A FLEXIBLE SOFTWARE FOR REAL-TIME CONTROL APPLICATIONS IN FUSION EXPERIMENTS

G. De Tommasi * **F. Piccolo** ** **A. Pironti** *
F. Sartori **

* *Associazione EURATOM/ENEA/CREATE*
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli "Federico II"
** *EURATOM/UKAEA Fusion Association*
Culham Science Centre
Abigdon, Oxon
OX14 3DB, United Kingdom

Abstract: *JETRT* is a framework of real-time software and simulation tools designed to help development of control systems in the JET fusion experiment environment. The main design choice is the complete separation of the target application from the hardware and plant interfaces. This architecture has been designed to maximize the reusability and to standardize the software development cycle of real-time control systems. Thanks to this design choice development costs have been reduced and even non-specialist programmers can easily contribute to a real-time project. *JETRT* also provides a set of powerful debugging and testing tools, some of them well integrated with the Matlab environment. This feature allows to reduce significantly the time spent on the plant for the commissioning of a new control system. *Copyright© 2005 IFAC*

Keywords: Real-time systems, object-oriented design and programming, software tools.

1. INTRODUCTION

JET tokamak (Wesson, 2000), though it was built more than twenty years ago, is still the world's biggest pulsed operated fusion experiment, where several computers interact in order to perform real-time control and monitoring services (Lennholm *et al.* (1999), Puppin *et al.* (1996) and Sartori *et al.* (2003)). Tokamaks are the most promising confinement devices in the field of controlled nuclear fusion: their principal object is the containment of a thermonuclear plasma by means of strong magnetic fields.
In such a complex environment achieving coding practice standardization and separation between the application software and its interfaces to the external systems, is the key to minimize development time and cost, and to maximize reusability and efficiency.
JETRT has been designed to separate the algorithmic part of a real-time control application (*User Application*) from the interface software (*JETRTApp*), and to standardize the software development cycle. Portability among the desired computer platforms has been considered as well, leading to a further increase of code reusability and to a reduction of the debugging efforts.
JETRT framework has been successfully used to develop and test many systems among the *Real Time Data Network* processing nodes (RTDN,

Felton *et al.* (1999)).
This paper gives an overview of the whole *JETRT* framework and describes in more details the real-time executor *JETRTApp* architecture. The next section introduces the JET experiment. Section 3 deals with design choices and carries out a comparison between our approach and other design methodologies and technologies for real-time systems. In section 4 an overview of the whole framework is given. Section 5 introduces *JETRTApp* architecture and timing issues, while the following section deals with the *User Application* plug-in. Eventually some concluding remarks are presented.

2. THE JET EXPERIMENT

In a fusion experiment, the main aim is to obtain a plasma (a fully ionized gas) with the desired characteristics. This result cannot be achieved simply by pre-programming the actuators. Because of the various types of instabilities manifested by the gas, several corrective actions must be taken, and their timing constraints must be always met: the systems containing these tasks can be classified as hard real-time systems, according with the operational definition given in Liu (2000). At JET typical sample times for control loops range between 50 μs (*Vertical Stability System*, Lennholm *et al.* (1997)) and 1 ms (*eXtreme Shape Controller*, Ambrosino *et al.* (2003) and Ariola *et al.* (2003)). While several distinct systems take care of every control and safety problems , the *Real Time Data Network* has been designed to coordinate their actions and to make possible the addition of further processing node. *RTDN* is a digital mechanism to exchange data between measurement systems, feedback controllers, and actuators based on ATM-AAL5 protocol. Each source node produces messages with typical rates range between 1000 and 20 messages per second.

JET is a pulsed machine: that means that every 20-60 minutes the plasma is formed and sustained for about a minute. Because of this, all the real-time applications operate into two different modes:

- **OFF-LINE**, before the experiment they receive the instructions, while after they return the collected information;
- **ON-LINE**, they perform real-time measurement, control or protective actions.

The off-line interfaces are the source of much of the technical complexity of the real-time applications. Before and after every pulse the *Supervisor* countdown task sends to the system change of state requests in order to synchronize the evolution of the various JET subsystems. As soon as the scientists have finished pre-programming a new experiment, the *Level-1* plant management system sends a packet containing new parameters. Finally, the information collected has to be available for sending to *GAP* (General Acquisition Program) data management system.
JETRT framework has also been created to help working in this environment, answering the need for a fast and reliable deployment of new systems.

3. JETRT DESIGN CHOICES

During its twenty years of life, several real-time systems have been deployed at JET. Since the very beginning, much of the code of a project was recycled in the implementation of the next one with the hope of saving development and testing time. While this practice was proving to be very helpful in reducing programming costs, it eventually appeared to have too many shortcomings:

- the hardware related details where intermixed with the application specific ones. In order to test the application on a different system it was necessary to emulate the target platform.
- Once a specific hardware platform had run out of commercial life, the migration to a new platform required an almost complete rewrite of the code.
- Only the people with enough knowledge of the target platform could benefit from reusing the existing software.
- Allowing excessive freedom in writing a program means that only the programmer is very efficient in maintaining it.

At the same time it was observed that, in the on line mode, all of the deployed systems, despite their complexity, could actually be reduced to a simple iterative model:

(1) Data acquisition.
(2) Processing.
(3) Output the results.

To realize this behaviour there isn't need of a complex and distributed control system, while a simple single-processor architecture can be used. The system architecture must satisfy two main requirements:

(1) ensure enough processing power, so to allow the satisfaction of the timing constraints;
(2) use as much COTS as possible.

In principle this model seems to be well suited for a programmable logic controller (PLC) based architecture: unfortunately none of this commercial devices can assure the needed timing constraints in terms of scan cycle of less than 1 ms (including I/O processing). For this reason the selected ref-

erence platform uses other types of off-the-shelf components, such as industrial processor cards running RTOS and I/O boards, which can ensure the timing requirements.
For example to attain the required 1 ms sample time, the eXtreme Shape Controller has been deployed on a VME board with a 400 MHz PowerPC CPU running VxWorks (see Sartori *et al.* (2004)). Once the standard hardware architecture has been chosen the real-time application must be designed to realize the iterative cycle introduced above when on line, while, in off line mode, it has to accomplish all the communications and ancillary tasks, without any timing constraint. Off line tasks are the same for all the JET applications, therefore, once a real-time executor that performs all of them has been developed, the definition of the on line processing task is the only thing to do when a new real-time application has to be created.
Since the hardware has been selected a priori, HW/SW design techniques, which have been successfully applied in various application fields (see Saoud *et al.* (2002)), are not necessary in the *JETRT* framework. Similarly, because the architecture is not distributed, there isn't need to use technologies such as RTCORBA as has been done in Tanabe *et al.* (2001).
Moreover, thanks to the simplicity of the proposed iterative model and thanks to the single processor choice, developing a new real-time application with *JETRT* framework is simpler than using other frameworks (e.g. Janka (2002) and Moore *et al.* (1999)). In fact, *JETRT* has proven to be extremely beneficial for the projects development providing the advantages of a simple and standardized application-programming model.

4. JETRT OVERVIEW

The *JETRT* framework is a cross-platform class-library designed to speed up the development of real-time control systems and providing validation tools to ease the test and commissioning phases.
Usually most of the time developing a new real-time system is spent testing the program. The task is clearly hindered because of the limited debugging facilities present in many target systems. In fact, even if several products enhancing the testing capabilities for the various platforms are available from the market, in our experience the problem mostly lies elsewhere. The commissioning time is actually spent more on the algorithmic part of the code rather than in the interfaces or in the real-time synchronization, because while the latter part can be tested at leisure in the laboratory, the former needs the running of a complete experiment in order to be tested. For this same reason, the mathematical algorithms are

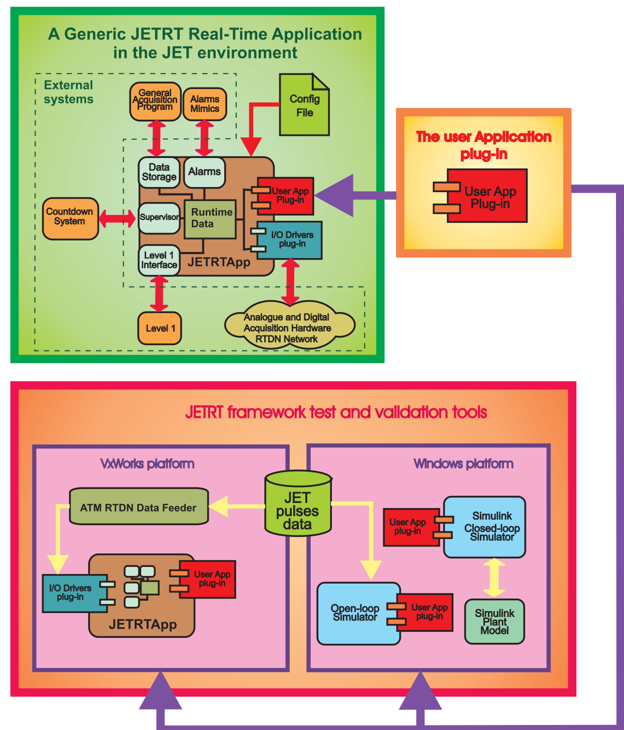

Fig. 1. JETRT framework overview

normally developed separately on specific simulation environments like Matlab, where many test and display facilities are available.

Having achieved the separation of the algorithms (*User Application*) from the interfaces (*JETRTApp*) it was then very easy to use the same application as a plug-in within any simulation environment, thus allowing the testing of the code using the same tools used in the early mathematical development phases. This is the major reason why the *User Application* must be written as a portable application.
In Fig. 1 is depicted the overall architecture of the *JETRT* framework. Special attention has been given to the role of the *User Application* plug-in and how the same piece of code that will be used on the plant can be used with the test and validation tools.
The open-loop simulator showed in Fig. 1 is the most used testing tool. It uses the information stored in the JET database containing the measurements from old experiments, to reproduce the data that the algorithm would have processed if it had been running at that time. Despite not been perfectly adequate for testing close loop systems, this method normally allows finding most of the problems in the code, especially because of the user friendliness of the debugger on an Intel/WinNT4 platform, compared to that of some target platforms (Motorola PowerPC/VxWorks).
A more thorough test can be performed by loading the application module into Simulink. In this environment it is either possible to compare directly the original model of the system with its own im-

plementation, or to execute the User Application code in close-loop using a model of the plant.
A data feeder based on the ATM Real-Time Data Network (Felton *et al.*, 1999) has been developed to test the applications on their target platforms. The feeder downloads, from the JET database, all the experimental inputs needed from the *User Application* and send them to *JETRTApp* via the real-time network. The data feeder allows testing a modified version of the *User Application*, which differs only for the I/O boards configuration. Since the real boards can be tested separately, the application can be tested almost completely even on the target platform.

5. JETRTAPP ARCHITECTURE

JETRTApp is a generic single-processor real-time application, which has been designed using object-oriented techniques. Its structure is very modular: it makes heavy use of threads to handle the different interfaces and of plug-ins to allow both working with different hardware and performing different algorithms.
The block diagram of Fig. 1 shows the connections of the real-time executor *JETRTApp* with the JET external systems and the other *JETRT* components. The I/O Drivers plug-in system allows the customization of the data acquisition. It is a collection of high level drivers that act as bridge between the low-level drivers and *JETRTApp*. The *User Application* implements the specific real-time control and diagnostic algorithm.
The *Runtime Data* block is the information exchanged between *JETRTApp* and its plug-ins during the real-time execution. The *Configuration File*, is a structured text file whose hierarchical structure reflects the internal *JETRTApp* object structure. At the system start-up, the file is opened and its content passed to all the internal components. Each object constructor routine extracts the relevant information and uses it to setup its parameters to create subcomponents and to start any necessary thread. Following the setting in the file, the system initializes the necessary I/O Drivers, loads the desired *User Application* plugin, allocates the data collector memory and starts the interfaces with the external systems. Once finished, it immediately begins the real-time periodic actions while waiting for a pulse start sequence, an operator or any other remote command.
Fig. 2 shows a block diagram of *JETRTApp* where the five most important components can be easily noticed:

- The Supervisor State Machine.
- The Real Time Thread.
- The Real Time Data Collector System.
- The External Boards Interface.

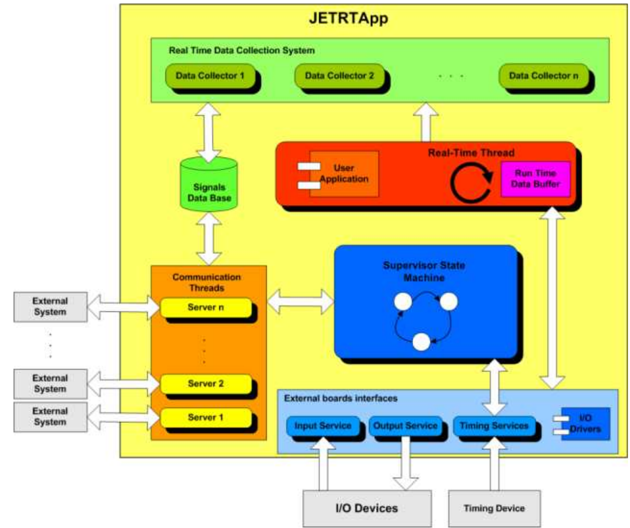

Fig. 2. Schematic of JETRTApp

- The Communication Threads.

The *Supervisor State Machine* is a finite state machine used to manage the overall state of the *JETRTApp*. It changes the state according to a set of rules and in response to external (start of the countdown, pulse trigger, end of JET pulse, start of data collection) and internal events (errors during the real-time computations). This state controls the overall functioning of the program, whether it is on-line or off-line, whether it is ready to operate or not. It also synchronizes the various threads within the application for instance disabling the data collection and the *Level-1* parameters processing during the real-time phase.
The *Real Time Thread* is responsible for the calling of the *User Application* plug-in during the JET pulse, while the *Real Time Data Collector System* stores all the requested data sending them to GAP after the end of pulse.
The *External Boards Interface* manages all the I/O boards by the means of the I/O drivers plugin common interface.
The *Communication Threads* handles all the communications between *JETRTApp* and the JET computing environment. It starts a thread for each system it is communicating to, and tries to keep the socket open until the remote system shuts it down. This means that there is a thread handling the messages for each external system. This component is a container for specific protocol message handlers. As soon as a message is received, it is dispatched to each of the handlers until one is willing to accept it and complete the transaction. The different subsystems are actually parts of the *JETRTApp* object as shown in Fig. 3.

The JETSupervisorStateMachine object implements the Supervisor State Machine and JETStatus is its state. CODASmessageReceiver manages the communication threads. JETUserApp object contains the *User Application* plug-in, which is

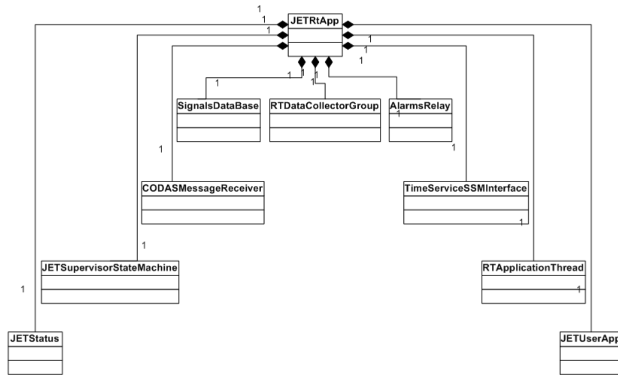

Fig. 3. UML diagram of the JETRTApp object

dynamically loaded at the object initialization. It also interfaces to CODASmessageReceiver in order to act as a message handler for the *User Application* messages.

The SignalDataBase and RTDataCollectorGroup objects implements the *Real Time Data Collector System*. SignalsDataBase stores the information about all the signals of interest for the external data collection: a physical input or output, a calculated measure, an internal state or an alarm. During the on-line phase several data collectors contained in the RTDataCollectorGroup object, selectively store part of the information flowing in the system. Each collector has different memory allocation, a different selection of signals, and it is configured to store data at varying rate during different acquisition time windows or around specific events. The settings for the acquisition are found in the Configuration File but can also be changed as part of the GAP interface.
The AlarmsRelay object sends asynchronous messages to the external systems containing information about limits, exceptions or generic alarms that have occurred during the previous experiment.
Eventually RTApplicationThread is the object that runs and manages the real-time *User Application*. It contains several important subcomponents, such as the ones dedicated to the timing service: *ExternalTimeTriggeringService* and *InternalCPUTimingService*.
In order to synchronize the activities of the many real-time systems, the time information and the real-time triggers are broadcast from the JET central server via optical fiber. Since this system provides the timing with only 1ms resolution, it was necessary to find a method to measure time more accurately while still maintaining synchronization with JET. The solution was to use any high-resolution time measuring available in the platform, whether that was a CPU internal counting register, or a chipset timer. The time information was then composed of two parts: the 1 ms precision time of start of the real-time computation cycle and the higher resolution time within the cycle. *ExternalTimeTriggeringService* manages the external timing triggering synchronization hardware while *InternalCPUTimingService* provides the high-resolution information. *ExternalTimeTriggeringService* is also a container of activities that have to be scheduled at a precise JET time.

6. THE USER APPLICATION

The *User Application* is normally a highly sophisticated mathematical code, implementing measurement, control or diagnostic system. Most of the time, the scientist writing the program does not want to know the technical details of the external world interfaces, since from their point of view the algorithm is simply a function reading some data and producing results. With this new system, this is now possible, since these details are hidden away in the *JETRTApp*. The interface of the *User Application* determines the complexity of the interaction between the user code and the external world.
The present version of the *User Application* interface implements the functions described in Table 1.
The *User Application* component has been implemented as C++ dynamic loadable object. Thanks to this separation only this plug-in has to be modified if a new real-time application is developed, without affecting the rest of *JETRTApp* code. Once the plug-in is loaded, it is initialized calling *Init()*. The *Level-1* parameters processing is left completely to the application, which must provide a proper *MessageProcessing()* function. This function can be used also to implement any other custom specific protocol. The data acquisition is active during the off-line phase. Data is collected at a lower rate and passed to the *User Application* using the *OfflineProcessing()* call. This call can be used to either make sure that the system outputs are set to safe values or simply to keep monitoring the inputs. Before entering the on-line phase *Check()* is called in order to verify the willingness of the *User Application* to begin the real-time action. If parameters are missing or invalid or if some plant readings are wrong, the application can abort the experiment. After the *PulseStart()*

API Functions
Init()
MessageProcessing()
Check()
PulseStart()
MainRealTimeStep()
SecondaryRealTimeStep()
SafetyRealTimeStep()
OfflineProcessing()
Menu()

Table 1. User Application API

is called and the experiment starts, *JETRTApp* performs these cyclical operations: first the code is synchronized to the JET timing, the acquisition is then completed, the *MainRealTimeStep()* is called, then the data is written to the outputs, the *SecondaryRealTimeStep()* is called and finally the data is stored on the data collectors. If during the on-line phase the *User Application* generates a non-recoverable internal error (by using a special call-back), then the system stops executing the standard sequence, and instead just calls the *SafetyRealTimeStep()* between the data input and data output.

CONCLUSION

A software architecture designed to improve the development of real-time control systems at JET and to reduce the deploying time has been presented.

Thanks to the separation between the control algorithm and all the common subsystems needed by a real-time application, *JETRTApp* has standardized the development cycle to create a new real-time system: only a new *User Application* plug-in has to be written and several parameters have to be set in the Configuration File.

JETRTApp code runs both on Motorola/VxWorks, which is the plant platform, and on INTEL/ WinNT4 by simply recompiling it. The latter is used as a powerful simulation and testing platform, allowing even to run the *User Application* within Matlab/Simulink environment. Therefore it is easy to ensure that the code of a new system is well tested before the deploying and this will reduce the request operational time for the commissioning.

In 2003 *JETRT* has been successfully used to develop the eXtreme Shape Controller (Albanese *et al.* (2004)) and the *Error Field Correction Coils controller* (Zanotto *et al.* (2004)).

The development work is by no means finished. There are several new activities in progress. *JETRT* framework is being ported to different platforms, among which real time Linux RTAI.

Moreover a real-time Simulink executor concept has been tested successfully for a simple system. This system uses off-the-shelf tools such as the Mathworks *Real-Time Workshop* to automatically generate the *User Application* from a Simulink diagram. Linux port is very interesting to look for new and cheaper hardware solutions, based on standard PCs rather than on expensive industrial computer boards, while automatic generation of the application plug-in is a step further into the reduction of development time.

REFERENCES

Albanese, R. et al. (2004). Design, implementation and test of the extreme shape controller (xsc) in jet. In: 23$^{\mathrm{rd}}$ *SOFT*. Venice, Italy.

Ambrosino, G., M. Ariola, A. Pironti and F. Sartori (2003). A new shape controller for extremly shaped plasmas in jet. *Fusion Enginnering and Design* **66-68**, 797–802.

Ariola, M., G. De Tommasi, A. Pironti and F. Sartori (2003). Controlling extremely shaped plasmas in the jet tokamak. In: 42$^{\mathrm{nd}}$ *Conference on Decision and Control*. Maui, Hawaii.

Felton, R. et al. (1999). Real-time plasma control at jet using atm network. In: *Proceedings of* 11th *IEEE NPSS Real Time Conference*. Santa Fe. pp. 175–181.

Janka, R. S. (2002). *Specification and design methodology for real-time embedded systems*. Kluwer Academic Publisher.

Lennholm, M. et al. (1997). Plasma vertical stabilisation at jet using adaptive gain control. In: *Proceedings of the* 17$^{\mathrm{th}}$ *SOFE Conference*. Vol. 1. pp. 539–542.

Lennholm, M. et al. (1999). Plasma control at jet. In: 2nd *IAEA Technical Commitee Meeting*. Lisbon.

Liu, J. W. S. (2000). *Real Time Systems*. Prentice Hall.

Moore, M. L. et al. (1999). Complex control system design and implementation. *IEEE Control Systems* pp. 12–27.

Puppin, S. et al. (1996). Real-time control of the plasma boundary at jet. In: *Proceedings of the* 16$^{\mathrm{th}}$ *SOFT*. Lisbon.

Saoud, S. Ben, D. D. Gajski and A.Gerstlauer (2002). Co-design of embedded controllers for power electronics and electric systems. In: *Proceedings of the 2002 IEEE International Symposium on Intelligent Control*. Vancouver. pp. 379–383.

Sartori, F., A. Cenedese and F. Milani (2003). Jet real-time object-oriented code for plasma boundary reconstruction. *Fusion Enginnering and Design* **66-68**, 735–739.

Sartori, F. et al. (2004). The system architecture of the new jet shape controller. In: 23$^{\mathrm{rd}}$ *SOFT*. Venice, Italy.

Tanabe, T. et al. (2001). Preliminary design of muses control system based on rt-corba and java. In: 8th *International Conference on Accelerator and Large Experimental Physics Control Systems*. San Jose, California.

Wesson, J. (2000). *The science of JET*. JET Joint Undertaking. Abingdon, Oxon.

Zanotto, L., F. Sartori, M. Bigi, F. Piccolo and M. De Benedetti (2004). A new controller for the jet error field correction coils. In: 23$^{\mathrm{rd}}$ *SOFT*. Venice, Italy.