Arrays in AADL

Peter Feiler

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution

Copyright 2017 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM17-0310

V2 Array Support

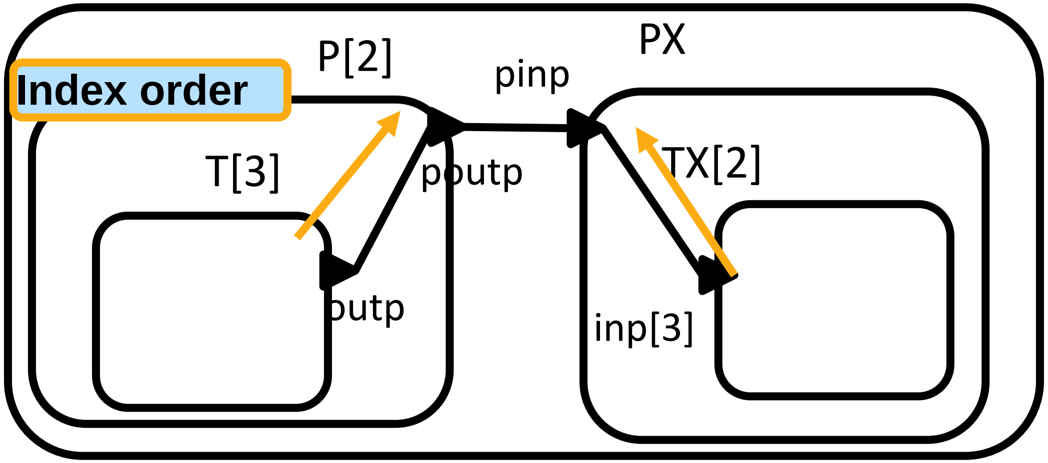
We have

- multi-dimensional arrays for components
- single dimension for features
  - Intended for components like voters
  - Feature arrays only at the leaves of the component hierarchy
- Users can specify the dimensions and later add each dimension size via refined to
  - Sizes can be supplied by property constant or as numeric value
- Arrays at different levels of the component hierarchy
- 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
- Currently array dimensions not reflected in enclosing interface features
  - We do reflect feature aggregation as feature group

© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Connection Instances and Arrays



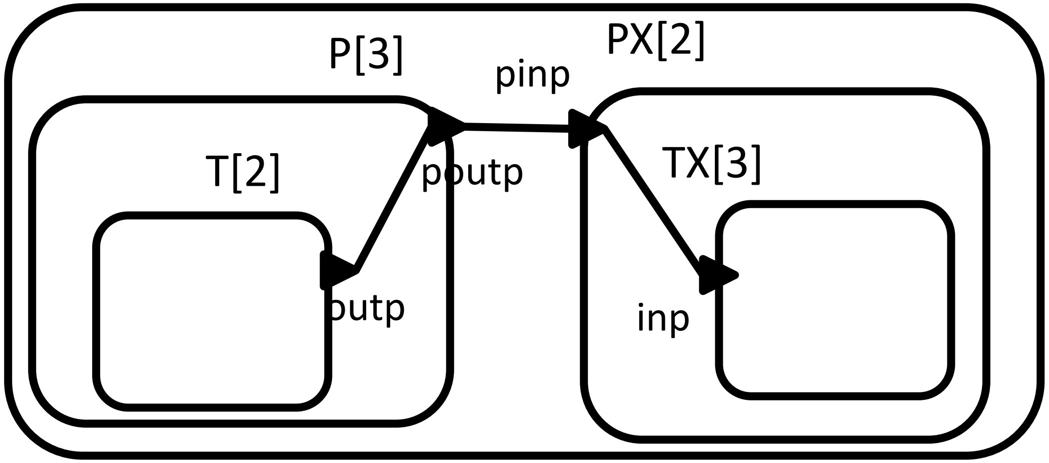
For P.T.outp [3][2] -> PX.TX.inp[3][2]

[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]

Swapped dimensions



© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Exposing Dimensions in Interface

Approach

- Expose externally visible dimensionality through interface
- Allows for connection declarations for specific array elements

Similar to exposing feature grouping in interface

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

© 2017 Carnegie Mellon University
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

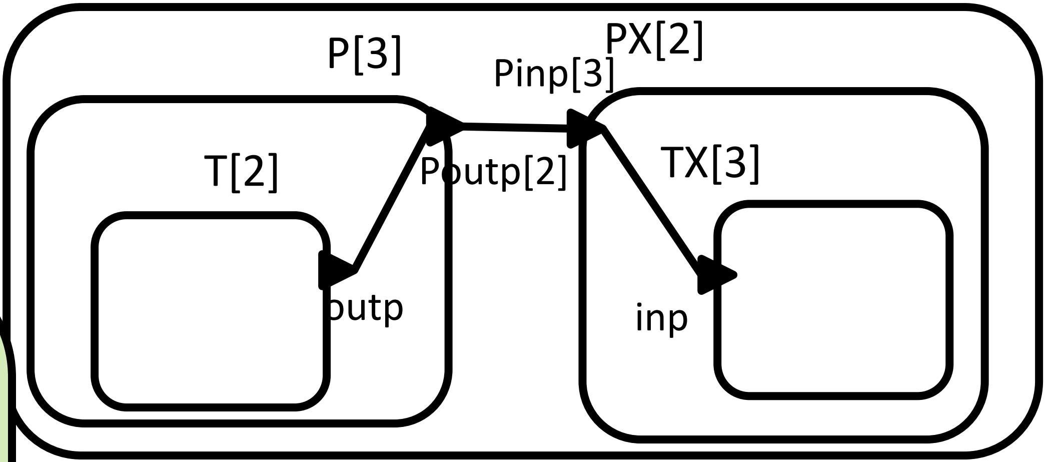
Expose Inner Dimensions as Feature array dimension

```
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
```

Up/down connection is one to one as default



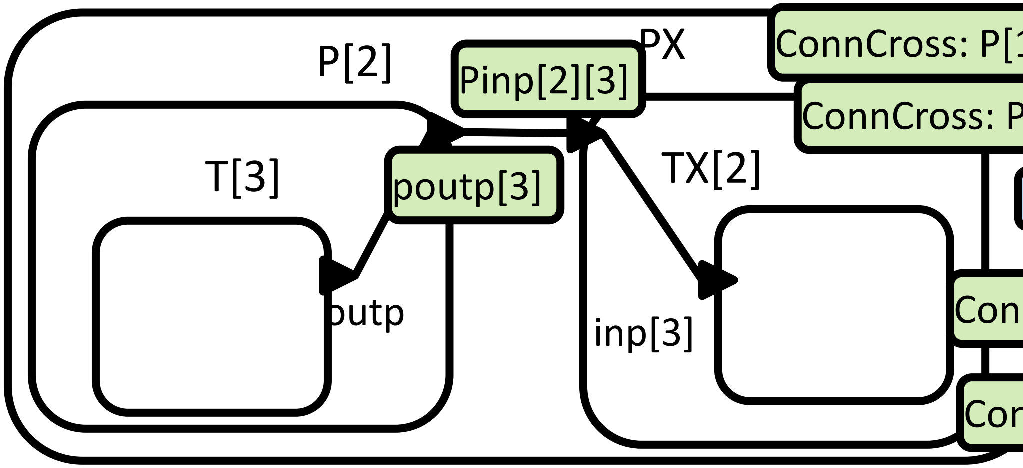
```
System implementation top.i
subcomponents
 p: system P[3];
Px: system PX[2];
Connections
C1: p[1].poutp[2] -> px[2].pinp[1];
Or
Cx: p[k].poutp[j] -> px[j].pinp[k];
```

Cross connection flips dimensions

© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Connection Instances and Arrays



ConnCross: P[1].poutp[2] -> PX.Pinp[1][2]

ConnCross: P[].poutp[] -> PX.Pinp[][]

Left to right dimension order

Conndown: pinp[1][2] -> TX[1].inp[2]

Conndown: pinp[][] -> TX[].inp[]

Need for specifying dimensions in connections representing the whole array?

© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Connection 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: port sub1 -> sub2
       {[1,2] == [2,1], [2,1] == [1,2]};
```

Reusable index mapping for connections between different instances of the same source and target

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

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

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 [];
```

If mapped to whole array allow []?

Connection Patterns

Applicable to Cross connections

- Same as in V2: pattern across all dimensions
- One-to-one, all-to-all, next, previous: within a dimension
- Changing dimension order (e.g., first to second & vice versa)

Dimension index or label plus pattern

Applicable to up/down connections

- Primary pattern: one-to-one
- Change in dimensionality: X[10].p == outerp[5][2]

What up/down mapping patterns make sense without making it overly complex? One-to-many, many-to-many, etc.?

Configuration of Array Sizes

- Configuration of array sizes

```
System implementation top.design
subcomponents
Sub1 : system S[];
Sub2 : system S[];

top.config configures top.design
( Sub1 => [10] ,
Sub2 => S.impl[15]);
```

Dimensions are declared for subcomponents and features

Sizes are configured in for existing classifier

Sizes are configured in together with specific classifier

Parameterized Configuration of Array Sizes

- Configuration of dimensions

```
System top
Features outp: out data port[2][];
```

Example with one dimension size already set

```
System implementation top.design
subcomponents
Sub1 : system S[];
Sub2 : system S[];
```

Internal subcomponent arrays mapped into feature array

```
connections
C1: port Sub1.outport[] -> outp[1][];
C2: port Sub2.outport[] -> outp[2][];
```

Acceptable values within range
Also allow power of 2(?): 2^(2..10)

```
top.config(copies: integer 2..10) configures top.design
( outp => [][copies],Sub1 => S.impl[copies] , Sub2 => S.impl[copies]);
```

Use in subcomponents

```
Topsub: system  top.config( copies => 5);
```

© 2017 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Parameterized Configuration of Array Sizes

- Configuration of classifier and size

```
top.config1(mysub: system s, copies: integer 2..10) configures
top.design
```

```
( outp => copies, Sub1 => mysub[copies] , Sub2 => mysub[copies]);
```

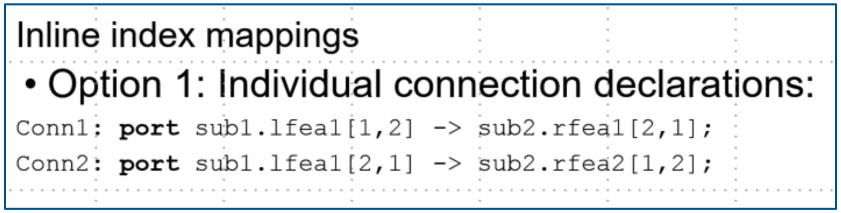
Use in subcomponents

```
Topsub: system  top.config( s.i, copies => 4);
```

Index-based Array Connections Revisited

How does this work when we have parameterized dimension sizes?

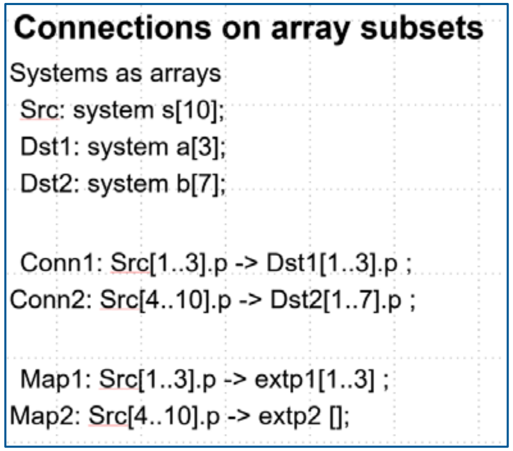
- Once the sizes are configured users can associate an index mapping as part of the configuration
  - Conn1: `{[1,2] == [2,1], [2,1] == [1,2]}`
- algorithmic specification operating on size parameters
  - Bren was going revive such a proposal



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];
```



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 [];
```