Software for Complex Robotics Systems

—

The Composition Pattern

Herman Bruyninckx
Eindhoven University of Technology
KU Leuven
http://people.mech.kuleuven.be/~bruyninc/

April 29, 2015

Overview of this lecture

- *design* = first **modelling**, then implementation.
  This lecture is about modelling of **system of systems**.
- systems have "Structure", "Behaviour" & "Activity"
- model for *Structure* = **Composition Pattern**
  model for *Behaviour* = *Task-Skill-Motion*
  model for *Activity* = *Port-based containers*
- to develop functionality = **to decouple** according to *"5Cs"*:
  Computation, Communication, Coordination, Configuration, Composition
- to develop systems = **to couple** functionalities:
  - map *Task-Skill-Motion* on *Composition Pattern* (architecture)
  - map *Composition Pattern* on *operating system* (deployment)

Structure, Behaviour & Activity

Behaviour: *"reacting to stimuli"*

- as seen from the "outside"
- software systems: interact via events and data
- hardware systems: interact mechanically, electrically,. . .

Activity: *"executing the code"*

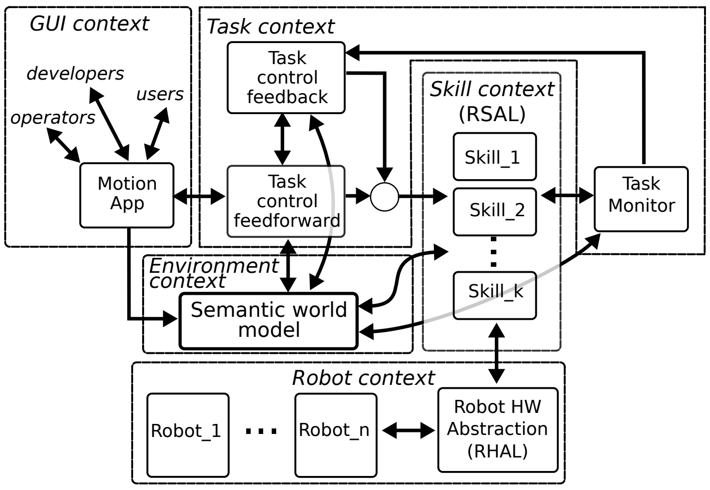
- how behaviour is realised "internally"
- software: CPU + RAM + bus
- hardware: mechanical, electrical,. . . , impedance

Structure:

- *interconnection* of Behaviours and Activities

= *system architecture*

Behavioural model: Task-Skill-Motion



Decouple behaviour: the 5Cs

Where is something computed, and communicated to/from?

Connection	
Coordination	When must components change their behaviour?
Configuration	What parameters define the behaviour of all components?
Computation	What functionality is computed?
Communication	How are results of computations being communicated?

Composition "contains", "connects"	
Coordination	event handling, decision making
Configuration	parameter setting, behaviour monitoring
Computation	data + algorithms + scheduling
Communication	data + patterns + protocols

Compute behaviour: the 5COPs

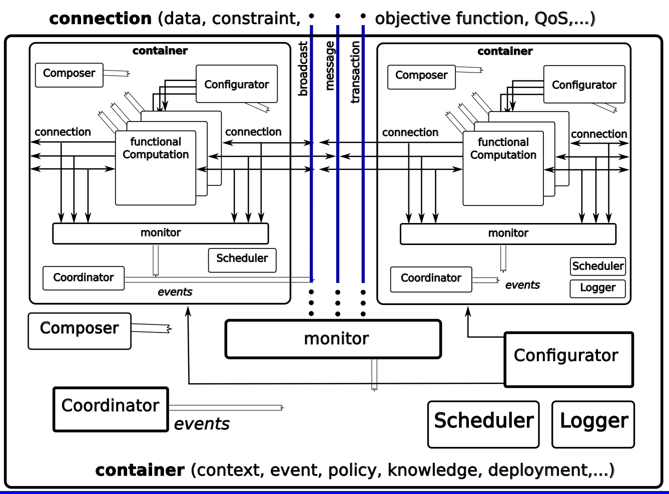
("COP" = *constrained optimization problem*)

Constrained Optimization Problem *FSM, configuration, monitoring, events*	
objective functions	$\sum_i w_i f_i(X,y)$
optimization variables	$\min_X$
constraints	$g(X,y) \leqslant 0$
tolerances	$\mathrm{dist}(X,X_{\min}) \leqslant \delta$

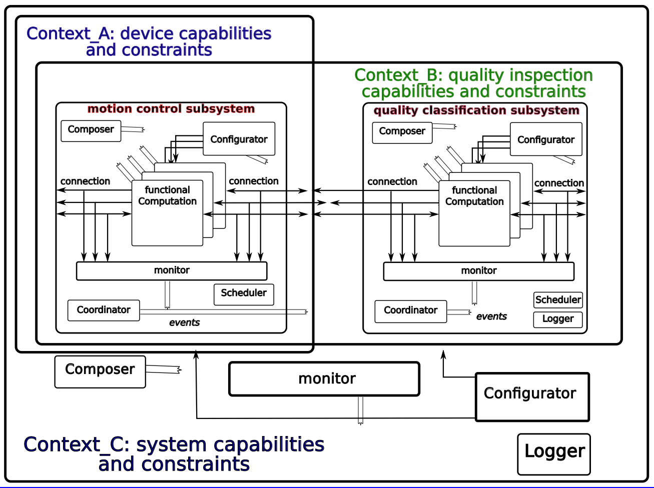
Advantages of a *constrained-based* design:

- composable!
- monitorable!
- tolerant!
- configurable!

Structural model: Composition Pattern (focus on "roles", not on "functionalities"!)

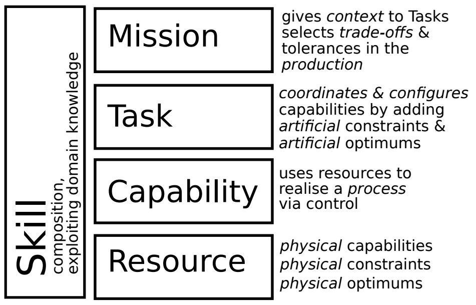


Structural model: Composition Pattern (2) —Where does "knowledge" fit in?—


Context_A: device capabilities and constraints
Context_B: quality inspection capabilities and constraints
motion control subsystem
quality classification subsystem
Composer
Configurator
connection
functional Computation
monitor
Scheduler
Coordinator
events
Logger
Context_C: system capabilities and constraints

Integration of TSM and CP

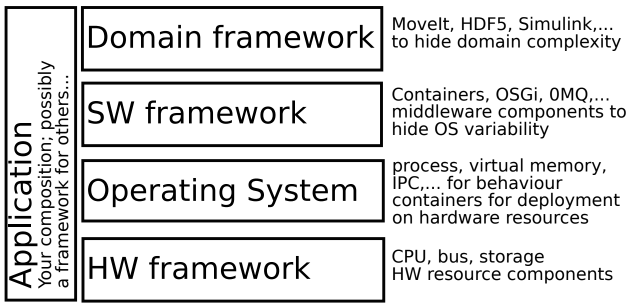
- every Task, every RHAL, every World Model is a *separate* CP.
- every Skill too, but it *couples* the CPs above, at various levels of **hierarchy**:


Skill
composition, exploiting domain knowledge
Mission
gives *context* to Tasks selects *trade-offs* & tolerances in the *production*
Task
coordinates & configures capabilities by adding *artificial* constraints & *artificial* optimums
Capability
uses resources to realise a *process* via control
Resource
physical capabilities *physical* constraints *physical* optimums

⇒ a Skill adds a *knowledge context* to a composition.

- several Skills can be active *at the same time*.

Activity model: port-based interaction


Application
Your composition; possibly a framework for others...
Domain framework
MoveIt, HDF5, Simulink,... to hide domain complexity
SW framework
Containers, OSGi, 0MQ,... middleware components to hide OS variability
Operating System
process, virtual memory, IPC,... for behaviour containers for deployment on hardware resources
HW framework
CPU, bus, storage HW resource components

Dangers of ROS:

- Domain = SW = OS ⇒ component = process = activity!
- Communication = *only* publish-subscribe via TCP/IP!
- Too "fat" components/nodes ⇒ too heavily coupled 5Cs!
- No dynamics or control ⇒ mechatronics "abstracted away"!

Single-threaded execution of Composition Pattern in an Activity

Common (but not absolute!) policy *to serialize* the execution of Behaviour in an Activity as follows:

```
when triggered      % by OS, or other CP
do {
    communicate()   % get latest events
    coordinate()    % react to them
    configure()     % possibly requiring reconfiguration
    schedule()      % now do one's Behaviour
    coordinate()    % execution could trigger new events
    communicate()   % that others might want to know about
    log()
}
```

Conclusions

- **what do all the arrows in (y)our diagrams mean. . . ?**
  - communication via data messages?
  - communication via shared memory?
  - knowledge integration via Configuration?
  - communication via observation?
  - decision making via reasoning?
  - . . .
- **monitor everything you expect to happen**. . .
  . . . **and be ready to react if it doesn't!**
- **skills = monitoring & coordination & configuration**