



Introduzione ad AADL

Piergiorgio Di Giacomo
Firenze, 07 giugno 2005

Note storiche

- All'inizio degli anni 90 l'attenzione di molte scuole di ricerca é stata rivolta alla struttura delle architetture software piuttosto che al codice
- I risultati attesi erano quelli di aumentare la qualità e la manutenibilità del software e dei sistemi complessi fornendo supporto alla creazione di vere e proprie architetture software
- Nacquero così i cosiddetti ADL (Architecture Description Language), degli approcci formali alla specifica delle architetture software.
- Di norma tali linguaggi descrivono le interfacce tra i componenti HW e SW di un sistema sia in termini di controllo e flusso di dati sia in termini di aspetti non funzionali del sistema (affidabilità, tempismo, livello di sicurezza, ecc.)

Note storiche

- Nel 1991 il DARPA (Defense Advanced Research Projects Agency) e l'esercito degli Stati Uniti decidono di patrocinare lo sviluppo del MetaH un ADL dedicato al dominio aeronautico.
- Il progetto, sviluppato presso Honeywell Labs produce oltre al linguaggio anche un set di tools (editor grafici e testuali, analizzatori di scheduling, affidabilità, timing) il cui impiego ha portato evidenti benefici nella progettazione di sistemi aerei militari.
- Per questo nel 2001 il MetaH é stato preso come base per la creazione di un nuovo standard: l'AADL (Avionics Architecture Description Language), patrocinato dal SAE (Society of Automotive Engineers).
- Successivamente, dato che le caratteristiche dell'AADL lo rendono adatto alla descrizione di qualunque sistema embedded real time, il significato dell' acronimo é stato cambiato in Architecture & Analysis Description Language

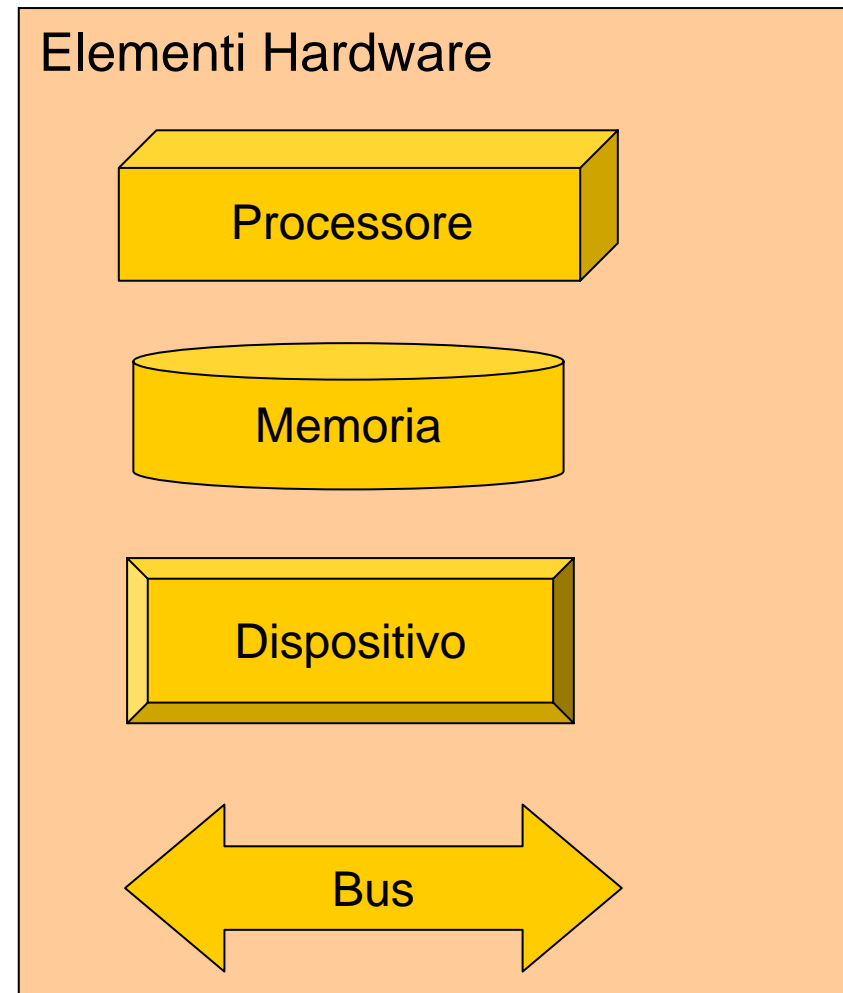
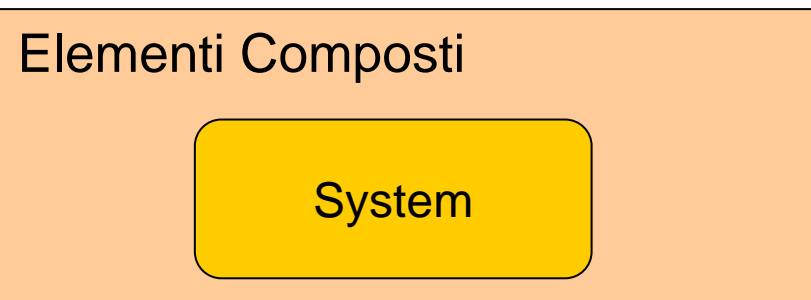
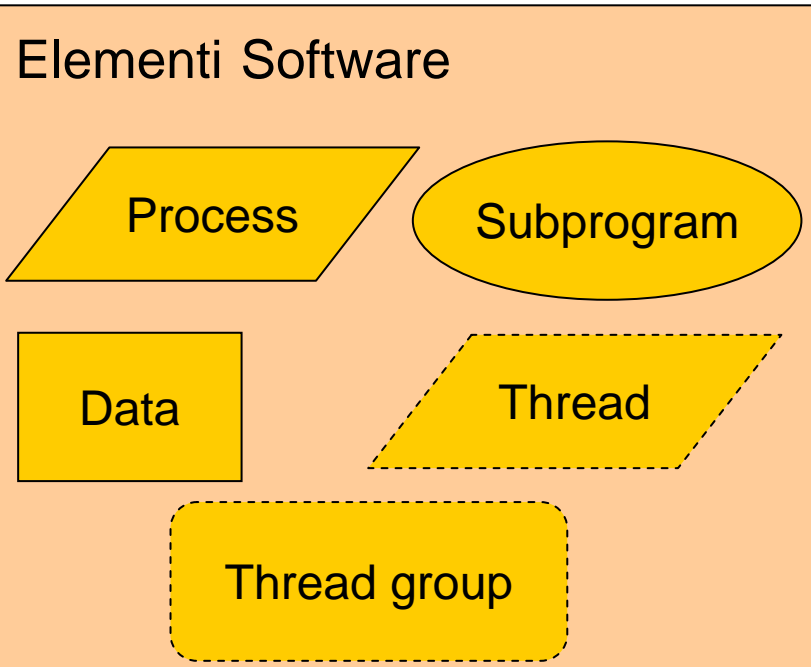
Che cos'è AADL

- L'AADL è prima di tutto un linguaggio descrittivo di tipo testuale;
- Lo standard nella versione 1.0 contiene anche una notazione grafica per facilitare la comprensione delle architetture descritte da un testo AADL
- La specifica di un profilo UML è tutt'ora in fase di approvazione
- La descrizione dell'architettura di un sistema con AADL consiste nella descrizione dei suoi componenti e della loro organizzazione in una struttura ad albero

Concetto di componente in AADL

- I componenti sono gli elementi fondamentali delle architetture AADL
- Essi appartengono a 10 categorie predefinite e possono essere:
 - Composti
 - Organizzati gerarchicamente
 - Interconnessi
- Le loro implementazioni possono contenere:
 - Subcomponenti
 - Proprietá
 - Features

Categorie di componenti

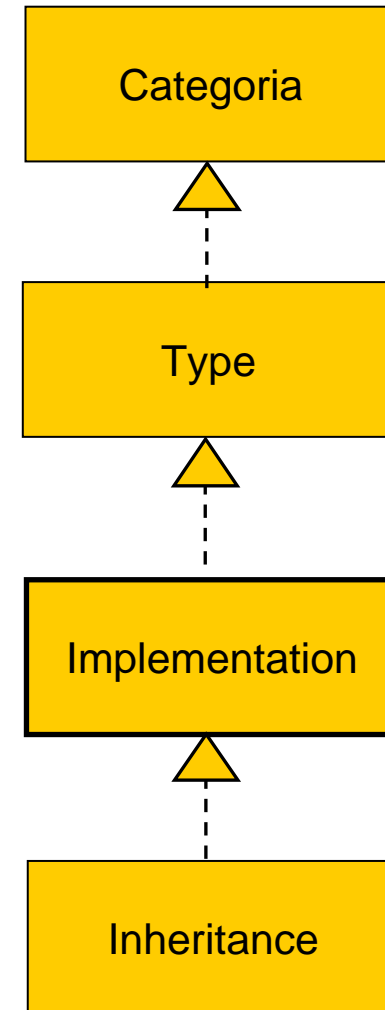


Tipi e implementazioni

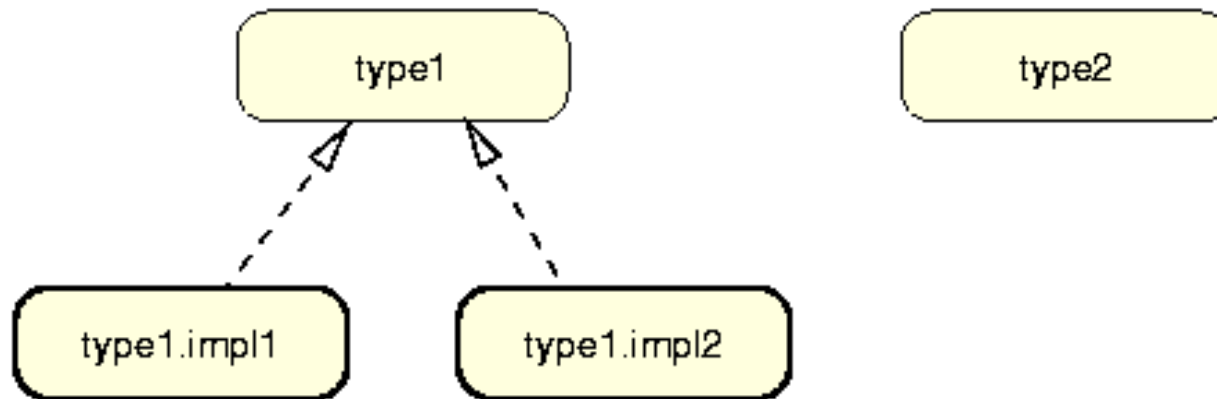
- Ciascun componente di un sistema é descritto in AADL in due fasi:
 - *Type*: rappresenta l'interfaccia funzionale del componente che é visibile agli altri componenti
 - *Implementation*: descrive il contenuto del componente (subcomponenti, proprietá, coneessioni, etc.)
- Tale divisione fa sí che in AADL sia possibile separare nettamente la descrizione del componente visto da fuori o da dentro. Ciò fa sí che la descrizione del tipo o della implementazione possa essere fatta in tempi diversi o da persone diverse ciascuna presa con diversi problemi a differenti livelli di astrazione.
- Esiste un meccanismo di ereditarietá (inheritance) dei tipi e delle implementazioni. Tale meccanismo é utile per raffinare una descrizione.
 - Un componente puó ereditare solo da un componente della stessa categoria
 - Allo stesso modo, un'implementazione puó ereditare solo da una implementazione della stessa categoria

Descrizione a tre livelli

- Categoria (predefinito)
- Type
 - Specifica l'interfaccia esterna
- Implementation
 - Specifica il contenuto
- Inheritance
 - Eredita un tipo o una implementazione



Esempio di descrizione di componenti



```
system type1
end type1;
```

```
system type2
end type2;
```

```
system implementation type1.impl1
end type1.impl1;
```

```
system implementation type1.impl2
end type1.impl2;
```

Ereditá tra tipi e implementazioni

```
system type1
end type1;
```

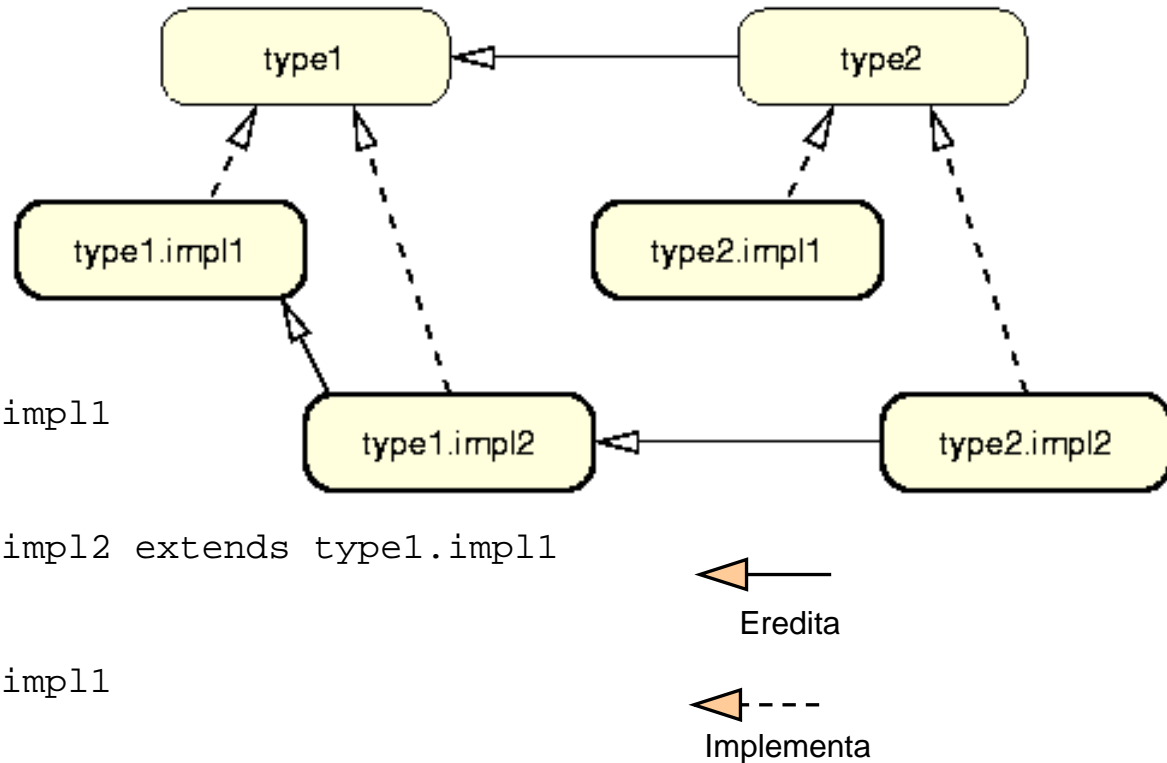
```
system type2 extends type1
end type2;
```

```
system implementation type1.impl1
end type1.impl1;
```

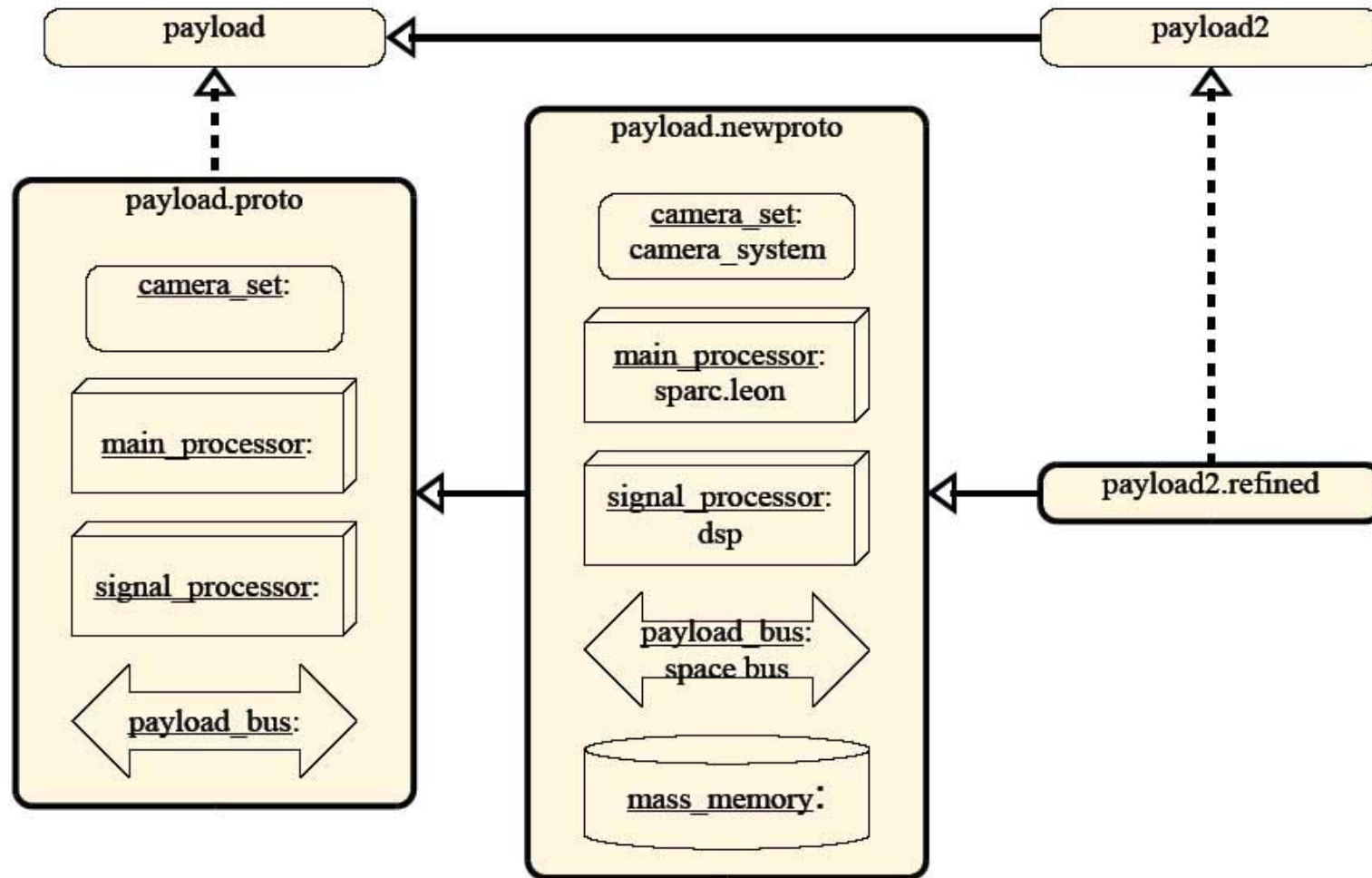
```
system implementation type1.impl2 extends type1.impl1
end type1.impl2;
```

```
system implementation type2.impl1
end type2.impl1;
```

```
system implementation type2.impl2 extends type1.impl2
end type2.impl2;
```



Esempio (1/2)



Esempio (2/2)

```

system payload end payload ;

system implementation payload.proto
  subcomponents
    camera_set : system ;
    main_processor : processor ;
    signal_processor : processor ;
    payload_bus : bus ;
  end payload.proto ;

system implementation payload.newproto
  extends payload.proto
  subcomponents
    camera_set : refined to
      system camera_system ;
    main_processor : refined to
      processor sparc.leon ;
    signal_processor : refined to
      processor dsp ;
    payload_bus : refined to
      bus spacebus ;
    mass_memory : memory ;
  end payload.newproto ;

system camera_system end camera_system;
bus spacebus end spacebus ;
processor dsp end dsp ;

```

```

processor sparc end sparc ;
processor implementation sparc.leon
end sparc.leon ;

property set space_properties is
  mass_t: type aadreal units mass_u ;
  mass_u: type units
    (g, kg => g*1000, t => kg*1000);
  mass_range_t: type range of mass_t;
  allowed_mass: mass_range_t applies to
    (memory, processor, bus, device, system);
  actual_mass: mass_t applies to
    (memory, processor, bus, device, system);
end space_properties ;

system payload2 extends payload
  properties
    space_properties::allowed_mass
      => constant 200.0kg .. 500.0kg ;
  end payload2 ;

system implementation payload2.refined
  extends payload.newproto
  properties
    space_properties::actual_mass
      => 300.0 kg ;
  end payload2.refined ;

```

Concetto di Feature

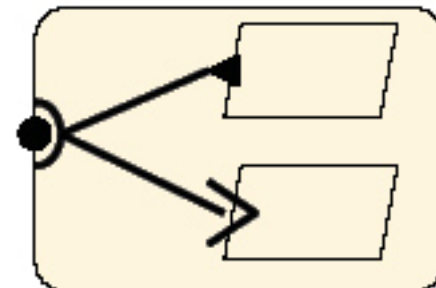
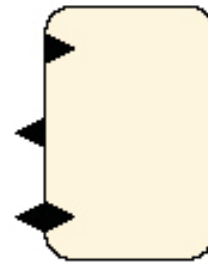
- In AADL una *feature* é una definizione di una parte del tipo di componente che definisce come ciascun componente si interfaccia con gli altri. Ci sono 4 tipi di feature:
 - Port
 - Subprogram
 - Parameter
 - Subcomponent access

Proprietá, porte e connessioni

- Ad ogni componente sono associate delle proprietá e i rispettivi valori
 - Alcune proprietá sono predefinite, ovvero sono fornite dal linguaggio (es. i thread hanno proprietá quali il periodo, la scadenza e il tempo di esecuzione)
 - Altre proprietá possono essere definite dall'utente e associate a (parte dei) componenti dell'AADL
- La descrizione dei flussi di dati e controllo é affidata alle porte e alle connessioni
 - Una porta é un punto di ingresso (uscita) di un componente in (da) cui passano dati o eventi;
 - Una connessione abilita il collegamento tra due porte, tra due subcomponenti o le porte di un subcomponente e il suo componente-genitore
 - Esistono meccanismi di verifica per assicurare la conformitá tra i tipi e direzioni delle connessioni e le porte che essi collegano

Caratteristiche delle porte

- Le porte possono essere di tipo diverso, avere una direzione ed essere raggruppate;
- Tre direzioni:
 - Input port (ingresso)
 - Output port (uscita)
 - Bidirectional port
- Tre tipi di porte
 - Data port
 - Event port
 - Event data port
- Un gruppo di porte raggruppa piú porte di tipo diverso



Proprietá, porte, connessioni

```

system system1
end system1;

system system1.impl
  subcomponents
    p1: process1.impl;
    p2: process2.impl;
  connections
    cn: data port p1.outport -> p2.inport;
end system1.impl;

process process1
  features
    outport: out data port;
end process1;

process process1.impl
  subcomponents
    t1: thread1.impl;
  connections
    cn: data port t1.outport -> outport;
end process1.impl;

process process2
  features
    inport: in data port;
end process2;

```

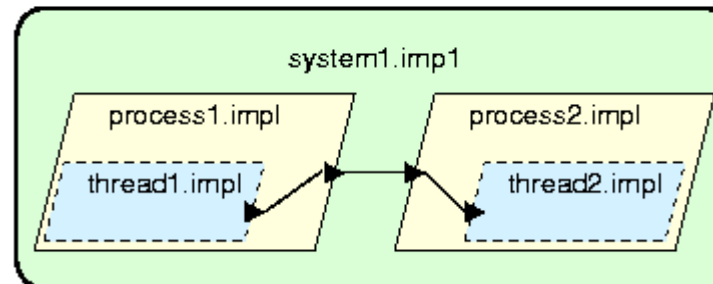
```

process process2.impl
  subcomponents
    t2: thread2.impl;
  connections
    cn: data port inport -> t2.inport;
end process2.impl;

thread thread1
  properties
    Period => 15 ms;
    Deadline => 10 ms;
  features
    outport: out data port;end thread1;
thread thread1.impl
end thread1.impl;

thread thread2
  features
    inport: in data port;end thread2;
thread thread2.impl
end thread2.impl;

```



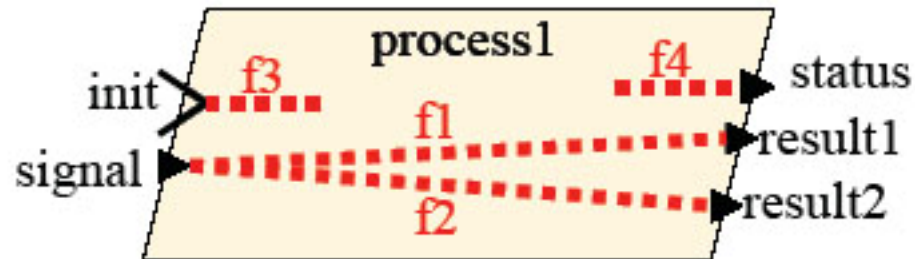
Flussi (flows)

- I flussi possono essere visti come insiemi di connessioni uniti tra loro a formare un percorso per certi tipi di dati/controlli
- La descrizione completa dei flussi comprende:
 - Flow specification
 - Flow implementation
 - End-to-end flow declaration

Specifica dei flussi

- A livello di tipo i flussi vengono specificati in modo da essere visibili all'esterno del componente che attraversano.
 - I flussi possono aver fine in un componente (flow sink)
 - Avere origine da un componente (flow source)
 - Attraversarlo partendo da una porta in o in/out e terminando in una porta out o in/out (flow path)

Specifica dei flussi: esempio



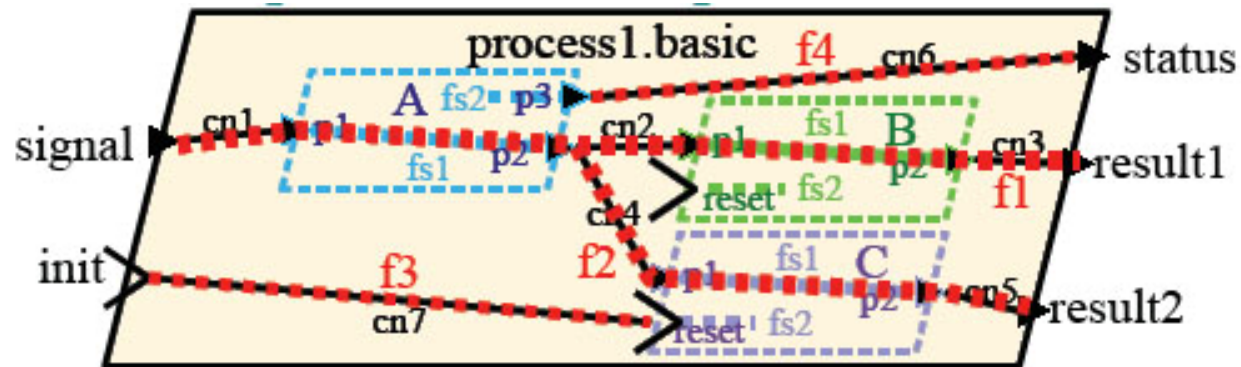
```

process process1
  features
    init: in event port;
    signal: in data port signal_data;
    result1: out data port position.radial;
    result2: out data port position.cartesian;
    status: out data port;
  flows
    f1: flow path signal -> result1;
    f2: flow path signal -> result2;
    f3: flow sink init;
    f4: flow source status;
end process1;
  
```

Flow implementation

- L'implementazione dei flussi descrive *come* è implementata ciascuna specifica dalla porta di ingresso a quella di uscita passando per i vari sottocomponenti che implementano il componente principale
- Per essere completa, l'implementazione di una componente deve provvedere una implementazione per ciascuna specifica di flusso associata al suo tipo
- Un flow *end-to-end* è un flusso che descrive il flusso logico di informazioni tra due subcomponenti (senza raggiungere il bordo del componente principale)

Implementazione dei flussi: esempio



```

process implementation process1.basic
subcomponents
  A: thread t1.basic; B: thread t2.basic; C: thread t2.basic;
connections
  cn1: data port signal -> A.pl;
  cn2: data port A.p2 -> B.pl;
  cn3: data port B.p2 -> result1;
  cn4: data port A.p2 -> C.pl;
  cn5: data port C.p2 -> result2;
  cn6: data port A.p3 -> status;
  cn7: event port init -> C.reset;
flows
  f1: flow path signal->cn1->A.fs1->cn2->B.fs1->cn3->result1;
  f2: flow path signal->cn1->A.fs1->cn4->C.fs1->cn5->result2;
  f3: flow sink init->cn7->C.fs2;
  f4: flow source A.fs2->cn6->status;
end process1.basic;

```

Modi

- AADL prevede la possibilità di specificare modi di funzionamento per i componenti del sistema
- In particolare, un componente può avere:
 - Valori delle sue proprietà dipendenti dal modo di funzionamento
 - Configurazioni dei subcomponenti e dei flussi interni dipendenti dai modi
- Naturalmente è possibile modellare la transizione tra due modi e il comportamento del sistema all'atto della transizione
- Ad ogni istante solo uno può essere il modo attivo.

Caso di studio

DOMANDE?

Piergiorgio Di Giacomo
R&D Area
Software Engineer
Piergiorgio.DiGiacomo@esi.es

Parque Tecnológico, # 204
E-48170 Zamudio
Bizkaia (Spain)
Tel.: +34 94 420 95 19
Fax: +34 94 420 94 20
www.esi.es