HAL
open science

ORCCAD, robot controller model and its support using Eclipse Modeling tools

Soraya Arias, Florine Boudin, Roger Pissard-Gibollet, Daniel Simon

To cite this version:

Soraya Arias, Florine Boudin, Roger Pissard-Gibollet, Daniel Simon. ORCCAD, robot controller model and its support using Eclipse Modeling tools. CAR 2010 - 5th National Conference on Control Architecture of Robots, May 2010, Douai, France. inria-00482559

HAL Id: inria-00482559

https://inria.hal.science/inria-00482559v1

Submitted on 10 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ORCCAD, robot controller model and its support using Eclipse Modeling tools

Soraya Arias, Florine Boudin, Roger Pissard-Gibollet and Daniel Simon
INRIA Rhône-Alpes
Inovallée, 655 avenue de l'Europe, Montbonnot
38334 ST ISMIER Cedex, FRANCE
http://orccad.gforge.inria.fr

Abstract

Orccad model has been designed to structure robot architecture according to two aspects : continuous time aspects, related to control laws, carefully merged with discrete-time aspects related to control switches and exception handling. CAD tools have been provided to help the users along the steps of the design, verification, implementation and exploitation processes. In this paper, we present how Eclipse Modeling tools are adapted to provide a complete CASE robotics tool-set to support this model from design to code generation. This approach allows to focus on the model rather than the infrastructure.

1 Motivation

The ORCCAD model allows to design robot software controllers using existing theory (Control and Discrete Events theory) in accordance with software tools based on existing software technologies which are already used in the embedded and real-time domains (RTOS, Object Oriented Language, Synchronous Language,...). We do not want to reinvent the wheel but to provide software tools adapted to co-engineering for robotic practitioners.

The first ORCCAD model has been described more than fifteen years ago [21] but we think it is still valid. Over the years software tools need to be constantly updated. Except for the real-time multi-tasks controller implementation, the ORCCAD software has not been updated for eight years. According to the evolution of the software engineering, (Model Driven Engineering MDE, Synchronous language [8], [7]. . .) ORCCAD tools now evolve from the existing model as we do not want to develop a yet another robotics software controller "from scratch".

A lot of robotics controllers software inspired from Component-Based Software engineering have been developed, for example : Claraty [29] , Cosarc [31] , Oscar [17], Orca [5], Orocos [6] or HRPOpen [20]. A good survey of robotic systems architecture can be found in [27]. All these approaches are done by research teams, often dedicated to a single robot type (mobile robot, humanoid robot or robotic arm) and are not adopted by a large community. Generally, focus is also put on the framework, its layers and its programming tools. To tackle the control robotics software complexity, we propose to define a model independently of the framework and of its programming tools.

The ORCCAD model can be used at different level and support different tools :

- it provides a C++ code template that can be used to write "manually" a robot controller. This code can be embedded in different frameworks for simulation or real experiments. For example this approach has been used for robotic visual-servoing [22] and 3D rover simulator [26], or for feedback scheduling and hardware-in-the-loop experiments [36], [2].
- specific robotics space tools designed by Trasys like Formid tools [23] to specify robot controllers behavior. Here the goal is to implement a generic formal mission specification and verification tool inside the DREAMS ESA robotic ground station control [24].
- finally, as a Computer-Aided Software Engineering (CASE) tool to model and generate a fully featured ORCCAD robot controller.

For our CASE tool, we propose to abstract the ORCCAD robot controller model using Eclipse Modeling tools based on Model-Driven Engineering (MDE) concepts. The robot controller software will only be a "instantiation" of our model

into a dedicated framework.

Our approach is more focused and less ambitious than project like Topcased [28], openembeDD [10] or Marte [16] which promotes model-driven engineering and formal methods as key technologies to design critical systems.

In the robotics field, the ANR project PROTEUS [30] has also a large ambition to define and implement experiments using Domain Specific Languages (DSL) allowing the description of the problems for specific part of the robotic domain.

In the next sections, we present the ORCCAD model, and then its implementation using Eclipse Modeling tools.

2 Eclipse Modeling Tools

To separate robot controller model and its code implementation we have decided to use Eclipse Modeling Tools [8]. The Eclipse Modeling Project focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tools, and standard implementations. These tools became extremely popular in the development of applications for a large number of domains.

The Eclipse Modeling Project is organized into projects that provide different capabilities : abstract syntax development, concrete syntax development, model-to-model (M2M) transformation and model-to-text (M2T) transformation.

The abstract syntax development tackles the conceptual elements of the model. This is performed by the Eclipse Modeling Framework (EMF). It provides an object graph for representing models, as well as capabilities for (de)serializing models in a number of formats, checking constraints, and generating various types of tree editors for use in Eclipse. Models can be specified using annotated Java, XML documents, or UML, then imported into EMF. Some tools as Object Constraint Language (OCL) [13] can be used to query and define constraints to apply on models.

The concrete syntax development tackles the graphical visual representation of the model. This can be performed by the Graphical Editor Framework (GEF) [9]. It provides the foundations for building graphical views for EMF models types. The Graphical Model Framework (GMF) for Eclipse reduces the complexity of developing in GEF/EMF.

The M2M project will deliver a framework and transformation engines for model-to-model transformation languages. It is a key aspect of model-driven development to develop gateway for different models.

The Model to Text (M2T) project focuses on the generation of textual artifacts (html page, source code,...) from models with tools like Xpand, Jet or Acceleo [11].

Using a subset of these modeling project technologies, it is possible to create our own DSL toolkit. An example of such designed DSL is Kermeta workbench [15]. It is a powerful meta-programming environment based on an object oriented DSL optimized for meta-model engineering.

The next section presents the ORCCAD model, model we want to build through Eclipse Modeling tools mechanisms.

3 The Orccad Model

[4] The formal definition of a robotic action is a key point in the ORCCAD framework. It is based on the following basic statements :

- In many cases the tasks to be achieved by robots can be stated as automatic control problems, and can be efficiently solved in real-time by using adequate feedback control loops. Let us mention the *Task-Function* approach[32] that was specifically developed to design robotic control systems;
- The design of theses feedback control loops (or control laws) is not enough to fully state a robotic action : starting and stopping events must be considered, as well as reactions to significant events observed during the task execution; this will be further used to design complex missions through actions composition.
- Since the overall performance of the system relies on the existence of efficient real-time mechanisms at the execution level, particular attention must be paid to their specification, verification and integration with control design.

From the users and programmers point of view, the specification of a robotic application must be modular, structured and accessible to users with different domain of expertize. The *end-user* is concerned with a particular application and must be provided with high level formalisms allowing to focus on mission specification and verification issues; the *control systems engineer* needs an environment with efficient design, programming and simulation tools to design and implement the control laws which then are encapsulated for the end-user.

Therefore Orccad proposes a bottom-up approach, based on a stack of models to hold the design process from elementary functions up-to complex application.

Orccad is structured mainly according to two abstraction levels : a functional level and a control level.

The *functional level* provides the elementary tasks to produce the robotic control loop (or control law). This level is modeled through the notion of *Robot Task* that is a cyclic data flow algorithm managed by an event-based logical behavior.

The *control level* aims at combining the elementary tasks in a logical way to define a robotic application or mission. This level is modeled through the notion of *Robot Procedure*.

3.1 Robot-Tasks

The **Robot-Task** (RT) models basic robotic control actions, where feedback control aspects are predominant. For example, let us cite hybrid position/force control of a robot arm, visual servoing of a mobile robot following a wall or constant altitude survey of the sea floor by an underwater vehicle. The RT characterizes in a structured way closed loop control laws, along with their temporal features related to implementation and the management of associated events [25, 35]. The feedback-control data flow is designed by a set of fully connected acyclic graph of algorithmic modules, like in the block-diagram in Figure 11c. Loops are forbidden and the diagram is closed via the physical resource. To avoid conflicts, a RT must have exactly one controlled physical resource i.e a robot (but the number of sensors is free). Temporal constraints must be defined to allow for real-time code generation.

Events associated to a RT define its logical behavior (or abstract view) they are structured with the following types :

- preconditions, which can be associated with measurements and watchdogs
- events and exceptions of three types :
  - synchronization signals reports the occurrence of a particular state in a RT, they can be used to synchronize different RTS;
  - type 1 exceptions are locally processed in the RT, e.g. by tuning a parameter of the control law;
  - type 2 exceptions signal that the RT cannot run to completion and must be switched for a new one, chosen by the supervision controller;
  - type 3 exceptions are fatal for the application and must, as far as possible, drive the system in a safe recovery state.
- post-conditions are emitted when the RT successfully terminates.

3.1.1 Module

A RT is composed of a several kind of modules that linked together will produce a specific robotic control loop. A RT is specified as a block-diagram, where elementary *algorithmic* modules are connected through their input/output ports, e.g. as in Figure 3.1.1. The structure and code execution of the module follow a systematic structure :

- the **init()** is run once, it is used to initialize the data and structures needed by the module;
- the **compute()** code cyclically reads the input ports, computes a function and posts the results on the output ports; it is repeatedly triggered by events which release a synchronization point, e.g. a periodic clock;
- the **end()** code is executed once at the end of the control loop task and performs the closing of the module ans of its associated resources.

Modules can exchange data only through oriented ports, and oriented links define data dependencies between modules. **Data** ports handle the flows of data generated by the cyclic execution of the modules. Data ports are typed using the various data types (e.g. double) and structures (e.g. vectors) supported by the host programming language. Events raised by some particular condition while running a module can be exported using **Event** ports. Finally bidirectional **Parameter** ports are used to import or export parameters values upon occurrence of some conditions triggered at a higher level.

A particular module, the *Physical-Resource* provides a gateway towards the physical system through its oriented **Driver** ports, which must be connected with algorithmic-modules Data ports. Indeed Driver ports call the drivers used to interface the control software with physical actuators and sensors every time they are read/written by the connected module. Physical-Resources can also hold Event and Param ports.

3.1.2 Temporal Constraints

The module specification defines an action from a continuous time point of view, i.e independently of time discretization and other implementation related aspects which are considered in a next design step. The transition from the above

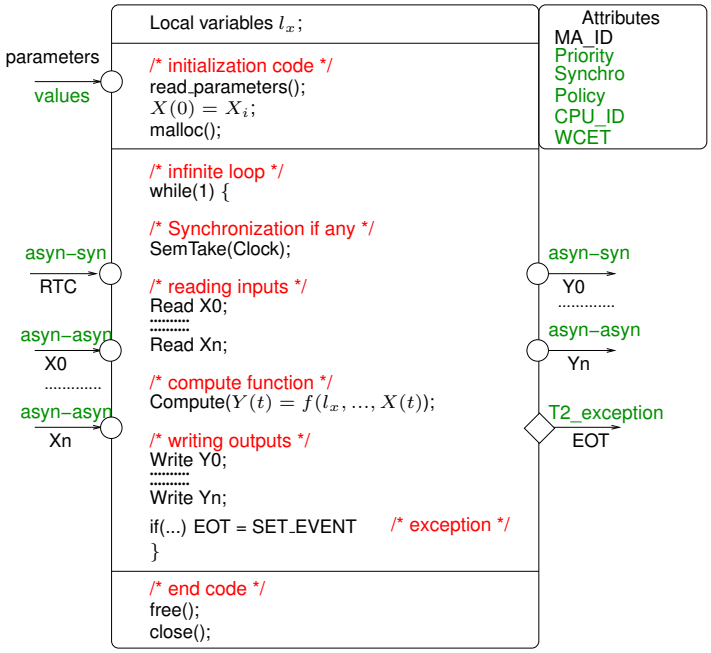


Figure 1: a Module with functional and temporal constraints

continuous time specification to a description taking into account implementation aspects is done by associating run-time properties to modules, i.e. sampling periods, assumed worst case execution time (WCET), communications and synchronizations between the processes. The set of temporal properties associated with a set of connected modules sharing the same constraints is modeled by a **Temporal Constraint (TC)**.

Modules can be synchronized in several ways :

- explicitly periodic modules are connected to a system clock through a hidden input port ;
- an input port can be synchronized on an output port of another module, thus inheriting its period ;
- it can be synchronized by an external source, e.g. by data incoming on a network socket.

Temporal attributes are associated to TCs to allow the configuration of the set of parametrized modules as a real-time executable process. The set of chosen attributes is assumed to be compliant with the API and features of most off-the-shelf operating systems with real-time capabilities.

Besides synchronization, another real-time related parameters is the TC **priority**. A **Policy** flag specifies one of the predefined recovery behavior used to handle timing violations at run-time, e.g. soft or hard real-time constraints. The assigned **CPU_ID** associated to a TC gives provision for multi-core implementation. The estimated worst case execution time (**WCET**) is also given to allow for further schedulability analysis.

3.2 Robot Procedures

When a robotic mission is to be defined, we are only interested on the behavior of the control loop (does it end normally, where there any error during the loop, does the control loop achieve its goal etc) to decide what is to be done at the next step of the mission.

For each RT, an *abstract view* is generated, hiding all specification and implementation details of the control law (Figure 2). This abstract view can be seen as the API of the RT. The characterization of the interface of a RT with its environment in a clear way, using typed input/output events, allows the composition of them in an easy way in order to construct more complex actions, the **Robot-Procedures** (RPs).

The RP paradigm is used to logically and hierarchically compose RTs and RPs in structures of increasing complexity. Usually, basic RPs are designed to fulfill a basic goal through several potential solutions, e.g, a mobile robot can follow a wall using predefined motion planning, visual servoing, or acoustic servoing according to sensory data availability. RPs design is hierarchical so that common structures and programming tools can be used from basic actions up to a full mission specification.

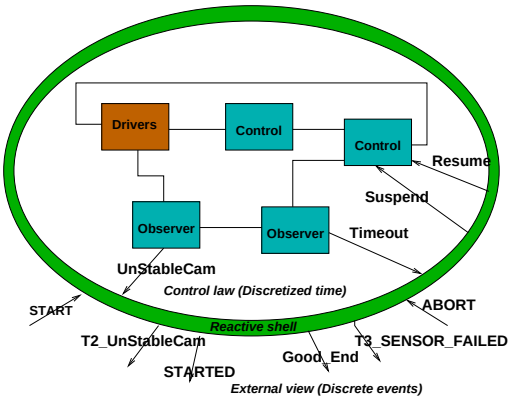


Figure 2: Encapsulation of the control law in a reactive shell

These well defined structures associated with synchronous composition, thanks to the use of the ESTEREL language [3], allows for the systematization and thus the automatizing of the formal verification on the expected controller behavior. This is also a key to design automatic code generators and partially automated verification. Formal definitions of RPs and RTs together with associated available formal verification methods may be found in [21].

4 Orccad tools using Eclipse modeling

4.1 EMF/GMF Framework

As presented in section 2, the Eclipse Modeling Project (EMP) is the base for the model driven software development. It's a technology dedicated to build Domain Specific Languages (DSLs). The structure of a DSL is captured in its meta-model. This meta-model is manipulated through an Ecore file in the EMF framework. The EMP regroups different frameworks dedicated to precise tasks. Together with EMF, we use GMF for the GUI part and Xpand for the code generation part. These frameworks are presented below.

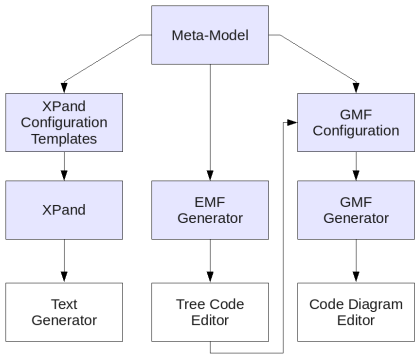
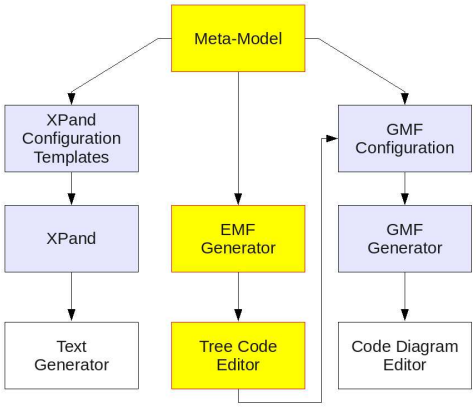


Figure 3: View of the Eclipse Modeling Project tools used for ORCCAD.

4.1.1 EMF

The **Eclipse modeling Framework** is the base of the modeling. This framework includes the Ecore meta-model (Figure 4). We choose to work with a partitioning structure (meta-model defined in several Ecore files) for the Orccad meta-model. It allows better maintenance, legibility and take-down on the meta-model. A Java tree editor is generated from the Orccad meta-model. The Orccad user can specify his Orccad model using the provided editor.



A tree editor example of the RobotTask structure (3.1) is given in Figure 5.

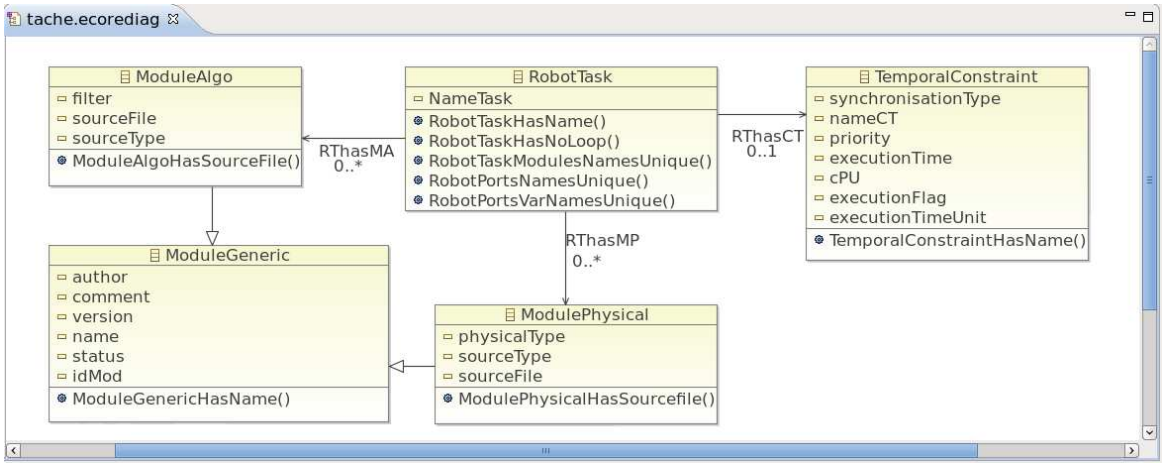
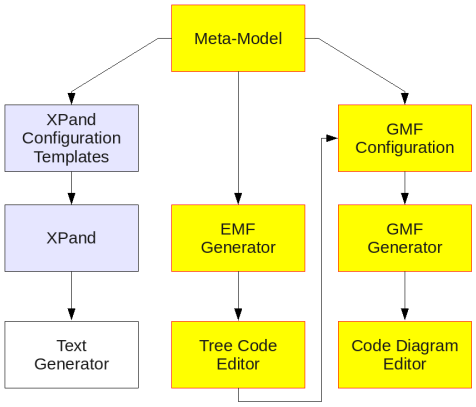


Figure 4: The Robot Task Ecore file, part of the global meta-model of Orccad.

4.1.2 GMF

The **Eclipse Graphical Modeling Framework** is a tool based on both EMF and GEF. The Graphical Editing Framework (GEF) [9] allows developers to take an existing application model and quickly create a rich graphical editor. GMF benefits from EMF and GEF. From the tree editor and with graphical configuration information, GMF generates a graphical user interface in Java. This interface can be afterwards customized according to user/developer preferences. The graphical configuration needed by GMF concerns the definition of the graphics elements manipulated in the graphical editor diagram (by example a module is represented as a box, an input/output port is represented as a triangle, etc).



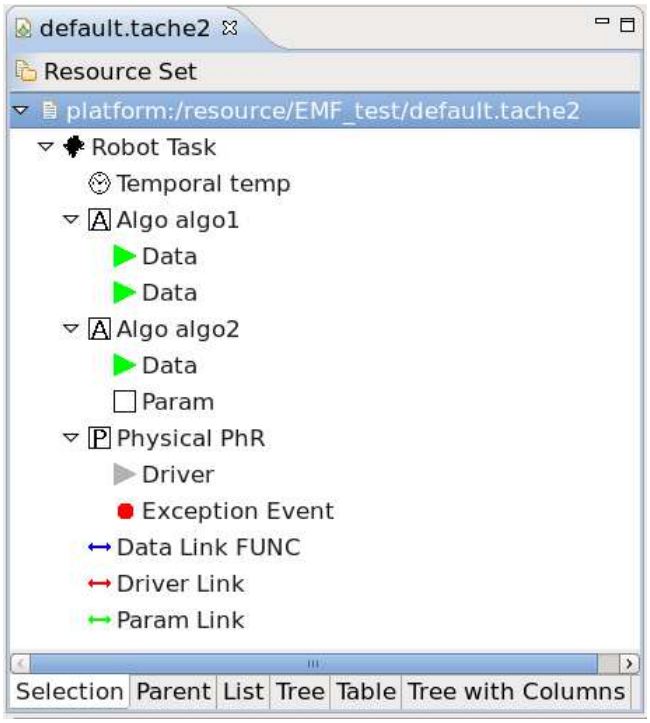


Figure 5: An Robot Task editor made with EMF.

Here is an illustration of the generated interface for the Robot Task diagram 6

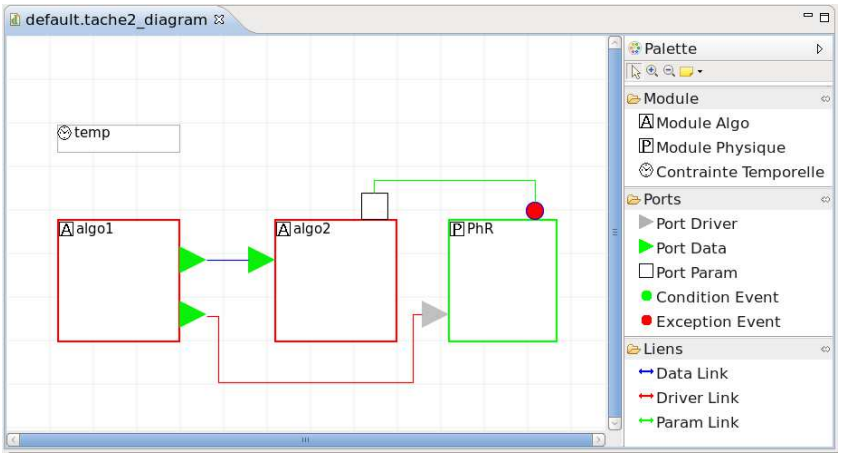


Figure 6: Graphical editor equivalent to the tree editor of Figure 5

4.2 Constraints

Different methods are provided by EMF to specify constraints, for example using OCL (Object Constraint Language) [13] or specify constraints in the meta-model. We choose the last one to keep the constraints in the meta-model for maintenance needs. We specify different types of constraints in the Ecore meta-model, in one hand, the structural constraints about the architecture, in the other hand, editor constraints (by example, names of modules non empty and unique). An example of editor constraint is given in Figure 7. Both kind of constraints are declared as EOperations in the Ecore meta-model. At the generation of the tree editor source code, specific functions must be filled by the developer in Java to define properly these constraints.

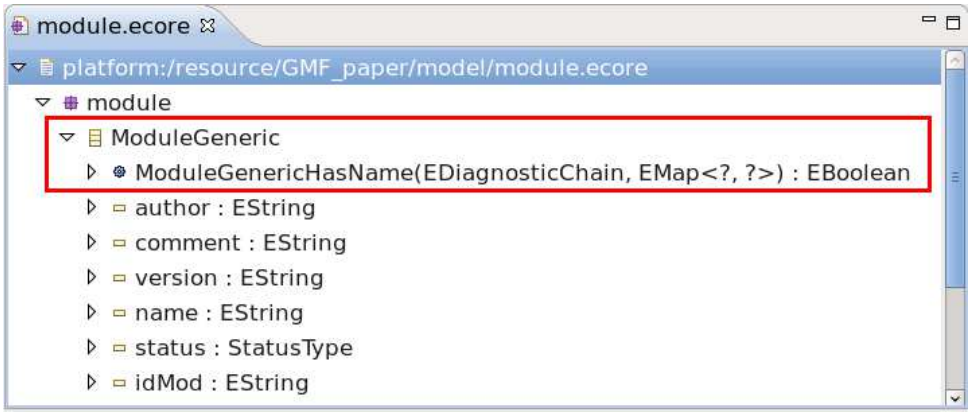


Figure 7: Specification of a constraint in the Ecore file.

An example of the filled generated function for a constraint declared in the figure 7 is given Figure 8. These defined

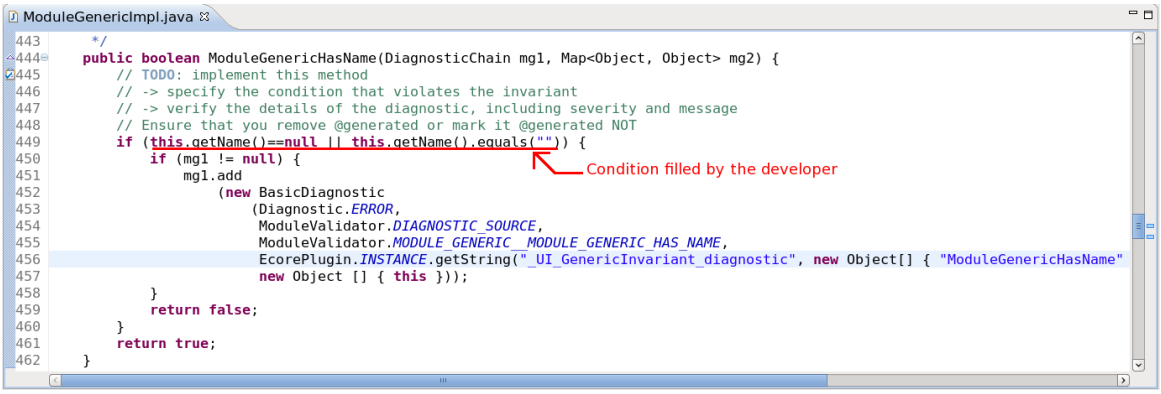
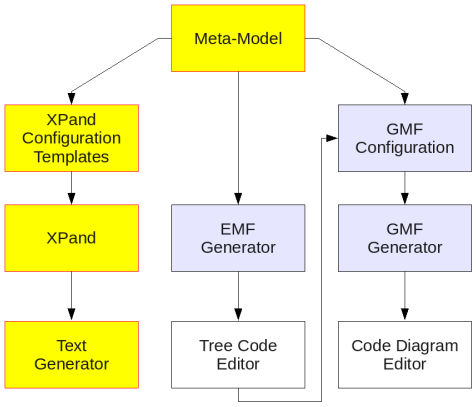


Figure 8: Example of a generated and filled constraint.

conditions are available in both EMF and GMF editor.

4.3 Code generation

The Eclipse Modeling Project includes a M2T (Model to Text Transformation) [14] section which provides different tools. We can find Acceleo, Jet and XPand [14]. We choose to use XPand [11] because of its simple syntax and it suits the EMF meta-model. The code generator is defined has a simple plugin added in the Eclipse Workspace.



We use the Xpand tool in such way to generate Esterel source code, C source code or/and HTML documentation from

the Orccad model specified by the user. As for GMF, generator configuration information must be given to specialized the generated code according to the needs.

The configuration of the Xpand generator needs templates defined in .xpt files. An example in the Figure 9 shows the template for a Robot Task HTML documentation page.

```
Root-.xpt
114
115«DEFINE RobotDef FOR RobotTask»
116
117«FILE  "TacheRobot/" +this.NameTask+ ".html"»
118<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
119    <head>
120        <title>Tache Robot «this.NameTask»</title>
121        <link rel="stylesheet" media="screen" type="text/css" title="RobotTask" href="../../RobotTask.css" />
122    </head>
123    <body>
124        <h2>Proprietes</h2>
125            <ul>
126                <li>Nom : « this.NameTask»</li>
127                <li>Appartient a la <a href="../Procedure/«((Procedure)this.eContainer).NameProcedure».html"></li>
128            </ul>
129    </body>
130</html>
131
```

Figure 9: Example of a Xpand template file.

The Eclipse IDE has the advantage to provide appropriate tools to produce binary executable from the generated source, for example the CDT (C/C++ development Tools)[12] or debugging tools. From model to binary executable, this single environment includes all the tools needed.

4.4 Runtime/Model execution

4.4.1 Runtime library

Once a robotic application has been fully defined and parametrized, run-time code can be generated thanks to the models used all along the design process. It is assumed that the execution is supported by an operating system with real-time capabilities, and the system calls used in the ORCCAD model were chosen to be generic enough to be found in most available RTOS. As control design is abstracted from any particular real-time target, the code generator provides the C++ code of the particular application, generated from the classes of modules, TC, RT and RP, together with the glue code to connect the end-users specific modules with the operating system. At this point system calls are still virtual, they are instantiated with the real system calls of the target at compile time. Therefore the run-time library is made of two classes of code templates :

- generic templates to define the C++ classes of ORCCAD components, e.g. modules, RT or physical resources;
- real-time targets specific definitions allowing to bind virtual system calls to concrete ones.

Up to now the targets taken into account are Linux (using the Posix API with real-time extensions) and Linux/Xenomai[1] (using its native API), the following table gives a few examples of bindings between virtual calls and their actual implementation.

	Orccad	Linux/Posix	Xenomai/Native
launch a real-time task	orcSpawn	pthread_create()	rt_task_spawn()
timer	orcTimer_t	timer_t	RT_ALARM
message queue	orcsgQ_t	mqd_t	RT_QUEUE
semaphore	orcSem_t	sem_t	RT_SEM

Table 1: Bindings from Orccad to Linux and Xenomai

[1]Xenomai is a real-time framework cooperating with the Linux kernel `www.xenomai.org`

4.4.2 Data flow execution model

In default mode, for basic control purpose, the designer specifies a single-loop, single-rate controller where all the algorithmic modules are gathered in a single real-time task, for which a single sampling period is defined. In that case the code of the modules included in the TC are ordered according to the data dependencies defined by the oriented links.

More complex timing behaviors, e.g. multi-rate control, make use of more or less tight synchronizations between modules and clocks. According to implementation requirements, modules and TCs can be mapped one to one, allowing them to be run asynchronously, where each module owns its private synchronization point. For efficiency purpose, modules which are strongly dependent, and which need to be tightly synchronized, can be gathered and embedded in a single TC, which runs in sequence the modules codes in the order defined by the data links. Data integrity between asynchronously connected modules is provided by asynchronous lock-free buffers [37] which are automatically inserted at code generation time.

In many cases simple design rules can be used to get a correct (deadlock free) timing diagram [34]. For more complex designs the synchronization and timing diagram can be built interactively and a Petri net model of the synchronization skeleton of the RT can be automatically extracted from the timed block-diagram to check for performance and deadlock freedom [33].

Beyond multi-rate scheduling, Feedback Scheduling can be specified and implemented in a particular module and TC, provided that some instrumentation (e.g. variable clocks generation and execution time measurement) is available from the real-time platform ([36]). Hooks are given in the run-time library to monitor, e.g., execution times and overrun status, and to control the scheduling parameters of the tasks set, e.g. timer intervals.

Note that feedback-control is inherently robust not only against plant parameters uncertainties but also against timing deviations such as occasional deadlines miss, data loss and jitter. Feedback controllers may run under weakly-hard (rather than hard) real-time constraints, which means that the timing execution must be managed by an exception handling policy. For example, in case of overrun the **Skip** policy allows the running task to finish its current execution beyond the assigned deadline, but to skip the next one. Some default policies are implemented in the run-time library, which is open and can be extended with user's defined policies.

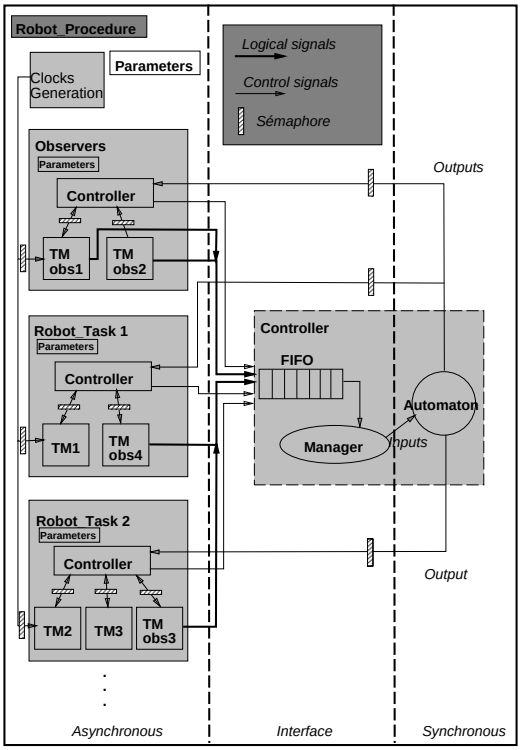
4.4.3 Event based execution model

As stated in the Orccad Model 3 section, a Robot-Task is made of a cyclic data flow algorithm managed by an event-based logical behavior. The RTs' logical behavior is specified by the ESTERELcode associated with the RTs which follow a generic sequence of initialization/computing/termination phases. Furthermore, at the application level the control code of all the underlying RTs and Procedures are gathered in a control automaton, which must be interfaced both with the data flow control structures and with the target operating system. Hence glue code is also generated to automatically provide the interface between the control application components.

As depicted in Figure 10a the main components allowing to run a specified ORCCAD application are :

- The Application Automaton (AA). It gathers all the ESTEREL control code, inside a particular, high priority real-time task;
- An event queue. It collects the events and exceptions generated during the execution of the controller; the AA is most often in a suspended state, waiting for events, it is awaken by the occurrence of events in the queue.
- In reaction to incoming events, the AA output function triggers actions of the controller. To this end the execution of the RTs phases are locked/unlocked by semaphores releases by the AA.
- The real-time data flow execution (the modules' functions and drivers) is implemented and runs using the features of the target operating system, in particular of according to its scheduler and other services like programmable timers. Status flags record the execution state of the real-time tasks, e.g. run-time overruns are detected and yield.

The sequence of events used to manage the life of a RT, together with the transition between successive RTs, follows the systematic scheme of Figure 10b, to ensure, e.g., that a control task can be started only when correctly initialized, to avoid conflicts between controllers and to ensure a safe termination of a controller whenever possible as stated in [25]. Thanks to the RT and RP models, and to the typing of events and exceptions, the set of run-time components used to implement the application control code is automatically generated.



Robot Task 1	Robot Task 2	Switching signals from/to automaton	Comments
•	→	post−condition or type 2 exception	Decision for switching to RT2
←	•	Transite	
	←	Resume	Resume real−time tasks (MTs)
	•	Resume_ok	
	←	Init	Task initialization Checking for pre−conditions
	•	Init_ok	
←	•	Control_stop	Stop of control application
•	→	Control_stop_ok	
	←	Control_start	Start of control application
←	•	Suspend	
•	→	Suspend_ok	

Figure 10: a) run-time architecture b) event based execution model

5 Case study : a *quadrotor* Drone

A *quadrotor* is a typical example of an embedded system whose safety is critical [2]. It consists of a mini helicopter, the four blades of which are actuated by four identical DC motors. It is equipped with an Inertial Measurement Unit (IMU) for system attitude positioning (Figure 11a). The quadrotor is a partially redundant structure, excellent to apply diagnostic methods and to design Fault Tolerant Control schemes. The initial phases of system modeling and control

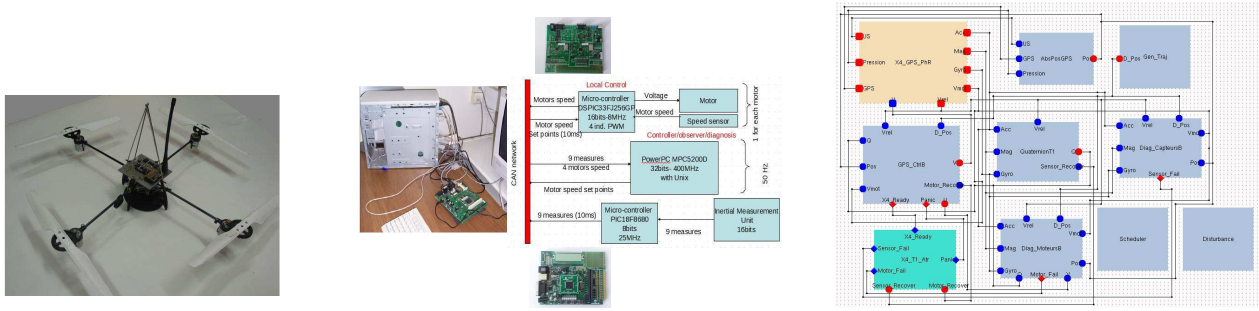


Figure 11: a) The quadrotor b) Hardware-in-the-loop c) Control design

design used modeling and simulation tools like Matlab/Simulink, which do not provide the efficient real-time, multi-tasks and multi-rate code needed to implement the embedded software. To provide more realistic results on the influence of the real-time implementation (including a CAN network) on the system, a hardware-in-the-loop experiment has been set up (Figure 11b) to make an additional step before testing with the real drone. Figure 11c describes the control and diagnostic setup used for testing purposes. In this block-diagram, the blue boxes represent the user-provided modules (i.e. functions) interconnected by their input/output ports (respectively blue and red dots).

The main parts of the functions network are described as follows :

- The attitude control path starts from the drivers of the quadrotor sensors (accelerometers Acc, rate gyros Gyr and

magnetometers Mag), which are the outputs of the interface module *X4-GPS-PhR*. The box *X4_GPS_PhR* (where PhR stands for Physical Resource) is the interface between the controller and the device to control : sending or reading data on its ports actually calls the drivers, i.e. the functions used to interface the real-time controller with the real hardware, or with the real-time simulator. The raw measurements are used by the Quaternion module *QuaternionT1* to estimate the drone attitude, which is forwarded to the *GPS_CtrlB* module to perform the attitude control. The computed motor velocities that are desired are then sent to the quadrotor via the V port;

- Provision is given for future enhancements of the sensor set, since a GPS-like position sensor and ultra-sonic sensors are expected to be integrated in the future. Therefore, a trajectory generator module *GEN_Traj* and a position estimator module *AbsPosGPS* are integrated in the control architecture to evaluate position control;
- The *Diag_CapteursB* module runs the diagnostic algorithm that isolates sensor failures. A failure is signaled by the Sensor_Fail weak exception to the X4_T1_Atr module and it is forwarded to the Quaternion module, so that the quaternion estimation algorithm can be adapted according to the reported failure;
- Similarly, the Diag_MoteursB module forwards motor failures to be handled by the X4_T1_Atr module;
- The Scheduler module implements a feedback scheduler : it monitors the controller's real-time activity and may react by setting on-the-fly the task-scheduling parameters, e.g. their firing intervals. For example, it can been used to implement a (m,k)-firm dropping policy [18], and to dynamically adapt the priorities of messages on the CAN bus [19].
- A Disturbance task allows users to generate an extra load either on the CPU or on the CAN bus, or to generate corruptions on the CAN bus.

Both computers communicate via a CAN bus : the driver ports located in the *X4_GPS_PhR* interface send and receive data using the Socket-CAN protocol[2]. From the real-time point of view, each module is implemented by a real-time task having its own programmable timer, with the exception of the *GPS_CtrlB* module which is triggered by the output of the Quaternion estimator to minimize the latency on the critical control path. All the other modules can be run asynchronously at their own (and possibly varying) sampling frequency. The task priorities are set according to their relative importance.

As the new version of Orccad design and generation tools are not yet fully-functional, some parts of this code has been written manually. In fact all the design process used the models defined in the ORCCAD approach, so that this kind of application is used as testbeds for the new run-time features and code generation tool. In particular, it lead to successfully design and run a real-time multi-tasks/multi-rate system with variable clocks and asynchronous communication. The Fault Tolerant Control application benefits from the on-line exception handling and on-line re-parametrization/re-configuration capabilities of the Orccad model. The new version of the run-time has been easily and successfully ported on different architectures (PC and PowerPC), under different operating systems and API (Linux and Xenomai), and UDP and CAN sockets have been integrated in the drivers library.

6 Summary and perspectives

The ORCCAD model allows to structure robot architecture experiments and the construction of robotic programming tools. In this paper, we have shown that Eclipse Modeling Tools are well adapted to provide a complete CASE robotics tool-set to support this model from design to code generation. This approach allows to focus on the model rather than the infrastructure.

Reusing the already existing and still fruitful Orccad model, a new Integrated Design Interface dedicated to the design and implementation of real-time control software, has been developed based on modern tools. It allows to work out all the steps of control design, from elementary functions specification until run-time code generation, inside the coherent framework provided by Eclipse.

The new modeling framework is entirely based on open-source tools, powerful editors and software management process. Therefore, even if the current experience with this software is limited, it is expected that feedback from experience and new features requirements could be easily and safely integrated.

Some features which were in former Orccad tools are not yet re-implemented or updated. This is for example the case for the Maestro mission level design language, which could be re-engineered, e.g. using the Kermeta workbench.

[2] `http://developer.berlios.de/projects/socketcan/`

Alternatives to the ESTEREL programming language for RTs and RPs design are also investigated, in particular with the Heptagon synchronous platform and its BZR extension, allowing for discrete events control synthesis, which is described in a companion paper [1].

References

[1] Soufyane Aboubekr, Gwenaël Delaval, Roger Pissard-Gibollet, Eric Rutten, and Daniel Simon. Automatic generation of discrete handlers of real-time continuous control tasks. In *5th National Conference on Control Architecture of Robots CAR'10*, Douai, May 18-19 2010.

[2] C. Berbra, S. Gentil, S. Lesecq, and D. Simon. *Co-design approaches for dependable networked control systems*, chapter 7 : Implementation: Control and Diagnosis for an Unmanned Aerial Vehicle. ISTE-Wiley, 2010.

[3] G. Berry. The esterel v5 language primer. Technical report, INRIA, http://www-sop.inria.fr/esterel.org/, 2000.

[4] J.J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The ORCCAD architecture. *Int. Journal of Robotics Research*, 17(4):338–359, april 1998.

[5] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback. Towards component-based robotics. In *Intelligent Robots and Systems IROS*, pages 163–168, august 2005. http://orca-robotics.sourceforge.net/.

[6] H. Bruyninckx. Open robot control software: the OROCOS project. In *IEEE International Conference on Robotics and Automation ICRA*, Seoul, may 2001. http://www.orocos.org/.

[7] J-L Colaço, B. Pagano, and M. Pouzet. A conservative extension of synchronous data-flow with state machines. In *ACM International Conference on Embedded Software (EMSOFT'05)*, Jersey city, New Jersey, september 2005.

[8] R. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Eclipse Series. Addisson-Wesley, 2009.

[9] Graphical editing framework (gef) project. http://help.eclipse.org/help33/topic/org.eclipse.gef.doc.isv/guide.html.

[10] http://openembedd.org/.

[11] Xpand website. http://wiki.eclipse.org/Xpand.

[12] Cdt website. http://www.eclipse.org/cdt/.

[13] Emf technology ocl project. http://www.eclipse.org/emft/projects/ocl.

[14] Model to text (m2t) website. http://www.eclipse.org/modeling/m2t/.

[15] http://www.kermeta.org.

[16] http://www.omgmarte.org/.

[17] http://www.robotics.utexas.edu/rrg/research/oscarv.2/.

[18] N. Jia, Y.-Q. Song, and F. Simonot-Lion. Graceful degradation of the quality of control through data drop policy. In *European Control Conference, ECC'07*, Kos, Greece, July 2007.

[19] G. Juanole, G. Mouney, and C. Calmettes. On different priority schemes for the message scheduling in networked control systems. In *16th Mediterranean Conference on Control and Automation*, Ajaccio, France, July 2008.

[20] F. Kanehiro, H. Hirukawa, and S. Kajita. OpenHRP: Open architecture humanoid robotics platform. *International Journal of Robotics Research*, 23(2):155–165, 2004.

[21] K. Kapellos. *Environnement de programmation des applications robotiques réactives*. PhD thesis, Ecole des Mines de Paris, Sophia Antipolis, 1994.

[22] K. Kapellos, F. Chaumette, M. Vergauwen, A. Rusconi, and L. Joudrier. Vimanco: Vision manipulation of non-cooperative objects. In *9th ESA Workshop on Advanced Space Technologies for Robotics and Automation ASTRA'06*, Noordwijk, The Netherlands, november 28-29-30 2006. ESTEC.

[23] K. Kapellos and F. Didot. Formid: A formal specification and verification for dreams. In *8th ESA Workshop on Advanced Space Technologies for Robotics and Automation ASTRA'04*, Noordwijk, The Netherlands, november 2-4 2004. ESTEC.

[24] K. Kapellos and F. Didot. Telemanipulation in the dreams ground control station. In *9th ESA Workshop on Advanced Space Technologies for Robotics and Automation ASTRA'06*, Noordwijk, The Netherlands, november 28-29-30 2006. ESTEC.

[25] K. Kapellos, D. Simon, M. Jourdan, and B. Espiau. Task level specification and formal verification of robotics control systems: state of the art and case study. *Int. Journal of Systems Science*, 30(11):1227–1245, 1999.

[26] K. Kapellos, D. Simon, R. Pissard-Gibollet, and B. Espiau. The orccad robot control architecture and tools in space applications. In *9th ESA Workshop on Advanced Space Technologies for Robotics and Automation ASTRA'06*, Noordwijk, The Netherlands, november 28-29-30 2006. ESTEC.

[27] D. Kortenkamp and R. Simmons. Robotic systems architecture and programming. *Springer Handbook of Robotics*, pages 187–206, 2008.

[28] OLC team M. Pantel, ACADIE team. The topcased project, a toolkit in open source for critical applications and systems design. In *TOOLS EUROPE / Model-Driven Development Tool Implementers Forum (MDD-TIF*, 2007.

[29] A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, , and D. Apfelbaum. CLARAty: Challenges and steps toward reusable robotic software. *submitted to International Journal of Advanced Robotic Systems*, 2006. http://keuka.jpl.nasa.gov/main/.

[30] B. Patin P. Martinet. Proteus: A platform to organise transfer inside french robotic community. In *3rd National Conference on Control Architectures of Robots (CAR)*, May 2008.

[31] R. Passama and D. Andreu. COSARC : Component based software architecture of robot controllers. In *1st National Workshop on Control Architecture of Robots: software approaches and issues*, Montpellier, 2006.

[32] C. Samson, M. Le Borgne, and B. Espiau. *Robot Control: the Task-Function Approach*. Oxford Science Publications. Clarendon Press, 1991.

[33] D. Simon and F. Benattar. Design of real-time periodic control systems through synchronisation and fixed priorities. *Int. Journal of Systems Science*, 36(2):57–76, 2005.

[34] D. Simon, E. Castillo, and P. Freedman. Design and analysis of synchronization for real-time closed-loop control in robotics. *IEEE Trans. on Control Systems Technology*, 6(4):445–461, july 1998.

[35] D. Simon, K. Kapellos, and B. Espiau. Control laws, tasks and procedures with ORCCAD: Application to the control of an underwater arm. *Int. Journal of Systems Science*, 17(10):1081–1098, 1998.

[36] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design: application to robot control. In *RTAS'05 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 118–127, San Francisco, March 2005.

[37] H.-R. Simpson. Multireader and multiwriter asynchronous communication mechanisms. *IEE Proceedings-Computer and Digital Techniques*, 144(4):241–244, 1997.