

Theory Applied to Real Embedded Systems



Franck Cassez

<http://www.irccyn.fr/franck>

National ICT Australia &

Centre National de la Recherche Scientifique, France

Supported by a Marie Curie International Outgoing Fellowship

7th European Community Framework Programme

December 15th, 2008

Outline of the Talk

- ▶ **Context of the Project**
- ▶ **Research Domain: Design of Real-Time Systems**
- ▶ **Theory Applied to Real Embedded Systems**
- ▶ **Theory & Tools**
- ▶ **Recent Case Study**

Context of the Project

▶ 7th EC Framework Programme/People/International Dimension

“The International Outgoing Fellowships (IOF) action aims to reinforce the **international dimension** of ... European **researchers** by giving them the opportunity to be **trained and acquire new knowledge** in a **third country high-level research organisation**. Subsequently, these researchers will return with the acquired knowledge and experience to an organisation in EU.”

▶ How it works: each year call for proposals, selection process, outcome (“yes”), implementation (end to end takes 1 year)

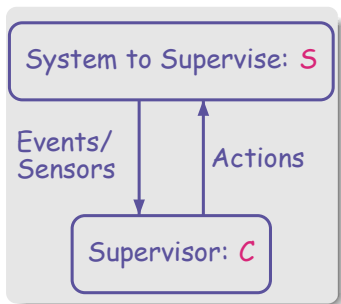
TARES: Aug. 14th, 2007 / Dec. 8th, 2007 / Sep. 1st, 2008

▶ Our proposal: **Theory Applied to Real Embedded Systems**

- ▶ high-level research organisation: NICTA
- ▶ return host: CNRS, Nantes, FR

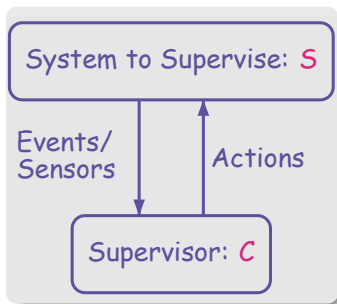
▶ Implementation: **Marie Curie Research Fellow** & CNRS Researcher **seconded** to NICTA from Sep. 2008 to Aug. 2010 and **re-integrated** into CNRS Sep. 2010 to Aug. 2011

Research Domain: Design of Real-Time Systems



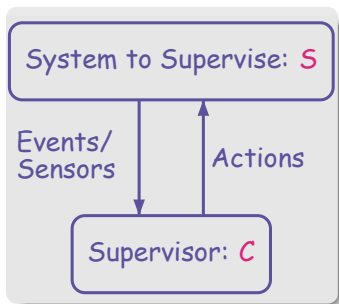
Research Domain: Design of Real-Time Systems

Build **Safe** Systems



Research Domain: Design of Real-Time Systems

Build **Safe** Systems



Property φ

Research Domain: Design of Real-Time Systems

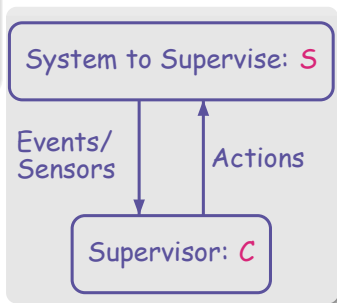
Modeling

Timed Automata

Time Petri Nets

Timed Logics

Build **Safe** Systems



Property φ

Research Domain: Design of Real-Time Systems

Modeling

Timed Automata

Time Petri Nets

Timed Logics

Build **Safe** Systems

System to Supervise: **S**

Events/
Sensors

Actions

Supervisor: **C**

Verification

Test

Theorem Proving

Model-Checking

Property φ

Research Domain: Design of Real-Time Systems

Modeling

Timed Automata
Time Petri Nets
Timed Logics

Build **Safe** Systems

System to Supervise: **S**

Events/
Sensors

Actions

Supervisor: **C**

Diagnosis & Control

Diagnosis
Control
Optimal Control

Verification

Test
Theorem Proving
Model-Checking

Property φ

Research Domain: Design of Real-Time Systems

Modeling

Timed Automata
Time Petri Nets
Timed Logics

Build **Safe** Systems

System to Supervise: **S**

Events/
Sensors

Actions

Supervisor: **C**

Diagnosis & Control

Diagnosis
Control
Optimal Control

Verification

Test
Theorem Proving
Model-Checking

Property φ

Implementation

Digital Supervisors
Continuous Systems

Theory Applied to Real Embedded Systems

▶ Why NICTA:

- ▶ **Real software** is developed at NICTA (e.g. L4 kernel)
- ▶ **Reasonable size** software
- ▶ Real-time software: **different types** of real-time problems scheduling, fault tolerance, ...
- ▶ **formal methods** to prove correctness of (parts of) the software

▶ Objectives of the Project:

- ▶ NOT to model-check the whole software
- ▶ Focus on **timing aspects**
 - ▶ scheduling
 - ▶ performance optimization
 - ▶ reconfiguration and fault detection
- ▶ Use **recently** developed techniques & tools

What kind of tools/problems?

Theory Applied to Real Embedded Systems

▶ Why NICTA:

- ▶ **Real software** is developed at NICTA (e.g. L4 kernel)
- ▶ **Reasonable size** software
- ▶ Real-time software: **different types** of real-time problems scheduling, fault tolerance, ...
- ▶ **formal methods** to prove correctness of (parts of) the software

▶ Objectives of the Project:

- ▶ **NOT** to model-check the whole software
- ▶ Focus on **timing** aspects
 - ▶ **scheduling**
 - ▶ **performance optimization**
 - ▶ **reconfiguration and fault detection**
- ▶ Use **recently** developed techniques & tools

What kind of tools/problems?

Theory Applied to Real Embedded Systems

▶ Why NICTA:

- ▶ **Real software** is developed at NICTA (e.g. L4 kernel)
- ▶ **Reasonable size** software
- ▶ Real-time software: **different types** of real-time problems scheduling, fault tolerance, ...
- ▶ **formal methods** to prove correctness of (parts of) the software

▶ Objectives of the Project:

- ▶ **NOT** to model-check the whole software
- ▶ Focus on **timing** aspects
 - ▶ **scheduling**
 - ▶ **performance optimization**
 - ▶ **reconfiguration and fault detection**
- ▶ Use **recently** developed techniques & tools

What kind of tools/problems?

UppAal-TiGA = UppAal for timed games

- ▶ Inherit UPPAAL expressiveness, data structures and GUI networks of TA, extended data types, ...
- ▶ Implements an **efficient forward/backward** on-the-fly algorithm
- ▶ Solves **safety** and **reachability** games
- ▶ Computes a timed controller (if one exists) otherwise computes a counter-strategy for the opponent + can be used in a command-line manner
- ▶ Planned Extensions:
 - ▶ **time-optimal** control: **DONE**
 - ▶ **partial observation 1**: **DONE** but unreleased
 - ▶ **partial observation 2**: **FORTHCOMING**
 - ▶ **Büchi** games: **DONE** but unreleased

What can we do with this tool ?

UppAal-TiGA = UppAal for timed games

- ▶ Inherit UPPAAL expressiveness, data structures and GUI networks of TA, extended data types, ...
- ▶ Implements an **efficient forward/backward** on-the-fly algorithm
- ▶ Solves **safety** and **reachability** games
- ▶ Computes a timed controller (if one exists) otherwise computes a counter-strategy for the opponent + can be used in a command-line manner
- ▶ Planned Extensions:
 - ▶ **time-optimal** control: **DONE**
 - ▶ **partial observation 1**: **DONE** but unreleased
 - ▶ **partial observation 2**: **FORTHCOMING**
 - ▶ **Büchi** games: **DONE** but unreleased

What can we do with this tool ?

UppAal-TiGA = UppAal for timed games

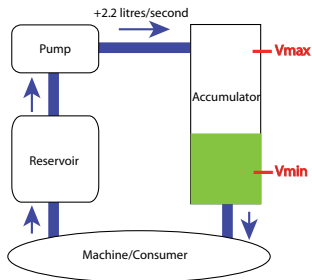
- ▶ Inherit UPPAAL expressiveness, data structures and GUI networks of TA, extended data types, ...
- ▶ Implements an **efficient forward/backward** on-the-fly algorithm
- ▶ Solves **safety** and **reachability** games
- ▶ Computes a timed controller (if one exists) otherwise computes a counter-strategy for the opponent + can be used in a command-line manner
- ▶ Planned Extensions:
 - ▶ **time-optimal** control: **DONE**
 - ▶ **partial observation** 1: **DONE** but unreleased
 - ▶ partial observation 2: **FORTHCOMING**
 - ▶ **Büchi** games: **DONE** but unreleased

What can we do with this tool ?

An Industrial Example: Oil Pump Control

Provider: HYDAC ELECTRONICS GMBH

European Project **QUASIMODO**

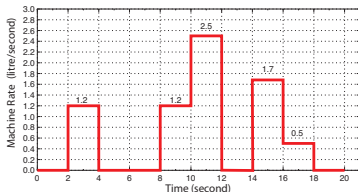


Assumptions

- ▶ Accumulator: **safety** requirement
 $R \equiv \forall t \geq 0, V_{\min} \leq v(t) \leq V_{\max}$
- ▶ Pump: **delay of 2 t.u.** between switches on/off
- ▶ When machine consumes: rate **fluctuates by ± 0.1 litre**

Control Objectives

- ▶ ensure **R**
- ▶ minimize energy:



$$E = \int_0^{\infty} v(t) dt$$

Two Solutions: 2-point and Smart Controllers

2-point Controller

- ▶ Switches on/off when bounds are reached
- ▶ Can ensure **R**
- ▶ Can be made **robust** (against fluctuations)
- ▶ Can be proved **correct** (with PHAVER)
- ▶ average **E** is **307**

Smart Controller

- ▶ Takes decision according to what happened in previous cycle
- ▶ Simulation with **Simulink**:
 - ① seems to reach a stationary regime in presence of fluctuations and ensure **R**
 - ② average **E** is **222**
Gain 28% / 2-point
- ▶ Can **NOT** be proved correct and robust

Both controllers measure time and volume **accurately**

Can we do **better** ?

Two Solutions: 2-point and Smart Controllers

2-point Controller

- ▶ Switches on/off when bounds are reached
- ▶ Can ensure **R**
- ▶ Can be made **robust** (against fluctuations)
- ▶ Can be proved **correct** (with PHAVER)
- ▶ average **E** is **307**

Smart Controller

- ▶ Takes decision according to what happened in previous cycle
- ▶ Simulation with **Simulink**:
 - ① seems to reach a stationary regime in presence of fluctuations and ensure **R**
 - ② average **E** is **222**
Gain 28% / 2-point
- ▶ Can **NOT** be proved correct and robust

Both controllers measure time and volume **accurately**

Can we do **better** ?

Two Solutions: 2-point and Smart Controllers

2-point Controller

- ▶ Switches on/off when bounds are reached
- ▶ Can ensure **R**
- ▶ Can be made **robust** (against fluctuations)
- ▶ Can be proved **correct** (with PHAVER)
- ▶ average **E** is **307**

Smart Controller

- ▶ Takes decision according to what happened in previous cycle
- ▶ Simulation with **Simulink**:
 - ① seems to reach a stationary regime in presence of fluctuations and ensure **R**
 - ② average **E** is **222**
Gain 28% / 2-point
- ▶ Can **NOT** be proved correct and robust

Both controllers measure time and volume **accurately**

Can we do **better** ?

Methodology

- ① Build an abstract model with **timed game automata**
- ② **UPPAAL-TIGA** to compute an **optimal** controller on **one** cycle
- ③ Check **correctness** and **robustness** with PHAVER
- ④ Evaluate efficiency with PHAVER and SIMULINK

Restrictions & Results

- ▶ Restrictions on the **power** of our controller:
 - ▶ We measure the volume at the beginning of each cycle
 - ▶ Volume is measured with imprecision of ± 0.06 litres
 - ▶ Controller can only switch on at most twice per cycle
 - ▶ Switch commands issued at t occur at time $t \pm 0.1$ seconds
- ▶ Results
 - Synthesis of 14 local & discrete controllers with UPPAAL-TIGA
 - Verification of correctness & robustness the 14 controllers in a continuous environment with PHAVER
 - Average E is 170: Gain 23% / Smart Controller (44% / 2-point)

Methodology

- ① Build an abstract model with **timed game automata**
- ② **UPPAAL-TIGA** to compute an **optimal** controller on **one** cycle
- ③ Check **correctness** and **robustness** with PHAVER
- ④ Evaluate efficiency with PHAVER and SIMULINK

Restrictions & Results

- ▶ Restrictions on the **power** of our controller:
 - ▶ We measure the volume at the **beginning** of each cycle
 - ▶ **Volume** is measured with **imprecision** of ± 0.06 litres
 - ▶ Controller can only switch on at most **twice** per cycle
 - ▶ Switch commands issued at t occur at time $t \pm 0.1$ seconds

▶ Results

- ① Synthesis of 14 **local & discrete** controllers with UPPAAL-TIGA
a controller gives 4 data: **start, stop, start, stop**
- ② Verification of **correctness** and **robustness** the 14 controllers in
a **continuous** environment with PHAVER
includes fluctuations and measure/time imprecisions
- ③ Average E is 170: Gain 23% / Smart Controller (44% / 2-point)

Methodology

- 1 Build an abstract model with **timed game automata**
- 2 **UPPAAL-TIGA** to compute an **optimal** controller on **one** cycle
- 3 Check **correctness** and **robustness** with PHAVER
- 4 Evaluate efficiency with PHAVER and SIMULINK

Restrictions & Results

- ▶ Restrictions on the **power** of our controller:
 - ▶ We measure the volume at the **beginning** of each cycle
 - ▶ **Volume** is measured with **imprecision** of ± 0.06 litres
 - ▶ Controller can only switch on at most **twice** per cycle
 - ▶ Switch commands issued at t occur at time $t \pm 0.1$ seconds
- ▶ Results
 - 1 Synthesis of 14 **local & discrete** controllers with UPPAAL-TIGA a controller gives 4 dates: start, stop, start, stop
 - 2 Verification of **correctness** and **robustness** the 14 controllers in a **continuous** environment with PHAVER includes fluctuations and measure/time imprecisions
 - 3 Average E is **170**: Gain 23% / Smart Controller (44% / 2-point)