Orccad,
a Model Driven Architecture and Environment for Real-Time Control

Soraya Arias
Florine Boudin
Roger Pissard-Gibollet
Daniel Simon
Orccad: status and motivations

Model:
• Control design oriented approach for robotics
• Mixed feedback and discrete events

Tools:
• Design & simulation/validation
• Real-time workshop

V4 modeling and software development:
• Aging version, based on proprietary tools (1998)
• Sound model & design approach
• Model Driven Architecture based on Eclipse Modeling Tools
• Open Source software
The Orccad model

Requirements capture from:
• End-users
• Control scientists
• Real-time experts
Bottom-up design

RobotTasks
• Feedback Control
• Cyclic real-time data flow
• Event-based view

RobotProcedures
• Discrete Events Control
• Incremental design
• Exception processing
• Mission definition
Quadrotor networked control & diagnosis

Networked system
- CAN bus
- Distributed diagnosis
- Fault tolerant control

Flexible scheduling
- Varying sampling
- (m,k)-firm
- Dynamic priorities

Hardware-in-the-loop
- Linux simulation
- PPC embedded

V4 Runtime update

(SafeNecs ANR)
Networked system
- CAN bus
- Distributed diagnosis
- Fault tolerant control
- Flexible scheduling
- Varying sampling
- (m,k)-firm
- Dynamic priorities

Hardware-in-the-loop
- Linux simulation
- PPC embedded
- V4 Runtime update
Orccad components: RobotTask

Feedback control action
- Control algorithm definition
- Invariant structure for RT life
- Modular design
- Functional parameters
- Timing parameters

Event based behaviour
- Precondition (opt. timeout)
- Synchronization
- Exception
  - Weak T1
  - Strong T2
  - Fatal T3
- Postcondition (opt. timeout)
Orccad components: Modules

Implement functions

Algorithmic
Phy_Resource (drivers)

Typed Input/Output ports
• Data
• Drivers
• Parameters
• Events

User defined C code

init(inputs)
forever{
compute(inputs)
}
end()
Orccad components: Temporal Constraint

Real time threads

- Task ID
- Modules ID
- Priority
- Synchronization
  - Clock
  - Output port
  - Extern event
- Overrun policy
  - Skip, Soft, Hard
  - User's defined
- WCET
- CPU ID
Orccad components: RobotTask

Validation:
- Names uniqueness
- Links coherency
- PhR uniqueness
- All modules are synchronized
- ...
Orccad components: RobotProcedure

- Composition of control actions
- Incremental design
- From exception processing to mission definition
- Currently written in Esterel

- Synchronous language with imperative style
- Modularity using parallelism at specification time
- Powerful escape mechanisms
- Multiform time
- Deterministic behaviour $\Rightarrow$ formal verification
- Inputs and outputs are signals (pure or valued)
- Communication based on synchronous diffusion
- Compiles into efficient deterministic automata
Orccad components: RobotProcedure

- `emit S`
- `await S`
- `p ; q`
- `p || q`
- `call P (X1,X2)(e1,e2)`
- `exec T ()() return R`
- `loop p end`
- `loop p each S end`
- `present S then p else q end`
- `abort p when S`
- `weak abort p when S`
- `diffusion of a signal`
- `await the next occurrence`
- `sequence`
- `parallel`
- `extern procedure call`
- `extern task launch`
- `infinite loop`
- `temporal loop`
- `branching`
- `strong preemption`
- `weak preemption`
Control task automaton:
Robot-Procedural specification:

- Incremental design of complex actions (nominal and degraded modes)
- Exception handling
- Mission specification: sequence, parallelism...
- Semi-automatic code generation from existing specs.
- We don’t have a FSM in mind at design time
Guarded alternative:

- actions are guarded by T2 emitted by other actions
- actions are mutually exclusive
- no dead-lock nor endless loop
Formal verification:

Specialized properties:

- Safety properties (fatal exceptions are always correctly handled)
- Liveliness properties (the goal is always reached in nominal executions)
- Conflicts detection (mutual exclusion)
- Conformity with the requirements
- Help to specification (abstract views)
Formal verification: customized tools

two control laws must not compete to control the same robot
MDA in Orccad

- Eclipse Modeling Project based on the idea of a Model (MetaModel)
- EMP offers different tools for different goals: EMF, GMF, Xpand...
- Principle of plug-in in the Eclipse Environment
Orccad by Developer & User

As a Developer

Orccad MetaModel → Software

[Images of robotic devices and a car]
Orccad by Developer & User

As a User
MDA : How it works

The Metamodel defines how a model is made.
Made by the developer.

The Model is realized by the user.
It matches to the meta-model and its constraints.

It generates the source code from the model, using templates defined by the developer.
MetaModel - an example

The graphical view is close to an UML model.
MetaModel - an example

Code is generated in Java, we find Java properties in the Ecore model.
MetaModel - an example

Code generated in Java, we find Java properties in the Ecore model.
MetaModel - an example
EMF – Tree Editor

- A plugin developed in the Eclipse project
- From a metamodel, generates a Tree Editor as a plugin
  - For Eclipse
  - RCP plugin
- Really useful to realize beta-version
- Constraints must be defined and filled at this step.
EMF – Tree Editor

- Generation of Code
  - Creation of a new Project (Plug-in)
  - Packages by functions
  - All the customization on eclipse plug-in are allowed

- Generated code must be modified and/or completed. With keyword, a re-generation of the code is safe.
EMF – Tree Editor
GMF – Graphical Editor

The Tree Editor must be generated before the generation of the Graphical Editor.

We specify through files:
- Graphical representations of elements and links
- Palette tool
- Mapping, the coherence between view, ecore and palette.

Then we can generate the Graphical Editor.
GMF – Graphical Editor

Graphical Interface Code:

- MVC design pattern

Model, Controller and View are independent for a easier maintenance.
GMF – Graphical Editor

Result of a quick Graphical Interface uncluttered -> customization!
GMF – Graphical Editor

Example of a simple customization
Xpand – The Code generator

Why Xpand?

➢ Xpand is proposed as a M2T (Model to Text) technology in the Eclipse Modeling Project
➢ It fits with the Ecore Metamodel
➢ Entirely customization for any type of file
➢ Templates have a simple syntax
➢ Code generator is independent from the source code
Xpand – The Code generator

- Meta Model .ecore
- XPAND
- Templates .xpt
- Source code
- Documentation
- Webpages
- Whatever you want
Runtime

Code generation
- C++ classes
- Virtual system calls

Compilation
- Binding to real calls
- Link with specific runtime library
  - Linux/Posix
  - Xenomai/Native
  - ...

<table>
<thead>
<tr>
<th></th>
<th>Orccad</th>
<th>Linux/Posix</th>
<th>Xenomai/Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>launch a real-time task</td>
<td>orcSpawn</td>
<td>pthread_create()</td>
<td>rt_task_spawn()</td>
</tr>
<tr>
<td>timer</td>
<td>orcTimer_t</td>
<td>timer_t</td>
<td>RT_ALARM</td>
</tr>
<tr>
<td>message queue</td>
<td>orcsqQ_t</td>
<td>mqd_t</td>
<td>RT_QUEUE</td>
</tr>
<tr>
<td>semaphore</td>
<td>orcSem_t</td>
<td>sem_t</td>
<td>RT_SEM</td>
</tr>
</tbody>
</table>
Drone controller

Attitude control
- Quaternion non-linear estimator
- Screws velocity control
- Attitude set point

Synchronized link
- Minimized latency

Asynchronous links
- Data integrity protection
Drone controller

Attitude control

Sensors/Motors diagnosis

• T1 exception
• Parameter update
Drone controller

Attitude control

Sensors/Motors diagnosis

Networked system
- Drivers
- UDP sockets
- CAN sockets
- Real testbed or HIL
Drone controller

Attitude control

Sensors/Motors diagnosis

Networked system

Feedback scheduler
- Varying clocks
  - MTSetSafeSampleTime(h)
- CAN priorities
- Overrun policies
  - Soft, Hard, Skip...
- API instrumentation
  - orcGetExecTime()
Feedback-scheduling a robot controller
http://orccad.gforge.inria.fr

Opening soon!

Questions ?