A Robotics Task Coordination Case Study

Davide Brugali, Luca Gherardi, Patrizia Scandurra

May 9, 2011 - Shanghai, China
Outline

- Introduction
- Separation of concerns: computation and coordination
- The case study
- High level solution
- Coordination FSM
- Implementation
Introduction

Goal: separate of concerns in the development of component based systems (in our case particular attention on Computation and Coordination).

Solution: model and implement Computation and the Coordination through 2 different frameworks (SCA and ASM).
Separation of concerns

- **Computation**: concerned with data processing algorithms required by an application.

- **Configuration**: determines which system components should exist, and how they are inter-connected.

- **Communication**: deals with the exchange of data.

- **Coordination**: concerned with the interaction of the various system components.
Computation Vs. Coordination

✿ Computation: **how** the functionalities are realized.

✿ Algorithms: data structures and operations.

✿ Coordination: **when** the functionalities are used.

✿ Interaction among components: cooperation and competition.

✿ They have to be modeled and implemented in a separate way.

✿ We chose two different frameworks.
Computation and SCA

We adopted the Service Component Architecture.

Set of specifications which describe a model for building applications.
Coordination and ASM

- We adopted the Abstract State Machines
- Evolution of the FSM
  - Formal Method
    - Automatic validation and verification of correctness.
    - Runtime monitoring and self-adaptation.
  - Provides rigor without mathematical overkill.
**SCA - ASM**

- Developed by P. Scandurra and F. Albani at University of Bergamo.

- Extension of the SCA Eclipse plugin (front-end).
- Extension of the Tuscany runtime (back-end).
The case study scenario

- Three participants:
  - Laser scanner: executes scans (n measures) on demand and writes results on a data buffer: scan(from, nSteps).
  - 3D Perception and Obstacle Avoidance applications.
The case study requirements

- A scan operation requires an amount of time that is not fixed.
- A client could request single and multiple scans.
- A scan request could be received while the scanner is already scanning.
  - Policy 1: discard requests.
  - Policy 2: queue requests.
Three possible situations

- Situation A: synchronous request
Three possible situations

Situation B: asynchronous request
Three possible situations

- Situation C: two or more asynchronous request

![Diagram showing Client 1, Client 2, and Laser Scanner]
High level solution A

* Applicable for situations A and B - Direct interaction
High level solution B

* Applicable for situation C - Direct interaction
Sensor Coordinator FSM
Policy 1

scan(from, nSteps, nScans)

IDLE

"Done"

&&

remScans = 0

BUSY

"Ack"

"Done"

&&

remScans > 0

SCANNING

lunedì 9 maggio 2011
Sensor Coordinator FSM
Policy 2

Scan (from, nSteps, nScans)

IDLE

"Ack"

"Done" AND remScans = 0 AND pendingReq = 0

SCANNING

("Done" AND remScans > 0)

("Done" AND remScans = 0

AND pendingReq > 0)

BUSY

("Done" AND remScans > 0)

OR

("Done" AND remScans = 0

AND pendingReq > 0)
The implementation

request\text{(from, nSteps, nScans)}

scan\text{(from, nSteps)}

update\text{(eventName)}
The Sensor Coordinator ASM Policy 1

//Rule invoked for the startup of the components main agent
rule r_init($a in SensorCoordinating) = //to initialize the components state
    par
    status($a) := READY
    ctl_state($a) := IDLE
    from($a) := 0
    steps($a) := 0
    remScans($a) := 0
endpar

//Main agent's program
rule r_SensorCoordinator =
    let($r = nextRequest(self)) //Select the next request(if any)
    in
        if isDef($r)
            then r_acceptRequest[self,$r] //Handle the request $r
        endif
endlet
The Sensor Coordinator ASM
Policy 1

```
1 rule r_acceptRequest ($a in Agent, $r in String) =
2   if (ctl_state($a)=IDLE and $r="r_request(Agent,Integer,Integer,Integer)")
3     then seq
4       //first scan
5       r_wreceive[clientSensorCoordinating($a),"r_request(Agent,
6           Integer,Integer,Integer)",paramScan($a)]
7       if (isDef(paramScan($a)))
8         then
9           r_request[$a,first(paramScan($a)),second(paramScan($a)),
10          third(paramScan($a))]
11       endif
12     endseq
13   else if (not ctl_state($a)=IDLE and $r="r_update(Agent,String)")
14     then seq
15       r_wreceive[clientEventObserving($a),"r_update(Agent,String)",
16           event($a)]
17       if (isDef(event($a)))
18         then r_update[self,event($a)]
19       endif
20     endseq
21   endif
```
The Sensor Coordinator ASM Policy 1

```java
//@Service
rule r_update($a in Agent, $event in String) =
  if (ctl_state($a)==BUSY and $event=="Ack")
    then ctl_state($a) := SCANNING
  else if (ctl_state($a)==SCANNING and $event=="Done" and remScans($a)>0)
    //continue with next scan
    then par
      ctl_state($a) := BUSY
      remScans($a) := remScans($a)−1
      r_wsend[laserScanning($a),"r_scan(Agent,Integer,Integer)",(from($a),
        steps($a))]
    endpar
  else if (ctl_state($a)==SCANNING and $event=="Done" and remScans($a)=0)
    then ctl_state($a) := IDLE
endif endif endif
```
The Sensor Coordinator ASM
Policy 1

```plaintext
1 definitions: //definitions of named ASM transition rules
2     //@Service
3 rule r_request($a in Agent,$from in Integer,$steps in Integer,
4     $nScans in Integer)=
5     par
6     ctrl_state($a) := BUSY
7     from($a) := $from
8     steps($a) := $steps
9     remScans($a) := $nScans - 1
10    r_wsend[laserScanning($a),"r_scan(Agent,Integer,Integer)",(from, $steps)]
11   endpar
```
Conclusions

- The use of two different frameworks for modeling and implementing computation and coordination:
  - Improves flexibility and reusability: we can modify the implementation without having to change the coordination and vice-versa.
  - We can reuse a simple composite into a bigger composite (hierarchical containment).
Conclusions

- The use of two different frameworks for modeling and implementing computation and coordination:
  - Improves flexibility and reusability: we can modify the implementation without having to change the coordination and vice-versa.
  - We can reuse a simple composite into a bigger composite (hierarchical containment).
- SCA and ASM promise good results also in robotics domain !!!
Thank you for your attention !!

* The research leading to the results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).