SDL-2000 Tutorial

Prof. J. Fischer
Dr. E. Holz
Dr. A. Prinz

Session B

Humboldt-Universität zu Berlin

Session Content

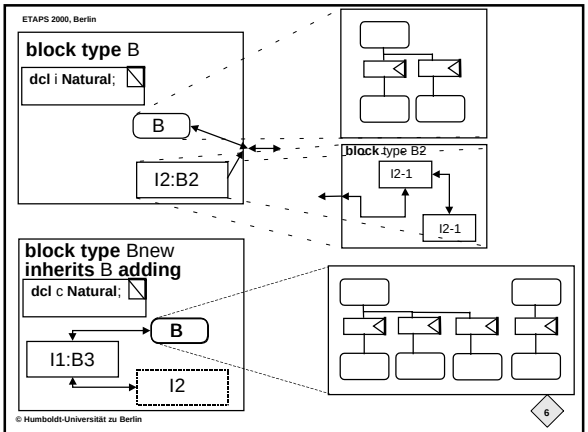
- Object Orientation
  - concepts: inheritance, context parameters, virtuality
  - structural specialisation
  - advanced behavioural specification
- Data Concept
  - interface
  - predefined and user-defined types

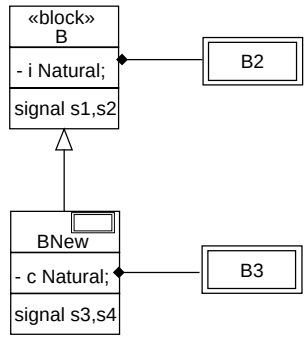
Object-Orientation in SDL

- structural typing concept allows to define the properties of a set of specification elements
- kinds of structural types
  - agent type
  - state type
  - signal (type)
  - procedures (type)
  - data types and interfaces

- type concept corresponds to class concept in other OO languages and notations
  - inheritance
  - virtuality
  - abstraction
  - instance definition &creation
- all instance definitions in SDL are either explicitly or implicitly based on a type

- inheritance allows the definition of a type basing on another (super-) type of the same kind
  - addition of new structural elements
  - addition of new behavioural elements
  - redefinition of virtual elements
- single inheritance supported by all types
- multiple inheritance supported by interface

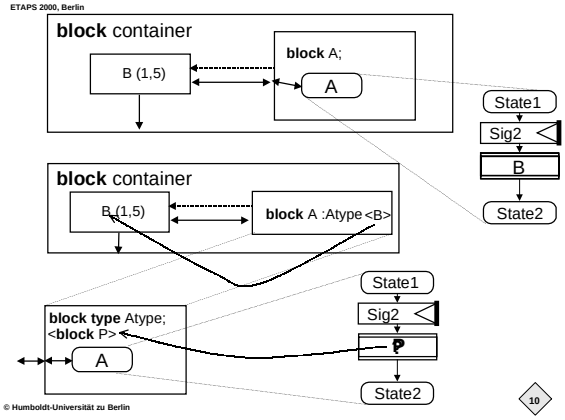




- type definitions for an element can be given in
  - the scope unit where the entity can be given
  - any surrounding scope unit or
  - any type definition for such a scope unit
  - a package
- instances of such a type can be defined
  - where the type is visible and
  - the element is allowed

Context Dependencies

- types may refer to instances and types in their defining context
  - definition of instances may be limited to the same scope unit (e.g. in case of instance references)
- types may refer to instances and types in their instantiation context
  - formal context parameters in defining context
  - actual context parameters in instantiation context



Context Parameter Kinds

- agent (type): block, process
  (type and instance set)
- composite state type
- procedure, remote procedure
- signal
- data type, interface, exception
- variable, remote variable, timer, synonym
- gate

Context Parameter of System Type

Parameter kinds
- block, and process type
- composite state type
- signal
- data type,
- exception,
- interface
- procedure,
- remote procedure
- synonym

not available kinds
- block
- process
- variable
- remote variable
- timer
- gate

Context Parameter

Parameter kinds
- block and process type
- block and process instance set
- composite state type
- signal
- data type, exception, interface
- procedure, remote procedure
- variable, remote variable, timer
- timer, synonym
- gate

can be used in
- block type
- process type
- composite state type
- procedure

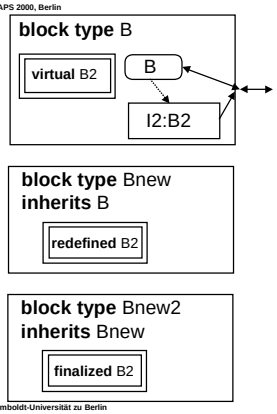
Data types — **support of**
- data type
- synonym

Signal — **support of**
- data type

Interface — **support of**
- signal
- data type,
- exception
- remote procedure
- remote variable

Abstract and Virtual Types

- types marked with the keyword abstract do not directly have instances
  - pure classification
  - used as super-type in an inheritance hierarchy
- virtual types local to another type may be redefined in a specialisation of that type
  - must be contained in a type definition
  - redefinition can be constrained
- system type can not be abstract or virtual

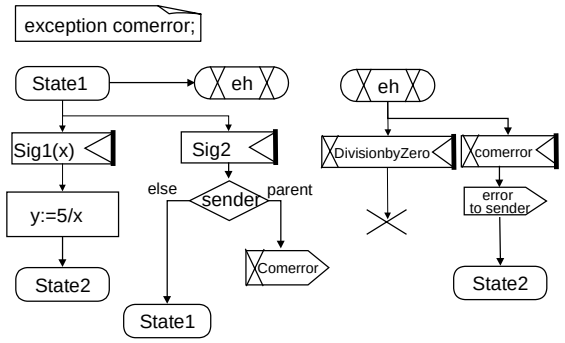


- I2 is instance set of virtual type B2
- I2 is instance set of redefined type B2
- I2 is instance set of finalised type B2
- no further redefinition allowed

- redefinition/finalisation must be
  - subtype of original virtual type or
  - subtype of virtuality constraint
    (**virtual block type** B **atleast** Base)
- references to a virtual or redefined type refer to the most recent redefinition
- finalised types can not be redefined further

Advanced State Machines

- exceptions are used to denote and handle unexpected or exceptional behaviour
- exception denotes the type of cause
- exception handler defines the behaviour to occur after an exception (handle-clauses)
- onexception attaches an exception handler to a behaviour unit
- raise forces a transition to throw an exception



Procedures

- procedures are a means to group and name recurrent behaviour
- procedures can be called during a transition
- notation corresponds to agent state machine
  - local states, inputs and transitions
  - local variables, parameters
- recursion allowed
- procedures are a type

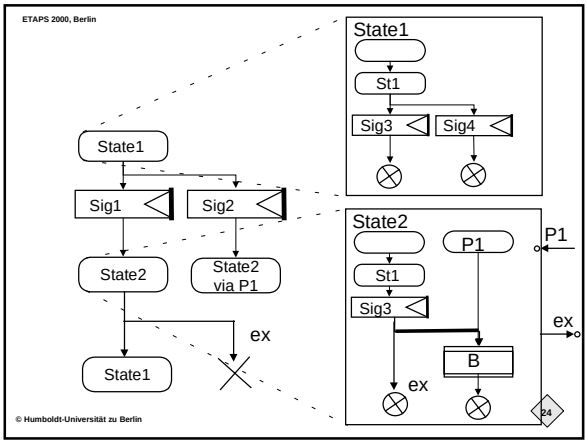
Remote Procedures

- an agent can make its procedures available for other agents
  - remote procedures
  - realized by two-way communication between caller and server (similar to remote variables)
- after a call to a remote procedure the caller is blocked until he receives the procedure return from the server

- remote procedure call may deadlock
  - can be prevented by time limit for execution which raises an exception
- server accepts calls for remote procedures in any state
  - execution may be deferred by *save*
  - execution may be denied by *reject*

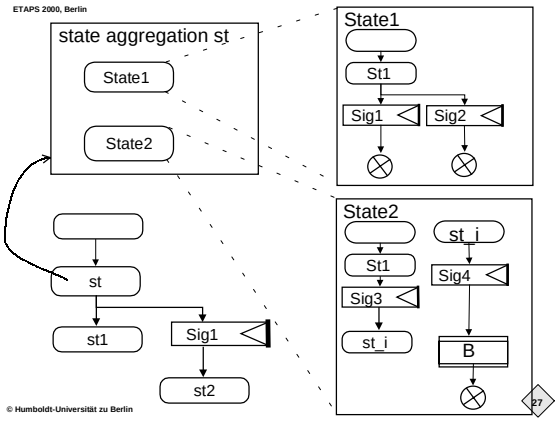
Hierarchical State Machines

- composite states are a means to hierarchically structure state machines
  - nesting of states
  - agent can be in more than one state at a time
  - Harel´s state charts
- composite state is itself a sub-state machine
- state machine of an agent is in fact a top-level composite state



- composite states share agent´s input queue
- internal transitions with the same trigger as external transitions have higher priority
- excactly one transition is executed
  - possibly concatenated with triggerless transitions
- special procedures may be used to define initialisation and finalisation
  - called implicitly upon entering/leaving a composite state

- state aggregations partition the state space of an agents state machine
- each partition handles a different set of the input stimuli
- excactly one partition is executing a transition at any point in time
  - multiple enabled transitions are executed in an interleaved manner

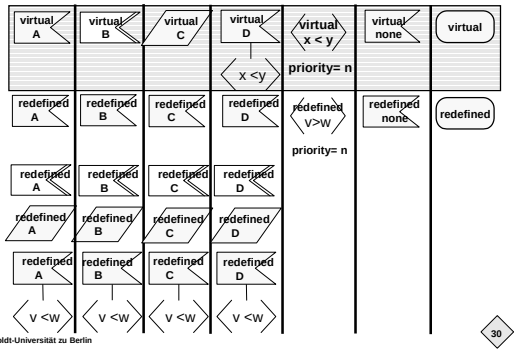


- composite states and state aggregations can be classified
  - composite state type definition
  - typebased composite states/state aggregations
- concept & notation similar to agent types
- instances are static
  - live&die with containing agent
  - multiple instances in the same scope must have different names

Virtual Behaviour Elements

- allow the redefinition or replacement of behaviour elements in a type specialisation
- redefinition and finalisation similar to structural elements
- available for
  - procedures
  - transitions
  - exception handle transitions

Redefinition of Virtual Transitions



Interface and Data Types

- pure typing concept used for typed communication between agents
- interface definition groups and names a set of
  - remote variable
  - remote procedure
  - signal definitions
- gates and channels paths can be typed by interfaces

Implicit Interface

- each agent and agent type introduces an implicit interface
  - same name as agent (type)
- contains all
  - signals accepted by the agents state machine
  - remote variables/procedures provided by agents state machine
- inherits all interfaces on gates connected to agents state machine

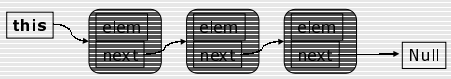
- each interface implies a Pid-Type, that can be used to refer to instances
  - variables
  - receiver in output statements
  - server in imports or remote procedure calls
- dynamic type check tests
  - assignment compatibility
  - provision of remote variables/procedures
  - acceptance of signals

Data Types

- two main kinds of data types
  - value types
  - object types (references)

  conversion possible
- data type definition specifies
  - data elements and structure
  - operators and methods for data manipulation
- package Predefined contains a set of general data type definitions

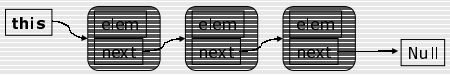

```
object type List <type Elem>;
   struct
        elem          Elem;
        private next  List;
   operators Make(Elem)->List;
   methods   add(Elem)-> List;
```




```
endobject type;
```



```
object type List <type Elem>;
```




```
   operator  Make(e Elem) { return (. e,Null .); }
   method    add(e Elem) {
        dcl last Elem;
        for(last=this,last.next/=Null,last.next);
        last.next := (. e .);
   }
endobject type;
```

predefined data types

data constructions

simple types

- Boolean, Integer, Natural, Real, Character, Bit, Octet
- Charstring, Bitstring, Octetstring
- Time, Duration, Pid

data templates

- Powerset, Bag, Array, Vector

predefined data types

data constructions

- inheritance
- context parameters
- operators, (virtual) methods with algorithmic syntax elements
- local data types, constants, exceptions
- visibility of data elements

literals

structures

choices

Data assignments

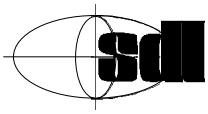
value types	object types
• strong type check • assignment restrictions for specialised types	• references local to agent • polymorphic assignments for specialised types • virtual methods • assignment attempts
• object creation	• value extraction


```
value type Charstring
... /* predefined */
endvalue type
```

```
dcl val_var Charstring,
    ref_var object Charstring;
```

`ref_var := Make('Hello world')`	creates a reference to an object
`val_var := ref_var`	value extraction
`ref_var := val_var`	creates a refererence and copies the value into the object
`val_var := 'foo'`	value assignment

SDL-2000 Tutorial



Prof. J. Fischer
Dr. E. Holz
Dr. A. Prinz

End of Session B

Humboldt-Universität zu Berlin