ATLAS

Introduction

Best Practices in integrating complex robotic systems

Gianni Borghesan
KU Leuven
Febraury 2019

Outline

Practical infos

Where

KU Leuven

When

24-28 of February 2020

Who

- Gianni Borghesan (KUL)
- Diego Dall' Alba (UNIVR)
- Albert Hernansanz (UPC)

Preparation

- A PC (also VM) with Ubuntu 16 + Ros Kinetic + Orocos
- Group: General concept of the system
- Individual: **pitch presentation** a description of the module that you would integrate (behaviour, life-cycle, data exchanged) – after this presentation.

Schedule

Monday to Thursday

- Two morning sessions of 1.5 hours (9:30-11:00, 11:15 - 12:45)
- Afternoon: practical sessions & Integration (2PM - 5:30PM or later)

Friday

- Morning - Integration
- Afternoon - Evaluation (close around 4PM)
- Happy Hour

Schedule

S#	Title	Description	Who
1	Introduction	Presentation of the course, Overview of the integration process, ESR presentations,	GB, All
2	Modular integration	Modular integration, Review of modules and group definition	HA
A	ROS +Practical	ROS intro, ROS hands-on 1	DDA
3	Real Time	RT systems, Scheduling, Link to Control	GB
4	Middleware 2	OROCOS (with ROS Integration)	GB
B	Practical	OROCOS hands-on, Integration.	
5	Best practices in programming	Programming in Safety-Critical systems, revisioning.	AH

GB: Gianni Borghesan (KUL). **DDA**: Diego Dall' Alba (UNIVR), **AH**: Albert Hernansanz (UPC)

Schedule

S#	Title	Description	Who
6	Data Visualization	Data recording and display.	DDA GB
C	Practical	ROS hands-on 2, teleop. example, integration.	AH
7	Visualization	Data Visualization in medical applications, sensor registering.	DDA
8	HRI	how to convey information, haptic feedback teleop, virtual and augmented reality.	AH
D	Practical	Integration	
E	Practical	Integration	
F	Evaluation	Evaluation and Feedback	

GB: Gianni Borghesan (KUL), **DDA**: Diego Dall' Alba (UNIVR), **AH**: Albert Hernansanz (UPC)

Goal of the week

Provide the fundamental notions of:

- How to approach integration
- Software integration of complex system, consisting of:
  - Hardware interfaces,
  - (RT) control,
  - Data treatment (classification, reconstruction),
  - Decision-making systems,
  - User Interfaces.
- Tools and Libraries for robotic software development.
- Software development management.
- Real-Time and scheduling.

Outline

Scenario example - the eye surgery system

Goal

Enable the cannulation of retinal veins, and inject a drug via a $30\,\mu\mathrm{m}$ O.D. needle

Issues

- Tremor of hand
- Eye ball movement due to force on sclera
- Long infusion time (5 to 30 min)
- poor depth perception with the stereo microscope
- double puncture: injection in the retina

Eye surgery Robotic System

Scenario example - the eye surgery system

Desired features - Mechanical

- Comanipulation system
- Tremor filtering - variable damping
- (Adjustable) Remote Center of Motion
- Locking system

Scenario example - the eye surgery system

Desired features - Augmented perception

- Distance measurement retina/needle $\rightarrow$ auditory clues
- Puncture detection $\rightarrow$ auditory clues
- Vein detection $\rightarrow$ augment image.

Scenario example - the eye surgery system

Hardware components

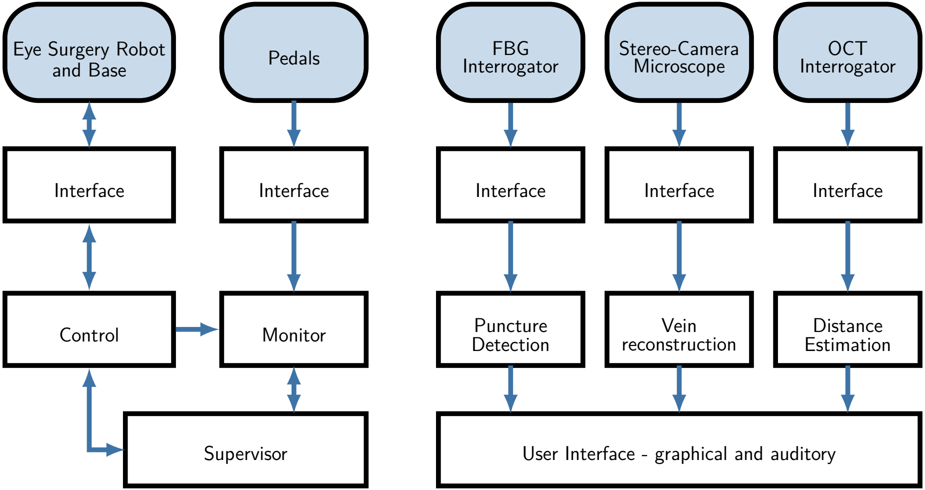
- Robot, Base for moving the RCM, Motor controllers, Pedals - on Ethercat
- Cameras on microscope with USB frame-grabber
- OCT interrogator (distance meas.)
- FBG interrogator (force meas.)
- Needle with OCT fiber and FBG integrated
- PCs

Scenario example - the eye surgery system

Software components

- Cameras interface
- OCT interface
- Robot and Pedals interface
- FBG interface
- GUI
- Auditory cues
- Image processing
- Distance estimation via OCT data.
- Robot control (different control modes)
- Puncture detection via FBG
- Supervisor(s)

Structure of the system



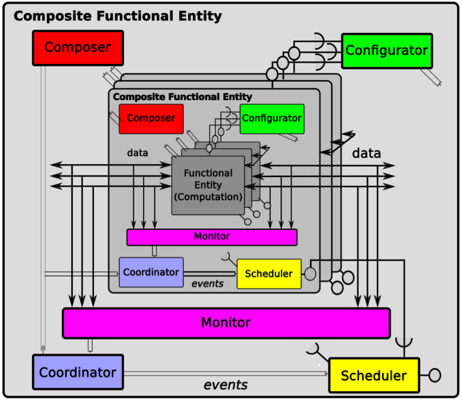
Identify the characteristics

- Which are the requirements of each part?
- Which data are required/provided, and how?
- Which are the steps to bring the system up?
- Is there a specific schedule?
- Is each system reliable? is it possible to measure?

Identify the concerns

A system can be roughly divided into 5 operations, [2]

- communication,
- computation,
- coordination,
- configuration, and
- composition.



from [10]

KU LEUVEN ATLAS

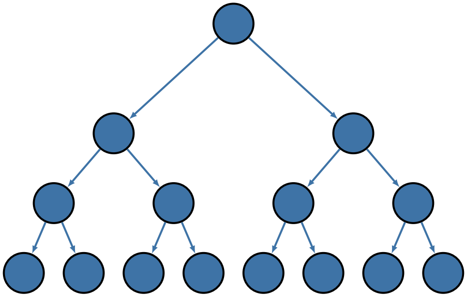
Outline

Top-Down system decomposition

The system should be decomposed iteratively in functional units

Eye Surgery system:

- Robotic system
  - robot interface
  - robot control
    - Locking (position) control
    - Tremor filtering (damping) control
  - base control
- ...



Top-Down system decomposition

Each system has one or more **functionalities**

Algorithms that compute:

- Kinematics
- Control actions
- Estimate of states of the environment
- ...

Top-Down system decomposition

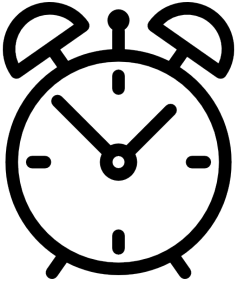
Each system/functionality has a **context** that

- Define the data that are "hidden" in the component
  - Configuration Parameters
  - States
  - ...
- Define the data that are exposed
- Define the data that are needed

Top-Down system decomposition

Each system/functionality needs to be triggered:

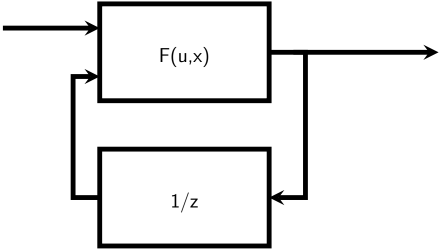
- Periodically?
- When new data arrives?
- Whenever there is time?



Example – Dynamical System

Computation

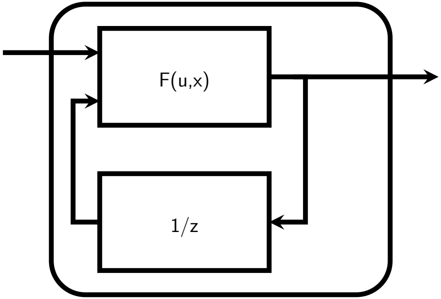
Algorithms implemented in some language.
Each algorithm is the collection and interconnection of general-purpose functionalities/libraries.



Example – Dynamical System

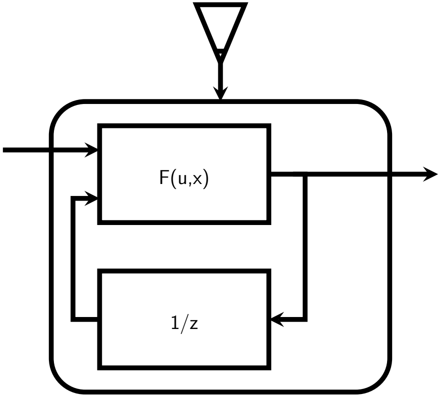
Context

Each algorithm needs data input and data output, plus persistent data (states). The context defines which is the boundary of these data.
Context enables modularity.



KU LEUVEN ATLAS

Example – Dynamical System



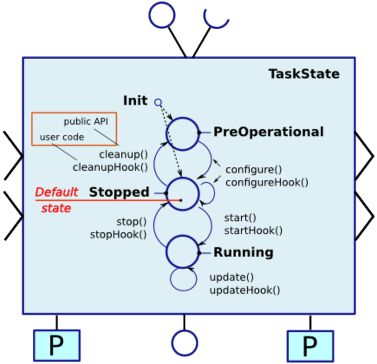
Activities

When these functions are called?

Top-Down system decomposition

Each system has (common) **Life Cycle**

- Operations that are needed to start-up.
- Different states that system should transition to.
- Conditions that trigger transitions.



Top-Down system decomposition

Each system can have a **specific Life Cycle**

Example: MaxPos Maxon ethercat motor driver:

- It has the states of each ethercat slave.
- It has its own states (in figure).

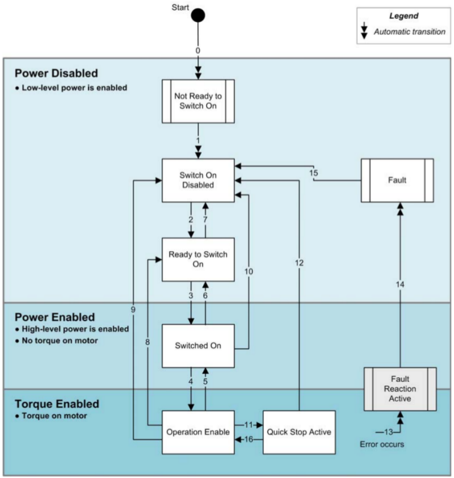


Figure 3-3 Device State Machine

Constraints of the subsystem

Each subsystem has specific requirements

- Input of data
- Timing of inputs
- State of the other components
- ...

Plan the work

Provide documentation

Rule out ambiguities

Iterate until consensus

Plan the work

Provide documentation

- interfaces
- offered capabilities
- required capabilities
- (risks)

Rule out ambiguities

Iterate until consensus

Plan the work

Provide documentation

Rule out ambiguities

- units,
- order,
- definitions,
- ...

$\rightarrow$ agree on a model.

Iterate until consensus

Plan the work

Provide documentation

Rule out ambiguities

Iterate until consensus

Verification that all the pieces works (in line of principle)

Recap - system decomposition

So far:

We agreed and documented

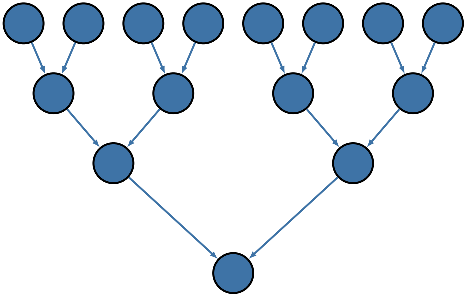
- functional decomposition in sub-systems, recursively
- breakdown of life cycles
- breakdown of requirements and capabilities

Bottom-Up system composition

To do

for each level we:

- Implement (following the specification)
  - Connect ?
  - Synchronise ?
- Verify (vs. specification) possibly with unit testing



One step back to the 5Cs (before implementation)

Communication

- Frequency ?
- Real time ?
- Size/Buffer ?

Coordination

- Synchronize
- Scheduling

Configuration

- Offline
- Online

Computation

- Real time?

Composition

One step back to the 5Cs (before implementation)

Communication

- Frequency ?
- Real time ?
- Size/Buffer ?

Coordination

- Synchronize
- Scheduling

Configuration

- Offline
- Online

Computation

- Real time?

Composition

Some operations that are common to all the subsystems

Re-use of same libraries/infrastructure!

Introspection

Introspection is the ability of a (sub-)system to make explicit its state.

- the life cycle is the first approximation
- Systems may have more complex/nested state $\rightarrow$ **Explicit Modelling**
- Systems should make aware other systems of their state for
  - logging and
  - feedback on coordination, fault recovery.

Monitoring

Monitors can evaluate:

- The state of world - from continuous domain to symbolic information
- The state of systems
  - watch-dog
  - compare output with expected output

Choice of tool(s)

Depending by the characteristics of each of the **5C**, each part of the subsystem can rely on different tools, mainly offered by:

- Language
- Library

Choice of tool(s)

Language

- Low-level, strongly-typed languages offers deterministic performances.
- Scripting languages are normally good for deployment (e.g. Lua) or non-realtime tasks (e.g. Python).
- Some languages are better supported by libraries (e.g. Python) or for specific tasks.

Choice of tool(s)

Libraries - computation

Stand-alone libraries (that depends also by language) Some examples:

- Kinematics: KDL, expressionGraph, ...
- Math/Albegra: Eigen, LAPACK, numpy, ...
- Visualization: Qt, pyside, pyqtgraph, ...
- Optmization: qpOASES, Casadi, ...

Important!

Choose wisely, because these are strong dependencies...

Choice of tool(s)

Libraries - communication, synchronisation, configuration, deployment.

This is typically a layer offered by a middle-ware. Some examples:

- ROS, ROS 2 [9]
- Orocos [1]
- Yarp [3]
- Miro [8]
- Taste [6, 7]
- MicroBLX [4]

Inter-process communication middle-ware that can be applied to robotics:

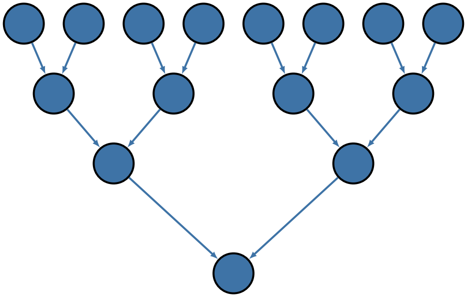
- ZeroMQ,
- Corba

Refer to [5] for more example.

Implementation

Algorithm

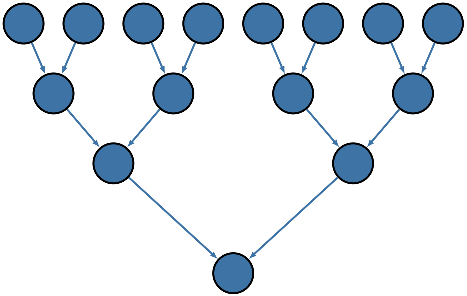
- **Independent** – no unnecessary dependencies with e.g. the middle-ware
- **Isolated** – It does a single thing, limit side effects
- **Testable** – make unit tests, if possible.
- **Documented** !



Implementation

Composition of Algorithms

- Does the algorithm has a life cycle ?
- Is it shared with others ?
- Data flow follows a strict causality ?
- Coordination ?



Division (or not) into activities and contexts !

Implementation

Composition of Algorithms – Example

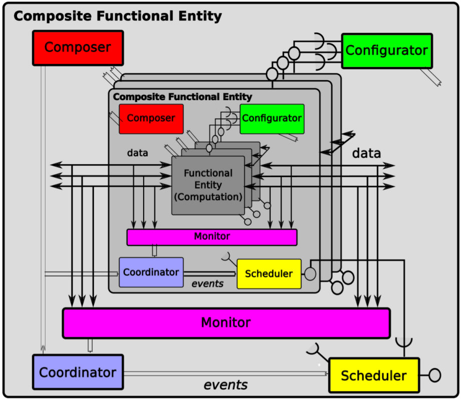
1. HW interface of robot – data in
2. Forward kinematics
3. Control action in user (Cartesian) space
4. Inverse/Transpose Differential kinematics
5. Control action in joint space
6. Safety control
7. HW interface of robot – data out

Implementation

Composition of Algorithms – Result

From outside, is a system with

- a functionality
- a life cycle
- a data flow
- a “behavioural” interface
- a trigger
- a configuration interface



from [10]

Composition of systems

Systems can be deployed

- in the same process
- in the same machine (IPC will be needed)
- in different machines

Issues of composition: communication and coordination

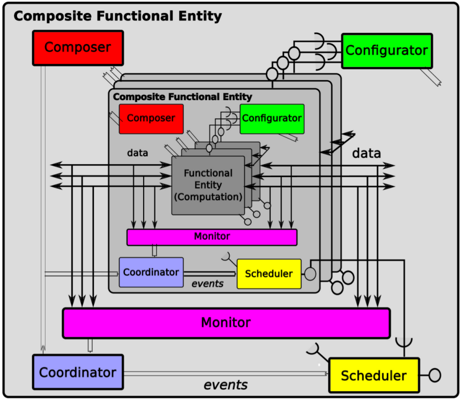
- Data sharing mechanism: latency, bandwidth, jitter, RT
- Explicit scheduling
- Additional coordination

Composition of systems

Implementation – Ideally

From outside, is a system with

- a functionality - given by other system
- a life cycle
- a data flow
- a “behavioural” interface
- a trigger
- a configuration interface



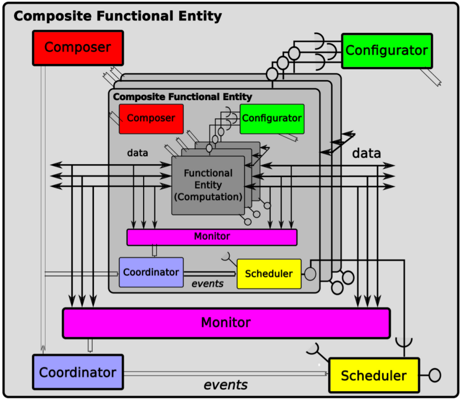
from [10]

Composition of systems

Implementation – In reality

Some aspects lack support for scaling

- behaviour are integrated
- communication is flat
- synchronization is flat
- introspection tooling is very limited
- ...



from [10]

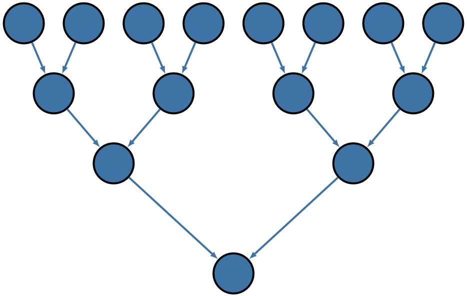
Verification

After each composition

- Verify (vs. specification) possibly with unit testing
- Document

if verification fails:

- Correct, or
- Relax specification, if possible.



Questions ?

Groups

Bibliography I

[1] H. Bruyninckx. “Open robot control software: the OROCOS project”. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 3. May 2001, 2523–2528 vol.3. DOI: 10.1109/ROBOT.2001.933002.

[2] Herman Bruyninckx et al. “The BRICS Component Model: A Model-Based Development Paradigm for Complex Robotics Software Systems”. In: Mar. 2013, pp. 1758–1764. ISBN: 9781450316569. DOI: 10.1145/2480362.2480693.

[3] Paul Fitzpatrick et al. “A middle way for robotics middleware”. In: *Journal of Software Engineering for Robotics* 5.2 (2014).

Bibliography II

[4] markus Klotzbücher. *microblx: real-time, embedded, reflective function blocks*. https://github.com/kmarkus/microblx.

[5] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. "Middleware for Robotics: A Survey". In: *2008 IEEE Conference on Robotics, Automation and Mechatronics*. Sept. 2008, pp. 736–742. DOI: 10.1109/RAMECH.2008.4681485.

[6] M. Perrotin et al. "Taste: A real-time software engineering tool-chain overview, status, and future". In: *SDL Forum*. 2011.

[7] *TASTE*. http://taste.tuxfamily.org. online - visited October 2019. 2019.

[8] H. Utz et al. "Miro - middleware for mobile robot applications". In: *IEEE Transactions on Robotics and Automation* 18.4 (Aug. 2002), pp. 493–497. DOI: 10.1109/TRA.2002.802930.

Bibliography III

[9] V.A. *Robot Operating System (ROS): The Complete Reference (Volume 3)*. Ed. by Anis Koubaa. Vol. 778. Studies in Computational Intelligence. Cham: Springer, 2018. ISBN: 978-3-319-91589-0. DOI: 10.1007/978-3-319-91590-6.

[10] Dominick Vanthienen, Markus Klotzbücher, and Herman Bruyninckx. “The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming”. eng. In: *JOSER: Journal of Software Engineering for Robotics* 5.1 (2014), pp. 17–35. ISSN: 2035-3928.