11.1 Port

(1) Ports are logical connection points between components that can be used for the transfer of control and data between threads or between a thread and a processor or device. Ports are directional, i.e., an output port is connected to an input port. Ports can pass data, events, or both. Data transferred through ports is typed..... Incoming events may trigger thread dispatch or mode transitions. Properties specify the input and output timing characteristics of ports. Actual event and data transfer may be initiated by the runtime system of the execution platform or by Send_Output runtime service calls in the application source text.

AADL distinguishes between three port categories: *data port*, *event port* and *event data port*.

From the perspective of the application source text, data ports are accessible in the source text as data variables, event ports represent event queues whose size is accessible, event data ports represent message queues whose content can be retrieved.

An example:

```
thread threadA
  features
    portA: in data port {Timing => immediate };
    portB: in event port;
    portC: out event data port dataA
           {Output_Time => (Completion, 0.0ns .. 0.0ns)};
end threadA;
```

11.1.1 Abstract syntax of *Port*

$$Port ::= Event\_port + Data\_port + Event\_data\_port \tag{7}$$

$$Basic\_port ::= portID \times Port\_direction \times \mathbf{opt}(\mathbf{list}(Port\_property)) \tag{8}$$

$$Event\_port ::= Basic\_port \tag{9}$$

$$Data\_OR\_Eventdata\_port ::= Port\_triggering \times \mathbf{opt}(Data\_reference) \times Basic\_port \tag{10}$$

$$Data\_port ::= Data\_OR\_Eventdata\_port([Port\_triggering = no]) \tag{11}$$

$$Event\_data\_port ::= Data\_OR\_Eventdata\_port([Port\_triggering = on]) \tag{12}$$

$$Port\_direction ::= \{in, out, \{in\ out\}\} \tag{13}$$

$$Port\_triggering ::= \{on, no\} \tag{14}$$

$$Data\_reference ::= dataID \tag{15}$$

Note:

1. A *Port* belongs to three categories: *Data_port*, *Event_port* and *Event_data_port* (7).
2. A *Event_port* is specified by a *portID*, *Port_direction* and an optional list of *Port_property* (8).

3. A *Data_port* is a *Data_OR_Eventdata_port* whose *Port_triggering* is *no* (11).
4. A *Event_data_port* is a *Data_OR_Eventdata_port* whose *Port_triggering* is *on* (12).
5. *Port_direction* could be either *in*, *out*, or *in out* (13).
6. *portID* adheres to the naming rules specified for all identifiers.
7. A *Port_property* could be *Input_Time*, *Output_Time*, *Timing*, *Fan_out_policy* property association and many others. The following table gives a brief view of properties that are associated to ports. The property should be applied to the corresponding port categories. Yue: the properties marked in red have been translated. (We mainly take into account the properties related to timing aspects and queues.)

Property	In port			Out port		
	Event	Data	Event data	Event	Data	Event data
Input_Time	X	X	X			
Output_Time				X	X	X
Fan_Out_Policy				X	X	X
Input_Rate	X	X	X			
Output_Rate				X	X	X
Timing		X			X	
Source_Name	X	X	X	X	X	X
Source_Text	X	X	X	X	X	X
Type_Source_Name	X	X	X	X	X	X
Device_Register_Address	X	X	X	X	X	X
Base_Address	X	X	X	X	X	X
Allowed_Memory_Binding_Class		X	X		X	X
Allowed_Memory_Binding		X	X		X	X
Actual_Memory_Binding		X	X		X	X
Compute_Entrypoint	X		X			
Compute_Entrypoint_Call_Sequence	X		X			
Compute_Entrypoint_Source_Text	X		X			
Compute_Execution_Time	X		X			
Compute_Deadline	X		X			
Overflow_Handling_Protocol	X		X			
Queue_Size	X		X	X		X
Queue_Processing_Protocol	X		X	X		X
Dequeued_Items	X		X			
Dequeue_Protocol	X		X			
Urgency	X		X			
Transmission_Type		X			X	

The detail of these properties will be explained in next section.

11.1.2 Standard properties

This section gives an explanation of some of the standard properties listed in the above table. We are mainly interested in the properties related to temporal specifications and queuing characteristics. These properties are translated in Signal programs. Other properties, such as properties related to memory binding and source specifications, are not included in the current translation. They could be presented as "pragma" to specify the related information.

1. Properties related to IO policy. These properties define some communication and timing related properties. We are interested in them, especially the timing properties.

   (a) **Input_Time** property can be used to explicitly specify an input time for ports. This property could have a list of values, which indicates that input is frozen multiple times for the execution during one dispatch. The default value is dispatch with zero offset.

   ```
   Input_Time: list of IO_Time_Spec ⇒ (Time ⇒ Dispatch;
         Offset ⇒ 0.0 ns .. 0.0 ns;) applies to (port);
   IO_Time_Spec : type record ( Offset : TimeRange;
         Time : IO_Reference_Time; );
   ```

   Each possible value is a pair of *Time* (possible values: *Dispatch_Time* by default, *Start*, *Completion* and *NoIO*) and a time range *Offset* (0.0 ns .. 0.0 ns by default).

   The **IO_Time_Spec** property specifies the amount of execution time *Offset* relative to a *Time* at which input or output occurs. The value consists of a reference point and time range pair.

   Input_Time possible ReferencePoint:

   - **Dispatch_Time**: (the default value) input is frozen at dispatch time; the time reference is clock time.
     $T = 0$.
   - **Start**: input is frozen at a specified amount of execution time into the execution. The time is within the specified time range. The time range must have positive values.
     $Start\_Time_{low} \leq C \leq Start\_Time_{high}$.
   - **Completion**: input is frozen at a specified amount of execution time relative to execution completion. The time is within the specified time range. A negative time range indicates execution time before completion.
     $(C_{complete}+Completion\_Time_{low}) \leq C \leq (C_{complete}+Completion\_Time_{high})$
     where $C_{complete}$ represents the value of c at completion time.
   - NoIO: input is not frozen. In other words, the port is excluded from making new input available to the source text. This allows users to specify that a subset of ports to provide input. The property value can be mode

specific, i.e., a port can be excluded in one mode and included in another mode.

The content of incoming ports are frozen at a specified time (Yue: Input_Time.) This means that the content of the port that is accessible to the recipient does not change during the execution of a dispatch (Yue: Input_Time, not really dispatch time) even though the sender may send new values.

Frozen: From the point of Input_Time on, any new arrived data (or event, or event data) is not accessible until the next Input_Time (Figure 11).

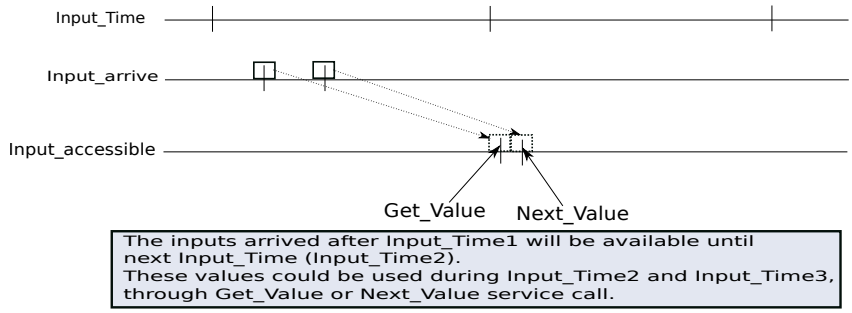

Figure 11: Input frozen

The input of other ports that can trigger dispatch is not frozen. Input of the remaining ports is frozen according to the specified input time.

In AS5506A v2, p136, "The Input_Time can be done for all ports by specifying the property value for the thread". But the Input_Time property is defined only applies to a port, but not to a thread (p262).

(b) **Output_Time** specifies the amount of execution time until completion at which output becomes available. This property indicates the time output is transmitted to connected components. The default value is completion with zero offset. Possible value is a pair of reference time *Time* (*Start*, *Completion* by default, *Deadline* and *NoIO*) and a time range *Offset* (0.0 ns .. 0.0 ns by default).

Output_Time: **list of** IO_Time_Spec ⇒ (Time ⇒ Completion;
Offset ⇒ 0.0 ns .. 0.0 ns;) **applies** to (port);

Output_Time possible ReferencePoint:

- **Start_Time**: output is transmitted at a specified amount of execution time into the execution. The time is within the specified time range. The time range must have positive values.
  $Start\_Time_{low} \leq C \leq Start\_Time_{high}$.
- **Completion**: output is transmitted at a specified amount of execution time relative to execution completion. The time is within the specified time range. A negative time range indicates execution time before completion.
  $(C_{complete}+Completion\_Time_{low}) \leq C \leq (C_{complete}+Completion\_Time_{high})$
  where $C_{complete}$ represents the value of c at completion time.

The default is completion time with a time range of zero, i.e., it occurs at $C = C_{complete}$

- **Deadline**: (the default value) ; output is transmitted at deadline time; the time reference is clock time.
  $T = Deadline$.
- **NoIO**: output is not transmitted . In other words, the port is excluded from making new output from the source text. This allows users to specify that a subset of ports to provide output. The property value can be mode specific, i.e., a port can be excluded in one mode and included in another mode.

The output will be transmitted immediately if it is called by a Send_output service call, otherwise it will be sent out at next Output_Time. (Figure 12)

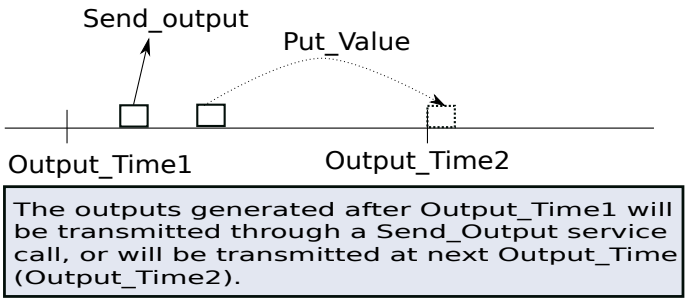

Figure 12: Output_Time

Input_Time and **Output_Time** can have a list of values. Two Signal events *InEvent* and *OutEvent* are used to represent the **Input_Time** and **Output_Time**. They must be guaranteed in the time range. A Signal process Between() is provided to ensure this. The details could be refered in Section 14.4.

(c) **Input_Rate** specifies the number of inputs per dispatch or per second of data, events, event data or subprogram calls. One input per thread dispatch by default.

```
Input_Rate: Rate_Spec ⇒ (Value_Range ⇒ 1.0 .. 1.0;
      Rate_Unit ⇒ PerDispatch; Rate_Distribution ⇒ Fixed) applies to (port);
Rate_Spec : type record (Value_Range : aadlreal;
      Rate_Unit: units (PerSecond, PerDisptach);
      Rate_Distribution : Supported_Distributions; );
```

(d) **Output_Rate** specifies the number of outputs per dispatch or per second of data, events, event data or initiations of subprogram calls. One output per thread dispatch and *fixed* distribution by default.

```
Output_Rate: Rate_Spec ⇒ (Value_Range ⇒ 1.0 .. 1.0;
      Rate_Unit ⇒ PerDispatch; Rate_Distribution ⇒ Fixed) applies to (port);
```

(e) **Timing** property specifies the connection type of a data port.

```
Timing : enumeration (sampled, immediate, delayed)
      ⇒ sampled applies to (port);
```

i. If Timing is declared as immediate, then Output_Time is Completion and Input_Time is Start. The defined Input_Time and Output_Time are ignored.

ii. If Timing is declared as delayed, then Output_Time is Deadline and Input_Time is Dispatch.

Timing	Input_Time	Output_Time
immediate	*Start*	*Completion*
delayed	*Dispatch*	*Deadline*

(f) **Fan_Out_Policy** property specifies how the output is distributed to multiple recipients of a port with multiple outgoing connections. Default value is *Broadcast*.

```
Fan_Out_Policy: enumeration (Broadcast, RoundRobin, Selective, OnDemand)
    applies to (port);
```

The Fan_Out_Policy property indicates whether the output is passed to all recipients (Broadcast), to the next recipient ready to be dispatched (OnDemand), or the output is distributed evenly to the recipients (RoundRobin). If the property is not specified the default is Broadcast. If the fan out policy is OnDemand, a queue may be associated with the port through the use of the appropriate queue properties.

PLG: the exact title for these queue properties is In port queue properties ; more over the complementary wording in 8.2.3 does not mention queues associated with output ????. Moreover an AADL-thread can probably wait for dispatch coming from several ports; if it is dispatched from one source, it should cancel the other demands,....???? it's a very costly protocol

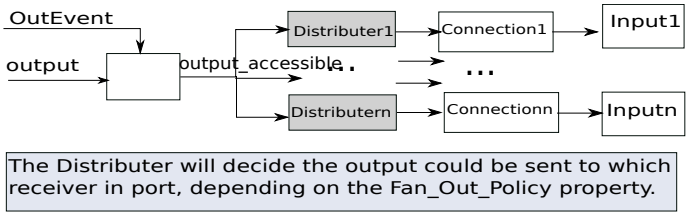

Figure 13: Fan_Out_Policy

A controler (Distributer) is needed to choose the recipients of an out port. (Figure 13.) For each connection, a Distributer is added to control that whether the output will be sent to this connection or not. A set of Distributer are defined in a library: Distributer_Broadcast, Distributer_OnDemand,etc. Depending on the Fan_Out_Policy, a corresponding Distributer will be used. The details are specified in Section 14.4.

```
process Distributer_Broadcast =
{integer Distributer_ID;}
(? i; ! o;)
```

```
    (|o := i|);

process Distributer_OnDemand =
{integer Distributer_ID;}
(? i; event demand; ! o;)
  (|o := i when demand |);
```

2. Properties related to external source text (...) These properties do not specify timing or communication properties. They could be translated as "pragma" or "spec" in Signal.

   Source_Name: **aadlstring applies to** (data, port, subprogram, parameter);
   Source_Text : **inherit list of aadlstring applies to**
   (data, port, subprogram, thread, thread group, process, system, memory, bus, device, processor, parameter, feature group, package);
   Type_Source_Name: **aadlstring applies to** (data, port, subprogram);

   The **Source_Name** property specifies a source declaration or source name within the associated source text that corresponds to a feature defining identifier.

   The **Source_Text** property specifies a list of files that contain source text.

   The **Type_Source_Name** property specifies a source declaration or source name within the associated source text that corresponds to a feature defining identifier.

3. Properties related to memory space (binding,...) These properties are related to hardware memory. We will not translate them in Signal.

   Device_Register_Address: **aadlinteger applies to** (port, feature group);
   Allowed_Memory_Binding_Class: **inherit list of classifier** (memory, system, processor);
   Allowed_Memory_Binding: **inherit list of reference** (memory, system, processor);
   Actual_Memory_Binding: **inherit list of reference** (memory);
   Base_Address: **aadlinteger** 0 .. Max_Base_Address;

4. Port specific compute entrypoint properties for event and event data ports:

   Compute_Entrypoint: **classifier** (Subprogram Classifier) **applies to**
   (thread, device, provides subprogram access, event port, event data port);
   Compute_Execution_Time: Time_Range **applies to** (thread, device,
   subprogram, event port, event data port);
   Compute_Deadline: Time **applies to** (thread, device,
   subprogram, subprogram access, event port, event data port);

   (4) Event (-data) ports may dispatch a port specific Compute_Entrypoint. This permits threads with multiple event or event data ports to execute different source text sequences for events arriving at different event ports (PLG: such an entry is a black box in a gray box) If specified, the port specific Compute_Execution_Time and Compute_Deadline takes precedence over those of the containing thread.

   Yue: how to translate these properties in Signal?

5. Properties related to queues. These properties are related to event or event ports.

   (a) **Queue_Processing_Protocol**. Queues will be serviced according to this property, by default in a FIFO order. An event (event data) port could be represented by a Signal queue, for example: FIFO.

   Queue_Processing_Protocol: Supported_Queue_Processing_Protocols ⇒ FIFO **applies to** (event port, event data port, subprogram access);

   (b) **Queue_Size**. The default port queue size is 1.

   Queue_Size: **aadlinteger** 0 .. Max_Queue_Size ⇒ 1 **applies to** (event port, event data port, subprogram access);

   (c) **Dequeue_Protocol**. This property specifies the dequeuing option to the receiving application.

   Dequeue_Protocol: **enumeration** (OneItem, MultipleItems, AllItems) ⇒ OneItem **applies to** (event port, event data port);

   - **OneItem**: a single frozen item is dequeued and made available to the source text unless the queue is empty.
   - **AllItems**: all items that are frozen at input time are dequeued and mad available to the source text via the port variable.
   - **MultipleItems**: multiple items can be dequeued one at a time from the frozen queue.

   (d) **Dequeued_Items**. This property specifies the maximum number of items that ate made available to the application when the input is frozen at input time.

   Dequeued_Items: **aadlinteger applies to** (event port, event data port);

   (e) **Overflow_Handling_Protocol**. This property determine the action, when an event (event data) arrives and the number of queued events is equal to the specified queue size.

   Overflow_Handling_Protocol: **enumeration** (DropOldest, DropNewest, Error) ⇒ DropOldest **applies to** (event port, event data port, subprogram access);

6. Other properties.

   Urgency: **aadlinteger** 0 .. Max_Urgency
   Transmission_Type: **enumeration** (push, pull);

   The **Urgency** property specifies the urgency with which an event at an in port is to be serviced relative to other events arriving at or queued at other in ports of the same thread.

   The **Transmission_Type** property specifies whether the transmission across a data port connection is initiated by the sender (*push*) or by the receiver (*pull*).

   Yue: how to translate them in Signal?

11.1.3 In out (common) port behavior

Rate properties (29) The Input_Rate and Output_Rate properties specify the rate at which input and output is expected to occur at the port with the associated property. By default the input and output rate of ports is the rate at which the thread executes. The rate can be fixed (periodic) or according to a distribution. An input or output rate higher than the dispatch rate of a thread indicates that multiple inputs or multiple outputs are expected during a single dispatch. An input or output rate lower than the dispatch rate of a thread indicates that inputs or outputs are not expected at every dispatch. If an Input_Time or Output_Time property is specified, then the number of values must be consistent with the rate. An input or output rate lower than the period indicates that input is not expected at every dispatch and that output is not expected to be transmitted at every dispatch.

Those rate properties will not generate anything but comments in Signal. The consistence between a (Input/Output)_Time list statically defined and a (Input/Output)_Rate given by a distribution is not obvious to understand as such. Moreover, a rate lower than the rate given by the size of the (Input/Output)_Time results in a non-deterministic behavior. Thus, we should assume that (Input/Output)_Time list gives the (maximal) number of (Input/Output) values between 2 dispatches in the current Mode. And we consider Rates as information for verification tools.

Input ports (13) Data, events, and event data arriving through incoming ports is made available to the receiving thread, processor, or device at a specified input time. From that point on any newly arriving data, event, or event data is not available to the receiving component until the next dispatch (PLG: Input_Time, not really dispatch), i.e., the input is frozen.

(17) The Input_Time property can have a list of values. In this case it indicates that input is frozen multiple times for the execution of a dispatch.

1. **Actual input**

   (15) The Input_Time property can be used to explicitly specify an input time for ports. This can be done for all ports by specifying the property value for the thread, or it can be specified separately for each port. (PLG: may some ports inheriting thread property while others have their own specification?) Yue: The property definition declares that this property could only apply to a port not to a thread. Then how to specify it for a thread?

   (40) A Receive_Input runtime service allows the source text of a thread to explicitly request port input on its incoming ports to be frozen and made accessible through the port variables....The Receive_Input service takes a mask parameter that specifies for which ports the input is frozen. (PLG: links with Input_Time ???)

   (42) A Get_Value runtime service shall be provided that allows the source text of a thread to access the current value of a port variable. The service call returns the data value. Repeated calls to Get_Value result in the same value to be returned, unless the current value is updated through a Receive_Input call or a Next_Value call.

PLG: as far as I understand these rules allow several freezing between two dispatches, not only at dispatch time as indicated in (13).

There are some questions concerning the consistency:

- Is it possible to freeze input after completion ? The reasonable answer is probably that a thread cannot emit complete before all Input_Time occurrences. But in AADL an Input_Time may follow the Completion_Time !!! How is this possible if the thread is not running (see (40) below) ??? Does this means that input freezing is done by some AADL implicit action ??? How this policy can be made consistent with hidden Receive_Input calls.
- Negative time associated with Completion_Time is generally not causal !!!

(9.1.4(28)) Arrival of events on event ports can also trigger a mode switch if the event port is named in a mode transition originating in the current mode (see Section 12). Events that trigger mode transitions are not queued at event ports.

2. **Input ports and Polychrony**

   The input port behavior induces an event Signal-signal InEvent for a port. InEvent has as many occurrences as given by Input_Time list. This occurrences may be dynamically generated according to queue size, TimeOffset, ...

Output ports (27) The Output_Time property can have a list of values. In this case it indicates that output is transmitted multiple times as part of the execution of a dispatch.

1. **Actual output**

   (38) A Send_Output runtime service allows the source text of a thread to explicitly cause events, event data, or data to be transmitted through outgoing ports to receiver ports. The Send_Output service takes a mask parameter that specifies for which ports the transmission is initiated. Send_Output is a nonblocking service. (PLG: links with Output_Time ???) Yue: for event (event data) ports, the output could occur anytime during the execution through a Send_Output service call.

   (39) A Put_Value runtime service allows the source text of a thread to supply a data value to a port variable. This data value will be transmitted at the next Send_Output call in the source text or by the runtime system at completion time or deadline.

   PLG: These rules allow several sending between two dispatches.

   There are some questions concerning the consistency:

   - Is it possible to send after completion ? The reasonable answer is probably that a thread cannot emit complete before all Output_Time occurrences. But in AADL an Output_Time may follow the Completion_Time !!! How is this possible if the thread is not running (see (40) below) ??? Does this means that output is achieved by some AADL implicit action ??? How this policy can be made consistent with hidden Send_Output calls.

- Negative time associated with Completion_Time is generally not causal !!!

2. **Output ports and Polychrony**

   The output port behavior induces an event Signal-signal OutEvent for a port. OutEvent has as many occurrences as given by Output_Time list. This occurrences may be dynamically generated according to queue size, TimeOffset, ... and Fan_Out_Policy.

   A Fan_Out_Policy that is not the standard Broadcast policy, will generate a Signal-process in charge of this policy

   Since the Fan_Out_Policy is closely related to its associated connections, a Distributer is added between the output and each connection. To ease the implementation, the Distributer is not included in the translated out port process, but inserted between the output and the connections. (the details translation of Fan_Out_Policy is specified in Section 14.4).

   Let $x \in Out\_port$, which could be either an Out_data_port,
   or an Out_event_port, or an Out_event_data_port
   $y \in Fan\_Out\_Policy$ is x's property
   $c = \{c_1, c_2, \ldots c\_n\}$ are the connections whose source is x
   $\mathbf{PortFanOutTranslation}(x, y, c)$ =
   $(|\mathbf{PortFanOutTranslation}(x, y, c_1)$
   $|\ldots$
   $|\mathbf{PortFanOutTranslation}(x, y, c_n)|)$

   An AADL in out port is translated into a front-end Signal-process depending upon the port properties to manage directed connections.

11.1.4 Data port

(9) Data ports are intended for transmission of state data such as signals. Therefore, no queuing is supported for data ports. A thread can determine whether the input buffer of an in data port has new data at this dispatch by checking the port status trough a Get_Count service call, which is accessible through the port variable through a Get_Value service call. If no new data value has been received the old value is made available.

(9.1(L10))A data port cannot be the destination of more than one semantic port connection unless each semantic port connection is contained in a different mode.

(5.4.6(71))...data port connections across synchronization domains are sampled connections.

1. **Aggregate data port**

   (8.3.1(15))The role of an aggregate data port is to make a collection of data from multiple outgoing data ports available in a time-consistent manner. Time consistency