Semantics of the DOL XML Schemata

Wolfgang Haid and Kai Huang
Revision 1069, July 23, 2008

1 Introduction

In this document, we describe how we can specify a system consisting of an application, an architecture and a mapping, using the XML schema definitions developed in the SHAPES project. In Section 2, the usage of the process network XML definition is described in detail. In Section 3, the usage of the architecture XML is described, and in Section 4 the usage of the mapping XML is described.

The specification (together with the application source code) allows to describe a multi-processor system and is the input for the generation of a functional simulation and hardware-dependent software (HdS) generation.

2 Process Network Specification

The process network XML schema definition describes an XML format which allows to describe applications at an abstract level. Using a process network, an application is described not only by source code but by both the process network and the associated source code.

The proposed process network definition can be used to describe applications as follows.

- A process network contains processes whose interfaces are described by ports. The behavior of single processes is defined in C/C++.
- A process network describes the interconnections between processes.
- Repetitive structures can be described by using so-called iterators. This opens the possibility of parameterizable process networks.

Fig. 1 shows an example of a simple process network and its basic components.

The process network description of an application for the DOL is contained in an XML file, which complies with the `processnetwork.xsd`

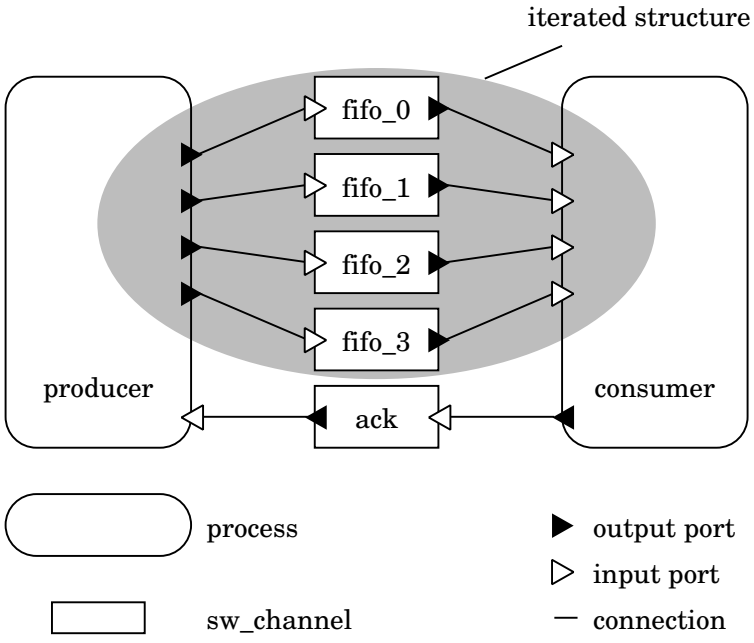

Figure 1: Example process network.

XML schema definition. In `processnetwork.xsd`, the syntax of the XML specification is described, whereas in this document, we focus more on the semantics of the XML specification document.

As a starting point, we consider the process network shown in listing 1. Please note, that unimportant parts of the definition are omitted. (The example does, therefore, not represent a valid XML document according to the schema definition.) From the schema definition we learn that the root element of the XML file containing an application description has to be <processnetwork>. This element has 1 mandatory attribute `name`. The root element can be followed by an arbitrary number of global elements <variable>, and <function>. These elements can be used to set variables and to define Java functions that can be referenced from within the XML description. We will look in more detail into <variable> in Section 2.1.5. The <function> element is described in Section 2.1.6.

After the global definitions, the actual elements of an application, i.e. <process>, <sw_channel>, and <connection> elements may occur in the document. To make process networks parameterizable to a certain degree,

these elements may occur inside <iterator> elements. The <iterator> element allows the description of repetitive structures.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <processnetwork name="example process network">
03   <variable value="4" name="N"/>
04   <function name="foo">
05     int foo(int input) {
06       return (int) Math.pow((double) input, 2.0);
07     }
08   </function>
09   <process name="generator">
10     ...
11   </process>
12   <sw_channel type="fifo" size="10" name="C2">
13     ...
14   </sw_channel>
15   <iterator variable="i"  range="N">
16     <process name="square">
17       <append function="i"/>
18       ...
19     </process>
20     <sw_channel type="fifo" size="10" name="C2">
21       <append function="i"/>
22       ...
23     </sw_channel>
24   </iterator>
25   <process name="consumer">
26     ...
27   </process>
28   ...
29 </processnetwork>
```

Listing 1: Example process network.

2.1 Elements for Application Specification XML

In this section, we describe the usage of the allowed elements in the process network specification. First, the elements <process>, <sw_channel>, <port>, and <connection> are described. With these elements, any application can be described. For large applications however, describing each process with a dedicated <process> element, for instance, becomes tedious and error-prone. Therefore, the <iterator> element has been introduced to describe repetitive application structures. The <iterator> element, which is closely linked with the <variable>, <function>, and <append> element is described in the second part of this section.
In simulation generation, the iterated elements need to be instantiated, a step we refer to as "flattening". In particular, we refer to flattening as the transformation of a process network with iterated elements to a process network containing only <process>, <sw_channel>, <port>, and

<connection> elements.
The <process>, <sw_channel>, and <connection> elements have an attribute `name` for identification of these elements in later design stages. Therefore, each name must only appear once in a process network. Also, the <port> element has a `name` attribute. The names of ports must be unique within the enclosing element, that is, <process> or <sw_channel>.

2.1.1 <process>

The element <process> is used to describe a process in the process network. It has 1 attribute `name`, and may contain four different child elements: <append> (described in Section 2.1.8), <iterator> (described in Section 2.1.7), <port> (described in Section 2.1.3), and <source> (described in this section). The sequence of the four child types is mandatory.

The <source> element has 2 attributes: `type` and `location`. The `type` attribute can have either the value `c`, or `c++` and denotes the type of the source code which describes the functionality of the process. The `location` attribute holds the path and name of the file containing the source code. This attribute is used by the simulation generator.

The <port> elements describe all ports of the process, being input or output ports. These ports can be connected through <connection> elements with <sw_channel> elements. Each <process> element must have at least one port, either as a child element, or as a child element of an enclosing <iterator> element.

```
01 <process name="square">
02   <append function="i" />
03   <port type="input" name="0" />
04   <port type="output" name="1" />
05   <source type="c" location="square.c" />
06 </process>
```

2.1.2 <sw_channel>

The <sw_channel> element is used to describe FIFO channels in the process network. It has 3 attributes `type`, `size`, and `name`. `type` describes the type of the channel. Currently, the type of a <sw_channel> must be `fifo`. No other values are allowed. The attribute `size` specifies the number of bytes a FIFO must be able to hold at least.
Consider the simple process network in Fig. 2 where two processes (processA and processB) are connected by two FIFOs (fifoA and fifoB). Assume that

processB waits (blocking read) until processA sends some data on fifoA and that processA writes data on fifoB before sending data on fifoA.

Then, fifoB should be able to hold all the data written to fifoB while processB is blocking on fifoA. Otherwise, a deadlock will occur.

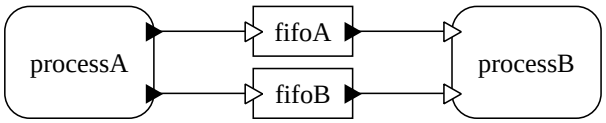

Figure 2: To the definition of the `size` attribute

`name` is the name of the <sw_channel>. The <sw_channel> element may have two type of child elements: <append> (described in Section 2.1.8) and <port> (described in Section 2.1.3). Each <sw_channel> element must have exactly two <port> elements.

```
01 <sw_channel type="fifo" size="10" name="C2">
02   <append function="i"/>
03   <port type="input" name="0"/>
04   <port type="output" name="1"/>
05 </sw_channel>
```

2.1.3 <port>

The <port> element may occur as child element of the following elements: <process>, <origin>, <target>, and <sw_channel>. The element may be iterated with an <iterator> element inside a <process> element, and may have <append> elements as children elements. The `name` attribute is used to assign a name to a port. The port name must be unique within the enclosing element. If the <port> element occurs within a <process> or <sw_channel> element (or an <iterator> element within a <process> element), it has an additional attribute `type`, which can have two values: `input` or `output`. Each port can only be connected once using the <connection> element.

```
01 <port name="0" type="input">
02   <append function="1"/>
03 </port>
```

Remark: HdS generation requires that port names are *integers*, not strings. The functional simulation will work for integer and string port names.

2.1.4 <connection>

The <connection> element is used to establish connections between processes and the FIFO channels. More precisely, a connection is established between two ports of the corresponding elements. Connections may only connect:

- a single output port of a <process> with a single input port of a <sw_channel>, or
- a single output port of a <sw_channel> with a single input port of a <process>.

All <connection> elements have 1 attribute: `name`. Further, the elements may contain three types of child elements: <append> (optional, described in Section 2.1.8), <origin> (mandatory), and <target> (mandatory).

Both, the <origin> element and the <target> element are used to denote the two elements of a process network that should be connected, i.e. one port of a <process> and one port of a <sw_channel> element. <origin> and <target> do not impose a direction of the dataflow over a connection. Rather, the direction of the connection is determined by the type of ports which are linked by <connection>.

The attributes and contents of <origin> and <target> are identical, so we just describe the <origin> element. It has 1 attribute `name` and may contain an <append> element to modify the `name` attribute. This attribute is used to denote the element to which a connection should be established. It has to be the name of an existing <process> or <sw_channel> element.

The <origin> and <target> element must contain a <port> element (described in Section 2.1.3) which defines the connection port.

```
01 <connection name="to_square">
02   <append function="i"/>
03   <origin name="C2">
04     <append function="i - 1"/>
05     <port name="1"/>
06   </origin>
07   <target name="square">
08     <append function="i"/>
09     <port name="0"/>
10   </target>
11 </connection>
```

2.1.5 <variable>

<variable> elements must occur as the first children of the root element. The schema definition does not allow any other location. The <variable>

element has 2 attributes: `name` and `value`. The `name` attribute contains a string that can be referenced from within the <append> element or the <iterator> element. During the flattening process the processor replaces the name placeholder with the corresponding value, an integer value stored in the attribute `value`.

```
01 <variable name="N" value="4"/>
```

While flattening, all occurrences of `N` in <iterator> or <append> elements will be replaced by the integer 4.

2.1.6 <function>

This element must occur right after the <variable> elements as a child of the root element. It cannot occur somewhere else in the XML file. The <function> element has 1 attribute `name` which contains a string denoting the name of the function. If present, the element must contain a complete Java function as content. See below for an example of a function element:

```
01 <function name="myFunction">
02   int myFunction(int input1, int input2) {
03     return (int) Math.pow((double) input1, (double) input2);
04   }
05 </function>
```

The function must return an `int` and may accept an arbitrary number of `int` parameters. The functions can be used to calculate more complicated `function` attributes for the <append> elements:

```
01 <append function="myFunction(in1, in2)"/>
```

2.1.7 <iterator>

Remark: Iterators are currently not supported by HdS generation. Therefore, iterators cannot be used in specifications that should run on the real or virtual SHAPES platform.

The <iterator> element can be used to replicate <process>, <sw_channel>, <port>, or <connection> elements several times. To do so, the `variable` attribute has to be set and in the `range` attribute the upper limit for the iterator has to be set. If the range is `N`, for instance, the nested elements inside the iterator will be replicated `N` times. To actually create different instance of the iterated elements, the <append> element is used (described in Section 2.1.8). When an iterator is flattened, the variable given in the `variable` attribute iteratively takes values from 0 to $\mathtt{N}-1$.

Below is an example that shows the usage of an iterator:

```
01 <iterator variable="i" range="N">
02   <process name="square">
03     <append function="i"/>
04     ...
05   </process>
06
07   <sw_channel type="fifo" size="10" name="C2">
08     <append function="i"/>
09     ...
10   </sw_channel>
11 </iterator>
```

Iterators may be nested to generate multi-dimensional arrays of iterated elements.

2.1.8 <append>

This element may be used within several elements to modify the element's name. The <append> element has 1 attribute `function` which is a string. The string must have one out of four formats and evaluate to an integer value:

- iterator variables (as specified in Section 2.1.7), e.g., `i`, `j`, `k`
- pure integers, e.g., `1`, `2`, `3`
- variables (as specified in Section 2.1.5), e.g., `N`
- functions (as specified in Section 2.1.6), e.g., `myFunction(2, 4)`

The `function` attribute may contain also simple arithmetical functions, such as `function="2 * i"` based on the format specified above.

When an iterator is used to create an instance of <process>, <sw_channel> or <port>, only the first form is permitted in an <append> element within this iterator, as shown in the example below. All four forms are allowed when the <append> element appears as a child in a <connection> element, or when it appears as a child of <process>, <sw_channel>, or <port> within a <connection> element.

The names of <process>, <channel>, <port>, and <connection> elements can be modified with more than one <append> element. The name of the iterated elements will be built according to the sequence of the <append> elements:

```
01 <iterator variable="i" range="2">
02   <interator variable="j" range="3">
03     <process name="square">
04       <append function="i"/>
05       <append function="j"/>
06     </process>
07   </iterator>
08 </iterator>
```

will be expanded during the flattening process to:

```
square_0_0
square_0_1
square_0_2
square_1_0
square_1_1
square_1_2
```

2.2 Data Input and Output using Device Drivers

This subsection deals with the question how data input and output via device drivers is considered in the DOL specification. The basic approach is to "tag" nodes in the process network such that the subsequent code generators can produce the according code. The concrete implementation is described in the following.

Fig. 3 shows a simple example for illustration. The top part of the figures shows an application that reads samples from an input interface and writes them to some output interface. Data input and output is handled by *driver processes* which are handled according to the kind of execution, that is, functional simulation or execution on the real or virtual platform.

From the application specification point of view, the implementation looks as follows:

- The process network is implemented as usual, but the functionality that is later implemented by the device driver is encapsulated in a separate *driver process*. This process is "tagged" as a driver in the process network XML specification by using a configuration element:
  `<configuration name="isDriver" value="true"/>`
  Otherwise, there is no difference between a driver process and an ordinary one, that is, the ports and the source code of the driver process need to be specified like for any other process.

- In functional simulation, a driver process is handled like any other process. Depending on the application, an input driver process could read data

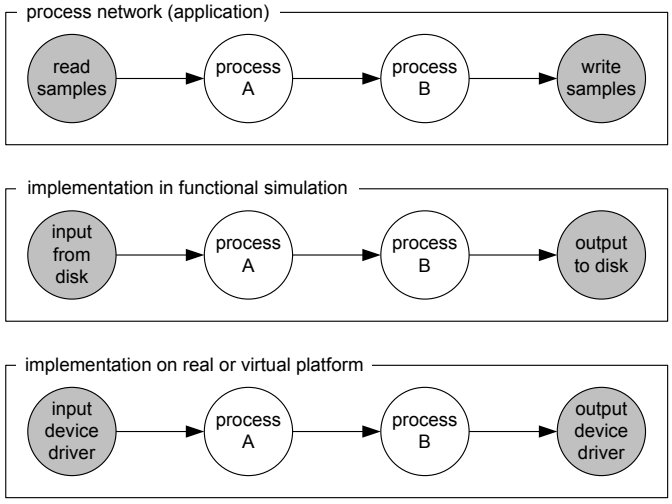

Figure 3: Handling device drivers in DOL using *driver processes* (shaded).

from a disk, generate random data, etc. An output driver process could write data to a disk, print the result to the console, etc. The code for doing so needs to be provided by the application programmer.

- For the implementation on the real or virtual platform, device drivers need to be provided that implement the required behavior. Input device drivers might read data from a digital interface and put it to some memory location, for instance. Output device drivers might read data from some memory location and forward them to an digital-to-analog converter, for instance. The device drivers are implemented in the hardware-dependent software layer.

From the mapping specification point of view (refer to Section 4 for details about the mapping specification), the implementation looks as follows:

- Driver processes are bound to the device that actually implements the behavior of that process (using a <`binding`> element, see Listing 2). Driver processes do not need to be scheduled (within a <`schedule`> element), however.

- The write and read paths for the channel of a driver process are the paths along which the data actually travel.

As an example, consider a synchronous serial channel (SSC) device in an architecture: <processor name="SSC" type="POT">. Furthermore, there exist read and write paths between the SSC registers and other memories (dxmsscreg and sscregrdm, for instance). For an input device, the driver process would be bound to SSC, and the channel to a write path emerging from SSC. The read path would end at a memory associated with a RISC/DSP, as shown in Listing 2.

```
01 <binding name="binding_inputdriver" xsi:type="computation">
02   <process name="inputdriver"/>
03   <processor name="SSC"/>
04 </binding>
05
06 <binding name="binding_inputchannel" xsi:type="communication">
07   <sw_channel name="inputchannel"/>
08   <writepath name="scregdxm"/>
09   <readpath name="dxmrdm"/>
10 </binding>
```

Listing 2: Example for device driver mapping.

3 Architecture Specification

The architecture schema definition describes an XML format which allows to describe a hardware architecture at an abstract level, which is sometimes referred to as virtual architecture level. In the XML, the structure of an architecture is described using computing and communication resources:

- Computation resources: processors, memories
- Communication resources: hardware channels

To describe these resources and their parameters, the architecture schema specifies the elements <processor>, <memory> and <hw_channel>.
The communication links between the memories of processors are described by communication paths, that is, end-to-end connections between those memories, instead of just local links. A communication path, see Fig. 4, consists of a write path and a read path which share the location of a common buffer, referred to as channel buffer. The write path is used to transfer data from the transmit buffer located in the transmitting processor's memory to the channel buffer. The read path is used to transfer data from the channel buffer to the receive buffer located in the receiving processor's memory. Write and read paths are described using the elements <writepath> and <readpath>.

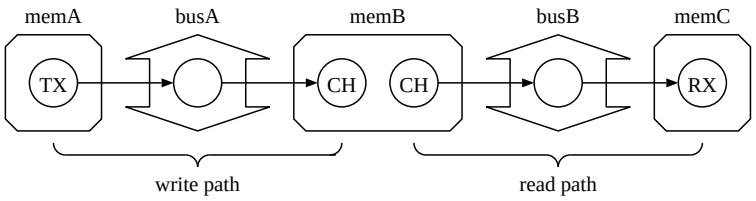

Figure 4: Communication path between two memories.

A communication path is any combination of a write path and a read path whose channel buffers are located on the same memory. According to this, Fig. 5 shows an example architecture with the five communication paths listed in table 1.

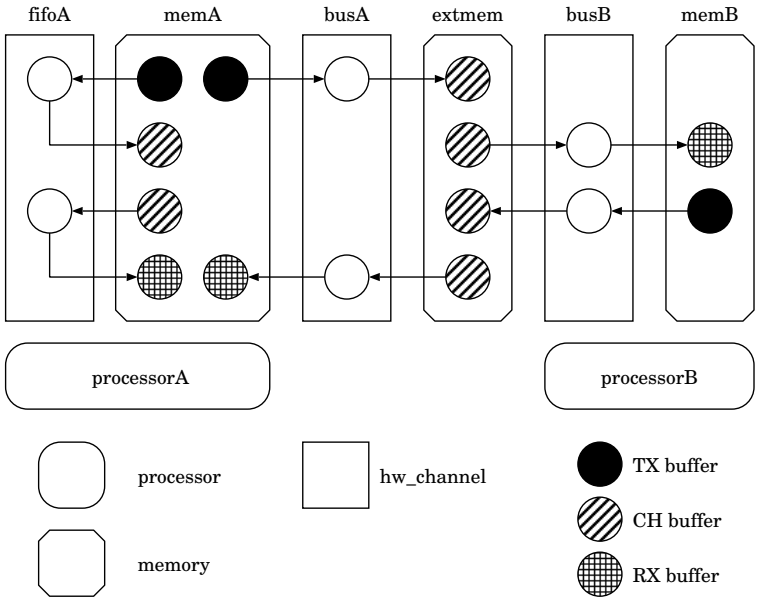

Figure 5: Example architecture with 5 communication paths.

TX buf	→ hw_channel	→ CH buf	→ hw_channel	→ RX buf
memA	fifoA	memA	fifoA	memA
memA	busA	extmem	busA	memA
memB	busB	extmem	busB	memB
memA	busA	extmem	busB	memB
memB	busB	extmem	busA	memA

Table 1: Communication paths of the example architecture shown in Fig. 5.

3.1 Elements for Architecture Specification XML

3.1.1 <processor>, <memory>, <hw_channel>

The <processor>, <memory>, and <hw_channel> elements are used to describe processing, storage, and communication resources, respectively. In particular,

- <processor> describes a processor or another computing resource like an FPGA, a special peripheral, etc.,
- <memory> describes a memory or another storage device,

- <hw_channel> describes a communication resource, such as a unidirectional link, a bus, or even an entire network on chip.

For each of these elements, the mandatory attribute `type` further specifies the type of the resource. The different types of resources are as follows:

- <processor>: `RISC`, `DSP`, `POT`
- <memory>: `ROM`, `RAM`, `REG`, `DXM`
- <hw_channel>: `FIFO`, `BUS`, `DMA`, `SPI`, `BRIDGE`

For each resource, a unique name must be specified in the `name` attribute. A resource can contain an optional <configuration> element to annotate parameters to the resource. The <configuration> element has two required attributes: `name` specifies the parameter name, and `value` specifies the parameter value. There are no restrictions on `name` and `value` which gives the possibility to specify an arbitrary set of valid parameters without changing the schema.
The listing below shows the XML of three resources of the architecture shown in Fig. 5.

```
01  <processor name="processorA" type="DSP">
02    <configuration name="clock" value="100"/>
03  </processor>
04
05  <memory name="memA" type="RAM">
06    <configuration name="size" value="48000"/>
07  </memory>
08
09  <hw_channel name="busA" type="BUS">
10    <configuration name="width" value="32"/>
11  </hw_channel>
```

3.1.2 <writepath>, <readpath>

The <writepath> and <readpath> elements are used to describe write paths and read paths. Their semantics is similar, so just the <writepath> element is described in detail here.
<writepath> has got one attribute `name` to specify an identifier for the path. The first element within <writepath> must be an element <processor> with a mandatory attribute `name`, specifying on which processor a process must be executing such that it can use this write path. The <processor> element is followed by an element <txbuffer> with a mandatory attribute `name`, specifying on which memory the transmit buffer of this communication path is located. After that one or more <hw_channel> elements with a mandatory

attribute `name` need to be listed, specifying the communication links used by that path. Finally, an element <chbuf> with a mandatory attribute `name` is used to specify the channel buffer for the write path. Like <processor>, <memory>, and <hw_channel> elements, a <writepath> element can be annotated with <configuration> elements.
The <readpath> element has the same semantics as the <writepath> element, except that instead of the <txbuffer> a <chbuffer> needs to be specified and that instead of the <chbuffer> a <rxbuffer> needs to be specified. The listing below shows the XML of one write path and one read path from the example architecture shown in Fig. 5.

```
01 <writepath name="memAextmem">
02   <processor="processorA" />
03   <txbuffer="memA" />
04   <hw_channel="busA" />
05   <chbuffer="extmem" />
06   <configuration name="latency" value="5" />
07   <configuration name="burst" value="1" />
08 </writepath>
```

```
01 <readpath name="extmemmemB">
02   <processor="processorB" />
03   <chbuffer="extmem" />
04   <hw_channel="busB" />
05   <rxbuffer="memB" />
06   <configuration name="latency" value="8" />
07   <configuration name="burst" value="2" />
08 </readpath>
```

3.1.3 <variable>, <function>, <iterator>, and <append>

The semantics of the <variable>, <function>, <iterator>, and <append> elements are the same as in the process network specification (see Sections 2.1.5 to 2.1.8). The difference concerning <iterator> is in the elements which can be iterated which are <processor>, <memory>, <hw_channel>, <writepath>, and <readpath>. Likewise, <append> can only appear within these elements.

4 Mapping Specification

The mapping defines where and how the components of an application are executed on a distributed hardware platform. Mapping needs to be done in the spatial domain, which is referred to as binding, and in the temporal domain, which is referred to as scheduling. In the DOL, the binding specification defines a mapping of processes to processors and software channels to communication paths. The scheduling specification defines the scheduling policy on each resource and the according parameters, e.g. time division multiple access scheme and the associated slot length, fixed priority scheduling and the associated priorities, or static scheduling and the associated ordering.

4.1 Elements for Mapping Specification XML

4.1.1 <binding>

The <binding> element serves for the definition of bindings. The <binding> element has got two attributes, `name` and `xsi:type`. The `name` attribute is used to specify an identifier for the binding. The `xsi:type` attribute can be either `computation` to define a binding of a process to a processor or `communication` to define the binding of a software channel to a communication path. Depending on the type of the <binding>, the child elements are different.
When the <binding> is of type `computation`, then it has two child elements <process> and <processor>, specifying which <process> (as defined in the process network) is bound to which <processor> (as defined in the architecture specification).
When the <binding> is of type `communication`, then it has three child elements <sw_channel>, <writepath>, and <readpath>, specifying which <sw_channel> (as defined in the process network) is bound to which communication path consisting of one <writepath> and one <readpath> (as defined in the architecture specification).
The <binding> element can occur inside an <iterator> element, see Section 4.1.4.
The listing below shows a <binding> of a <process> and a <sw_channel> to illustrate the usage of these elements.

```
01 <binding name="process_binding" xsi:type="computation">
02   <process name="processA"/>
03   <processor name="processorA"/>
04 </binding/>
```

```
01 <binding name="channel_binding" xsi:type="communication">
02   <sw_channel name="channelA"/>
03   <writepath name="memAextmem"/>
04   <readpath name="extmemmemB"/>
05 </binding>
```

4.1.2 <schedule>

The <schedule> element is used to define the schedule of processors and hardware channels. The <schedule> element has got two attributes, `name` and `type`. The `name` attribute is used to specify an identifier for the schedule. The `type` attribute which can be one of `fifo`, `fixedpriority`, `roundrobin`, `static`, or `tdma` defines the scheduling policy. The <schedule> element has a child element <resource> which is used to specify the resource for which resource the schedule is. This is done by specifying the name of the <processor> or <hw_channel> in the `name` attribute of the <resource> element.
The scheduling itself is defined by adding <origin> elements as children to the <schedule> element. In the context of the <schedule> element, the <origin> element has one attribute `name` to refer to a specific <process> or <sw_channel> element. Each <process> is bound to exactly one <processor>, thus each <process> of the application specification occurs once in exactly one <schedule> element. Each <sw_channel>, however, can be bound to several <hw_channel> elements, namely, when either the write path or the read path (or both) to which the <sw_channel> is bound contain several <hw_channel> elements. Thus each <sw_channel> can occur in different <schedule> elements. Also, a <sw_channel> element can occur more than once in a single <schedule> element when both the write path and the read path to which the <sw_channel> is bound contain the specific <hw_channel> element. In this case the first occurrence of that <sw_channel> in an <origin> element refers to the write path and the second occurrence of that <sw_channel> in an <origin> element refers to the read path. Each <origin> element can have <configuration> elements as child elements to annotate scheduling parameters, see Section 4.1.3.
The <schedule> element can occur inside an <iterator> element, see Section 4.1.4.
The listing below shows a <schedule> element to illustrate the usage of the elements.

```
01 <schedule name="bus_schedule" type="fixedpriority"/>
02   <resource name="bus"/>
03   <!-- the following origin element refers to the <writepath>
```

```
04          of channelA -->
05    <origin name="channelA">
06      <configuration name="priority" value="2"/>
07    </origin>
08    <!-- the following origin element refers to the <readpath>
09          of channelA" -->
10    <origin name="channelA">
11      <configuration name="priority" value="1"/>
12    </origin>
13  </schedule>
```

4.1.3 <configuration>

The <configuration> element has two required attributes: `name` specifies the parameter name, and `value` specifies the parameter value. The <configuration> element is used to annotate additional information to <origin>, and <schedule> elements.

4.1.4 <variable>, <function>, <iterator>, and <append>

The semantics of the <variable>, <function>, <iterator>, and <append> elements are the same as in the process network specification (see Sections 2.1.5 to 2.1.8). The difference concerning <iterator> is in the elements which can be iterated which are <binding> and <schedule> in the mapping specification.
When iterators are used, the <append> element is used as the first child element of the <binding>, or <schedule> element to define a unique name for the iterated element. The <append> element can also be used to adapt the `name` attribute of <process>, <processor>, <sw_channel>, <writepath>, <readpath>, <resource>, and <origin> elements within <binding>, <schedule> elements.