A Software Infrastructure for Robotic Skill Learning and Cognition

(Or: how to improve your PhD students' software attitude)

Herman Bruyninckx
Dept Mechanical Engineering, KU Leuven, Belgium

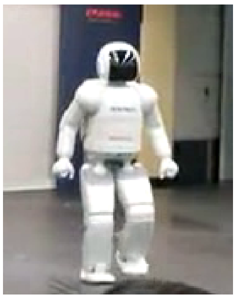
BRICS (*Best Practices in Robotics*)
Rosetta (*RObot control for Skilled ExecuTion of Tasks*)

Lund, April 17, 2012

My background — research interests

- making robots "more intelligent"
  (. . . as all of you do, I guess :-))
- got involved in software engineering aspects about a decade ago, because I got tired of the extreme **lack of reuse** of software modules in robotics.
  (Anyway, that would only take a year or three-four. . .)
- **Reuse**:
  - multi "vendor"
  - multi robot
  - multi framework (ROS, Orocos, OpenRTM, OPRoS,. . .)
  - multi programming language
  - not just code, but also knowledge: **models**!
- **Ambition:** to bring robotics software development to **industry grade** quality levels

Overview — Problem statement



- Cognitive. . . ? . . . rather: smartly written code!
- Learning. . . ? . . . rather: adaptable parameters!

Still **"cognition/learning for robots by humans"**
instead of **"cognition/learning by robots for humans". . .**
⇒ revolution in Software Engineering needed
(. . . among other things!)

Overview take-home messages

- Trend 1: **"Look ma, no hands!"** hackatons
- Trend 2: software made by **unsupervised PhD students**
- Trend 3: **software is law** (*implementation* = semantics)
- Problem 1: required **complexity** is not yet supported
- Problem 2: PhDs with too little **system knowledge**
- Problem 3: abundant **Not Invented Here attitude** in control, learning, planning, modelling, middleware,...
- Problem 4: current trends **not scalable & maintainable**
  - class libraries: **too deep** hierarchies
  - frameworks (Orocos, ROS,...): **too flat** architectures
  - distributed world models: **too few** ontologies
- Need 1: from *"it works!"* to **"it is reusable!"**
- Need 2: change **focus** from *code* to **models/standards**
- Need 3: **more smaller SW modules** needed

"Look ma, no hands" hackatons

Problem"

- **Goal** number one: make it work!
- **Means** number one: agile development...

  ...but the abused version: look into each others' code and adapt, adapt, adapt,..., till it, indeed, works...
- **Team** number one: very homogeneous with shared "hacker" mentality + Sense/Plan/Act decoupling
- Moderns applications: dozens to hundreds of nodes, components, modules! Future applications: thousands!

⇒ "agile" inter-module adaptation won't scale anymore!

"Look ma, no hands" hackatons

Solution: Model-Driven Engineering:

- works in other domains (aerospace, automotive, mechatronics, embedded, medical,...)!
- knowledge is in the model, code is generated
- knowledge brings structure: hierarchy, stable "agent" sub-systems,...
- knowledge brings discipline: reference architectures, standards,...
- domain brings complementary experts together

Problem for robotics: model = closed world assumption
⇒ methodologies required for "opening up" models, on-line

⇒ robotics will (**have to**) drive new ICT paradigms!

Software made by unsupervised PhD students

- Peer review...?
  Seniors read the papers of the juniors, but do they read their code?
  Do they co-design their software architectures?
  Do they make them share data structures!
- Seniors want it "to work": *Just code it...*
- Macho attitude: "real men write code not documentation"
  (One of the many open source myths, sigh.)
- PhDs optimize their incremental progress, not others' long-term maintainable solutions

Comparison/Solution: typical coding team in aerospace = 3-5 people, average age 50+, average lines-of-code-a-day 3–5, code-by-model,...

Software is law

Problem:

- **no** robotics software modules exist whose behaviour/semantics can be fully predicted by information in documentation/model
- **instead**: one has to execute and observe

Solution: systematic introduction of semantic models, "*also known as ontologies*":

- common sense & physics
- robot system architecture = interactions between planning, sensing, control, world modelling,...
- tasks, affordances, perception networks,...

Class Libraries: too deep hierarchies

Problem illustrated by means of inverse dynamics class:

- v1: Newton-Euler, by inward/outward "sweeps" over kinematic chain with ideal 1DOF joints:
  `tau = ID_NE (q, qdot, F)`
- v2: what about posture control?
  `tau = ID_NE_PC (q, qdot, F,tau_p)`
- v3: what about damped least-squares singularity robustness?
  `tau = ID_NE_PC_DLS (q, qdot, F,tau_p,lambda)`
- v4: what about joint limit avoidance?
  `tau = ID_NE_PC_DLS_JL (q, qdot, F,tau_p,lambda,K)`
- v5: what about N-DOF joints? mobile platforms? configuration of all parameters?...

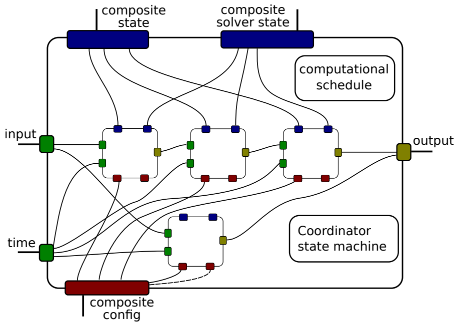
Severe open source **"vendor" lock-in**!

Class Libraries: too deep hierarchies (2)

Solution: introduce data flow computational composites

- refactor libraries in functions that are atomic & composable
- tooling to embed them in port-based components. . .

. . . with (runtime!) configurable computational scheduling depending on triggered ports ($\gg$ Simulink!)



- Finite State Machines for life cycle and inter-composite coordination

(No software framework already provides all of this.)

Not Invented Here syndrom

Problem: *"I have my hammer and can hit all nails!"*

- planners: *"faster planning avoids control"*,. . .
- perception: *"SLAM avoids planning and control"*,. . .
- control: *"ILC is control"*, *"MPC is control"*,. . .
- learning: *"reinforcement learning avoids modelling"*,. . .
- **all** of them: world model is **hidden inside**!

$\Rightarrow$ close to no real "multi-vendor" integration taking place

- communication middleware:

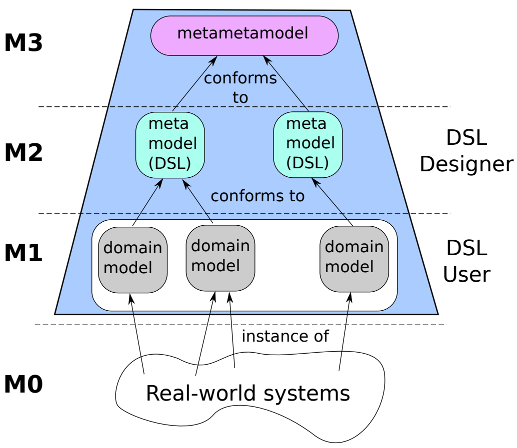
  where was robotics the **last 20 years**. . . ?!?!

Solution:

- **system**-level education (. . . also in seniors!)
- refactoring code to atomic libraries & components:

  $\rightarrow$ *"do one thing only, and do it perfect!"*

BRICS/Rosetta: MDE in robotics



Self-awareness models: key to cognition & learning **by** robots!

Model-Driven Engineering (2)

M3 = **m**eta-**m**eta **m**odel (no domain-knowledge!)
M2 = **m**eta **m**odel (Domain Specific Language)
M1 = **m**odel (domain model encoded in DSL)
M0 = implementation
(in specific programming language(s))

- **M3–M0**: is an ontology, i.e., a formal representation of knowledge about a domain!

⇒ necessary for giving our robots the ability to interpret their own actions, and their interactions with the world.

BRICS: "5C" separation of concerns

Composition (How are computations, communications, configurations, and coordinations interacting?)	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?

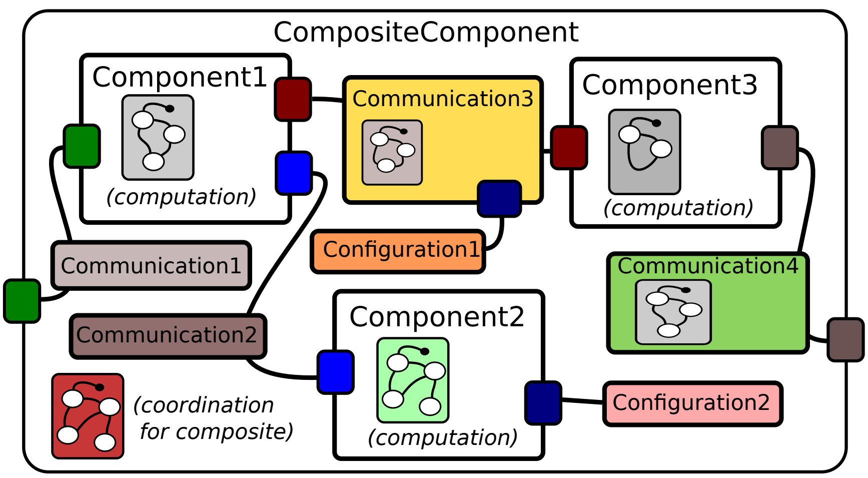
"4Cs"[1] for **decoupling**, "Composition" for **coupling**

5C "meta-model" helps to separate

- **framework** (Orocos, ROS. . .) from **functional code**.
- **elementary types** of functionality from each other.

[1]Thanks to Klas Nilsson, Lund, for introducing me to this concept!

Example of a "5C" architecture



Added value in "5C": Coordination & Configuration

Our spin-off `Intermodalics.eu` makes 90% of its money by adding 5% C&C code to available open source software, in context of advanced industrial integration projects...
Only possible when other functionalities are nicely separated in the software repositories

Rosetta: Task-Skill-Motion

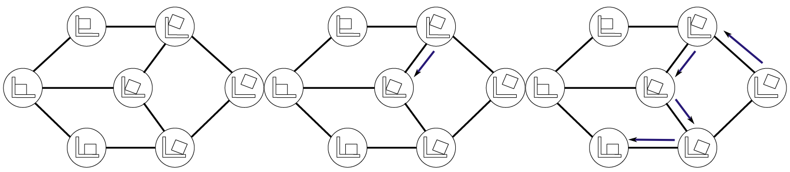
MDE models of "**platforms**":

- Task: specification with only manipulated **object** properties
- Skill: adds **type** of robots, sensors, sensor processing, trajectory generation,...
- Motion: adds properties of **concrete** robot, sensor, sensor processing,...

Data structures:

- Constraint Graph:
  node = {constraints on objects}, edge = context
- Scene Graph: node = object, edge = position
- Skill Graph: node = "control", edge = transition

TSM — Dualities Contact–Skill–Constraint Graphs



Constraint Graph   + Motion   + Skill

Under uncertainty:

- one Motion can lead to multiple Constraints
- transitions in Skill Graph get probabilistic

Typically in current software:

- Constraint Graph is not made explicit!

⇒ although necessary for online reasoning...

- Scene Graph: many projects need such software...

Conclusions

- Robotics is going to lead the SW engineering field, because of the incomparable challenges of cognitive learning robots
- we are still putting the intelligence in the programmers, not in the robots
- revolution needed in **attitudes**: reusability & **modelling**
- revolution needed in **standardization**!
- revolution needed in **tooling**!

⇒ *"Coders of the world, unite!"* (and not (just) your code. . .)

Stop the hackatons!
Start the knowledge modelling!