Connection Instances and Arrays

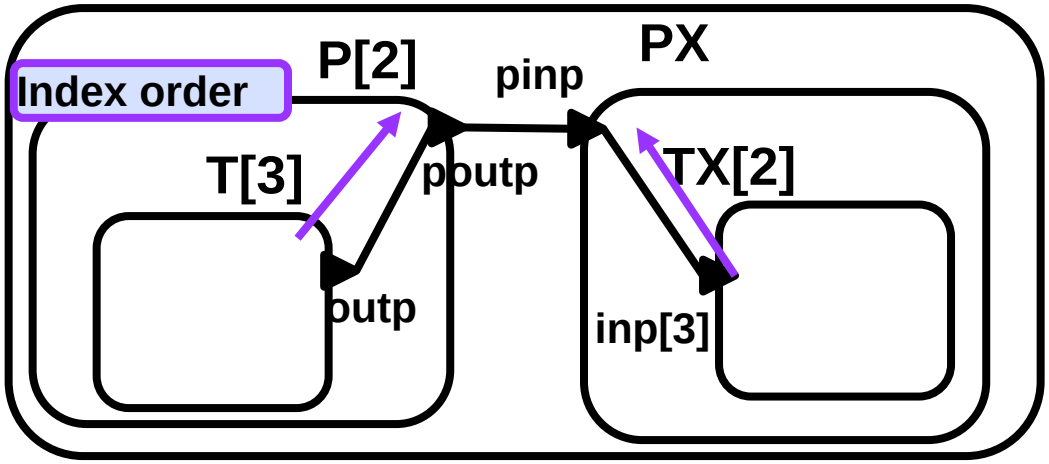
We have

- multi-dimensional arrays for components
- single dimension for features
- Arrays at different levels of the component hierarchy
- Feature arrays only at the leaves of the component hierarchy
  - At higher levels in the hierarchy the features are not declared as arrays
    - They would reflect the dimensionality of lower levels

Array declarations at different levels of the hierarchy result in multi-dimensional instance arrays for the leaf components

- We configure connection instances for resulting arrays in instance model

Connection Instances and Arrays

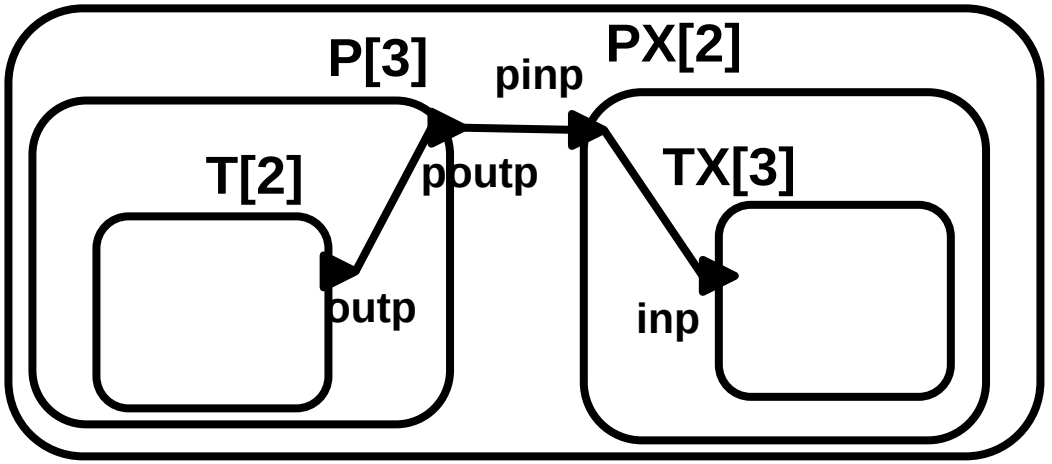


For P.T.outp [3][2] -> PX.TX.inp[3][2]

[1,1] => [2,1]
[3,2] => 1,2]

For P.T [2][3] -> PX.TX[3][2]

[1,1] => [1,1]
[2,1] => 1,2]
[3,1] => [1,3]



Array Index based Connection Declarations

Currently done via Connection_Pattern property or array indices for each connection instance (Connection_Set property

```
        lrus : system HostNodeDual.basic[8];
        brain_bus: system brainDual.braid {BrainPS::BrainSize =>8;};
        lruA : system HostNodeDual.basic;
        lruB : system HostNodeDual.basic;
        lruC : system HostNodeDual.basic;
        lruD : system HostNodeDual.basic;
        brain_bus1: system brainDual.braid {BrainPS::BrainSize =>4;};
connections
    -- each host gets connected to two neighbouring BIU nodes.
      na: bus access brain_bus.hc -> lrus.hc
      {Connection_Pattern => ((Odd_To_Odd,One_To_One),(Even_To_Even,Cyclic_Next));};
    -- manual connection of individual host components to the brain
    -- a single host connects to node N ch2 and node N+1 ch1
Allow- bus1.hc[2][1] <-> lruA.hc[2] and bus1.hc[1][2] <-> lruA.hc[1]
      naA: bus access brain_bus1.hc <-> lruA.hc
      {Connection_Set => ([src=>(2,1);dst=>(2);],[src=>(1,2);dst=>(1);]);};
      -- connected to node 2/3
      naB: bus access brain_bus1.hc <-> lruB.hc
      {Connection_Set => ([src=>(2,2);dst=>(2);],[src=>(1,3);dst=>(1);]);};
      -- connected to node 3/4
      naC: bus access brain_bus1.hc <-> lruC.hc
      {Connection_Set => ([src=>(2,3);dst=>(2);],[src=>(1,4);dst=>(1);]);};
      -- connected to node 4/1
      naD: bus access brain_bus1.hc <-> lruD.hc
      {Connection_Set => ([src=>(2,4);dst=>(2);],[src=>(1,1);dst=>(1);]);};
```

Index Mapping

Inline index mappings

- Option 1: Individual connection declarations:

```
Conn1: port sub1.lfea1[1,2] -> sub2.rfea1[2,1];
Conn2: port sub1.lfea1[2,1] -> sub2.rfea2[1,2];
```

- Option 2: mapping inline with interface connection:

```
Conn1: feature group sub1 -> sub2
       map [1,2] == [2,1], [2,1] == [1,2];
```

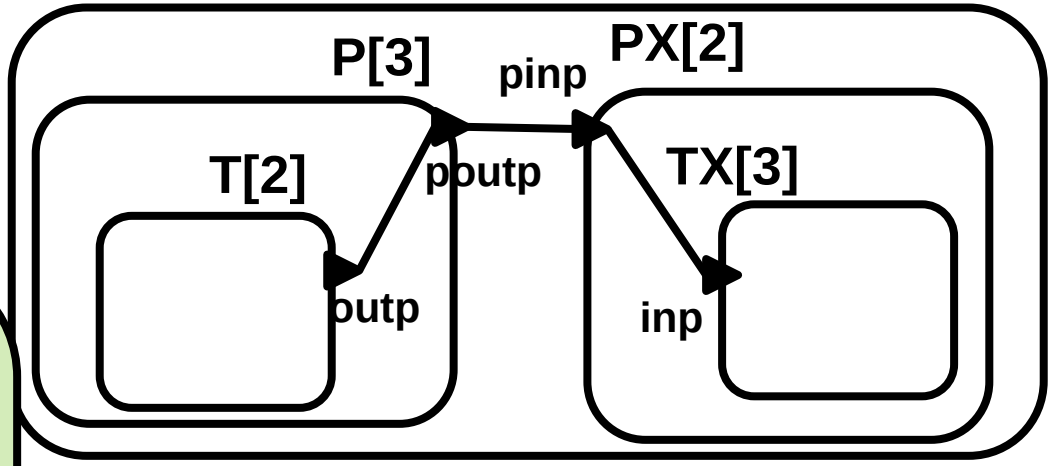
Reusable index mapping for

```
map1: mapping
[1,2] == [2,1], [2,1] == [1,2]
end mapping ;
```

Directional and inverse or bi-directional?

Expose Inner Dimensions as Feature array dimension

System p
Features
Poutp: out event port [2]



System px
Features
Pinp: in event port [3]
End px;

System implementation px.i
Subcomponents
Tx: system tx[3];
Connections
 pinp[1] -> Tx[1].inp;
 pinp[2] -> Tx[3].inp;
 pinp[3] -> Tx[2].inp;
Or
Pinp[] -> Tx[].inp; -- one-to-one

System implementation top.i
subcomponents
 p: system P[3];
Px: system PX[2];
Connections
C1: p[1].poutp[2] -> px[2].pinp[1];
Calt1: p.poutp[1,2] -> px.pinp[2,1];

Combination of feature indices (inner dimensions) and subcomponent indices

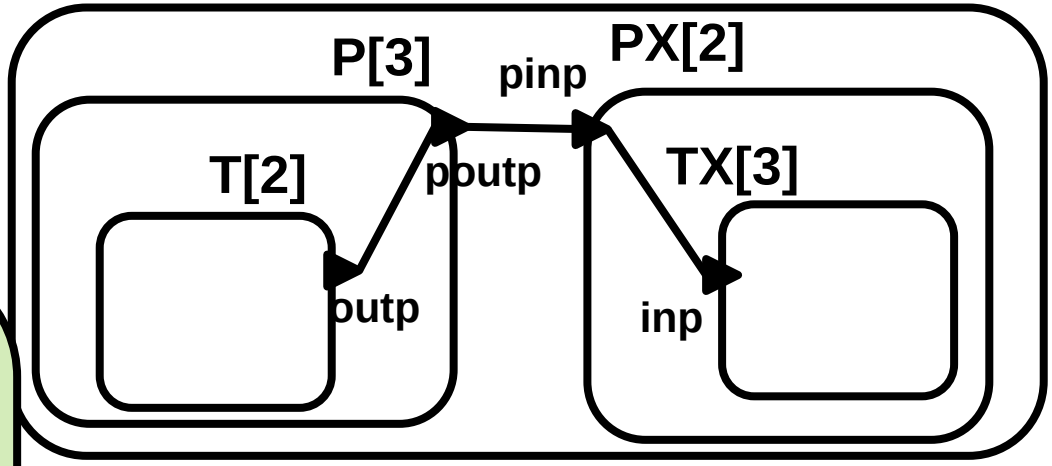
Dimensions and dimension sizes are exposed to the top

Specify indexed access compositionally

System p
Features
Poutp: out event port

System px
Features
Pinp: in event port
End px;

System implementation px.i
Subcomponents
Tx: system tx[3];
Connections
 pinp[1] -> Tx[1].inp;
 pinp[2] -> Tx[3].inp;
 pinp[3] -> Tx[2].inp;
Or
Pinp[] -> Tx[].inp; -- one-to-one



System implementation top.i
subcomponents
 p: system P[3];
Px: system PX[2];
Connections
C1: p[1].poutp -> px[2].pinp;

Mapping of [3,4] -> [4,3] (change in index order) cannot be expressed.

Connections on array subsets

Systems as arrays

```
Src: system s[10];
Dst1: system a[3];
Dst2: system b[7];

Conn1: Src[1..3].p -> Dst1[1..3].p ;
Conn2: Src[4..10].p -> Dst2[1..7].p ;

Map1: Src[1..3].p -> extp1[1..3] ;
Map2: Src[4..10].p -> extp2 ;
```


Exposing Dimensions in Interface

Brendan example

Approach

- Expose externally visible dimensionality through interface

Similar to exposing feature grouping in interface

- Desire to connect elements within nested feature groups at the top level connection

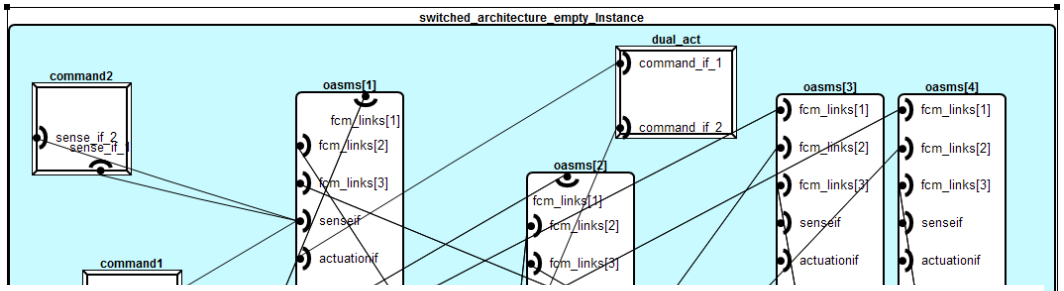
Issue

- Exposes implementation choices
- Handling of different dimensions in different subcomponent variants

Example Model

Array differencing

We would like to refine transparently




```
system tte_fcm
  features
      -- physical connections
      oasm_links:  feature group  ttethernet::ethernet_bi_link[4];
      cross_link_right : feature group ttethernet::ethernet_bi_link[1];
      cross_link_left : feature group ttethernet::ethernet_bi_link_inv[1];

end tte_fcm;
```

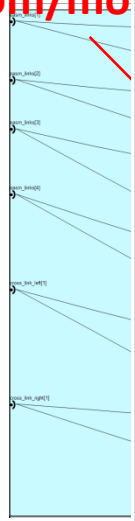
Without defining internals this model allows the connectivity within OSATE

Refining The Internals of the FCM

Internal t

- For eas

6 port com/mo



TTE_FCM.aadl | *TTE_FCM_tte_fcm_i_Instance.imv | TTE_RootSys.aadl | *TTE_RootSys_switched_architecture_empty_Instance.imv

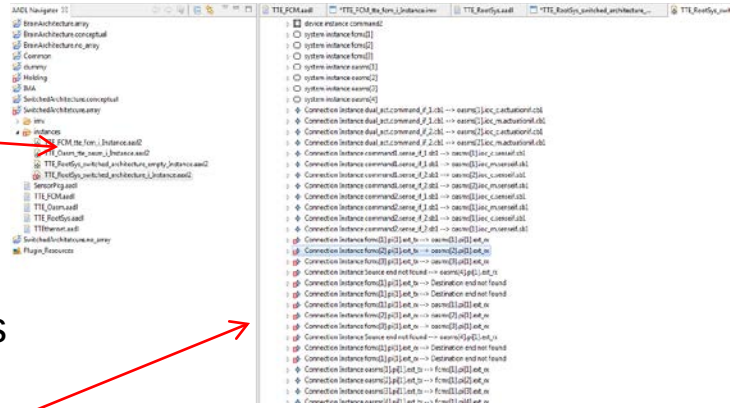
```
system implementation tte_fcm.i
subcomponents
    snitch_c: system ttethernet::snitch.i;
    snitch_m : system ttethernet::snitch_mon.i;
    pi : system ttethernet::passive_isolator[6];
    tx_links : bus ttethernet::link[6];
    proc_com :processor common_components::proc.arm;
    proc_mon :processor common_components::proc.ppc;
    mem_com : memory common_components::edac_mem;
    mem_mon : memory common_components::edac_mem;
    dpram : memory common_components::dpram;

    mon_hb : bus common_components::host_bus;
    com_hb : bus common_components::host_bus;
connections

    oasm_tx : bus access pi.ext_tx ->  oasm_links.tx{ connection_set => ([src =>(1);dst =>(1);],
                                                                         [src =>(2);dst =>(2);],
                                                                         [src =>(3);dst =>(3);],
                                                                         [src =>(4);dst =>(4);]
                                                                        );};

    left_tx :bus access pi.ext_tx -> cross_link_left.tx{connection_set => ([src =>(5);dst =>(1);]);};
    right_tx :bus access pi.ext_tx  -> cross_link_right.tx{connection_set => ([src =>(6);dst =>(1);]);};
    oasm_rx : bus access   pi.ext_rx -> oasm_links.rx { connection_set => ([src =>(1);dst =>(1);],
                                                                         [src =>(2);dst =>(2);],
                                                                         [src =>(3);dst =>(3);],
                                                                         [src =>(4);dst =>(4);]
                                                                        );};

    left_rx :bus access    pi.ext_rx->cross_link_left.rx  {connection_set => ([src =>(5);dst =>(1);]);};
    right_rx :bus access   pi.ext_rx ->cross_link_right.rx {connection_set => ([src =>(6);dst =>(1);]);};
    tx_pi : bus access  pi.int_tx -> tx_links;
    rx_pi : bus access oasm_links.rx -> pi.ext_rx{ connection_set => ([src =>(1);dst =>(1);],
                                                                         [src =>(2);dst =>(2);],
                                                                         [src =>(3);dst =>(3);],
                                                                         [src =>(4);dst =>(4);]
                                                                        );};


    int_txc : bus access  snitch_c.tx -> tx_links;
    int_rxc : bus access  pi.int_rx ->snitch_c.rx;
```

In isolation this component connectivity is also OK in OSATE

Integration

Empty component connectivity works

No error



Refining Empty components, results in connectivity errors due to the introduction of the *internal* structure of the FCM.

```
system implementation switched_architecture.i extends switched_architecture.empty
subcomponents
    fcms: refined to system tte_fcm::tte_fcm.i[3];
    oasms: refined to system tte_oasm::tte_oasm.i[4];
end switched_architecture.i;
```

To make connectivity work we need to refer to the 6 port internal array when making connections at the level above

```
subcomponents
    fcms:  system tte_fcm::tte_fcm.i[3];
    oasms: system tte_oasm::tte_oasm.i[4];

connections
```

5 and 6 do not exist at the higher-level context

```
    fcm_x_lane: feature group fcms.cross_link_right <-> fcms.cross_link_left
                                                    { connection_set => ([src => (5,1);dst =>(6,2);],
                                                                         [src => (5,2);dst =>(6,3);],
                                                                         [src => (5,3);dst =>(6,1);],
                                                                         [src => (6,1);dst =>(5,3);],
                                                                         [src => (6,2);dst =>(5,1);],
                                                                         [src => (6,3);dst =>(5,2);]
                                                                         ) ;};
```

Emerging Proposal for Arrays

View associating array dimensions as a configuration issue.

- This avoids making array dimensions visible in component types at all levels
  - Visibility of dimensions in component type exposes implementation detail and creates issues with different dimensions for different variants
- This aligns with treating the connection pattern as a configuration specification

Idea for array specification

Use configuration specification

- Configuration parameter

```
System configuration top.design
Prototypes
 SubSize1: integer; SubSize2: integer;
Arrays
Sub1 [12, 4];
Sub2.subs2.port1 [3];
Sub2[2].subs2.port1 [3];
Sub2[2];
Sub2.subs2.port1 [3];
```

Looks like dependency injection a la Google

