An UML based design tool for Fault-Tolerant Real-Time Linux

Agnes Lanusse, Patrick Vanuxeem, Yann Tanguy And Sebastien Gerard
CEA - SACLAY
LIST/DTSI/SOL/L-LSP
91191 GIF-YVETTE CEDEX
FRANCE
{agnes.lanusse, patrick.vanuxeem}@cea.fr
{yann.tanguy, sebastien.gerard}@cea.fr

Abstract

Real-time embedded systems used to be hand tailored to fit requirements and constraints of applications. But the increasing complexity of applications and the rapid evolution of hardware make these practices unbearable to face nowadays industrial reality with very fast evolution of requirements. The development process of real-time embedded applications must be supported by specific tools to facilitate their reuse, evolution and maintainability. Initiatives based on UML-RT provide such tools to model applications and automatically generate code. In this paper we present one such approach providing a simple domain model for the design of Fault-Tolerant Real-Time applications. A first implementation has been achieved with the OCERA-Fault-Tolerant framework developed within the OCERA IST project as target. This framework is a specialized framework based on RTLinux.

1 Introduction

The growing demand for embedded systems in the industry has drastically changed the conditions of development of real-time embedded systems. The introduction of electronic and software in very competitive domains such as automotive industry or mobile devices among others, with pushing demand for new services at ever reduced costs and shorter time to market has two main consequences. On one hand, there is an emerging need for off-the-shelf small, efficient, reliable realtime operating systems, and, on the other hand, for new development processes that facilitate scalability, evolutivity and maintainability of embedded systems. Moreover, the challenges faced by industry put more and more stress on high level system correctness and ask for verification tools fully integrated in the development process.

The first point is a good opportunity for the real-time Linux community which has been very active in recent years providing new dedicated kernels such as RTLinux [20], OCERA [23], RTAI [22], L4 [21] and others as well as in providing new facilities in Linux mainstream such as Linux-HA[24]. Current users of such systems,

specialists programmers have been successful in developing applications that have demonstrated the level of maturity and efficiency of these kernels.

But these are generally hand tailored applications that require skilled programmers working directly at language level whereas industry is demanding for high level software design environments that would support large scale development and rapid adaptation to various target environments (hardware + programming language + OS).

Model-driven engineering, an active trend in software engineering, has led recently to the emergence of new tools supporting model-level design and automatic or semi-automatic code generation. We believe that the linux kernel community can take benefit from these results to get a chance to promote the use of real-time kernels by a wider range of developers.

Moreover, with a close cooperation between the two communities, in particular with the UML real-time community, it will be possible to set up efficient and safe patterns of implementation for real-time embedded applications, hence leading to active support for better practices in the domain.

In this paper we present a small example of model-based code generation that permits to implement simple fault-tolerant applications in OCERA

RTLinux starting from UML modeling. The prototype described here produces RTLinux C code and uses specialized middleware insuring transparent fault-tolerance handling. The goal of this experimentation is to show the feasibility of the approach and to put the basis of a methodology for a wider and richer code generation process.

In the next section we introduce the context of the experimentation, the framework used in OCERA to support fault-tolerant applications development. Then, in section 3, we describe the methodological approach chosen and the tools developed to support it. In section 4, we illustrate the process of code generation on a simple example. Finally we show how the methodology can be extended.

2 Experimentation context

In order to test and demonstrate our approach we needed a small but complete target real-time platform; we choosed the Framework for Degraded Mode Management developed OCERA (Open Components for Embedded Real-time Applications). The goal of this IST project was to provide improved support to real-time embedded applications based on Linux kernel.The OCERA architecture is based on Linux kernel and RTLinux executive which provides support for real-time applications that require very demanding response time. The added facilities offered cover four categories: Scheduling (EDF, CBS, Application Defined Scheduler), QoS(Resource Reservation Scheduling, Feedback Scheduling), Fault-Tolerance (Degraded Mode Management, Redundancy Management) and Communication (ORTE real-time Ethernet, RTCAN). In addition, an extensive work has been done in order to improve POSIX compliance. Thus many Posix basic facilities have been added to RTLinux (POSIX Timers, POSIX Barriers, PosixMessageQueues, POSIX Tracing...). Support for other languages then C such as Ada and Java have also been provided.

The different components developed cover a large range of real-time needs and can be applied to different Linux/RTLinux levels. The degraded mode management framework relies on specific middle-ware components named hereafter FT components (*ftappmon* and *ftcontroller*) that run within the OCERA RT kernel.

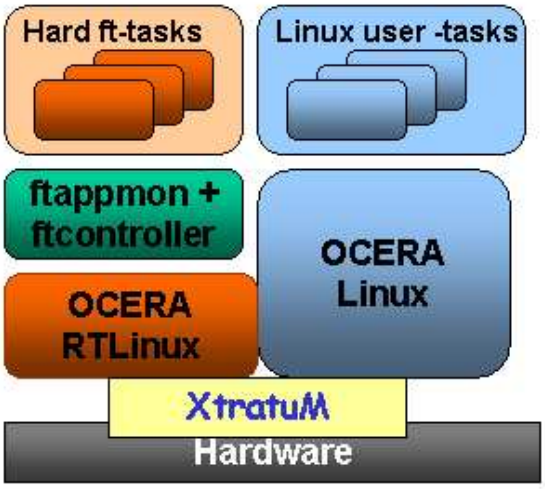

FIGURE 1: *FT components in OCERA kernel*

Its goal is to help real-time developers to design and implement dynamically reconfigurable applications. It is restricted for the moment to the implementation of graceful degradation management of applications but could be easily extended to more general multi-mode management.

This framework constitutes a perfect testcase: first of all, it is a typical example of a domain where separation of concerns during design is essential; secondly, it relies on a well defined simple task model, yet it provides abstractions that are far enough from direct OS coding to make model-driven approaches interesting; finally, it runs on the OCERA real-time Linux kernel, so it is a good example of target specific coding requirements.

2.1 The OCERA Degraded Mode Management Framework

OCERA FT components are providing user support for the implementation of embedded real-time fault-tolerant applications. Among all the numerous issues related to fault-tolerance, a selection has been made of general basic mechanisms that we believe can be the most useful for the development of small real-time embedded systems.

In this paper we focus on the facility offered by the Degraded Mode Management Framework. It has been designed to offer transparent management of dynamic reconfiguration of applications on detection of faulty situations (software or temporal, i.e. deadline miss, errors). Continuity of service is maintained in case of partial failure through graceful degradation management. This facility covers not only primary/backup recovery, but also global application mode management[6], [7].

The Degraded Mode Management Framework is an integrated set of tools and components offering :

- Off-line application design support;
- Code generation;
- FT-API;
- Run-time components (ftappmon and ftcontroller).

2.1.1 Design/build tool

The *ftbuilder* permits the specification of application entities, real-time constraints, different possible application modes along with related transition conditions. From this specification, code generation is achieved in order to instantiate internal control databases of run-time components (*ftappmon* and *ftcontroller*) and to provide application model files.

2.1.2 Asumptions and Tasks Model

Model characteristics will be detailed in the next section but here are the main points:

- Periodic tasks
- Synchronized communication model through shared data. (One writer/several possible readers. A reader task reads values emitted during previous period of writer task).
- Two possible behaviors (normal and degraded) for each task are defined by the developer. These behaviors are implemented in two alternative threads that are activated or suspended depending on application mode.

2.1.3 FT-API

A specific but reduced API has been defined for manipulating so called ft_tasks. It provides three main functions: ft_task_init(), ft_task_create, ft_task_end(). These ft_tasks are actually encapsulation of periodic RTLinux tasks. Other functions are used to init internal data-bases. Besides these specific functions, application developers can use RTLinux programming features.

2.1.4 Run-time components

The architecture proposed for the Degraded Mode Management Framework in OCERA consists of two complementary components: *ftappmon* and *ftcontroller* that insure transparent handling of fault-detection, recovery mechanisms and mode management. They provide respectively global monitoring of application and local control of execution.

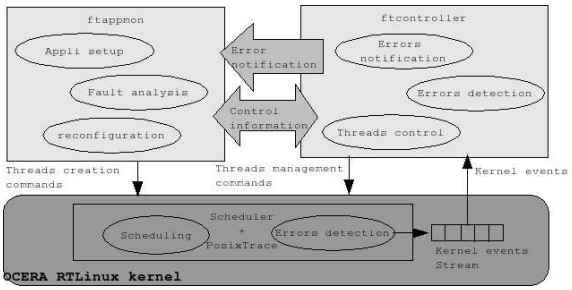

FIGURE 2: *OCERA FT architecture*

- *ftappmon*

  The *ftappmon* component is devoted to global application handling. It is in charge of overall application setup and of reconfiguration decisions. It contains information on different possible application modes and on transition conditions. When an error is detected and notified by the *ftcontroller*, the ftmonitor analyzes the event and issues reconfiguration orders (stop, awake, switch ft_task behavior) towards the *ftcontroller*.

- *ftcontroller*

  The *ftcontroller* is in charge of the direct control of application threads. It provides: error detection (kill or timing error), notification towards *ftappmon* and executes reconfiguration orders (issued by *ftappmon*) at task level.

These two components are linked together in a module *ftappmonctrl* that is loaded into the RTLinux kernel space.

2.2 FT Model

The framework currently available relies on a simplified model of applications. According to this model only applications with periodic tasks are handled (at the moment). Though these are indeed quite restrictive hypotheses, they represent a large range of effective current real-time embedded applications.

2.2.1 Application modes

The application is defined as having possibly several modes of execution. These are predefined at design and specify the possible degraded modes of functioning. This choice is similar to OSEK/VDX.

An application mode defines a specific configuration of active tasks. That is, the specification of the tasks that must be active when the mode is selected along with the relevant behavior for each one. This results in a list of pairs (task,behavior). More

precisely, a concept of *ft_task*, which provides an abstraction layer above the tasks or threads entities manipulated by operating systems has been introduced. It will be described in details in the next section.

The init mode is the mode in which the application will be started (initial configuration). Transitions between modes are triggered by the detection of specific events. In the current implementation, the possible triggering events are: *Kill* and *deadline_miss* events. It is planned that, in the future, *User* events can also be triggering conditions for mode change.

Transition conditions and target mode activation are explicitely specified statically during design. All this declarative information is gathered into an application model that is loaded at the beginning of the application into the run-time components. Dynamic reconfiguration can then be handled automatically according to these specifications when a faulty event occurs during execution. It is important to notice that the application behavioral control is totally separated from tasks coding which means that changes in requirements from the behavioral point of view does not impact application code.

2.2.2 FT_Tasks

As said earlier, an application consists of a set of periodic tasks: the *ft_tasks*. These *ft_tasks* are abstract entities that encapsulate real threads implementation.

ft_task scheduling parameters

Real-time attributes are assigned to each *ft_task* such as period, ready-time, expected duration (WCET), deadline and the scheduling policy to be applied (EDF or Priority policies are possible).

ft_task behavior

From the designer point of view, a *ft_task* may have several possible behaviors. In the current implementation, two behavior definitions are expected : a *NORMAL behavior* and a *DEGRADED behavior*.

From the implementation point of view, a behavior relates to a routine that will be executed when the *ft_task* will adopt this behavior. So the designer has to define two behavior routines for each *ft_task*.

The implementation model of a *ft_task* consists in creating two threads, one for each possible behavior. On application initialization, two threads are created and suspended for each *ft_task*; then the behavior corresponding to the behavior that is relevant for the application mode selected is made running. When an error occurs that implies mode change and consequently behavior change for the *ft_task*, the running behavior thread is killed and the suspended thread is made running (waiting for the next period of the *ft_task*).

ft_task communication

Within the application, *ft_tasks* do not communicate directly between each other. They use a specific synchronized model communication model based on a shared data concept. They can thus be considered as independant. Communication between *ft_tasks* are restricted to data exchange on a cyclic basis with an observability horizon of one period. Data produced by a *ft_task* are updated at each end of the execution cycle and made readable for client *ft_tasks* at each start of new cycle. There is only one writer, the data owner. Client *ft_tasks* read the data elaborated during the previous period. If a fault occurs during one cycle, a default value based on data elaborated during previous cycle can be provided.

No other synchronization is defined between *ft_tasks*.

2.2.3 Error events

Application modes transition are triggered by the detection of an abnormal event (software or temporal error). The current handled events are the following: *KILL* event resulting from the detection of a thread abortion by the kernel due to a software error; *DEADLINE_MISS* event resulting from the detection of a deadline_miss. This event is currently issued by the OCERA EDF scheduler (deadline is detected by the scheduler).

2.2.4 Application mode transition

On detection of one of the above abnormal events, the application mode can be automatically shifted to an other mode. This transition is defined at design time by the application developer. An application mode transition is defined by a triggering event, the *ft_task* on which the event occurs, the source application mode, the target application mode. When fired an application mode transition has the following effect : the termination of *ft_tasks* that are specified TERMINATED in the target application mode; the change of behaviors of all the *ft_tasks* that are present in the target application mode with a different behavior; The start of *ft_tasks* that were created but NOT_STARTED in the current application mode and that must be active in the target application mode. Application consistency during mode change is preserved thanks to the periodic functionning of the application and to the synchronized communication model. Of course, the definition of the different application modes and transition conditions must be carefully designed in order to get a consistent application model.

2.3 FT development process

The development process proposed to the application developer follows three steps:

- Application design is achieved interactively using the OCERA *ftbuilder* tool. The user describes tasks, behaviors, modes and mode transitions. From these descriptions two files are generated : ft_appli_model.c and ft_appli_model.h.
- User Coding is done manually by the application developer. It consists mainly in writing the code of routines for the application threads identified during previous step.
- The third step Compilation combines files issued by the two previous steps and links it with OCERA ft components.

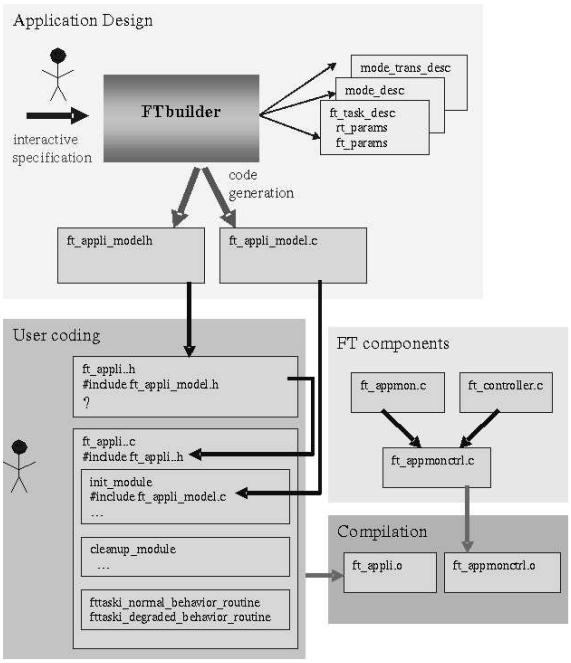

FIGURE 3: *Application development process*

The *ftbuilder* is a TCL/TK acquisition tool designed to provide user support for the implementation of embedded real-time fault-tolerant applications. It is a major element in the Degraded Mode Management Framework. It permits : the specification of application real-time constraints, different possible application modes along with related transition conditions and code generation facilities.

The *ftbuilder* permits the description of ft-applications in terms of :

- *ft_tasks*. Acquisition of *ft_tasks* entities and their associated features : real-time parameters (period, ready-time, expected duration, deadline); and behaviors (normal and degraded).
- *Application modes*. An application mode is a particular configuration of *ft_tasks*, i.e. the specification of the relevant behavior for each *ft_task*.
- *Application mode transitions*. An application mode transition is described by a triggering event (kill or deadline_miss), the faulty *ft_task*, the source mode, the target mode.

From the specification, code generation is achieved in order to provide application model files that will be used to instantiate internal control databases of run-time components (*ftappmon* and *ftcontroller*).

3 Towards UML based code generation

The recent advances in software engineering are converging towards a common idea, that systems should be designed at a model level, independant from implementation languages and that code can be generated afterwards in the most appropriate language depending on the target platform. In this context, the notion of target platform can be understood at different levels of abstractions, one can for example, address first an execution model, than a virtual machine and finally several target implementations of the virtual machines or he can directly address a particular machine with a given Operating System.

This approach has many advantages, it permits to :

- support separation of concerns which is a very important design issue in particular for fault-tolerance;
- support early validation from model analysis;
- support traceability of model evolutions;
- maintain several implementation streams from a single model.

The interest of many software communities has led to the definition of the OMG MOF and XMI standards and to new open tools that provide means for describing systems, manipulate models, transform them and produce code. Thanks to such tools it can be possible to adapt models to particular application domains and specialize the develoment process for this domain.

The actors involved in this field are system engineering that promote Architecture Design Languages

such as AADL for avionics systems or EAST-ADL for the automotive industry, and UML community.

In this paper we adopt a very basic approach of such methodology based on a subset of UML2.0 notations and on the tools provided by the Eclipse workbench. This first experience will be completed in the future in order to go further and take full advantage of the new facilities offered by UML2.0 [10] and its profiles related to real-time: UML Profile for Schedulability, Performance, and Time [12]; UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [11].

3.1 Method and tools adopted

Our goal is to transpose within the MDA paradigm, the development process defined for the design and implementation of real-time fault-tolerant applications in OCERA-FT.

Actually, the idea was to replace the *ftbuilder* in a first place, and enrich it so that we could produce more complete code and/or achieve more sophisticated verification at design stage. The choice of this test case was dictated by the fact that the underlying task model was a simple one, and thus the code generation component could be rapidly written. This has been verified, and we have been able to build a first demonstrator in quite a short time which provides already more than the previous *ftbuilder* did.

We present hereafter, the methodology adopted and the tools used.

Three main steps have been followed :

- domain modeling, i.e. provide an UML2.0 representation of the OCERA-FT Model described in the previous section;
- realization of a specialized editor for the modeling of OCERA-FT applications starting from this UML FTModel;
- development of a code generator which browses a user application model and produces corresponding application code conform to the specified FT code structure.

Normally in a full MDA approach, the last step would require the definition of a model for the target platform along with the definition of implementation patterns. For this first experimentation, code generation has been directly written for the specific OCERA target.

3.1.1 UML Modeling

The goal of this stage has been to define a domain meta-model corresponding to the FT conceptual model described above. A basic UML model consists generally in at least three types of diagrams describing complementary aspects of the design: the structural model, the interaction model and the behavior model.

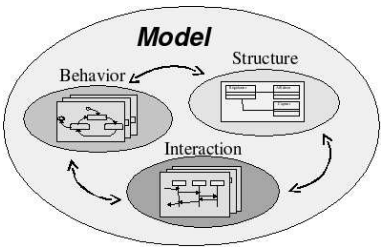

FIGURE 4: *Basic UML Model*

The first one describes the main entities of the application; the second one defines a model of interaction between entities; and the third one describes behaviors expected from entities. Depending on the stage of the development process, other diagrams may be necessary; use case diagrams for instance are very useful in the preliminary analysis notably to capture requirements.

Modeling can be done with any Case Tool supporting UML, while the definition and management of specialized profiles would have needed more advanced tools such as (Objecteering[17], Poseidon[18], RSA[19],...) which provide meta-modeling facilities and good coverage of UML2.0. These are very powerful modeling tools and provide good support for documentation and standard code generation.

However, since we needed to define specialized code generation patterns we looked for a more open environment. We chose thus to rely on the Eclipse environment for its openness, and its powerful EMF and JET facilities. The Eclipse Modeling Framework (EMF) in particular, offers a set of tools that support models design and manipulation, validation process and code generation when associated with the JET facilities (see Eclipse Framework book + site [16]).

Moreover, this environment which is becoming very popular, provides a lot of valuable plugins as well as facilities to develop our own ones.

So, we have used the EclipseUML (Omondo) free plugin which provides graphic facilities to enter models and have made an intensive use of EMF and JET facilities of Eclipse framework for the next two steps. We could have used also the UML2 plugin which is also available from the Eclipse UML2 project and is very complete but doesn't offer a graphical interface for diagrams.

EMF itself has its own core metameta-model named Ecore which is a subset of the OMG MOF (Meta Object Facility) API. All model manipulation tools rely on Ecore, thus UML models are converted to Ecore.

It is important to know that EMF uses XMI (XML Metadata Interchange) as its canonical form of a model definition and offers importation facilities that permit to create an Ecore model from an XMI document, an XMI document produced by a modeling Tool (Rational Rose for instance), annotated Java, or XML Schema. So it is possible to use other modeling tools provided they offer compatible XMI representation.

The result of the UML modeling is shown in the following figure.

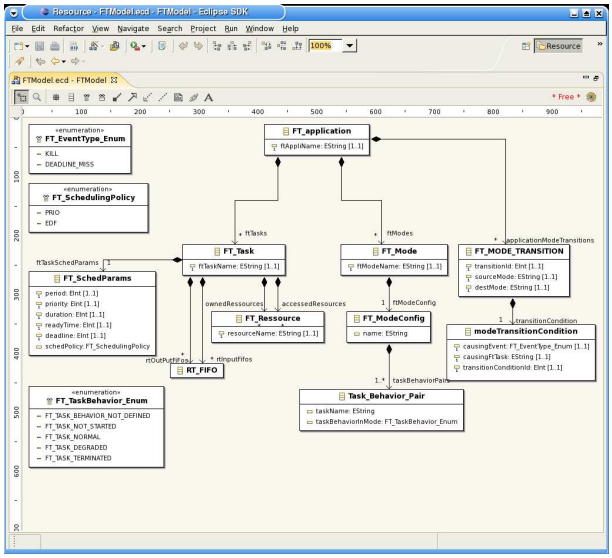

FIGURE 5: *FTModel described in UML*

We recognize, the entities of the OCERA FT Model: an FT_application is composed of FT_tasks, FT_Resources,FT_Modes and FT_Mode_Transitions. FT_tasks have FT_SchedParameters and are associated to FT_Resources.

Though this UML model is restricted to a class diagram; as it is, it represents perfectly the current FTModel.

Though limited, it is sufficient for our purpose in this first experiment. There were several reasons for this choice: first, this mapping is close to what was done with the *ftbuilder* which can be useful for a first comparison; second, the specificity of the model make interaction between *ft_tasks* limited to read operations, synchronization issues are not handled directly by the user but by the underlying implementation model of the *FT_entities*; third, the control data is embodied in the specification of transition conditions described in the structrural view.

Of course behaviors and mode transitions could have been represented with State Machines which is the classic UML mean to represent behaviors, but we decided to postpone such representation to a next version.

The resulting model corresponds to the expected conceptual model of the OCERA FT model for degraded mode management. Starting from this, we should normally build a specialized UML profile, that is an extension of standard UML2.0 meta-model, where added stereotypes on some well chosen model elements can provide a specialized view of UML for describing the features of our conceptual model. We can find a nice example of such approach for the tentative definition of an UML profile for OSEK from Artisan company.

In this experience though, we stick to the domain meta-model as it is, since it does not change the nature of the work to do in the next two steps, and it is sufficient for this experimentation.

3.1.2 FT Model Editor

The FTModel editor has been implemented quite easily thanks to the Eclipse EMF framework. This tool encompass generic predefined plugins that are able to produce java code for a specialized Editor starting from an Ecore model.

So to get an OCERA-FTModel editor, all we had to do was to specify the corresponding UML model, build an EMF project and import the model. Then EMF converts it to Ecore and provides a facility that produces a .genmodel file which is used to produce automatically a Java specialized editor corresponding to the model. It also offers facilities to specify rules that permit to extend generic model validation procedures. This specialized editor is actually, a plugin, the FTModel editor, that once installed within Eclipse provides a specific menu whose actions permit to : add, remove model entities, and validate model.

3.1.3 FT Code generator

The code generator uses the JET technology offered by the Eclipse Framework. It consists of a generic code generation engine that produces java code implementing code templates from user defined Jet Templates.

A simple syntax permits to specify text scripts corresponding to expected code production. JET produces then an equivalent Java template with a generate method.

The idea is then to browse a model and produce proper code corresponding to model entities by calling the various generate methods.

In our case, we had to produce four files: the application source and header files, plus two other files, namely ft_appli_model.h and ft_appli_model.c, which are the files describing global application FT control and are used to instanciate the internal data

bases of the runtime components (*ftappmon* and *ftcontroller*).

The application source code has to verify the standard RTLinux structure, i.e. contain an init-module, a cleanup module and routines to be run within RTLinux threads. So we had to define an implementation pattern that follows this structure and provides a compliant application skeleton. Moreover, since threads manipulation operations in our case are encapsulated within the primitives defined in the FT_API; at code generation, these primitives are used instead of direct RTLinux primitives.

To summarize we have developed two Plugins integrated into the Eclipse workbench : the FTModel editor and the FT2OCERA code generator.

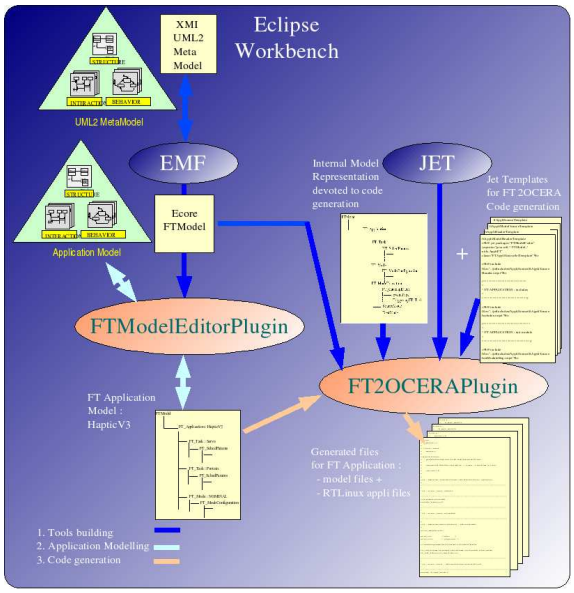

FIGURE 6: *Eclipse workbench and added Plugins*

Let's see now how we can use them for developing a simple application.

4 Building a simple application

We have chosen to test the development process with a very simple example which is a simulation of a real robotic application. This application has two periodic tasks : a *servo task* which is a standard servo controller and a *proto_in* task which is used to receive feedback from the environment.

The development process consists in the following steps:

- specify the application model using the FTModel editor plugin;
- validate the model;
- generate corresponding code skeleton;
- complete manually operation code;
- compile and link with OCERA-FT.

The FTModel Editor offers an interface permitting to specify the model elements of the FT application and to save it under the form of an Ecore model. (It is possible to define a more convenient user interface using Eclipse GEF plugin, but this is not done yet in our example).

The designer defines the tasks, their scheduling parameters and data ressources. Then he defines the modes definitions and the mode transition conditions.

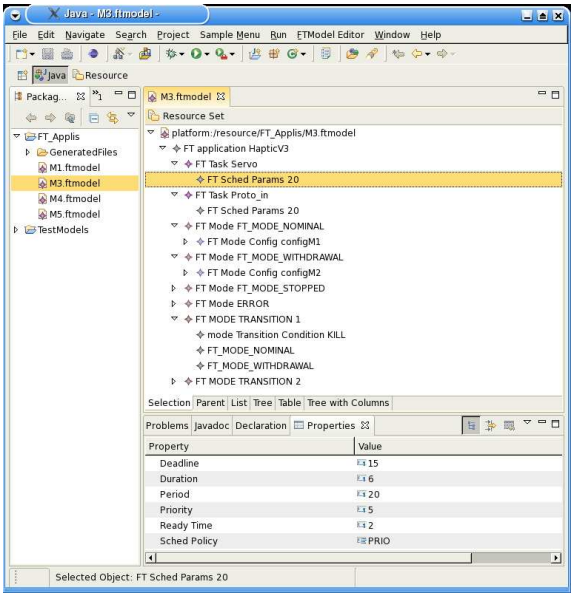

FIGURE 7: *Defining application Model with FTModel editor*

Once the model is entered, it is possible to check it's validity according to certain criteria. For the moment only structural properties of the model are checked, but the framework offers the possibility of adding rules to extend this verification process.

In the future it will be also possible to link the model with other tools such as UML MAST to perform schedulability analysis. Indeed FT_sched_parameters as defined are already compatible with the specification of SAction in the SPT profile defined by the OMG for schedulability analysis.

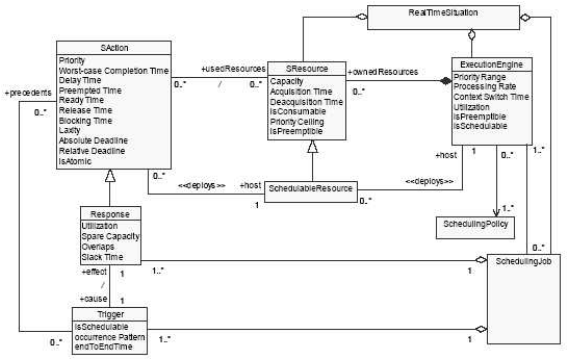

FIGURE 8: *SPT profile for schedulability analysis*

When, the model is stable and valid, the code can be generated.

Just activate the generate action in the Eclipse bar menu and the FT2OCERA code generator produces the files corresponding to the application.

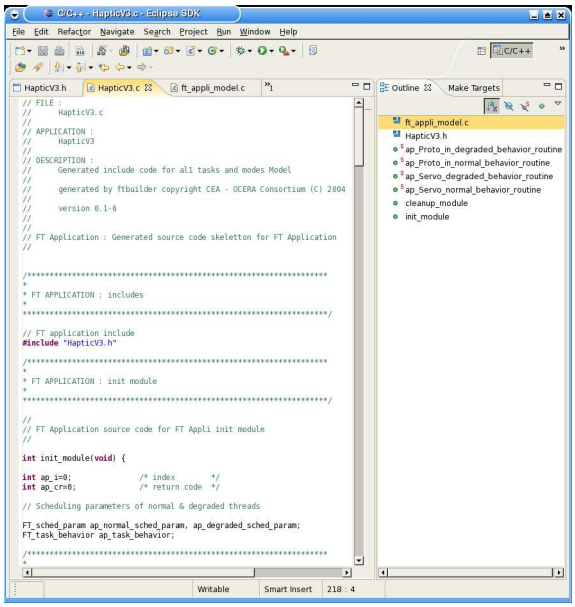

FIGURE 9: *Code Generated by FT2OCERA Plugin*

The code generated can be compiled and linked against OCERA FT-API. We obtain an application module that can be loaded in the kernel. The code produced is executable without any other manipulation. Of course, if we want the application to perform something interesting, the developer has to add its code within routines since this step is not automated. But, the overall control structure that implements the application global behavior is fully generated and can thus be changed quite easily using the model editor.

5 Going further

The process described in the previous section produces almost all the application code along with connection with underlying middleware components that insure transparent fault-tolerance facilities.

At this stage, this environment is very powerfull since any change at high level design can immediately be propagated and the code regenerated.

However, there is still a step that cannot be handled automatically, i. e. the actual code of operations has to be entered manually.

This is a hole in the global objective of having a model independant from the implementation. Moreover, this implies that the model itself is not complete, and that possibilities of verification or validation are limited[1]. Worse, user programming of routines might compromise the soundness of the model.

One solution to this problem is to provide an action language, that can be used to describe operation code at the modeling stage. Such approach has already been used in the past notably with SDL for the modeling of asynchronous communicating systems. Tools such as Tau from Telelogic could generate equivalent C code for target implementation.

Similar initiatives have been taken recently within the UML community with the definition of an action language which provides means for describing operations using predefined action types. A graphical syntax has been defined which can be used within Activity diagrams.

We present in the next section an example of this approach through the ACCORD/UML experience developed in our laboratory.

5.1 The ACCORD/UML process example

The ACCORD/UML methodology has been proposed several years ago by the CEA for the prototyping of real-time embedded systems[5]. It covers a full design process starting from preliminary analysis to full implementation. It uses UML2.0 notation to describe the models manipulated all along the design and development process. Particular entities have been defined to handle signal management and concurrency control. Thus, the notion of real-time object is an abstraction that permit to implement active objects able to handle real-time constraints on requests and to react to signal events according to specified potocols.

The design process is supported by specific metamodeling and code generation developed for several Case Tools: Objecteering, Poseidon, RSA. It is completed by the ACCORD runtime platform on which the application model is applied. It consists of a core

component that implements high level concepts such as real-time objects and a virtual machine. This virtual machine has been ported on Solaris, Linux and VxWorks.

Applying the methodology for a specific domain consists of three parts: applying specific design patterns relating to real-time issues; full code generation (structure + behavior) towards the Accord runtime platform; the Accord platform itself implementing high level concepts of the methodology.

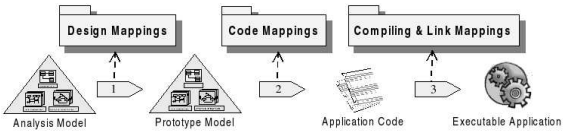

FIGURE 10: *ACCORD/UML Process*

5.2 From models to code in ACCORD/UML

The OMG UML2.0 standard defines a specific package which provides syntax and semantics for the description of application Actions[9]. With this facility, it is possible to specify an Action language that will permit to produce an executable model. Actually, the Action language gives an abstract form for describing code.

A subset of these actions are being used in the ACCORD/UML modeling framework and have been specialized so that each basic Action in the framework comes with a graphic syntax and an equivalent textual form.

The main basic actions handled are: ConditionalAction, SwitchAction, SendSignalAction, BroadcastSignalAction,CreateObjectAction; and are used within Activity diagrams to describe the code of operations.

A first implementation of this proposal is available within the ACCORD platform and permit to produce C or C++ code[3],[4].

Moreover, code for different optimization purposes can be generated from the same model. Currently, the user can manually make the choice of a code generation pattern.

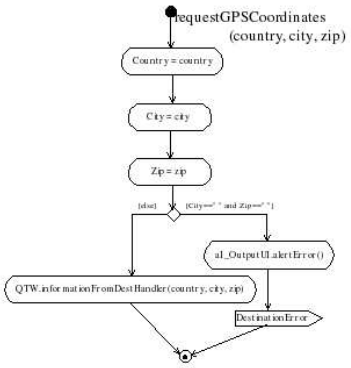

FIGURE 11: *Example of Action Language in Activity Diagram*

The code generated from this diagrams in C++ gives the following.

```
void DestHandler::requestGPSCoordinates(
  String country,
  String city, String zip) {
  Country = country;
  City = city;
  Zip = zip;
    if(City == "" && Zip == "") {
   aI_OutputUI->alertError();
   DestinationError();
     }
    else
     QTW->informationFromDestHandler(country,
              city, zip);
}
```

From the same diagram it is possible to generate a different target language code. Here is the corresponding code generated for C.

```
void requestGSPCoordinates(DestHandler *self,
  gstring country, gstring city, gstring zip) {
  Country = country;
  City = city;
  Zip = zip;
    if(City == "" && Zip == "") {
   alertError(self->aI_OutputUI);
   DestinationError(self);
  }
    else
     informationFromDestHandler(self->QTW,
               country, city, zip);
}
```

5.3 Perspectives

The ACCORD/UML project is a good example of the MDA approach and we can gain a lot from it. Several directions will be investigated:

- improvement of user oriented modeling.

  The next step of our developments will to be to extend our FTModel and redefine it so as to be compliant with UML2.0 new facilities to describe real-time concurrent systems and define a dedicated profile. For example, StateMachines will be used to describe mode transitions. This profile will encompass parts of recent new profiles devoted to specific real-time needs [2]: UML Profile for Schedulability, Performance, and Time [12];UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms[11]. The expected results will be to offer : a full UML Modeling and profile definition for OCERA-FT; a subset of action language to describe operations (and generate code from it).

- improvement of platform modeling.

  In the current implementation, target platform constraints are taken into account directly at the templates level during code generation. This has been done in this first implementation in order to test the feasibility and demonstrate the benefits of model-based approaches. However, the right way to achieve code generation is to rely on formalized coding patterns and target platform. We have thus to work on three main issues:

  - Formalization of implementation patterns (*FT_Tasks*, *FT_SharedData*,...)
  - Formalization of FT components (modeling of ftappmon and ftcontroller)
  - Formalization of RTLinux implementation pattern (modeling of code structure and specific features)

  The FT2OCERA code generator will be rewritten in order to take these models into account. This will permit to envisage the following step that would be to offer various target platforms for an FT application and produce code for RTAI or an other kernel.

6 Discussion

A part from some specific safety-critical domains such as space, avionics, air-traffic control or nuclear power plants, the design process of embedded systems was not very well supported by software tools other than IDEs, cross-compiling and debugging facilities.

Model-based approaches bring support for higher level design environment that can be very promising especially for preserving separation of concerns. They are a way to improve maintainability and portability of applications as well as a very good support for early validation process.

It is one of the reasons why this research domain is very active in the comunities of system engineering that promote Architecture Design Languages such as AADL for avionics systems or EAST-ADL for the automotive industry, and in the UML community. Already several profiles for specialized domains such as avionics or automotive industry have been defined, and new proposals such as SysML [8], an extension to UML to support Architecture design are arising.

An other major interest of the MDA is that it can help produce good quality optimized code. For instance, it is quite easy to produce C code conforming to development guidelines such as for example MISRA C for the automotive industry.

Moreover code generation can be adpated to target platforms so that a same application can be generated for different kernels, in different languages. This is made possible thanks to the separation between what is called Platform Independant Model(PIM) and Platform Specific Model(PSM). The idea is that given a PIM, one can build a PSM using a Platform Model (PM) by successive transformation and/or merging and thus produce code for the final target.

Though this may appear quite futurist to expert real-time developers, it is precisely were we think that the two research communities can gain the most from each other. Real-time development expertise from skilled programmers can be captured in the realization of generic implementation patterns which once tested and verified can be integrated in the development tools. This way, handler programming for instance could be made easier and safer.

In the same manner, with a small cooperation effort with kernel developers to help defining a specialized platform model, use of dedicated kernels could be facilitated, and hence help promote their dissemination.

The ACCORD/UML initiative is one of the many projects devoted to provide generic tools permitting the implementation of such development process. At the moment full C code can be generated.

As said all along this paper, the experimentation done for the OCERA FTModel was just a first experience to test the approach and have some first feedback. This experience has been very positive, and we are ready now to serve a more ambitious objective and work towards a full scale development environment. The example and experience gained

from ACCORD/UML and from other work such as the work done by the TRAME team make us confident in the feasibility of this project.

7 Conclusion

In this paper, we have described an experimentation whose goal was to test the feasibility of model-based approaches for the development of real-time embedded systems. This experimentation has been done on a specific target domain, that of fault-tolerance, and more particularly degraded mode management. We have provided a design environment that permits to produce automatically from an UML model, application skeleton for the OCERA FT target relying on RTLinux kernel.

We are confident that this approcah can be generalized so that it can produce more complete code.

Moreover, a close cooperation with expert kernbel develpers could be very fruitful and permit to provide smart code generation and set up safe scripts templates for tricky coding parts such has handlers.

References

[1] H. Dubois, S. Gerard, C. Mraidha, 2005 *Un langage d'action pour le developpement UML de systemes embarques temps rel.*, Proceeding of IDM05 Paris June 30 - July 1st 2005.

[2] S. Gerard, H. Dubois, H. Espinoza, 2004, *UML2 et ses profils pour le temps-reel*, Proceedings of ETR05.

[3] C. Mraidha, S. Gerard, and Y. Tanguy,H. Dubois, and R. Schneckenburger, 2004, *Action Language Notation for ACCORD/UML.*,CEA Internal Report DTSI/SOL/LLSP/04-163/HD.

[4] C. Mraidha, S. Robert, S. Gerard, David Servat, 2004, *MDA Platform for Complex Embedded Systems Development.*, Proceedings of IFIP Conference on Distributed and Parallel Embedded Systems (DIPES) 2004.

[5] A. Lanusse, S. Gerard and F. Terrier, 1998, *Real-time Modeling with UML: the ACCORD Approach.*, Proceeding of UML98: Beyond the Notation..

[6] A. Lanusse, P.Vanuxeem, 2005, *Towards providing fault-tolerance facilities in RTLinux : the OCERA Degraded Mode Management framework*, Proceedings of RTS Embedded Systems Paris, France 2005

[7] D. Servat, S. Gerard,A. Lanusse,P. Vanuxeem,F. Terrier. Author(s), 2003, *Doing Real-Time with a Simple Linux Kernel*, Proceedings of RTLWS'2003 Valencia Spain.

[8] M.C. Hause, F. Thom, 2005, *Building Embedded Systems with UML2.0/SysML*, Proceedings of RTS Embedded Systems Paris, France 2005

[9] OMG, 2001, *UML Action Semantics*,OMG.

[10] OMG, 2004, *UML2.0 Superstructure Specification*,OMG.

[11] OMG, 2004, *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*,OMG, ptc/04-09-01.

[12] OMG, 2005, *UML Profile for Schedulability, Performance, and Time, v1.1*, OMG, formal/05-01-0.

[13] OMG, 2005, *UML Profile for Modeling and Analysis of Real-Time and Embedded systems RFP*,OMG, realtime/05-02-06.

[14] OMG, 2005, *UML Profile for Modeling and Analysis of Real-Time and Embedded systems RFP*,OMG, realtime/05-02-06.

[15] Eclipse,Eclipse Foundation, ,*http://eclipse.org*

[16] Eclipse Modeling Framework, *http://eclipse.org/emf/*

[17] Softeam,Objecteering, , *http://www.obecteering.com*

[18] Poseidon, Gentleware, ,*http://www.gentleware.com*

[19] Rational Software Architect,IBM, ,*http://www.ibm.com/software/awdtools/architect/swarchitect/*

[20] RTLinux, FSMLabs, *http://www.fsmlabs.com/*

[21] L4, Dresden TU, , *http://os.inf.tu-dresden.de/L4/*

[22] RTAI, RTAI, ,*http://www.rtai.org*

[23] OCERA, OCERA consortium and UPVLC,,*http://www.ocera.org*

[24] Linux-HA, Linux-HA project,,*http://www.linux-ha.org/*