- 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

in this context means that if a set of periodic threads is dispatched at the same time to operate on data, then the recipients of their data see either all old values or all new values. This is accomplished by declaring a data port, whose data classifier has an implementation with data components corresponding to the data of the individual data ports.

The functionality of an aggregate data port can be viewed as a thread whose only role is to collect the data values from several in data ports and make them available as an aggregate data record; on the receiving side an equivalent thread takes passes on the elements of the aggregate data record on to the respective out data ports of receiving threads....

2. **Behavior**

   It seems that data ports can have multiple output and multiple input during an AADL thread dispatch. Figure 14 gives an example of one input time during a dispatch. The in data port could receive multiple values during one dispatch, but these values could not be accessible immediately. At input time, the latest value is accessible, and this value would be used during this input time.

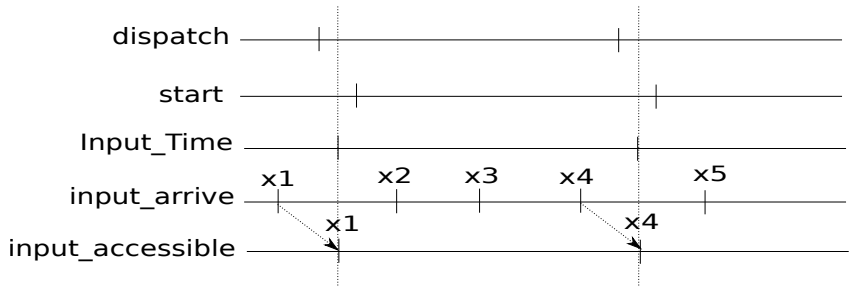

Figure 14: In data port (one input time)

(24) By default, the output time, i.e., the time output is transmitted to connected components, is the completion time for data ports.

3. **Data ports and Polychrony**

   Data port can be represented using cell Signal-process. A data port is close to a Signal-signal (several data ports can be synchronous).

   An aggregate data port can be implemented as indicated in AADL-8.1(7), as a Signal-process that builds a struct before sending values, and counterpart one that breaks the struct before delivering individual flows.

   So we need a Signal-process model for input data port and a Signal-process model for output data port. These models may be a unique common model..

   A data port supports only one value. If no new data value has been received, the old value is made available. A data port could be represented by a buffer, where a data written to the buffer remains there, until it is overwritten by a new one.

```
process buffer=(? i; ! o)
(| interleave(i, o)
 | o := current{false}(i, ^o)
 |)

process current =
{boolean v0;}
(? wx; event c; ! rx;)
(| rx := (wx cell c init v0) when c
 |)

process interleave =
(? x, sx;)
(| b:= not (b$1 init false)
 | x ^= when b
 | sx ^= when (not b)
 |) where boolean b; end;
```

4. **In data port and Polychrony**

   The latest value of in data port buffer is frozen at InEvent time, and this value is memorized in M.

```
process M = (? i; ! o)
(| o := AT(i, ^o)
 |)

process AT = (? i; event h; ! o)
(| o := (i cell h) when h
 |)
```

   The Frozen_data_ port() copies the latest value of the buffer at specified time instant (InEvent).

```
process Frozen_data_port =
(? ii; event InEvent; ! oo;)
(| oo := AT(ii, InEvent) |)
```

   A notation **DataPortTranslation**()is used to represent the syntax translation schema for *data port*, and $V(p)$ is used to represent the value of a property p, and $Count(v)$ represents the number of elements of value v (if v is a list).

   Yue: An in data port could have the following related properties: Input_Time, Input_Rate, Timing, Transmission_Type and source related and memory related properties. Since in this document, we are interested in the timing aspects, so the Timing

and Input_Time property are translated. The Input_Rate property is related to Input_Time, so it is not translated directly. The Transmission_Type specifies whether the data port connection related to this port is sender or receiver initiated, which could be interpreted in data port connection section. The other properties could be interpreted later, or implemented as Signal "pragma".

$$\begin{aligned} \text{Let } x &= (x_1, x_2, \{x_3, x_4, \dots\}) \in In\_data\_port \\ \text{where: } x_1 &\in portID \\ x_2 &\in Data\_reference \\ x_3 &\in Input\_Time \\ x_4 &\in Timing \end{aligned}$$

The in data port is separated into three cases according to the **Timing** property: *sampled, immediate* and *delayed.*

(a) $V(x_4)$ = **Sampled**.
The **Timing** property is specified as *sampled* or not specified. A sampled data port could be separated into two cases according to the number of IO_Spec_Time values of the **Input_Time** property: **Input_Time** contains only one reference time value, and **Input_Time** contains several reference time values.

i. $Count(V(x_3)) = 1$. **Input_Time** contains only one value, or **Input_Time** not specified.

$$\begin{aligned} \text{Let } x_3 &= (x_{31},\ x_{32}, x_{33},\ x_{34}, x_{35}) \in Input\_Time \\ \text{where: } x_{31}, x_{33} &\in integer \\ x_{32}, x_{34} &\in Time\_Unit \\ x_{35} &\in IO\_Reference\_Time \end{aligned}$$

An input written to the buffer remains there until it is overwritten by a new arrived input instance. At InEvent (given by Input_Time property), the current element in the buffer is copied (Frozen) to a memory M. The value in M will be used by Get_Value or other service call. The buffer may be overwritten multiple times before next InEvent occurs.
The *InEvent* (the actual input time) is under time constraint (*Between*) (Figure 15). The *InEvent* must be in the time range of [*ReferenceTime + min_offset , ReferenceTime+max_offset*]. *ReferenceTime* is the reference time (Specified in the **Input_Time** property, which could be *Dispatch*, *Start* ...). Time unit *unit1* (resp. *unit2*) is the unit of time offset *min_offset* (resp. *max_offset*).

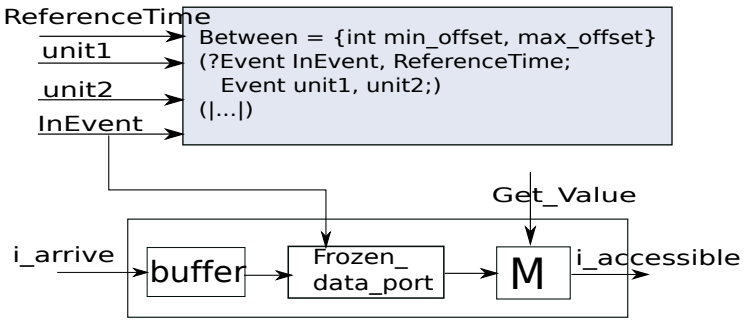

Figure 15: In data port (sampled)

```
process Between=
{ integer min_offset, max_offset }
(? event InEvent, ReferenceTime, unit1, unit2;)
(|...|)
```

Let data port x is in the context of a component, for example: a thread. We use a notation $\mathscr{C}$ to present this context. An in data port in a context $\mathscr{C}$ is presented as an instance of a Signal process.

$$\textbf{DataPortTranslation}(x, \mathscr{C}) =$$
$$x'_1 :: i\_accessible := InDataPort\{m, n\}$$
$$(i\_arrive,\ InEvent,\ Reference\_Time,\ unit1,\ unit2)$$

where i_arrive, $InEvent$, $Reference\_Time$, $unit1$, $unit2$ come from the context $\mathscr{C}$

$$Reference\_Time = \begin{cases} Dispatch, & \text{if } V(x_{35}) = \text{Dispatch} \\ Start, & \text{if } V(x_{35}) = \text{Start} \\ Complete, & \text{if } V(x_{35}) = \text{Complete} \end{cases}$$

$$m = \textbf{PropertyTranslation}(x_{31})$$
$$n = \textbf{PropertyTranslation}(x_{33})$$
$$x'_1 = \textbf{IDTranslation}(x_1) = x_1$$
$$unit1 = \textbf{PropertyTranslation}(x_{32})$$
$$unit2 = \textbf{PropertyTranslation}(x_{34})$$

The *InDataPort* process is defined in a library AADL_DATAPORT.

```
process InDataPort =
 { integer min_offset, max_offset;}
  (?  i_arrive;
      event InEvent, ReferenceTime, unit1, unit2;
   !  i_accessible;)
  (| o1 := buffer(i_arrive)
   | o2 := Frozen_data_port(o1, InEvent)
```

```
| Between{min_offset, max_offset}
        (InEvent, ReferenceTime, unit1, unit2)
| i_accessible := M(o2)
| )
where
  o1, o2;
end;
```

The *buffer(), Frozen_data_port()* and *M()* processes are defined in library AADL_DATAPORT. The translation of ID, Data_reference and Port_property will be introduced in later sections in detail. (The detailed **Input_Time** property translation can be found in Section 14.4.)

ii. $Count(V(x_3)) = n > 1$. **Input_Time** contains a list of values.
For example:
Input_Time: **list of** (Dispatch, 0.0ns .. 1.0ns) (Start, 0.0ns .. 1.0ns) **applies** to portA;
This case is left for further study.

(b) $V(x_4) = Immediate$.
If the **Timing** property is declared as *immediate*, then the *InEvent* is *Start* and zero *Offset* (the **Input_Time** value is ignored if it is defined). (Figure 16)

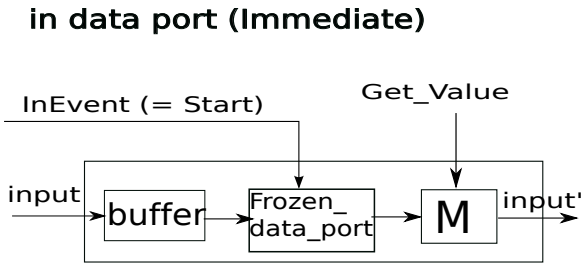

Figure 16: In data port (immediate)

In this case, the translation is similar to sampled data port (with only one input value). The reference time $Reference\_Time = Start$, offset is 0, and $InEvent = Start$ (InEvent is generated in another thread control process).

$$\mathbf{DataPortTranslation}(x, \mathscr{C}) =$$
$$x_1' :: i\_accessible := InDataPort\{0,0\}(i\_arrive,\ InEvent,\ Start,\ true, true)$$

where i_arrive, $InEvent$, $Start$ come from the context $\mathscr{C}$

(c) $V(x_4) = Delayed$.
If the **Timing** property is declared as *delayed*, then the *InEvent* is *Dispatch* and zero *Offset* (the defined **Input_Time** value is ignored).

In this case: the translation is similar to sampled data port (with only one input value). The reference time $Reference\_Time$ = $Dispatch$, offset is 0, and $InEvent$ = $Dispatch$ (InEvent is generated in another thread control process).

$$\textbf{DataPortTranslation}(x, \mathscr{C}) =$$
$$x'_1 :: i\_accessible := InDataPort\{0,0\}(i\_arrive,\ InEvent,\ Dispatch,\ true, true)$$

where i_arrive, $InEvent$, $Dispatch$ come from the context $\mathscr{C}$

5. **Out data port and Polychrony**

An out data port could specify the following properties: Output_Time, Fan_Out_Policy, Output_Rate, Timing, Transmission_Type, source related properties and memory related properties. The temporal properties are considered in out translation, i.e., Output_Time, Fan_Out_Policy and Timing. The Output_Rate is related to Output_Time, it is not translated. And the other properties are not considered yet.

$$\text{Let } x = (x_1, x_2, \{x_3, x_4, x_5 \dots\}) \in Out\_data\_port$$
$$\text{where: } x_1 \in portID$$
$$x_2 \in Data\_reference$$
$$x_3 = Output\_Time \in Port\_property$$
$$x_4 = Timing \in Port\_property$$
$$x_5 = Fan\_Out\_Policy \in Port\_property$$

(Figure 17.) The Output is sent out (Send) when OutEvent is true. A Distributer will select the recipients depending on Fan_Out_Policy. (To ease the implementation, a distributer is added between the output and each connection.)

```
process Send = (? i, OutEvent; ! o;)
  (| o := AT(i, OutEvent) | )
```

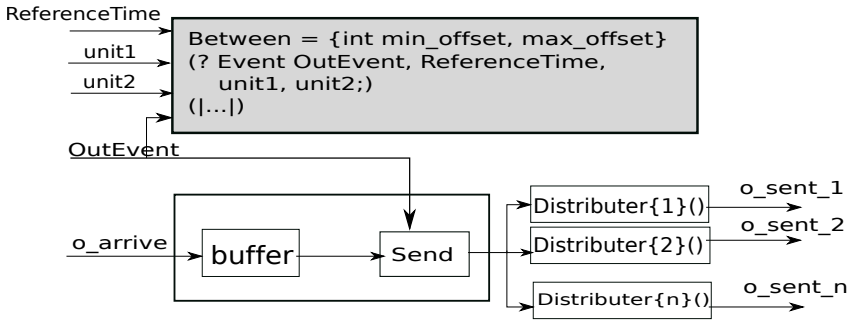

Figure 17: Out data port (sampled)

The out data port is separated into three categories depending on the **Timing** property: *sampled, immediate* and *delayed.*

(a) **Sampled**. $V(x_4)$ = $Sampled$.

The **Timing** property is specified as *sampled* or not specified (as default value).

The *OutEvent* is restricted by a constraint *Between* (or *SeveralBetween*), depending on the number of reference times the *Output_Time* property specifies.

In case of Output_Time specifies only one reference time value, $Count(V(x_3) = 1)$.

Let $x3 = (x_{31}, x_{32}, x_{33}, x_{34}, x_{35}) = Output\_Time \in Port\_property$

where: $x_{31}, x_{33} \in aadlintegerinteger$

$x_{32}, x_{34} \in Time\_Unit$

$x_{35} \in IO\_Reference\_Time$

The *OutEvent* must satisfy the time constraints (*Between()*, the detailed translation of **Output_Time** is specified in Section 14.4).

Let out data port x is in the context of a component, for example: a thread. We use a notation $\mathscr{C}$ to present this context. An out data port in a context $\mathscr{C}$ is presented as an instance of a Signal process.

$\mathbf{DataPortTranslation}(x, \mathscr{C})$ =

x'_1 :: o := $OutDataPort\{m, n\}$

$(o\_arrive, OutEvent, Reference\_Time, unit1, unit2)$

where o_arrive, $OutEvent$, $Reference\_Time$, $unit1$, $unit2$ come from the context $\mathscr{C}$

$$Reference\_Time = \begin{cases} Dispatch, & \text{if } V(x_{35}) = \text{Dispatch} \\ Start, & \text{if } V(x_{35}) = \text{Start} \\ Complete, & \text{if } V(x_{35}) = \text{Complete} \end{cases}$$

m = $\mathbf{PropertyTranslation}(x_{31})$

n = $\mathbf{PropertyTranslation}(x_{33})$

x'_1 = $\mathbf{IDTranslation}(x_1)$ = x_1

$unit1$ = $\mathbf{PropertyTranslation}(x_{32})$

$unit2$ = $\mathbf{PropertyTranslation}(x_{34})$

The *OutDataPort* process is defined in library AADL_DATAPORT.

```
process OutDataPort =
 { integer min_offset, max_offset;}
  (?  o_arrive;  event OutEvent, Reference_Time, unit1, unit2;
   !  o;)
  (| o1 := buffer(o_arrive)
   | o := Send(o1, OutEvent)
```

```
| Between{min_offset, max_offset}
          (OutEvent, Reference_Time, unit1, unit2)
| )
where
  o1;
end;
```

The Fan_Out_Policy x_5 and related connections will be translated to link the output and receiving connections (inputs).

In case of Output_Time specifies a list of reference time values, n = $Count(V(x_3) > 1)$, for example, it contains two reference time. This case is left for further study.

(b) **Immediate**. $V(x_4) = Immediate$.

If the **Timing** property is declared as *immediate*, the *OutEvent* is *Completion* . The value of **Output_Time** is ignored.

In this case: the translation is similar to sampled data port (with only one reference time value). The reference time is Completion, offset is 0, and $OutEvent$:= $Completion$

$$\mathbf{DataPortTranslation}(x, \mathscr{C}) =$$
$$x_1' :: o := OutDataPort\{0,0\}(o\_arrive,\ OutEvent,\ Completion,\ true,\ true)$$

where o_arrive, $OutEvent$, $Completion$ come from the context $\mathscr{C}$

$$x_1' = \mathbf{IDTranslation}(x_1) = x_1$$

(c) **Delayed**. $V(x_4) = Delayed$.

Similar as *immediate*. **Output_Time** is ignored. The *OutEvent* is *Deadline*, $OutEvent$:= $Deadline$.

$$\mathbf{DataPortTranslation}(x, \mathscr{C}) =$$
$$x_1' :: o := OutDataPort\{0,0\}(o\_arrive,\ OutEvent,\ Deadline,\ true,\ true)$$

where o_arrive, $OutEvent$, $Deadline$ come from the context $\mathscr{C}$

$$x_1' = \mathbf{IDTranslation}(x_1) = x_1$$

6. **In out data port and Polychrony**

In out data port is separated into in and out two ports?