FEEDBACK: CAPELLA IN TEACHING

Emmanuel Grolleau
ISAE-ENSMA
LIAS – Laboratoire d’Informatique et d’Automatique pour les Systèmes
Context
Ecole Nationale de Mécanique et d’Aérotechnique
Engineering in aeronautics, spatial, transportation, energy, environment

- School in 3 years recruiting mainly in CCP (MP, PC, TSI, PT)
- ~200 students per promotion
- Strong research basis (P’ and LIAS)
  - R&D oriented training
- Mainly common core in year 1 and year 2
- 6 possible specializations in year 3
  - Aerodynamics, Energetics, Thermics, Structures, Advanced materials, Computer science for avionics
Common core around computer science in Y1 and Y2

- Programming (Ada in Semester 1)
- Architecture and operating systems (POSIX C, LabVIEW in S1)
- Introduction to data engineering (SQL in S2)
- Numerical methods (FORTRAN, Python in S2&S3)
- Signal processing (LabVIEW, Matlab in S2)
- Control theory (LabVIEW, Matlab in S3)
- **Embedded systems** (C, LabVIEW, Ada in S4)
  - Choice of year 3 specialization

- Yearly, 25 to 30 students choose to specialize in CS for avionics
Embedded Systems class

- S4 common core ISAE-ENSMA
- Exported in Masters of CS in Univ. Poitiers
- Exported in common core SIAE Tianjin
- Exported in polytechnics universities of Hanoï and HCM City (Programme Francophone d’Ingénieurs d’Excellence au Vietnam)
- Exported in CNAM ingénieurs aéronautique et spatial by apprenticeship
- Very frequent discussions and exchanges with J. Hugues (ISAE)

2005

2015

2017
Audiences in 2016-17

- Mechanical engineering: ~200 students
- Apprenticeship: 17 students
- Computer Science: 15 students
- Capella in Emb. Sys.
Taught concepts
Methodology

- Main goal
  - Design a software system executing on off-the-shelf hardware component(s) meeting functional and non-functional requirements

- Means
  - Functional hierarchical decomposition
    - Mixing non-formal and formal parts
    - Mainly dataflow-like diagrams
  - Formal parts based on finite automata (introduction to model checking)
  - Mapping of functions to threads
    - Trade-off resource consumption vs. end-to-end delays
  - In this class, unicore systems (multi/many-core addressed in advanced embedded systems class)
History : 1997-2007

- SA-RT for functional decomposition & FSM
- Limited tooling & high tooling cost
- Semantics holes & subtlety
  - continuous vs. discrete flows
  - data flow or event flow
- Non-hierarchical & non-orthogonal FSM
  - Lot of time spent on synchronized products of FSM
- Use of Petri nets abandoned in 2000
  - Complex model for our students
  - Not very trendy after the end of SCEPTRE
- Often the Control Process is split into the resulting threads
  - The synchronized product is useless in the final system
- Mapping from data flow diagram to threads in DARTS
- Design Approach for Real-Time Systems (H. Gomaa)
- No tooling
  - In class, pen & paper only
Historical perspective: SA-RT functional decomposition

- **Context diagram**
- **Preliminary diagram**
- **Data dictionary**
- **Non-formal specification**
- **Data-flow diagram**
- **Control process**
  - Formal specification as FSM
- **SysC**
  - Do ...
- **SysA**
  - Do ...
- **SysB**
  - Do ...
Historical perspective: DARTS diagram

Exemple : Contrôle du système de freinage

- **Lire** État_demande_ABS
- **Ecrire** Niveau_freinage
- **Contrôler** application
- **Afficher** État bouton ABS
- **Affichage ABS**
- **Commander** freinage
- **Commande freinage**
- **Acquérir** demande freinage
- **Déetecter** glissement roue
- **HTR (100 ms)**
- **HTR (150 ms)**
- **HTR (1000 ms)**

Capella in teaching
Evolutions 2007-15

- SA-RT to SysML
  - Better tooling perspective
  - But necessity to declare a Block with its ports prior to its use as a Part is making the use tedious for students
- SysML industrial integration
- More expressive FSM and tooling
  - UML FSM based on Harel automata
  - Orthogonality avoids manual synchronized product
  - Orthogonality allows ascending design to be mixed with descending design
- DARTS to AADL
  - Same basic thread-related concepts
  - Potential tooling
    - OSATE2 used with the Y3 CS for avionics students
    - Too complex for Y2 common core class
Main SysML diagrams used: IBD for dataflow diagrams, BDD for typing (including units) and blocks&ports definition.
Why did I choose Capella over SysML?

In order to create the previous Internal Block Diagram I need to declare the blocks in a Block Definition Diagram.
Why did I choose Capella over SysML?

- FlowPorts are deprecated since SysML 1.3
- The standard recommends using Ports
  - The flow direction is given by an attribute of the carried element (item flow)
  - Two connected ports are either conjugated or not, their direction being given by the combination `isConjugated` and the direction of the carried element
- Too complex for my students who are not familiar yet with UML
- While I just wanted them to represent a dataflow diagram...
Let’s summarize

- I want to express:
  1. Dataflow diagrams showing functional interaction, hierarchically created
  2. Typing, if possible using units
  3. UML FSM
  4. Mapping a dataflow diagram of functions over threads, express how threads are triggered (periodic, sporadic, watchdog, aperiodic, etc.) and threads over CPUs, local function communications over blackboard or mailbox, global function communications over networks (CAN, AFDX, etc.)
  5. Run some performance analysis, schedulability tests, etc.

- Capella allows to easily address points 1&2. It is starting in v1.1.1 to address point 3.
- Some future extensions will address points 4&5 (Time4Sys & Waruna FUI project).
Capella in the methodology
Capella in teaching

requirements

Capella UML class diagram

UML Finite State Machines (in 2016-17, Yakindu SCT)

Functional decomposition & typing & units

Formal behavior

AADL (pen&paper)

Software/hardware architecture

NXC, LabVIEW, Ada, (POSIX), (Osek), (VxWorks)

Program
First exercise: Minepump

- Maintain water level between Low and High
- Trigger an alarm if MS>MS_L1
- Block the pump if MS>MS_L2
Operational Analysis

[OCB] Operational Capabilities

[OAIB] Alert

- Miner
  - Alert

- Excavator
  - Pump

[OAIB] Alert

- Read_methane_level
  - Methane_level

- Operate_alarm

[OAIB] Pump

- Read_Water_Level
  - Water_level

- Pump_control
Final dataflow logical diagram (LAB)
Using an external tool for UML FSM
- Computing worst-case end-to-end delays
- Assuming that a task having a deadline of $D$ meets its deadline
- Assuming that a task with a period $T$ has a deadline $D=T$
Variant with USB methane sensor and fault tolerance
After mapping functions to threads
Lecture & Tutorial classes (TD)

- Capella OCB, OAIB, SDFB, LAB presented in Lecture in 1h15
  - For small classes, 2h30 for letting the students do it on their computer at the same time
- Specifically using Capella:
  - 3x1h15 for letting the students create ODB, OAIB, SDFB, LAB, CDB on an example (minepump)
  - Most difficulties experienced for mechanical engineering students: CDB/LAB couple
  - Most difficulties for CS Masters students who know UML: reasoning with sensors/actuators/input-output in general
  - Majority of students able to work by themselves with a guide on a simple system’s design 1x1h15 (inverted pendulum)
- Manipulating UML FSM
  - For now using Yakindu, 2x1h15 + 1x3h
- From dataflow diagrams to design
  - 2x45 minutes
Exam

Second boîtier envoyant des consignes au 1er via bluetooth

Capteur infrarouge
Carte bluetooth intégrée
Boîtier de contrôle
Moteurs

Axe rotatif X
Axe latéral Y

gyros
accél
baro
GPS
Wi-Fi
avant
baro

Capella in teaching
Conclusion

- More students’ participation than when using pen&paper
  - A part was passive in tutorial class, waiting for the solution
- More students’ complaints because they cannot fail an exercise or they cannot carry on
  - Important to have a « ready-to-import » project file
  - These students may have been part of the prior passive group
- Students’ worries about having an exam with pen&paper while training using a software
- Problem of juggling between Capella/Yakindu/Pen&Paper with three different languages
  - Capella improved its UML FSM diagrams between v1.0.1 and v1.1.1
  - Future Time4Sys viewpoint in Capella
    (https://www.polarsys.org/projects/polarsys.time4sys)

- I recommend using Capella for software design class, and its possibilities will even improve in the near future