SDL-2000 Tutorial

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

Humboldt-Universität zu Berlin

Agenda of the Tutorial

- Session A
  - language overview and basics
  - agents and communication
  - behaviour and state machines (I)
- Session B
  - object-orientation
  - behaviour and state machines (II)
  - data, interface
  - context parameter
- Session C
  - introduction to ASMs
  - formal semantics of SDL

Overview and Basics

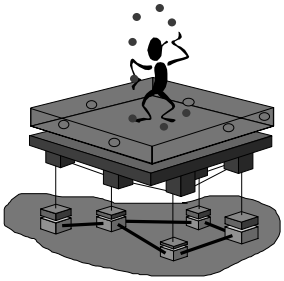
- SDL
  - Specification and Description Language
  - developed and standardised by ITU-T /SG10

  ITU-T Recommendation Z.100

The Influence of DOT

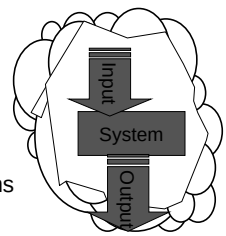
DOT will dominate the IT world in forthcoming years

- objects co-operate via infrastructure
- encapsulation hides complex data behaviour
- interoperability
- re-use of complete applications on different platforms
- extensibility
- implementation language independence



Target Application Area

- distributed object-oriented systems
- reactive systems
- real-time systems
- discrete communication
  - telecommunication systems
  - automotive systems
  - aerospace systems

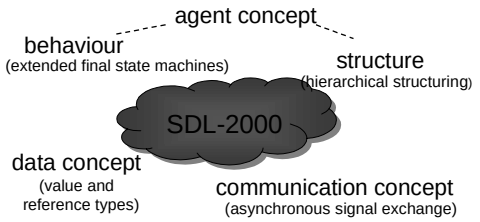


SDL Characteristics

- developed and standardised by ITU-T /SG10
  - long tradition, used in practice (ITU, ETSI, industry)
  - executable formal technique
  - backwards compatibility
- graphical representation of
  - concurrent processes / agents,
  - state-based behaviour
- commercial tools
  - editors, syntax and semantic checkers, simulators, code generators, test sequence generators, ...
- alignment with other techniques
  - ASN.1, OMG- IDL (ITU- ODL), OMG- UML

Basic Concepts

unification/generalisation: system, block, process, service



History

- Z.100 standard is revised every 4 years
- initial investigations starting 1968
- 1976 first standard
- 1988 formal and steady version
- 1992 introduction of object-orientation
- 2000 major language revision

History

1976 first standard: Orange Book
basic graphical language

- improvement of representations
- integration of data
- formal description of the language semantics
- textual syntax (SDL/PR)

1988 standard: Blue Book (SDL-88)
well-defined language
effective tools

History

1988 standard: Blue Book (SDL-88)
well-defined language
effective tools

object-oriented concepts
- inheritance
- virtuality
- abstract generic types
- modules

defined by transformation to SDL-88

1992 standard: White Book (SDL-92)
object SDL

History

1992 standard: White Book (SDL-92)

- revised data model
- simplification and unification of concepts
- harmonizing object modelling support
- improved implementation support
- alignment with UML
- new formal semantics

2000 revised standard SDL-2000

History

1992 standard: White Book (SDL-92)

1995 SDL in combination mit ASN.1 (Z.105)

1996 Amendment 1 (SDL-96)

2000 revised standard SDL-2000

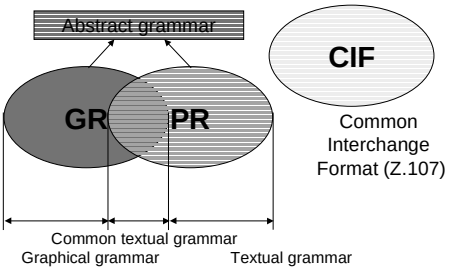
Structure of ITU Recommendation Z.100

- **main text recommendation Z.100** syntax, semantics

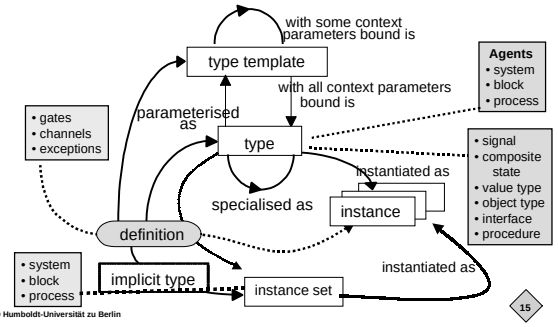
is accompanied by annexes:

- **Annex A** Index of non-terminals
- **Annex D** SDL Predefined data
- **Annex E** Reserved for examples
- **Annex F** Formal Definition (further study needed for SDL 2000)
- **Appendix I** Status of Z.100, related documents and Recommendations
- **Appendix II** Guidelines for the maintenance of SDL
- **Appendix III** Systematic conversion of SDL-92 to SDL-2000
- **Supplement 1** SDL+ Methodology

Representations



Definition of Type Template, Type, Instance, Instance Set



SDL (Base) Types

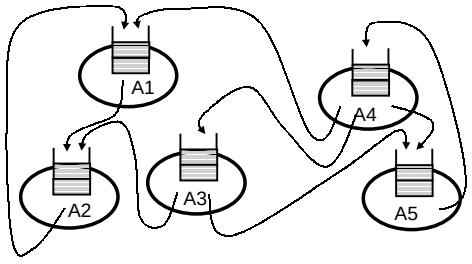
types	inheritance/ context param.	instance creation
• agents		
– system	single/ yes	*implicit*
– block	single/ yes	create, *implicit*
– process	single/ yes	create, *implicit*
• state	single/ yes	*implicit*
• signals	single/ yes	output
• procedure	single/ yes	call
• data types		
– value types	single/ yes	*initialization*,make
– object types	single/ yes	make, clone
– interfaces	multiple/yes	*no*

Agents

- basic specification concept
- model active components of a system

- an agent instance is an extended finite communicating state machine that has
  - its own identity
  - its own signal input queue
  - its own life cycle
  - a reactive behaviour specification

Communication Structure



Communication Principles (1)

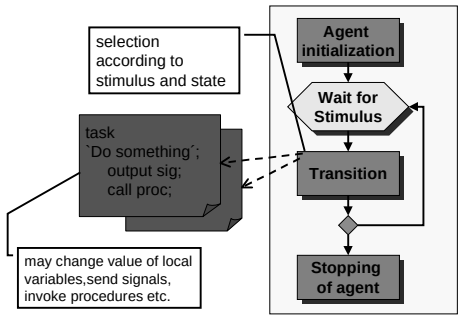
- events: sending and receiving of signals
- signals: carrier of information
  (signal type, signal parameter)
- communication: asynchronous by signal exchange
  between agent instances
- communication channels: with and without
  time delays
- priorities: not for signals
  but for signal processing
- processing: run-to-completion

Agent Composition

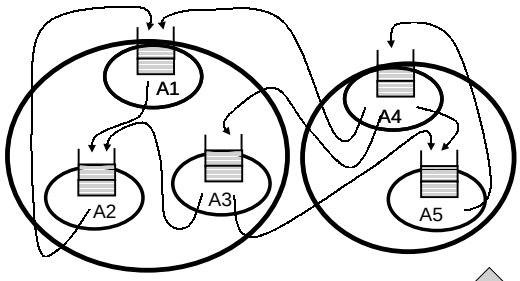
- three main parts
  - attributes
    parameters, variables, procedures
  - behaviour
    implicit or explicit state machine
  - internal structure
    internal agents and communication paths

parts may be optional

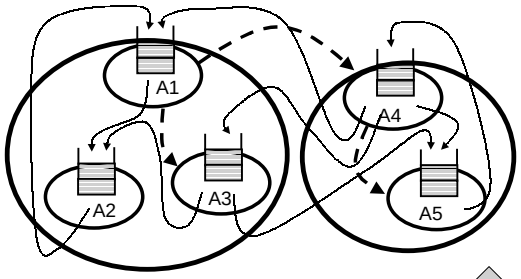
Agent Lifecycle (Behaviour)

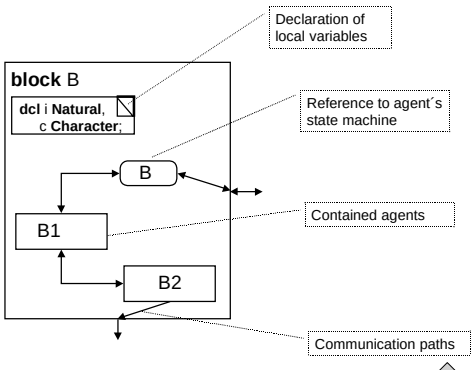


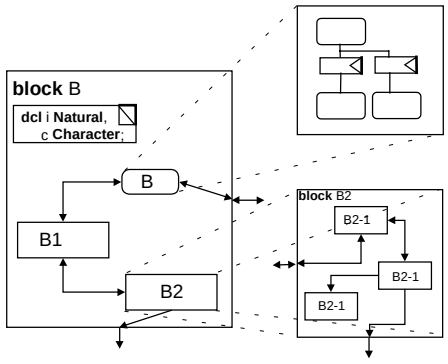
Agent (De-)Composition



Dynamic Agent Structures



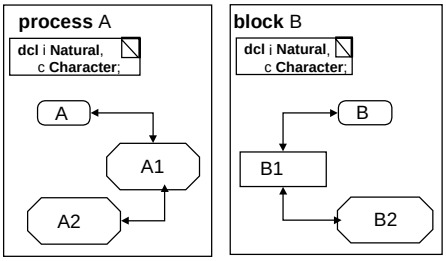




De-Composition of Agents Behaviour

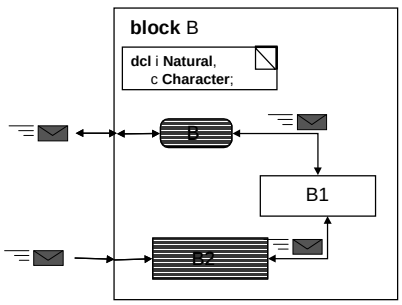
- structural decomposition into internal agents implies also decomposition of behaviour
- container of an agent determines behavioural semantics of its contents
  - concurrent agents: **block**
  - alternating agents: **process**

Agent Kinds



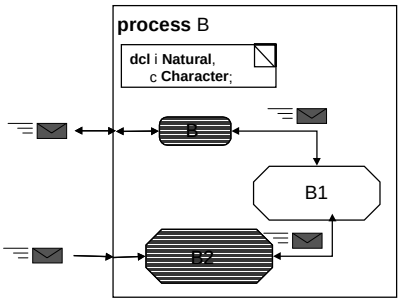
Block as Container Agent

- all contained agents execute concurrently with each other and with the agents state machine
  - multiple threads of control
  - concurrent execution of multiple transitions
  - transitions execute with run-to-completion
- contained agents may be
  - blocks or processes



Process as Container Agent

- all contained agents execute alternating with each other and with the agents state machine
  - at most one transition is executed at any point in time
  - selection is non-determined
  - transitions execute in run-to-completion
- contained agents
  - may be of kind process only

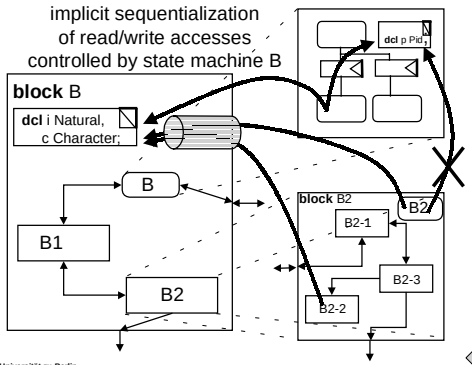


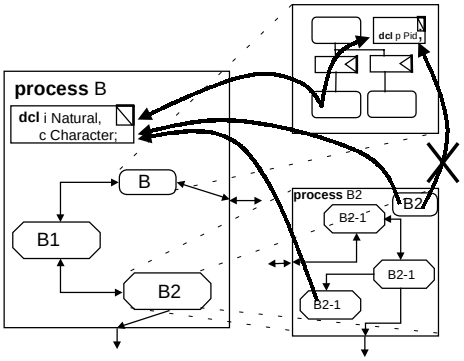
System as Container Agent

- a system is one special (block) agent
  - must be the outermost agent
  - defines the border to the environment
  - defines communication primitives exchanged with the environment

Agent Variables

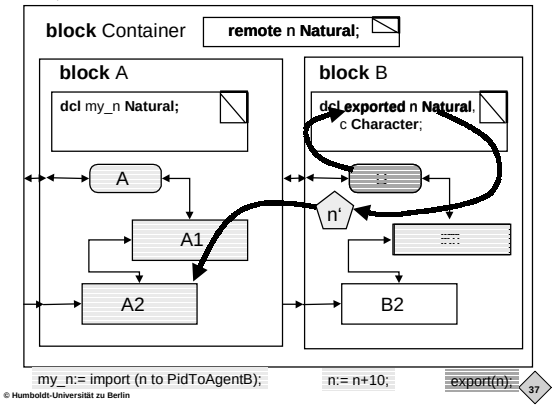
- variables
  - store data local to an agent
  - owned by agents state machine
- private variables, visible to
  - agents state machine
- public variables, visible to
  - agents state machine
  - contained agents
- exported variables, accessible to
  - agents state machine
  - agents which import them





Remote Variables

- short hand notation for a typical communication scenario
  - substitutes signal interchanges with the only purpose to get the value of a variable of another agent instance
  - access with import operation



Communication Principles (2)

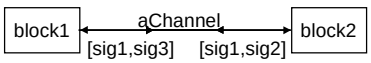
- all communication is based on signal exchange
- a signal carries
  - kind of signal (signal name)
  - user data
  - sender identification (Pid value)

signal aSignal (Natural, Character);

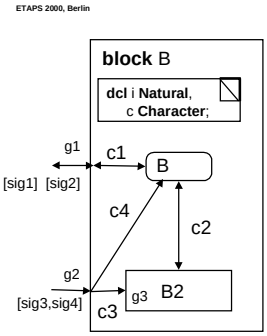
- communication requires a complete path from sender to receiver consisting of
  - gates
  - channels
  - connections
- path may be defined
  - explicitly
  - or implicitly derived

- channel
  - uni- or bi-directional communication path between two endpoints
  - safe and reliable (no loss, no re-ordering,...)
  - delaying or non-delaying transmission
  - restricted to selected set of signals


```
channel aChannel nodelay
        from block1 to block2
        with sig1, sig2;
        from block2 to block1
        with sig1, sig3;
endchannel;
```



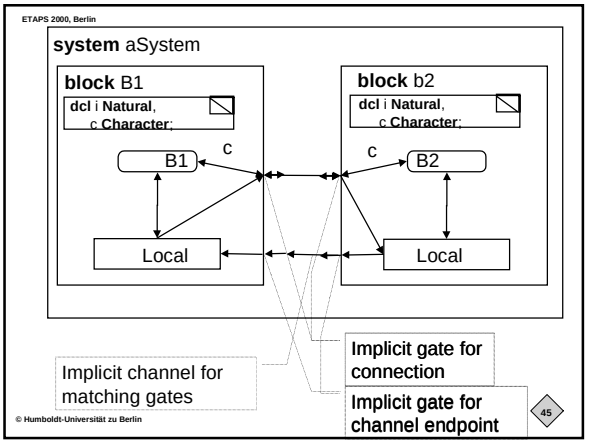
- gate
  - potential named endpoint for a channel at an agent or a state machine
  - uni- or bi-directional
  - constrained by set of signals or by interface (more in session B)
- connection
  - joining/splitting of channels at implicit gate




```
block B;
gate g1 in with sig2;
          out with sig1;
gate g2  in with sig3,sig4;
state B referenced;

block B2 referenced;
channel c1
          from env via g1 to this;
          from this to env via g1;
endchannel;
channel c2        from this to B2;
                  from B2 to this;
endchannel;
channel c3        from env via g2
                  to B2 via g3;
endchannel;
channel c4
          from env via g2 to this;
endchannel;
endblock;
```


- implicit gates
  - introduced for connections
  - introduced for unconnected channel endpoints
  - signal lists derived from channel signal lists
- implicit channels
  - introduced for unconnected gates
  - gates have to have matching constraints



Advanced Communication

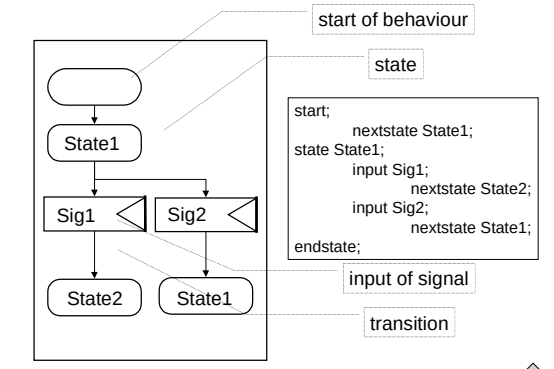
- two-way communication
  - (interrogation) implicitly mapped onto signal exchange
- kinds
  - remote variables
    - read access to variables of other agent
    - no containment relation required
  - remote procedures
    - execution of a procedure by a different agent

Simple State Machines

- behaviour of an agent
  - is specified by a state machine
- two main constituents:
  - states
    - particular condition in which an agent may consume a signal
  - transitions
    - sequence of activities triggered by the consumption of a signal

State Transition

- a state machine is always either
  - in a state waiting for a signal
  - or performing a transition
- a transition results in
  - entering a new state or
  - stopping the agent

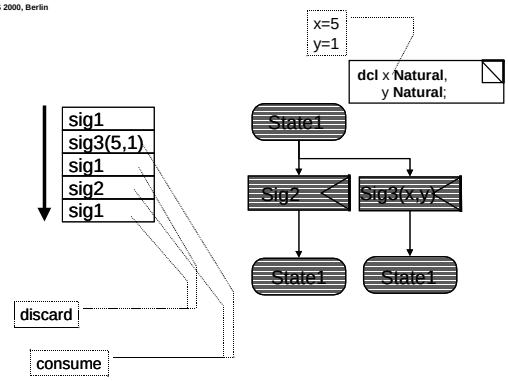


Complete Agent State

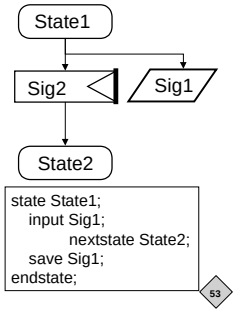
- current complete state is defined by
  - state(s) of the state machine
  - content of the signal buffer
  - values of the local variables
  - state(s) of activated timers
  - cross product of the state of
    - all contained agents and
    - all contained communication channels with delays

Trigger of Transitions

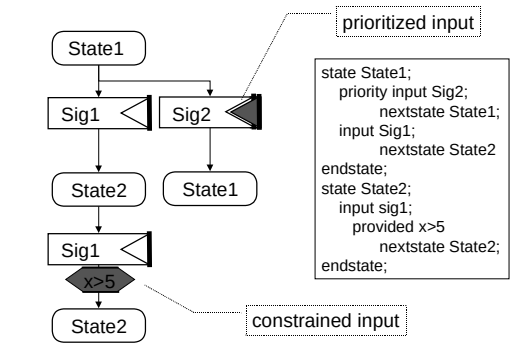
- input removes always the first signal of the input queue
- if there is no input for the first signal, it is discarded
- during an input data carried by a signal may be copied onto local variables
- reference to originating agent is stored in implicit variable **sender**



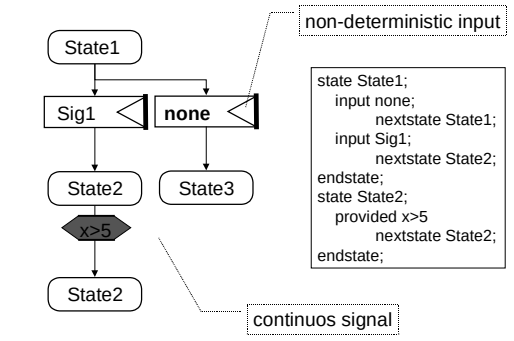
- consumption of signals can be deferred to a later state
  - special saved signals
  - valid until a new state is entered
  - avoids implicit discard



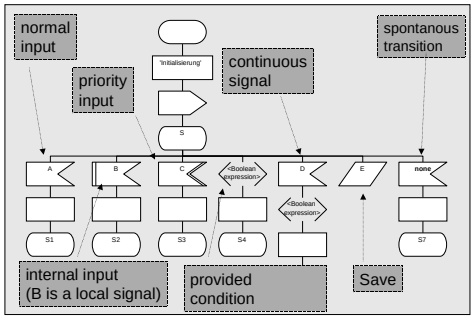
- input of selected signals can be preferred or constrained
  - priority input
    - transition will be selected even if the signal is not the first in the queue
  - enabling condition
    - transition will be selected only in case the attached condition is true (otherwise save)



- transitions may also be triggered without an (explicit) signal
  - continuos signal
    - transition will be selected if attached condition is true and no other transition can be selected
    - queue is empty or all other signals are saved
  - non-deterministic transition
    - transition will be selected non-deterministically and independent from any other transition

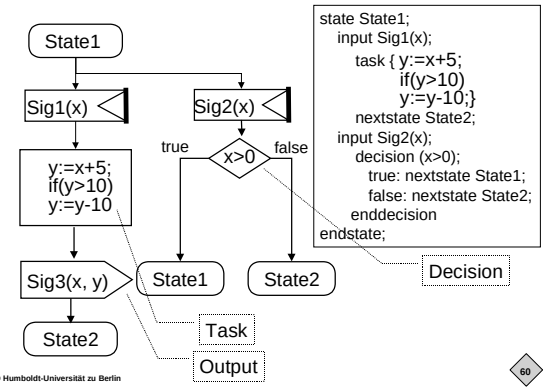


Kinds of Transition Stimuli



Transition Actions

- output
  - generation and adressing of signals (identification of receiver or communication path)
- task
  - sequence of simple or compound statements
  - informal text
- decision
  - branching a transition into a series of alternative paths



Time in SDL

- real-time systems (derived from SDL spec.) must respond within certain time limits
- consumption of time depends on the underlying execution system
- the global actual time can be accessed via special operator now
- progress of time can be observed

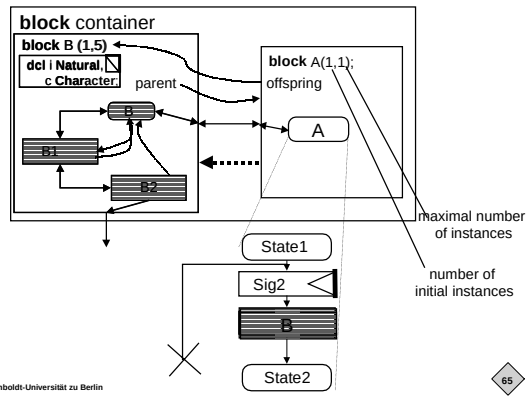
Time in SDL

- agents may have timers (stop watches) with expiration time
  timer t1, t2:= 10;
- timeout signals are inserted in agent signal queue on expiration
- timer operations
  - **set**(now + 25, t1)
  - **reset**(t2);
  - **dcl** b Boolean:= **active**(t2);
  - **input** (t);

Creation and Stopping of Agents

- the execution of a stop results in entering a special implicit stopping-state
  - no more execution of transitions
  - if agent contains no further agent instances, it will cease to exist
  - otherwise
    it will only handle the implicit get and set operations for the agents variables

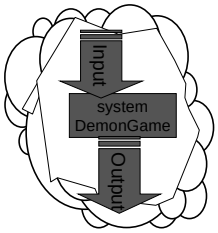
- performance of a create-request action results in the existence of a new agent instance in the indicated agent set
  - creators implicit offspring variable references newly created instance
  - createe´s implicit parent variable references creator agent



Example: DemonGame

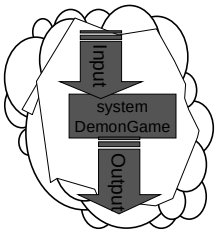
Reactive system

- unlimited (unknown) number of players
- player can play against the system
  - after registration
  - until closing the game
- multiple independent players

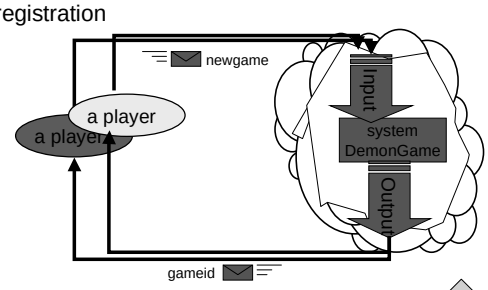


Game

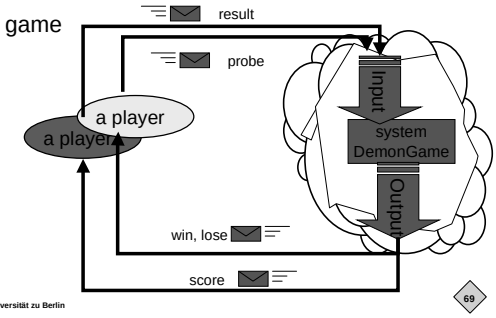
- the value of a hidden variable owned by the system is changed non-deterministically
- a player has to guess whether or not the the value is odd
  - if so, he will win a point
  - if not, he will lose a point
- a player can ask for his score any time



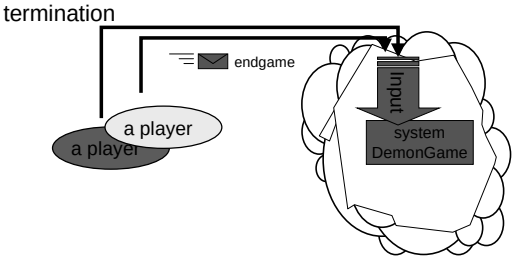
registration



game



termination



used SDL concepts

- **system, block, process (instance set)**
- **dynamic process generation**
- **signal addressing per Pid values (using sender, offspring, parent)**
- **data type: set of Pid values**
- **spontaneous transition**

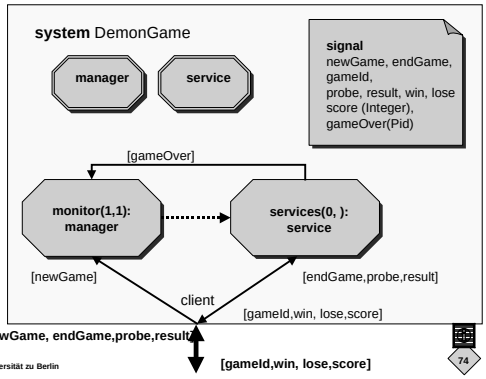
Predefined Generic Type for Definition of Sets

```
value type Powerset <type Itemsort>
  operators
  empty                                -> Itemsort;          /* empty set */
  "in"   (Itemsort, this Powerset)  -> Boolean;           /* is member */
   Incl  (Itemsort, this Powerset)  -> Powerset;          /* include item */
   Del   (Itemsort, this Powerset)  -> Powerset;          /* delete item */
   "<"   (this Powerset, this Powerset) -> Boolean;       /* proper subset*/
   ">"   (this Powerset, this Powerset) -> Boolean;       /* proper superset*/
   "<=" (this Powerset, this Powerset) -> Boolean;        /* subset */
   ">=" (this Powerset, this Powerset) -> Boolean;        /* superset */
   "and" (this Powerset, this Powerset) -> Powerset;      /* intersection */
   "or" (this Powerset, this Powerset) -> Powerset;       /* union */
    ...
endvalue type Powerset;
```

Concrete Data Type: Set of Pid Values

value type PIdSet **inherits** Powerset <Pid>{};

dcl player PIdSet:= empty;



Differences between SDL-92 and SDL-2000

- SDL-2000 is case-sensitive
- two spellings for keywords
  all uppercase or all lowercase
- additional keywords of SDL-2000

  abstract, aggregation, association, break, choice, composition, continue, endexceptionhandler, endmethod, endobject, endvalue, exception, exceptionhandler, handle, method, object, onexception, ordered, private, protected, public, raise, value

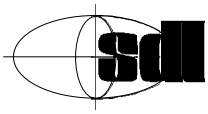
- removed keywords

  all, axioms, constant, endgenerator,endnewtype, endrefinement, endservice, error, fpar, generator, imported, literal, map, newtype, noequal, ordering, refinement, returns, reveal, reverse, service, signalroute, view, viewed

- not available constructs in SDL-2000

  view expression, generators, block substructures, channel substructures, signal refinement, axiomatic definition of data, macro diagrams

SDL-2000 Tutorial



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

End of Session A

Humboldt-Universität zu Berlin