EdROOM. Automatic C++ Code Generator for Real Time Systems Modelled with ROOM.

Polo (1), O. R., De la Cruz (2), J.M., Giron-Sierra (2), J.M., Esteban (2), S.

(1) Dept Arquitectura de Computadores y Automatica. Universidad Europea de Madrid
Villaviciosa de Odon, Madrid, Spain.

Tf: (34) 91 6647800 (721); e-mail: opolo@darc.esi.uem.es

(2) Dept. Arquitectura de Computadores y Automatica, Fac. Fisicas. Universidad Complutense de Madrid

28040 Madrid, Spain.

Keywords: Real-time software engineering, real-time software development, real-time code generation

Abstract

The development of real-time control systems is usually part of a design cycle, involving implementation, experimental control system testing, and re-design. For a faster and better complete development, a software engineering approach, based on a modelling language, is recommended. A successful alternative is ROOM: a modelling language with a graphical syntax. In this paper EdROOM is presented: an environment we developed to graphically edit ROOM models and to automatically generate from the models C++ real-time control code, for direct experimental application. The code generated is modular, easy to debug and to modify. The paper describes the design of the environment, with the key aspects of the modelling methodology. EdROOM has been used by our research team for several applications, with satisfactory results. One of these applications concerns the control of actuators of a fast ship for vertical motion smoothing: this is described as an example of EdROOM use.

1. INTRODUCTION

A quick review of the evolution of programming shows how the initial problems of non-structured code (meaning long debugging efforts), led to solve them by developing and using methodological approaches of software engineering, to be able to program reliable complex applications. Following a complete discipline, which starts from models of the software application to be created, the generation of modular, well-structured code is promoted. This is the purpose, for the particular case of real-time systems, of modelling languages such STATEMATE [1], ROOM [2] and Real-Time UML [3], with a graphical syntax to express the relationships between the system components and the specification of the behaviour of all of them. Figure 1 shows the main aspects to be considered by a real-time application.

Taking ROOM to analyse some of the practical control applications we had to solve during our research, we noticed that after a model is obtained using the ROOM method, the coding task was left to the user. We got the idea of bridging the gap between the graphical syntax and the real-time code, by creating a visual interactive environment, we denoted EdROOM, for diagram-based modelling and automatic real-time control code generation.

EdROOM provides a solution for fast development, clear graphical expression of the design ideas, homogenous code generated, easy debugging, design components reusability, encapsulation of detail levels and easy redesign. EdROOM has been successfully used for the development of real time control systems. An interesting example of application has been the development of a real-time control system to carry on experiments with a fast ship, at towing tank scale.

The paper has three sections. First, the design methodology is summarized. Second, the EdROOM environment is described. Finally, an example of application (the fast ship) is given.

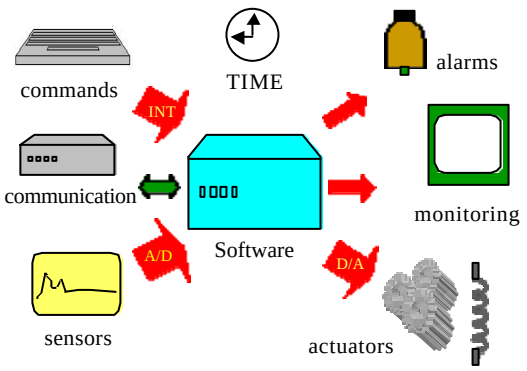

Figure 1. Real-time control systems

2. THE ROOM METHODOLOGY

The main feature of real time control systems is that they are in constant interaction with their environment. The diversity of simultaneous tasks these systems perform determines their complexity. The different tasks must be scheduled with different priorities and according to events. A concurrent model must be used to define the system.

The design of this type of systems must provide the following elements: (1) the determination of the set of concurrent tasks; (2) the definition of the communication between them, including its topology and the valid protocols; (3) the specification of the behaviour of each of the tasks and the system scheduling policy. ROOM provides graphical means to define these three elements.

2.1. Tasks: actors and actor classes in ROOM

In ROOM each task is assigned to an agent called **actor**. The actors are the basic entities that are used to define the ROOM models. ROOM uses a multilevel definition of the structure, since each actor may contain other actors within itself. Figure 2 shows a part of the hierarchical structure of a ROOM model. The first level of the model, composed by five actors, is graphically represented by the top left scheme in figure 2. The arrow that originates at one of its actors points at the scheme on the right that represents the second level of actors contained in it The other arrow points at the scheme of third level actors.

ROOM uses an object oriented solution to organise the relationships between the actors, which is based on the definition of actor classes.

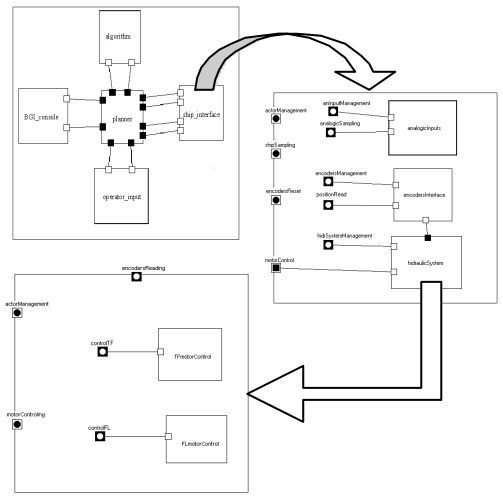

Figure 2. Actors hierarchy in ROOM

2.2 Communication among actors

The actors establish communication between them by means of message passing. To determine the set of messages that actors may exchange between them, the model includes some **protocol classes**. Each one defines a set of input messages and another of output messages that shall be used for communication.

An actor uses a protocol class when it adds a **port** of that class to the interface of its actor class. Figure 3 shows an actor class with 3 ports in its external interface(port1, port2 and port4) and one more in its internal interface (port3).

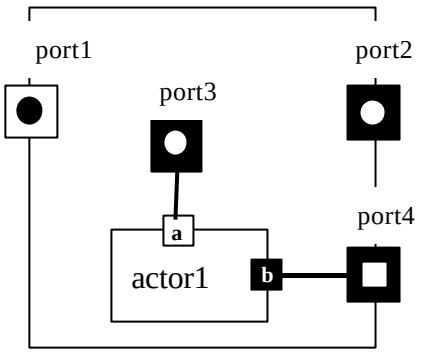

Figure 3. Communication ports

Communication between actors is established trough ports. There are several types of ports identified by different graphical representations. The ports with a dark border (port2, port3 and port4) use its protocol class in a conjugate way, taking the input messages as output messages and vice versa. Those with the rectangular internal border, such as port4, are relay ports and they allow to export a port from a component actor.

2.3. Behaviour and scheduling. ROOMCharts and priority of messages

To define the actor behaviour, each actor class has a state chart called ROOMChart. ROOMCharts are based on Harel's Statecharts formalism [4]. They determine the transition that will be triggered by a message received at the present state. Each transition has one signal, one reception port and one guard associated to it. The transition is triggered only if there is coincidence for the signal and port of the message received and the transition, and the guard is true. The ROOMCharts allow several definition levels. This is possible because any state may contain other sub-states and define its own context. Figure 4 shows a two level ROOMChart example where A and B are states that define their own context.

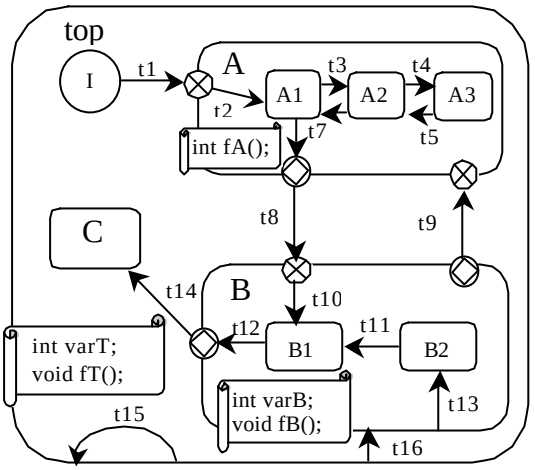

Figure 4. ROOMChart example

In each context, it is possible to define a set of variables and functions. These are used for the treatment of the received message when the actor is in one state of that context. State entries and exits, as well as transitions, may have associated functions that are executed when a message arrived triggers them. These functions may either belong to this context, or to any of its higher contexts..

The scheduling policy is established by means of the priority of messages sent. Each actor supports a message queue arranged by priority. In a multitasking environment, the actor that has the message with the highest priority must catch the processor time for handling it. ROOM allows synchronous and asynchronous message passing.

3. EdROOM: A GRAPHICAL EDITOR OF ROOM MODELS AND AUTOMATIC CODE GENERATOR

In order to take advantage of the ROOM modelling potential, the EdROOM environment has been developed. EdROOM is a CASE tool for editing ROOM models that integrates an automatic code generator

3.1. Model edition

EdROOM supports the management of the actor classes, the protocol classes and the data classes by employing windows and mouse. The figure 5 shows the edition canvas of the actor class structure. To add an component actor, or a port, to the actor class it is enough to drag the actor class, or the protocol class, from the lists located on the right and drop it into the canvas. The tool bar on the left is used to connect actors.

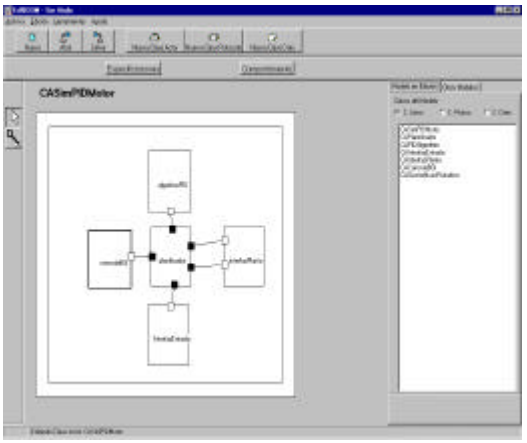

Figure 5: class Actor edited with EdROOM.

With respect to the behaviour, EdROOM includes a window for editing the state chart of each actor class. The tool bar located on the top is used to add states and transitions to the chart. The figure 6 shows this window. By double click over one state it is possible to add its sub-states and define a new context. This is the way to specify several levels in the behaviour. Besides, each transition includes a menu for adding its trigger condition(signal, port, guard) .

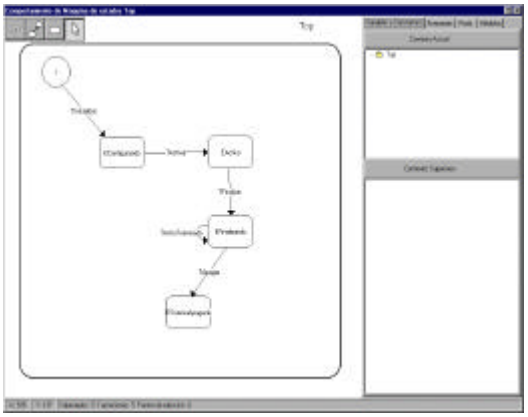

Figure 6 ROOMChart edited with EdROOM.

3.2. Implementation: the detail level functions and the ROOM services

In EdROOM, the detail level functions used to handle the input messages are integrated within the design. To do that, the behaviour edition window (Figure 6) is used to add variables and functions to the different contexts. These functions are linked to the transitions and to the state entry and exit. EdROOM includes also text edition windows to define the prototype and the body of these functions.

There is also a set of basic services that have been provided to simplify the detail level implementation. These services are: communication, scheduling, timing, and memory control. We have developed a library called mv_rtk.lib that contains them. This library is linked with the generated code and it implements all the primitives that are needed for making these services transparent to the user.

In ROOM, the scheduling is specified by assigning priorities to the messages. The library mw_rtk.lib provide the following primitive to send across a port an asynchronous message with a specific priority.

port.send(signal, priority, dataP, poolP);

The Timing service is accessible using the following primitives:

Absolute: ROOMtimer.InformAt(time, dataP, poolP, priority);

Relative: ROOMtimer.InformIn(interval, dataP, poolP, priority);

InformAt uses a absolute time parameter while InformIn uses a relative time parameter, namely an interval.

3.3. Code generation. Requeriments and capabilities

The generated code is a set of C++ source files that implement the actor classes of the model. From each actor class one C++ class with the same name is generated. The generated code includes the implementation of the structure, the communication and the behaviour defined in the design. The detail level functions added to the behaviour are correctly inserted in this code. To obtain the executable file, the source files are compiled and linked with the mv_rtk.lib library. This library has been implemented to run on the low cost real time kernel RTKernel [6]. This kernel works on Intel-based systems, and supports MS-Windows as a service. We are also developing other versions of the library, for other real-time operating systems, in order to provide portability of EdROOM to other computing platforms.

The generated code is clear and coherent with the design. This makes easy the debugging of the program. As an example we show the main function of a system that performs a PID control of the speed of a D.C. motor (figure 7).

```
#include "mv_rtk.h"
#include "PIDControlE.h"
void main(){
  Kernel kernel(ROOMVeryHighPriority);
  PIDControl MainActor ("top", NULL, 10,
                ROOMNormalPriority,1024);
  MainActor.ROOMConfig();
  MainActor.ROOMStart();
}
```

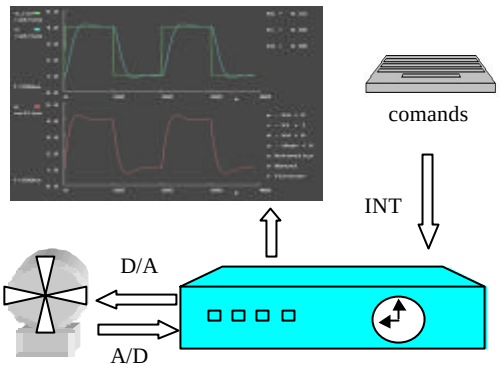

Figure 7: PID control of a DC motor

Figure 8 shows the diagram of actors designed to accomplish the PID control of the DC motor.

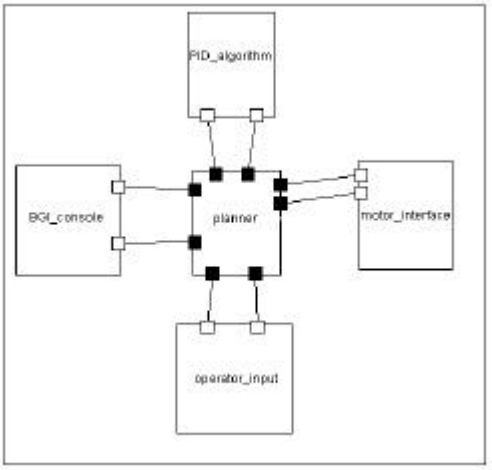

Figure 8. PID control ROOM model

The requirements to perform the development are not stringent: a Pentium 100 with 16 MB of memory, with Windows 95/98, is enough to run EdROOM. The compiler used to obtain the executables was Borland 4.52 or 5.02. (it is free distribution). The RTKernel licence needed to develop as many projects as you want costs only 550$. Finally, the computer needed to run the executable is a standard PC.

4. APPLICATION EXAMPLE: CONTROL EXPERIMENTS WITH A SHIP ON A TOWING TANK

Part of our research concerns a fast ferry. Vertical motions at high speed can have negative effects on the ship and the passengers comfort. By means of transom flaps and a T-foil near the bow, we can counteract the motions induced by waves. The flaps and the T-foil can move under control. The problem is to get a control strategy for optimal motion smoothing. A scaled down replica, 4.5 meters long has been built, for experimental studies in a towing tank institution (CEHIPAR, Madrid). The replica has flaps and T-foil. There is a step motor to move the T-foil wings and other to move the Flaps. The sensors located in the replica measure the following variables: heave, pitch, the height of the arriving wave, the drag forces (starboard and port) and the accelerations in several points of the replica [5].

We have used an industrial PC to run the control program. The PC includes the Advantech PCL812PG data acquisition card, to perform the A/D conversions of the sensors measurements and to generate the pulses that control the step motors. Another card, the TE5312, is also connected to the PC bus to read the motor encoders value. The control system has been developed to interact with the operator commands and to display the signals on a monitor. Figure 9 shows a schematic of the whole system. The clock inside the computer means the ROOM timing service has been used to perform the time synchronization tasks.

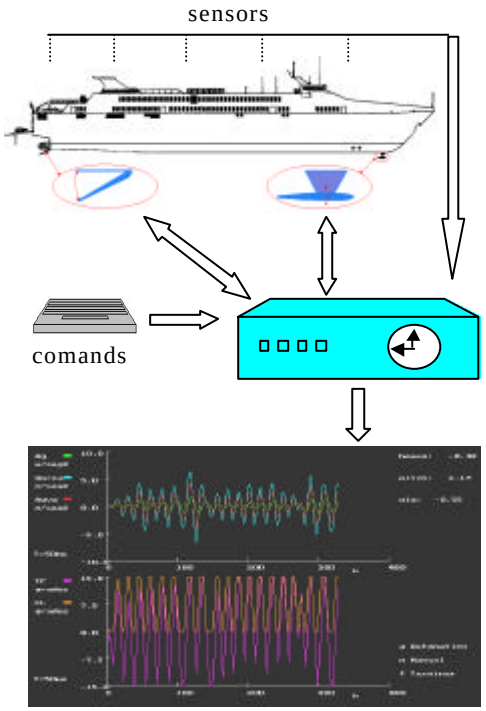

Figure 9. Control of the ship replica

4.1. Actors structure

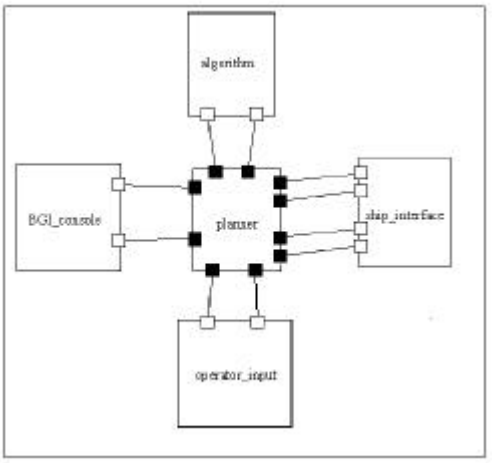

Figure 10. The five main actors of the ROOM model of the control program.

The model of the experimental control of the replica includes 5 inter-connected main actors (figure 10).

The function of these actors are the following:

-"ship_interface" is employed to provide the interface with the step motors and the sensors.

-"BGI_console" displays the main variables in the console.

-"operator_input" is employed to handle the operator commands given with the keyboard.

-"algorithm" executes the control algorithm.

-"planner" coordinates the work of the rest of the actors in each sampling period. It manages the start-up and termination of the whole system.

The "planner" also contains other actor, called "periodic_Sampling", which manages the sampling timing. The "ship_Interface" has three actor components inside too: One of them performs analog measurements sampling with the PCL812PG card. Other provides access to the TE5312 card and reads the motor encoders value. The third is devoted to move the appendages in accordance with the calculated commands. This last includes also two actors more in its internal structure. One of them generates the pulses to move the T-foil and the other moves the flaps.

The hierarchical strategy promoted by EdROOM, with the definition of several levels of actors, is an important feature. A complex task can be divided into simpler ones. This has been a key point for the success of the development.

4.2. Experience with EdROOM

One of the advantages that we wanted to obtain was to make easy the re-design of the control program. With EdROOM its possible add actor components in a advanced stage of the development to extend the functionality of some class actor.

Before the experiments on the water, the control system has been developed and tested directly with the replica (hanging from a crane). Actually, the replica was manually balanced, to see if sensors and actuators worked correctly. Some problems (calibration, signs, noise from motors) were solved by using EdROOM on the field to modify the control program. Once the replica was on the water, new problems appeared. The control strategy (taking signals from accelerometers and applying a digital PID) has to be re-designed to include some filtering. Again, EdROOM was used for a quick re-design of the control on the field (in the towing tank, time is money).

The protocol classes and the behaviour can be added or extended without problems. Other important benefit is the possibility of checking several alternatives. This can be made by a direct substitution of an actor with another actor which has the same interface.

The strict interface definition of the ROOM actors assures the correct integration of the new actors in the system. This fact is also the key point of the reusability of the ROOM actors: we can say that actors work like “plug and play” software components. As the result of all these aspects the development of real-time systems can be stated like the evolution of a basic prototype that runs properly in each stage.

5. CONCLUSIONS

A visual CASE tool for real time automatic control code generation has been created. The tool, called EdROOM, runs under Ms-Windows. It follows an object oriented methodology of software engineering, called ROOM, based in the use of actors. The use of ROOM guarantees a structured and clear design of control programs and allows for the reuse of software components at design level. The executable code is generated quickly and can be used in Intel-based systems. This code uses a low cost real time kernel called RTKernel. The library mv_rtk.lib implements the memory control, communication, timing and scheduling services, necessary to execute the ROOM model, using RTKernel primitives. This library is linked with the C++ code generated to create the executable. The implementation of this library for other RTOS paves the way to obtain platform independent applications and reusable components.

REFERENCES

[1] Harel, David. H. Lanchover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, 1990. "*STATEMATE: A Working Environment for the Development of Complex Reactive System*," IEEE Transactions on Software Engineering 16 (1990) pp 403-414.

[2] Selic, Brian, Gulleckson, Garth., and Ward, Paul T. 1994. *Real-Time Object Oriented Modelling. NewYork"*, John Wiley and Sons.

[3] Powel Douglass, B. 1998. *Real Time UML. Developing Efficient Objects for Embedded Systems.* Addison Wesley.

[4] Harel, David. July 1987. *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming 8 :231-274.

[5] R.Polo, O., Esteban, S., Grau, A., De la Cruz. J.M. 2001. *" Control Code Generator used for Control Experiments in Ship Scale Model"* Accepted for presentation in the IFAC Conference CAMS2001 July, 2001.

[6] RTKernel 4.0 and RTKernel 4.5 Real-Time Multitasking kernel for C/C++. User's Manual.