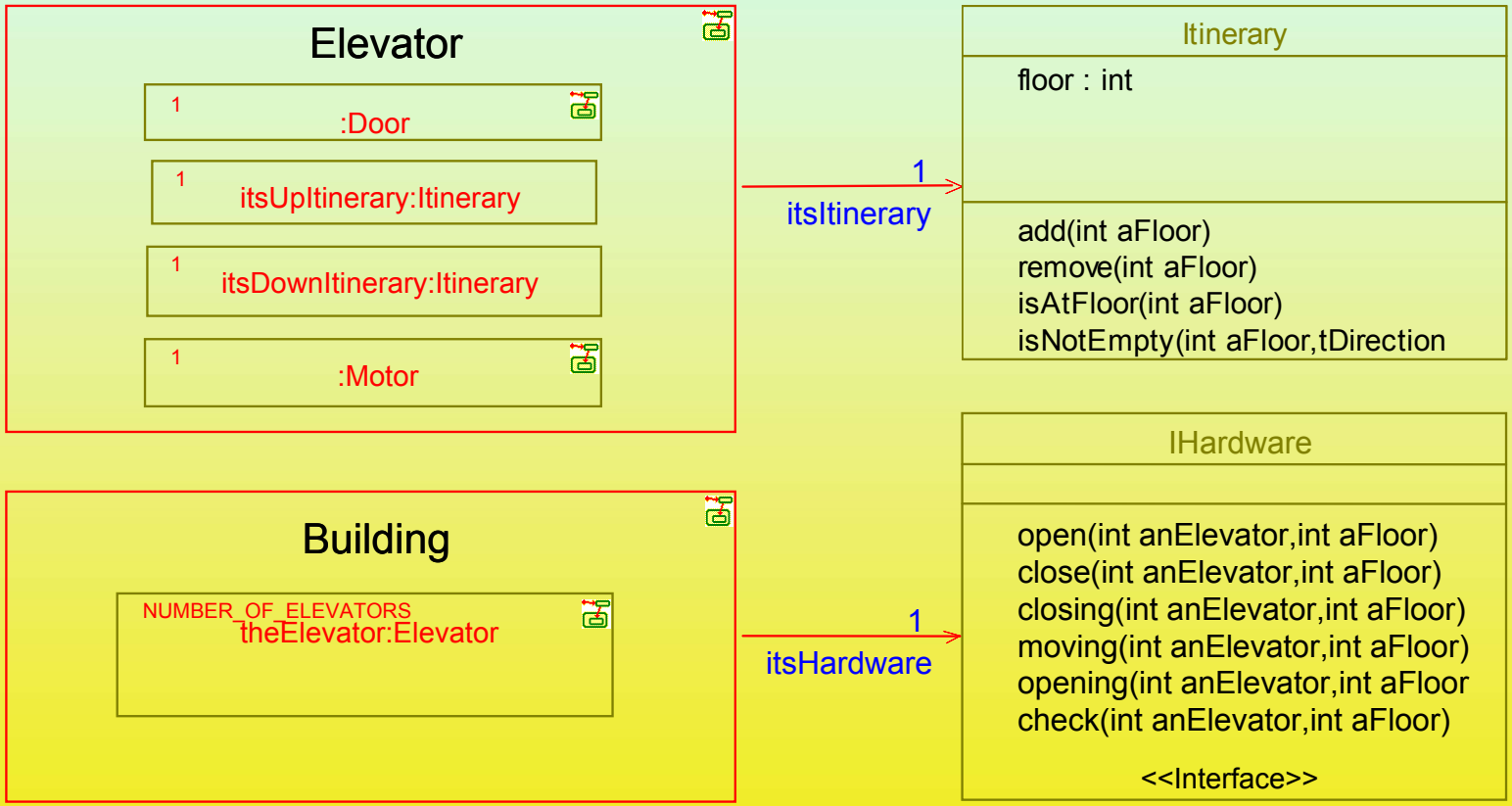


Statechart: Elevator(Auszug)



Object Model Diagram

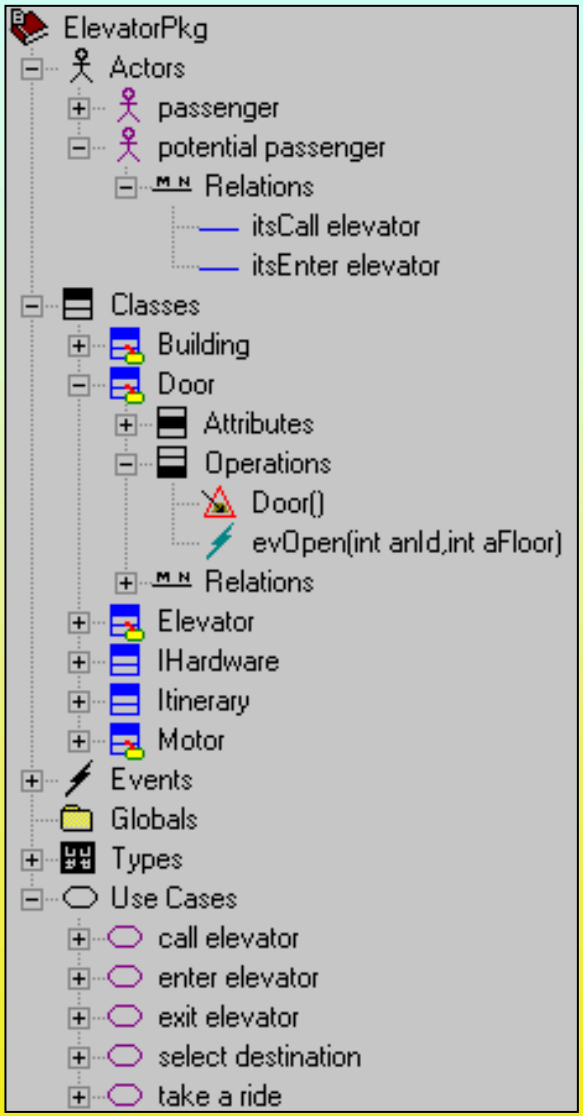
- Statische Struktur des Systems
- Beziehungen zwischen Klassen

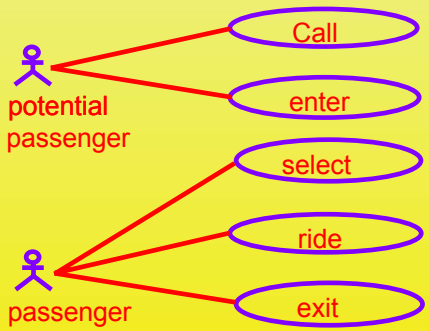


Packages

Zusammenfügen von
- Klassen
- Events
- Typen
- Akteuren & Use Cases

zu einer **Package**



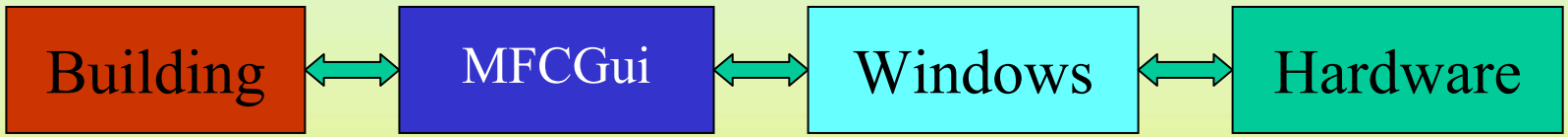


Hardwareanbindung

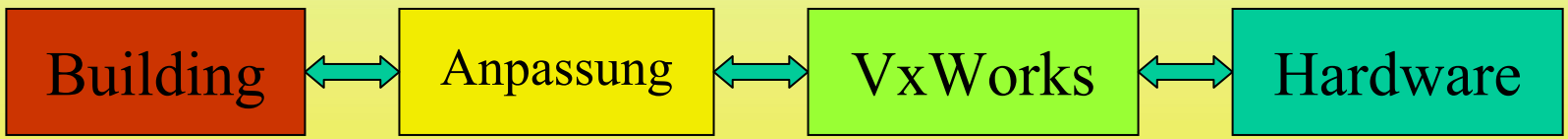
Erfolgt durch Betriebssystem/Environment

- Schnittstelle zwischen Building & Environment anpassen

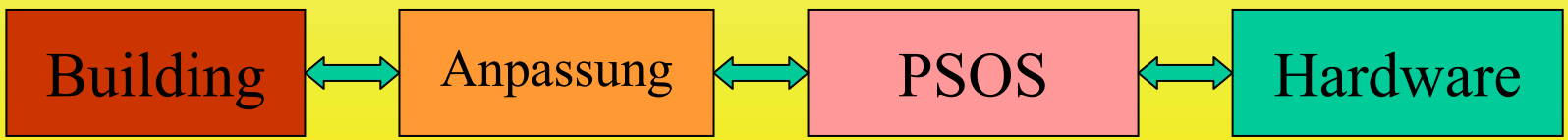
Environment: Microsoft



Environment: VxWorks



Environment: PSOS



Codegeneration

- Anwendung von Design-Pattern

  Bsp: Iterator, State
- Multithreading
- Standard Template Library STL
- Dokumentierter Sourcecode
- Rhapsody-Basisklassen (OXF)

  Bsp: OMReactive, OMThread, LeafState

Hello World - Source

```
//----------------------------------------------------------------------------
// MainHelloWorld.cpp
//----------------------------------------------------------------------------
int main(int  argc, char*  argv[]) {
   if(OXFInit(argc, argv, 6423))
      {
         Display* p_Display = new Display;
         p_Display->startBehavior();
         OXFStart();
         delete p_Display;
         return 0;
      }
   else
      return 1;
};
```

Hello World - Source II

```
int Display::rootState_dispatchEvent(short  id)
{
    int res = eventNotConsumed;
    switch(rootState_active)
    {
        case Writing:
        {
            if(id == Timeout_id)
                {
                    if(((Timeout*)event)->getTimeoutId() == Display_Timeout_Writing_id)
                        {
                            myThread->unschedTm(Display_Timeout_Writing_id, this);
                            //#[ state ROOT.Writing.(Exit)
                            cout<<"Exit : Display::HelloWorld!"<<endl;
                            //#]
                            rootState_subState = Writing;
                            rootState_active = Writing;
                            //#[ state ROOT.Writing.(Entry)
                            cout<<"Entry: Display::HelloWorld!"<<endl;
                            //#]
                            myThread->schedTm(2000, Display_Timeout_Writing_id, this, "ROOT.Writing");
                            res = eventConsumed;
                        }
                }
            break;
        };
    };
    return res;
};
```

Animation

- Debugging zur Laufzeit
  - Setzen von Breakpoints
  - Analyse von Instanzen
  - Beobachten von Statecharts
  - Auslösen von Events
- Siehe Beispiel: Door

Reports

- **Classes:**
  - **Building**

Relations:

itsHardware

Association with IHardware, Multiplicity of 1, Uni-directional

theElevator

Composition of Elevator, Multiplicity of NUMBER_OF_ELEVATORS, Uni-directional

Operations:

Building

Constructor , Public

Body

```
int id=0;
OMIterator< Elevator* > iElevator( theElevator );
for ( iElevator.reset(); *iElevator; ++iElevator )
    (*iElevator)->setId ( id++ );
```

configure

Primitive-operation , Public, Return type is void

Args:

'IHardware*' aHardware

Body

```
setItsHardware ( aHardware );
OMIterator< Elevator* > iElevator( theElevator );
for ( iElevator.reset(); *iElevator; ++iElevator ) {
    (*iElevator)->setItsHardware ( aHardware );
    (*iElevator)->getItsDoor()->setItsHardware(
      aHardware );      }
```

Interfaces

- Class-Importer
  - Importieren von Klassen im Sourcecode
  - Nur logische & physikalische Struktur, keine Relationen
- Rose-Importer
- Doors Interface
- Codedebugging mit Tornado-IDE
- Klasseneditor

Programmbeurteilung

+ Intuitive Bedienung
+ Umfangreiches Tutorial
+ Sehr gute Online-Hilfe
+ Umfassende Anbindung externer Programme
+ Stabil
- Kaum Literatur
- Kein Undo
- Nur eine Instanz startbar
- Nur WindowsNT

Rhapsody in C++

The End

Neugierig geworden?