Towards SDL Markup Language

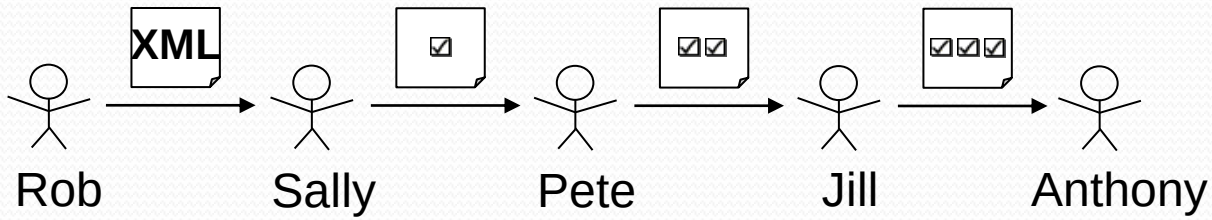
M. Bagic Babac, M. Kunstic, D. Jevtic
University of Zagreb, Croatia
Faculty of Electrical Engineering and Computing

Content

- Introduction
- SDL Markup Language
- SDL-ML Example
- Conclusion

Introduction: XML applications

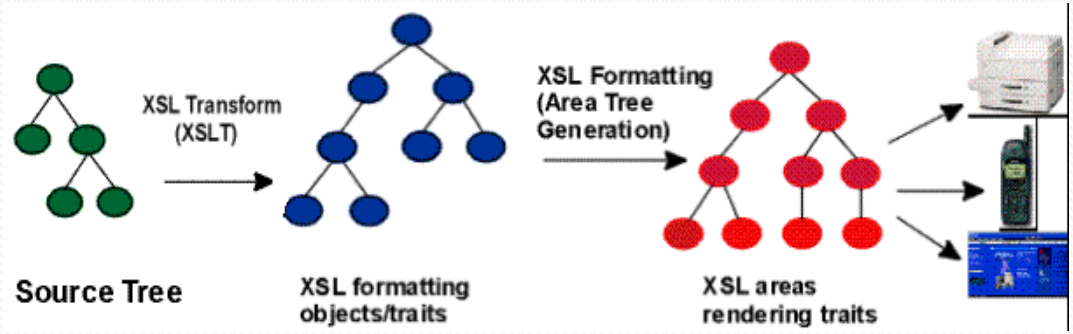
- **An XML based version of SDL for data interchange.**
- XML is widely used for data storage and data transmission:
  - web based,
  - extensible,
  - license-free,
  - platform independent,
  - easy display formatted XML data in browser,
  - can be used with database queries that return XML, etc.



XML transformations

- Media for data interchange
- B2B transactions on the Web
- Workflow applications - applications where documents are moved around a community of people who each perform on it

- Transformation and presentation:



[Source: Extensible Stylesheet Language Version 1.0, http://www.w3c.org/TR/xsl/]

SDL Markup Language

- An XML based interchange format for SDL
- **Z.106 ITU-T, Common Interchange Format (CIF) for SDL, 2002**. (SDL-PR)

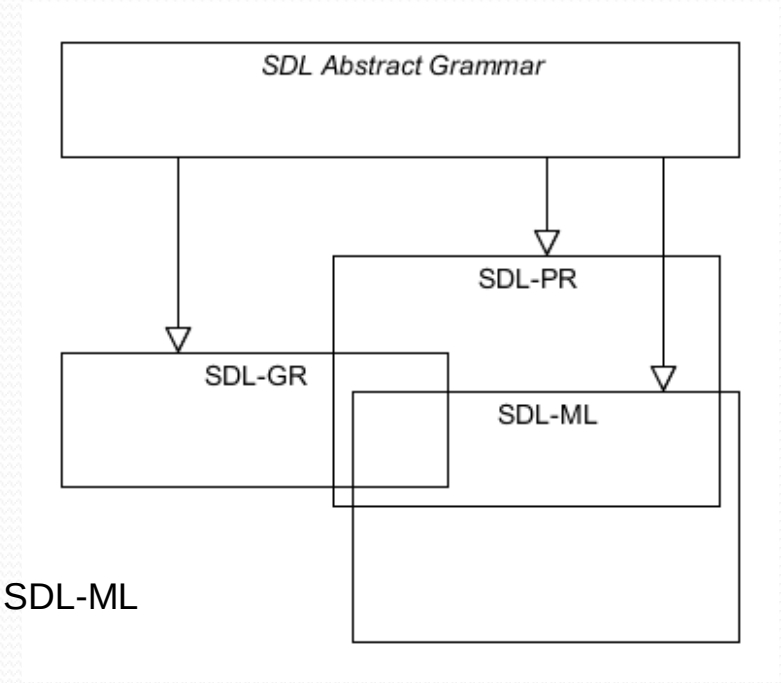


Figure 1. SDL-PR, SDL-GR, SDL-ML

SDL-PR, SDL-GR, SDL-ML

- SDL-ML (vs. CIF):
  - independent of tools and platforms,
  - supported with XML technologies,
  - a data storage tools do not need a dedicated SDL-PR parser,
  - a new self contained language, easily extended,
  - also based on SDL-PR syntax
  - conforms to the abstract grammar of SDL,
  - no “end…” closing keywords, but has closing tags.

- SDL-PR, SDL-GR, SDL-ML are equivalent.

SDL to SDL-ML

- Process interaction diagrams can be included in **blocks.**
- The root block is called **system.**

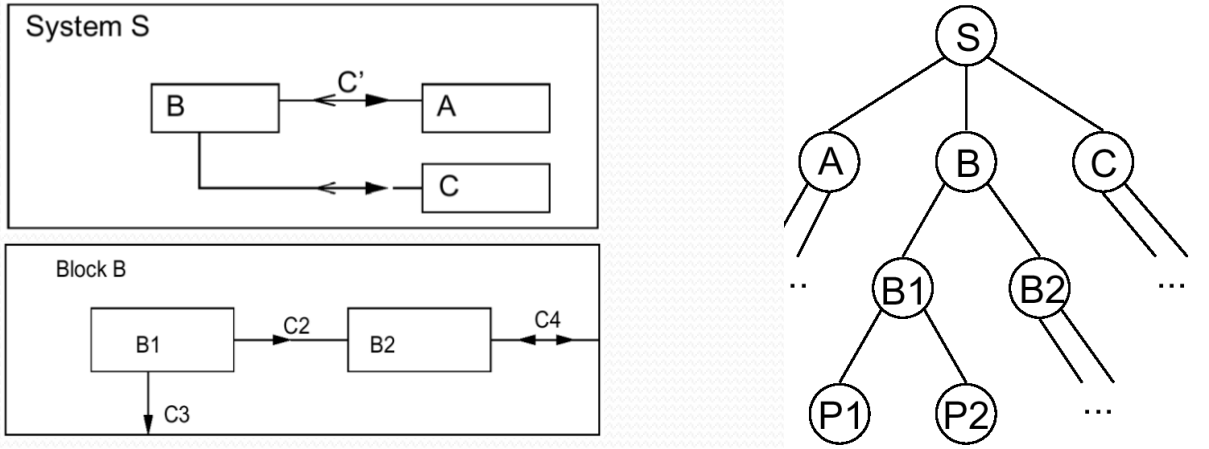


Figure 2. SDL and XML Hierarchy

SDL-ML Grammar and Lexic

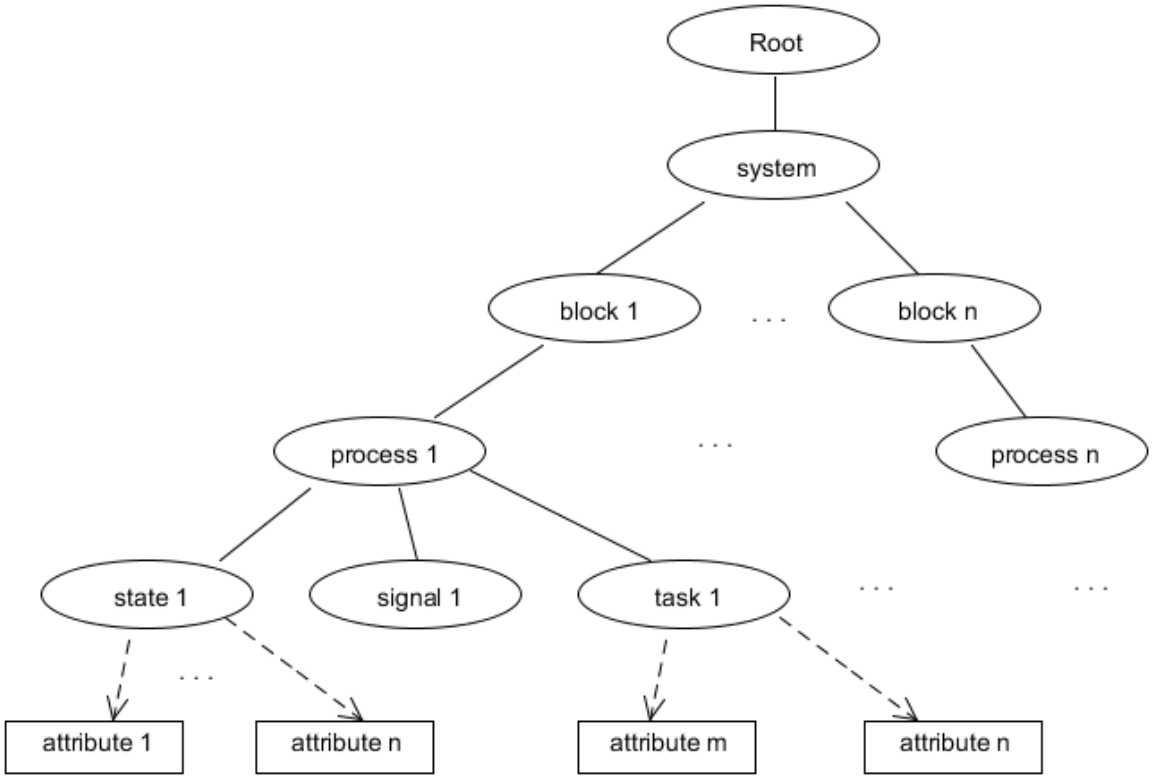


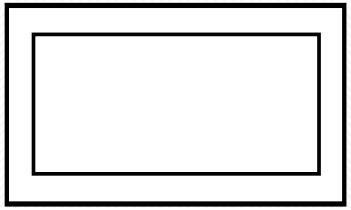
Figure 3. SDL-ML Tree

- SDL covers the basic aspects of OO:
  - **identity**
  - **classification**
  - **polymorphism**
  - **inheritance**
- SDL allows for the specification of:
  - system/ block/ process type
  - service/ procedure type
  - signal /data type

System (type)

```
<sdl:system>
   <sdl:use>
      <sdl:package>
         <sdl:name>packetABC</sdl:name>
      </sdl:package>
   </sdl:use>
   <sdl:name>SystemABC</sdl:name>

   <sdl:system_type>
      <sdl:name>typeABC</sdl:name>
      <sdl:fpar>
         <sdl:parameter>
            <sdl:name>par1</sdl:name>
         </sdl:parameter>
      </sdl:fpar>
      <sdl:inherits>  .... </sdl:inherits>

      <sdl:process_type> ... </sdl:process_type>
      <sdl:service_type> ... </sdl:service_type>
      <sdl:procedure>    ... </sdl:procedure>

   </sdl:system_type>

</sdl:system>
```

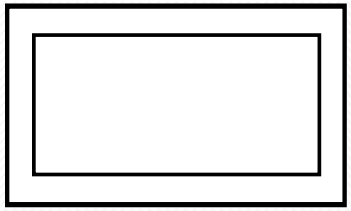
- Contains:
  1. system declarations part
  2. type declarations part
  3. block interaction part
- System type can be parameterised or a specialisation of another system type.
- Block specifications or instantiations of block types



Block (type)

```
<sdl:block>
   <sdl:name>blockABC</sdl:name>
   <sdl:no_of_instances>3</sdl:no_of_instances>
   <sdl:block_type>
      <sdl:name>typeABC</sdl:name>
      <sdl:fpar>
         <sdl:parameter>
            <sdl:name>par1</sdl:name>
         </sdl:parameter>
      </sdl:fpar>
      <sdl:inherits>  ... </sdl:inherits>

      <sdl:process_type> ... </sdl:process_type>
      <sdl:service_type> ... </sdl:service_type>
      <sdl:procedure>    ... </sdl:procedure>

   </sdl:block_type>
</sdl:block>

<sdl:gate>
   <sdl:name>gateC</sdl:gate>
   <sdl:out>
      <sdl:to>
         <sdl:with>sig</sdl:with>
      </sdl:to>
   </sdl:out>
   <sdl:in>
      <sdl:from>
         <sdl:with>
```

- Block type is a local specification within a system (or a system type) specification, or a remote specification within a package (a block type reference).
- Block type can be parameterised or a specialisation of another block type.



Process (type)

```
<sdl:process>
   <sdl:name>processABC</sdl:name>
   <sdl:no_of_instances>
      <sdl:init>3</sdl:init>
      <sdl:max>10</sdl:max>
   </sdl:no_of_instances>

   <sdl:process_type>

      <sdl:name>typeABC</sdl:name>
      <sdl:fpar>
         <sdl:parameter>
            <sdl:name>par1</sdl:name>
         </sdl:parameter>
      </sdl:fpar>
      <sdl:inherits>  .... </sdl:inherits>

      <sdl:service_type> ... </sdl:service_type>
      <sdl:procedure>    ... </sdl:procedure>
      .....

      <sdl:state> ... </sdl:state>

   </sdl:process_type>

</sdl:process>
```

- A process type reference may appear in a package, a system specification, a system type specification, a block specification, or a block type specification.
- Can be parameterised or a specialisation of another process type.

```
<sdl:gate>
   <sdl:name>gateB</sdl:name>
   <sdl:out>
      <sdl:to>
        <sdl:with>sig</sdl:with>
      </sdl:to>
   </sdl:out>
   <sdl:in>
      <sdl:from>
         <sdl:with>sig<sdl:with>
       </sdl:from>
   </sdl:in>
</sdl:gate>
```

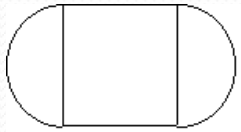
Service (type)

```
<sdl:service>

   <sdl:name>serviceABC</sdl:name>

   <sdl:service_type>
      <sdl:name>servicetypeABC</sdl:name>
      <sdl:fpar>
         <sdl:parameter>
            <sdl:name>par1</sdl:name>
         </sdl:parameter>
      </sdl:fpar>
      <sdl:inherits>  .... </sdl:inherits>

      ...
      <sdl:procedure>    ... </sdl:procedure>
      <sdl:state> ... </sdl:state>

   </sdl:service_type>

</sdl:service>
```

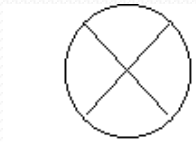
- A service type reference may appear in the specifications of a package, system, system type, block, block type, process and process type.
- A service type specification may be instantiated in a process specification or a process type specification.

Procedure

```
<sdl:procedure>

   <sdl:name>serviceABC</sdl:name>
   <sdl:fpar>
      <sdl:parameter>
         <sdl:name>par1</sdl:name>
         <sdl:sort>in</sdl:sort>
      </sdl:parameter>
   </sdl:fpar>
   <sdl:dcl>
      <sdl:variable>
         <sdl:name>var1</sdl:name>
         <sdl:sort>Integer</sdl:sort>
      </sdl:variable>
   </sdl:dcl>

   <sdl:inherits>
      <sdl:procedure>
         <sdl:name>proc</sdl:name>
      </sdl:procedure>
      <sdl:adding> ...
      </sdl:adding>
   </sdl:inherits>
      ...
   <sdl:state> ... </sdl:state>

</sdl:procedure>
```



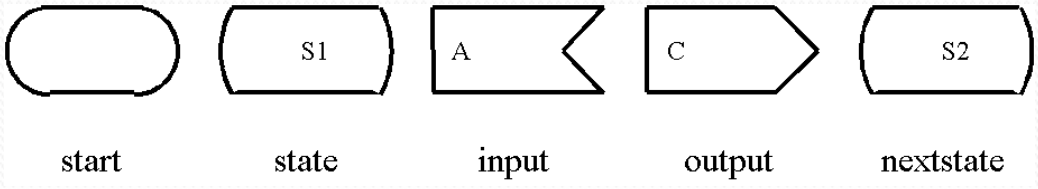
- Procedure is a specification of a type by itself.
- Procedure is instantiated when a procedure call in a process, service or procedure is interpreted



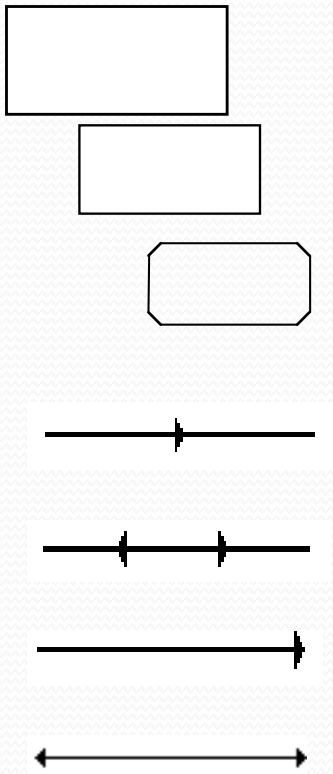
Signal (type)

```
<sdl:signal>

   <sdl:name>signalA</sdl:name>
   <sdl:sort>Integer</sdl:sort>

   <sdl:refinement>
      <sdl:subsignal>subA</sdl:subsignal>
   </sdl:refinement>

   <sdl:inherits>
      <sdl:signal>
         <sdl:name>signalBasic</sdl:name>
      </sdl:signal>
      <sdl:adding>
         <sdl:parameter>
            <sdl:name>newPar</sdl:name>
            <sdl:sort>newSort</sdl:sort>
         </sdl:parameter>
      </sdl:adding>
   </sdl:inherits>

</sdl:signal>
```

- Signal type is a specification of a type by itself.
- Signal type is instantiated when an output statement in a process, service or procedure is interpreted.
- Signal type can be parameterized, or be a specialisation.

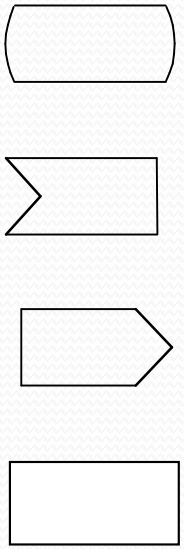


SDL-ML examples



```
<sdl:system id="31">
   <sdl:name>HelloWorld</sdl:name>

   <sdl:block id="51">
      <sdl:name>HelloWorld</sdl:name>

      <sdl:process id="63">
         <sdl:name>HelloWorld</sdl:name>

    <sdl:channel id="6">
       <sdl:name>Ch_Game_Env</sdl:name>
       <sdl:type>delaying</sdl:type>
       <sdl:direction>one</sdl:direction>
       <sdl:from_to>
          <sdl:from>env</sdl:from>
          <sdl:to>Gate_b</sdl:to>
          <sdl:with>
             <sdl:signal>
                <sdl:name>Yes</sdl:name>
             </sdl:signal>
          </sdl:with>
       </sdl:from_to>
    </sdl:channel>
```

Extended FSM



```
<sdl:state id="42">
   <sdl:name>Winning</sdl:name>
   <sdl:input>
      <sdl:name>Probe</sdl:name>
      <sdl:parameter>Val</sdl:parameter>
   </sdl:input>
   <sdl:output>
      <sdl:name>Result</sdl:name>
   </sdl:output>
   <sdl:nextstate>idle </sdl:nextstate>
</sdl:state>

<sdl:start id="1">
   <sdl:task id="23">
      <sdl:statement>Val:=Val+1;</sdl:statement>
   <sdl:join>labelA</sdl:join>
</sdl:task>
```

From DTD sdl.dtd

```
<!ELEMENT system (name, use*, newtype*, syntype*, signal+, system_type*, channel+, block+)>
<!ELEMENT block (name, no_of_instances*, newtype*, block_type*, route+, process*)>
<!ELEMENT process (name, gate+, dcl*, synonym*, timer*, signal*, process_type*, start?, state*)>
<!ELEMENT system_type (name, fpar*, inherits?, process_type*, service_type*, procedure*)>
<!ELEMENT block_type (name, fpar*, inherits?, process_type*, service_type*, procedure*)>
<!ELEMENT process_type (name, fpar*, inherits?, service_type*, procedure*, state*)>

<!ELEMENT channel (name, type?, direction?, from_to)>
<!ELEMENT from_to (from, to, with)>
<!ELEMENT with (signal+)>
<!ELEMENT signal (name, sort?)>

<!ELEMENT state (name, input+, nextstate*)>
<!ELEMENT start (task*, set*, nextstate*, join*)>
<!ELEMENT input (name, parameter*, join? | decision? | task* | nextstate?)>
<!ELEMENT decision (question, answer+)>
<!ELEMENT answer (output+)>
<!ELEMENT output (name, parameter*, set* | nextstate? | join?)>
<!ELEMENT question (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sort (#PCDATA)>
```

The Game Automata Example

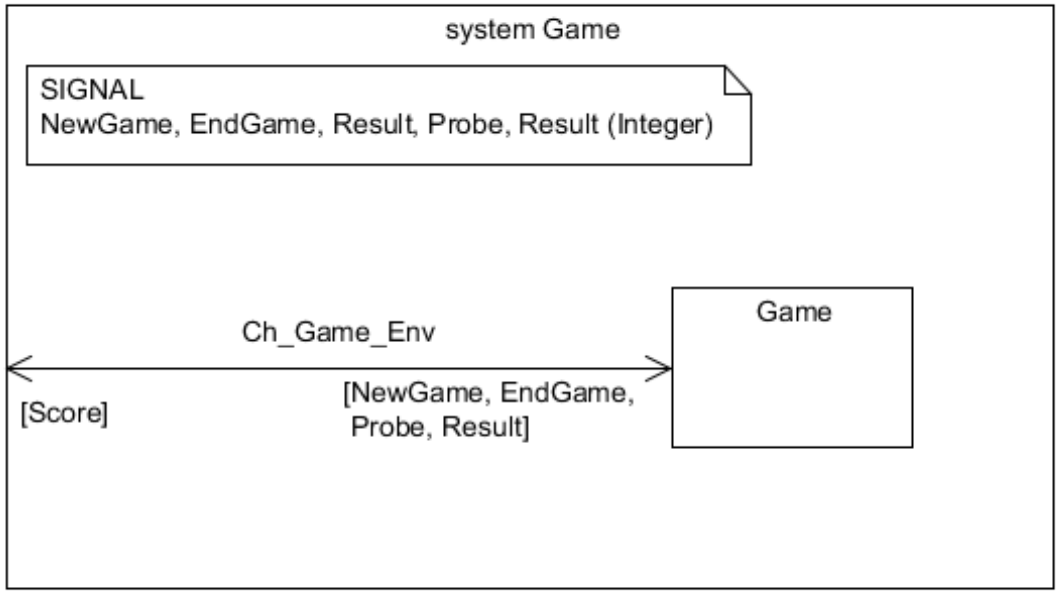


Figure 4. Game Automata System

Block Game
= contains the game
Channel Ch_Game_Env
= communication between the player and the system
Signals
= NewGame, EndGame to start and finish the game
= Probe to play the game
= Result to obtain the result

Block Game

Process Control
= an interface between the user and the game

Process Game
= instantiated by process Control when playing game

Routes
= Rt_Cont_Ch,
Rt_Game_Ch,
Rt_Cont_Game

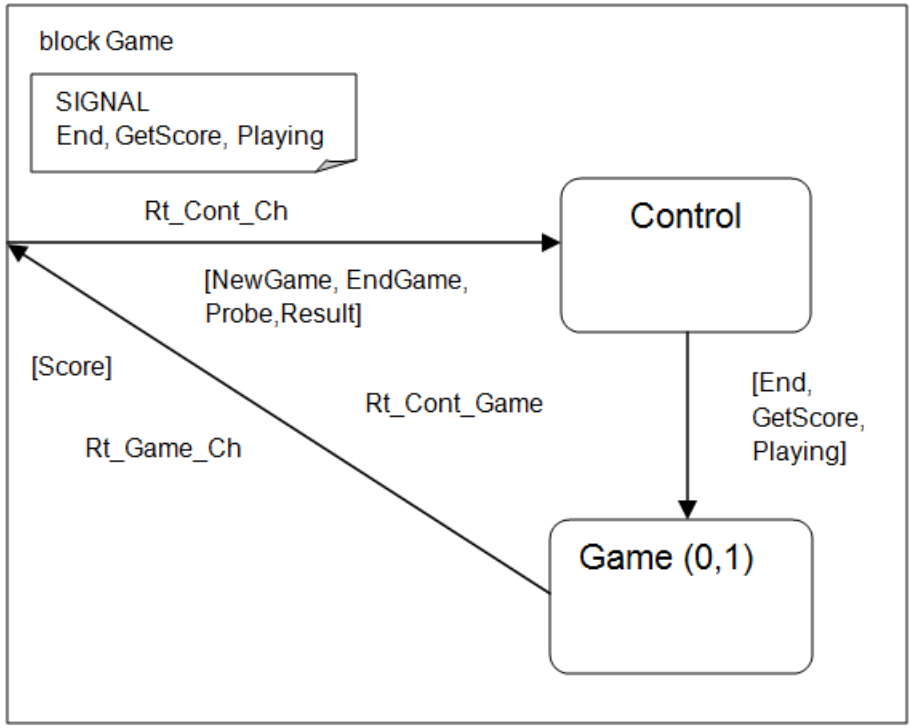
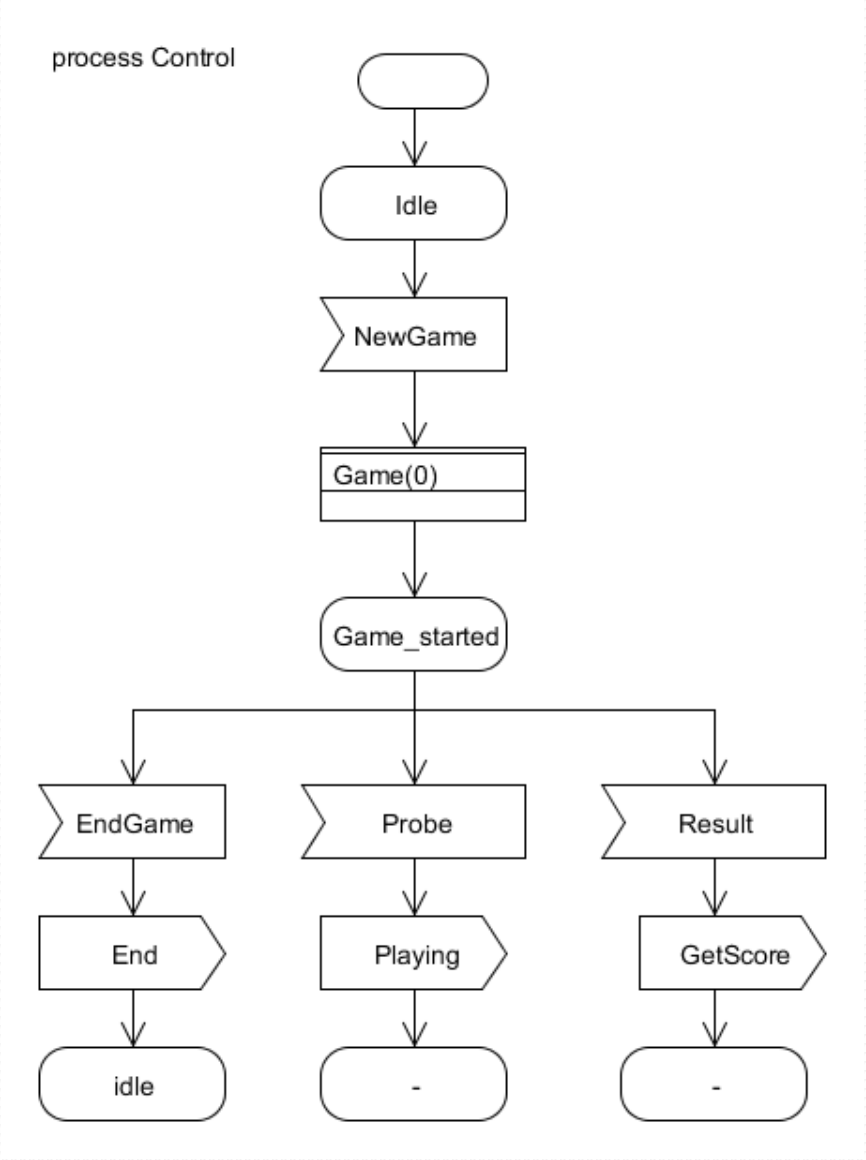


Figure 5. Game Automata Block

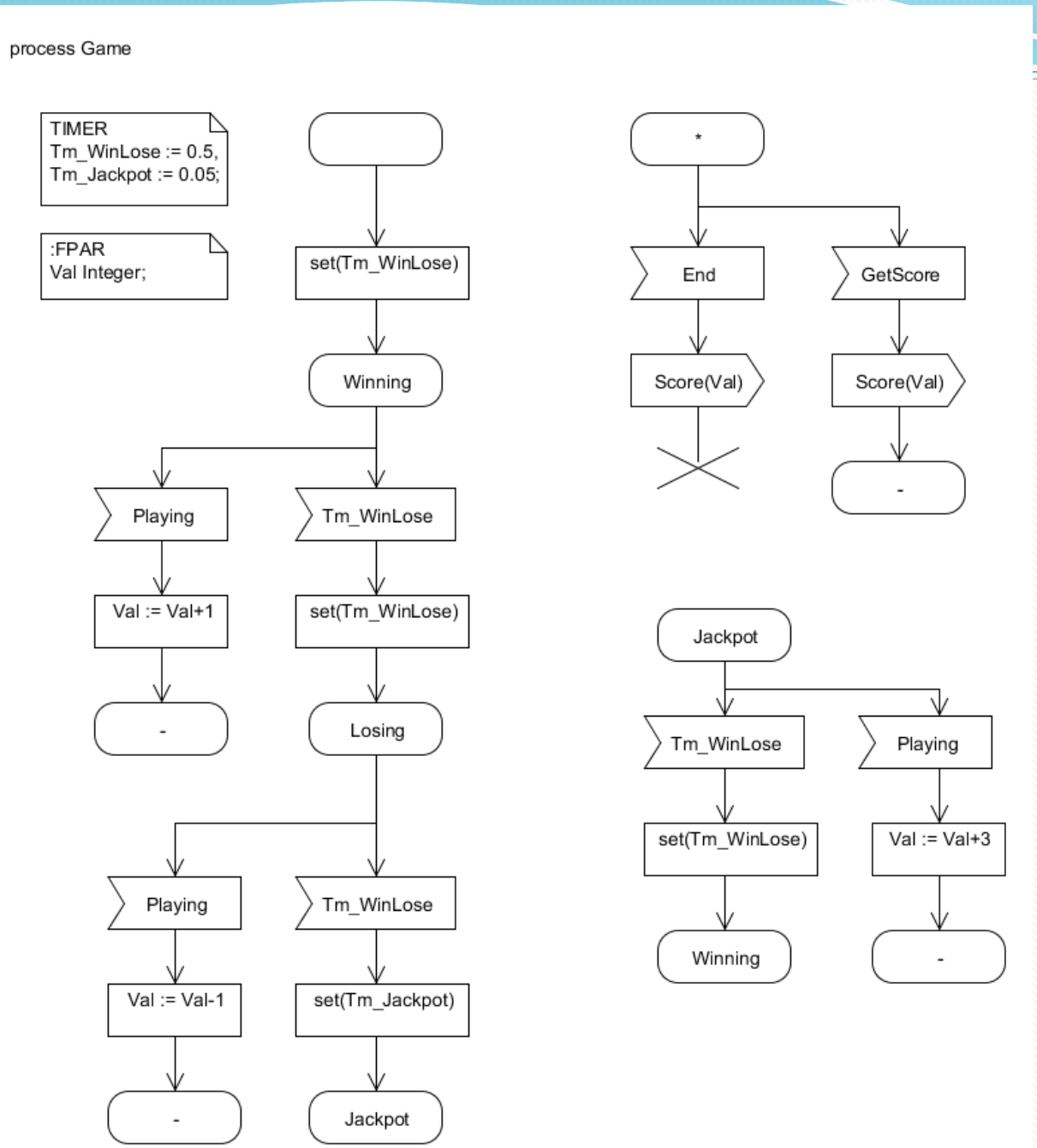


Process Control

Process Control
= creates a new Game instance
= tranfers the signals to the process Game

Process Game
= starts with Val=o;

Figure 6. Process Control



Process Game

Process Game

= simulates the game

= uses two timers

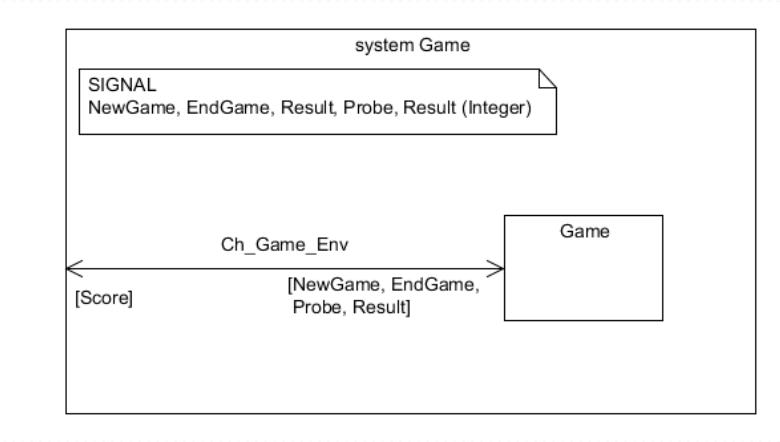
= returns the result with Val value.

= exits on the receival of End signal

Figure 7. Process Game

A Piece of SDL-ML Code

```
<sdl:system id="1">
   <sdl:channel id="6">
      <sdl:name>Ch_Game_Env</sdl:name>
      <sdl:direction>bi</sdl:direction>
      <sdl:type>nondelaying</sdl:type>
      <sdl:from_to>
         <sdl:from>Game</sdl:from>
         <sdl:to>env</sdl:to>
         <sdl:with>
            <sdl:signal>
               <sdl:name>Score</sdl:name>
               <sdl:sort>Integer</sdl:sort>
            </sdl:signal>
         </sdl:with>
      </sdl:from_to>
      <sdl:from_to>
         <sdl:from>env</sdl:from>
         <sdl:to>Game</sdl:to>
         <sdl:with>
               ...
               <sdl:signal>
                  <sdl:name>Result</sdl:name>
                  <sdl:sort>Integer</sdl:sort>
               </sdl:signal>
         </sdl:with>
      </sdl:from_to>
   </sdl:channel>
.....
```



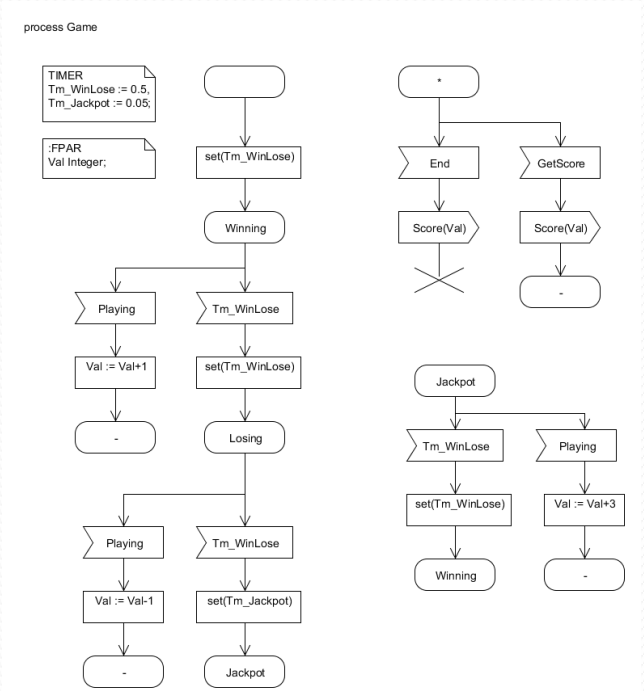
A Piece of SDL-ML Code for Game Process Specification

A Piece of SDL-ML Code

```
<sdl:system id="1">
   <sdl:name>Game</sdl:name>
   <sdl:block id="2">
      <sdl:name>Game</sdl:name>
      <sdl:process id="3">
         <sdl:name>Game</sdl:name>
         <sdl:state id="4">
             <sdl:name>Winning</sdl:name>
             <sdl:input>
                <sdl:name>Playing</sdl:name>
                <sdl:task>
                    <sdl:statement>Val:=Val+1;</sdl:statement>
                <sdl:nextstate>Winning</sdl:nextstate>
             </sdl:input>
             <sdl:input>
                <sdl:name>Tm_Win_lose</sdl:name>
                <sdl:set>
                   <sdl:timer>
                        <sdl:name>Tm_Win_Lose</sdl:name>
                   </sdl:time>
                </sdl:set>
                <sdl:nextstate>Losing</sdl:nextstate>
             </sdl:input>
         </sdl:state>
```

.....

A Piece of SDL-ML Code for Game Process Specification



Conclusion and Future work

- SDL-ML is
  - An XML-based format of SDL,
  - Not opposed to CIF (SDL-PR),
  - Implies XML tools support→ new graphical tools, model checking tools, data storage tools… all based on XML syntax (not SDL-PR).

- For future work
  - Determine the lexic units to include all of SDL elements, e.g. formal context parameters, data types, etc.,
  - Determine between the elements and the attributes,
  - Graphics to be developed.