

# Flows

Peter Feiler

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Copyright 2019 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](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM19-0625

# Flow Specifications and Sequences

## Flow specification

- Flow source, sink, path
- For features and element in named interface features

## Flow implementation

- Assignment of flow sequence to flow specification

```
interface control is
  insignal: in port;
  outaction: out port;
  processflow: flow path insignal -> outaction;
end;

process control.impl is
  dofilter: thread filter;
  docompute: thread compute;
  extin: mapping insignal => dofilter.insignal;
  ftoc: connection dofilter.outsignal -> docompute.insignal;
  extout: mapping outaction => docompute.outsignal ;
  processflow => flow dofilter.filterpath -> ftoc -> docompute.computepath ;
end;
```

## End to end flow sequence

```
controltoactuate: connection processing.outaction -> actuate.inp;
etef: end to end flow sense.reading -> sensetocontrol-> processing.processflow -> controltoactuate -> actuate.action;
```

# Flow Sequence Specification

Currently (V2)

- Alternating component.flowspec and connection
- Alternating component and connection
  - Flow spec inferred from connection end points
  - Flow related property inferred from value assigned to component

## Additional flexibility

- Component.flowspec sequence only
  - Infer connections
- Connection sequence only
  - Infer component and flow spec
- Reach down for components without flow spec
  - E.g., nested subcomponents
- End to end flow starting and endpoint
  - Define flow path with end points
  - Assign flow sequence as we do for flow spec

# Flows at Platform Level

- Flow sequence as target of connection binding

# Flow Graphs

Objective: Forward and backward traceability

- Forward: variation in latency/age at all end points
- Backward: variation in latency/age from all contributing sources
- Auto-generate from flow specs and connections
  - As we do for propagation graphs

Fan-in/out logic for each component (Merge point semantics)

- Fan in across ports
  - Flow path with multiple inputs (AND)
  - Separate flow paths as alternatives (OR)
  - Pre and post conditions on input/output
- Interpretation of BA logic
  - Input on several ports triggers dispatch
  - Fan in at single port with multiple incoming connections
- Fan out to multiple ports
  - All vs. alternative (Not needed) The fan-in takes care of everything. John Hatcliff discussion on canonical)

# Flow Patterns

- End to end flow spec (endpoints only)
  - Etef1: **flow** sys.proc.thread1.fsrc -> \* -> sys.proc2.thread4.snk
  - Endpoint spec as reference down the hierarchy
  - Infer all possible paths
  - Etef2: **flow all**(GPS.fsrc) -> \* -> **all**(Displays.view)
  - Infer all instances, i.e., all paths between any GPS and Display
- Flow impact
  - Impact : **flow** sys.gps.signalsrc -> \*
  - **All**(FlowSpec) -> \*
- Flow contributors
  - Effector : **flow** \* -> sys.actuator.cmd
  - Effector : **flow** \* -> sys.actuator.cmd