Available online at www.sciencedirect.com
SCIENCE DIRECT®
Control Engineering Practice 14 (2006) 1387–1393
CONTROL ENGINEERING PRACTICE
www.elsevier.com/locate/conengprac

A flexible software for real-time control in nuclear fusion experiments

G. De Tommasi[a,*], F. Piccolo[b], A. Pironti[a], F. Sartori[b]

[a]Associazione EURATOM/ENEA/CREATE, Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, Via Claudio 21, I-80125 Napoli, Italy

[b]Euratom/UKAEA Fusion Ass. Culham Science Centre, Abingdon OX14 3EA, UK

Received 8 February 2005; accepted 7 October 2005
Available online 10 November 2005

Abstract

JETRT is a software framework particularly suited for implementation of both real-time control and data acquisition systems. It is especially designed to work in a complex experimental environment such as the JET nuclear fusion facility. This new architecture maximizes the software reusability. The project-specific algorithm is compiled into a separate software component, in order to achieve a separation from the plant interface code. *JETRT* provides a set of tools to perform most of the validation phase on a Windows running desktop PC. Thanks to these design choices, both the development costs and the commissioning time have been reduced and even non-specialist programmers can easily contribute to the deployment of a new real-time system.

Keywords: Real-time systems; Object-oriented design and programming; Software tools

1. Introduction

Tokamaks are the most promising confinement devices in the field of controlled nuclear fusion: their principal objective is to contain a thermonuclear plasma by means of strong magnetic fields.

JET tokamak (Wesson, 2000) is the world's largest 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).

Due to the complex environment it is important to standardize the coding practice as well as to separate the application software from its interfaces to the external system. Such an approach is the key to minimize development time, cost and to maximize reusability and efficiency.

JETRT is a new software framework used at JET to develop both real-time control and data acquisition systems. *JETRT* differs from the hardware and software architectures used in other nuclear fusion experiments (Behler et al., 1999; Luchetta and Manduchi, 1999; Moulin et al., 1998) because it has been developed to separate the algorithmic part of a real-time application (*User Application*) from the plant-interface software (*JETRTApp*). This design choice has been done to standardize the application development, to achieve portability among the different computer platforms, and to increase the code reusability.

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). For example the eXtreme Shape Controller (Ambrosino et al., 2003; Ariola et al., 2003) runs on a reliable PowerPC/VxWorks architecture with a latency time of 1 ms. Non-safety critical data acquisition systems run on a cheaper INTEL/WinNT4 platform with a latency time of 2–10 ms.

This paper gives an overview of the whole *JETRT* framework and describes in more details the architecture of real-time executor *JETRTApp*. 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

*Corresponding author. Tel.: +39 0817683853; fax: +39 0817683816.
E-mail addresses: detommas@unina.it (G. De Tommasi), fpiccolo@jet.uk (F. Piccolo), apironti@unina.it (A. Pironti), fisa@jet.uk (F. Sartori).

0967-0661/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.conengprac.2005.10.001

is given. Sections 5, 6 introduce *JETRTApp* architecture, and timing issues. A short overview about how *JETRTApp* works with different I/O boards is given in the next section. Section 8 deals with the *User Application* plug-in, while the following section introduces the internal debugger feature available within the *JETRT* framework. 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 by simply pre-programming the actuators. Because of the various types of instabilities shown by the plasma, several corrective actions have to be taken.

The constraints on the latency times of the control systems have always to be met: the systems containing these tasks can be classified as hard real-time systems, in accordance with the operational definition given in Liu (2000). At JET typical cycle times for control loops range between 50 μs (Vertical Stability System, Lennholm et al., 1997) and 1 ms (eXtreme Shape Controller).

JET is a pulsed machine: this means that an experiment is performed every 20–60 min, during which the plasma is formed and sustained for about 1 min. As a consequence of such a cycle, all the real-time applications operate in two different modalities: ON-LINE and OFF-LINE. During the experiment, the systems are in the ON-LINE mode and they perform real-time measurement, control or protective actions. In this phase all the communications are disabled and the hard real-time constraints on the latency time have to be met, while there is no need to check such constraints in the OFF-LINE mode.

When a system is in the OFF-LINE mode it can interact with the other JET subsystems. These interfaces are the main sources of technical complexity for the applications. Before and after every pulse the *Plant Supervisor* sends to all JET subsystems a change-of-state request to synchronize their evolution. As soon as the scientists have finished to programme a new experiment, the *Level*-1 plant management system sends a message to each real-time node, containing the new set-up parameters. Eventually, the information collected during the pulse is sent to General Acquisition Program (*GAP*), which is the data management system.

JETRT framework, helps working in this environment, answering the need for a fast and reliable deployment of new systems.

3. JETRT design choices

Along the last 20 years, several real-time systems have been deployed at JET. Since the very beginning, most of a project code was recycled during the implementation of the next one, hoping to save development and testing time. While this practice proved to be very helpful in reducing programming costs, it eventually appeared to have too many shortcomings:

- the hardware-related details where mixed with the application ones. In order to test the same application on a different platform, it was necessary to emulate the target hardware.
- Once a specific hardware platform had run out of its commercial life, the migration to a new platform required an almost complete re-writing of the code.
- Only the people with enough knowledge of the platform could re-use the existing software.

Since JET is an experimental environment, each real-time system is realized only once, therefore, the costs of a prototype cannot be diluted as in a mass production. For this reason the system architecture needs to satisfy two additional requirements:

(1) use components on the shelf as much as possible;
(2) ensure enough processing power, such as to satisfy the timing constraints.

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 the simple iterative model shown in Fig. 1.

Once the standard hardware architecture has been chosen, the real-time application has to be designed to realize the iterative cycle introduced above, when in the ON-LINE mode. In the OFF-LINE mode, the application has to accomplish all the communications and ancillary tasks. These tasks are the same for all the JET applications and therefore this part of the code can be re-used when a new system is developed.

It follows that the definition of the ON-LINE processing task is the only thing to do when creating a new real-time application.

The scheduling policy adopted in the *JETRT* framework is the *priority-based pre-emptive* (Liu, 2000) with the granularity equals to the minimum allowed by the platform (10 ms for INTEL/WinNT4 and 200 μs for PowerPC/

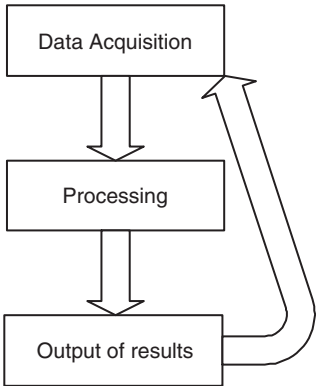

Fig. 1. JETRTApp iterative model.

VxWorks). 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).

Since the INTEL/WinNT4 platform has a 10 ms granularity, it cannot be used to develop a safety-critical control system.

Thanks to the simplicity of the proposed iterative model, *JETRT* has proven to be extremely beneficial for the project development providing the advantages of a standardized programming model.

Since the target hardware has been selected a priori, HW/SW design techniques, which have been successfully applied in various fields (Cuatto et al., 2000; Saoud et al., 2002), are not suitable in the *JETRT* framework.

4. JETRT overview

The *JETRT* framework is a cross-platform class-library designed to speed up the development of the real-time control systems. It provides validation tools to ease both the testing and commissioning phases.

When developing a new real-time system, most of the time is spent to test the program. The task is clearly hampered due to the limited debugging facilities available in many target systems. The market offers several products enhancing the testing capabilities for the various platforms, nonetheless the commissioning time is actually spent more on the algorithmic part of the code rather than on the interfaces or on the real-time synchronization. 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.

Once separated the algorithms (*User Application*) from the interfaces (*JETRTApp*), then it is very easy to use the same application as a plug-in within any simulation environment. It is then possible to test the code with the same tools used in the early design phases (e.g., the Matlab environment). This is the major reason why the *User Application* needs to be written as a portable application.

Fig. 2 shows the overall architecture of the *JETRT* framework. Special attention has been given to the role of the *User Application* plug-in and on how the same piece of code, which will be used in the plant, can be used with the test and validation tools.

The open-loop simulator in Fig. 2 is the most used testing tool. It takes the measurements from old experiments, stored in the JET database, to feed the algorithm under examination. Despite this method is not perfectly adequate for testing closed-loop systems, it 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 other target platforms.

A more thorough test can be performed by loading the application module into Simulink. In this environment it is possible either to directly compare the original model of the system with its own implementation, or to execute the *User Application* code in closed-loop using a model of the plant.

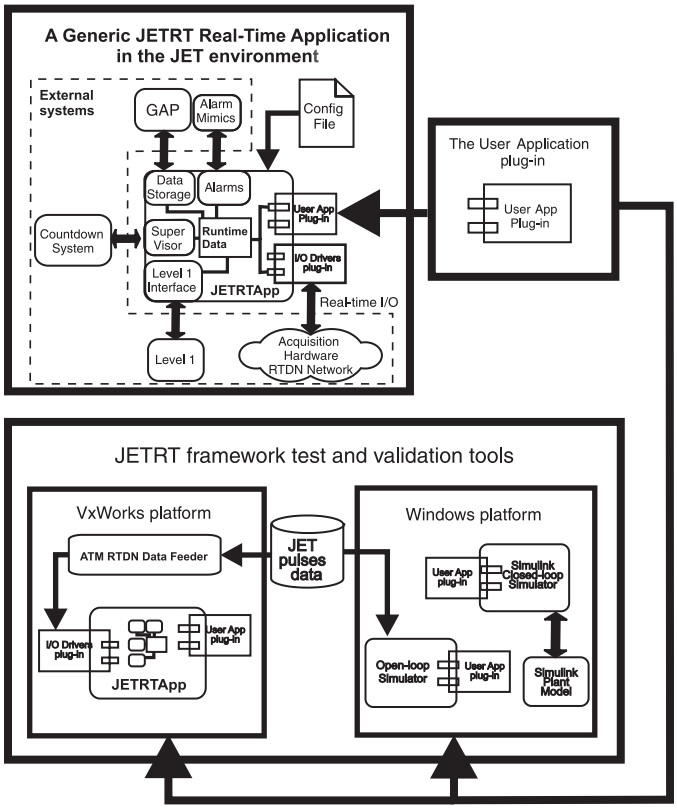

Fig. 2. JETRT framework overview.

A data feeder based on the ATM Real-Time Data Network 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 sends them to *JETRTApp* via the real-time network.

JETRTApp allows to save the current time at the end of each data acquisition phase. Once the acquisition hardware has been chosen and the processing routine has been written, the iterative model shown in Fig. 1 is highly predictable. Therefore, when this feature is enabled, the resulting information can be exploited to verify the hard real-time constraint on the latency time.

5. JETRTApp architecture

JETRTApp is a generic single-processor application, which has been designed by using object-oriented techniques. *JETRTApp* architecture 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 in Fig. 2 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 which acts 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 represents 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 structure of the *JETRTApp* object. At the system start-up, the Configuration File is opened and its content passed to all the internal components. Each object constructor routine extracts the relevant information and uses it to set up its parameters, to create the sub-components and to start any necessary thread. Following the setting in the file, the system initializes the necessary I/O Drivers and loads the desired *User Application* plug-in.

Fig. 3 shows a block diagram of *JETRTApp* where the five most important components are outlined:

- The Supervisor State Machine.
- The Real-Time Thread.
- The Real-Time Data Collectors System.
- The External Boards Interface.
- 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 ON-LINE phase). This state machine controls the overall behaviour of the program, synchronizing the various threads within the application. The Supervisor State Machine sets *JETRTApp* in the ON-LINE mode, raising the priority of the real-time thread to the highest possible value. The result is that all the threads, which are performing the communications with the external systems, are turned off during the ON-LINE phase.

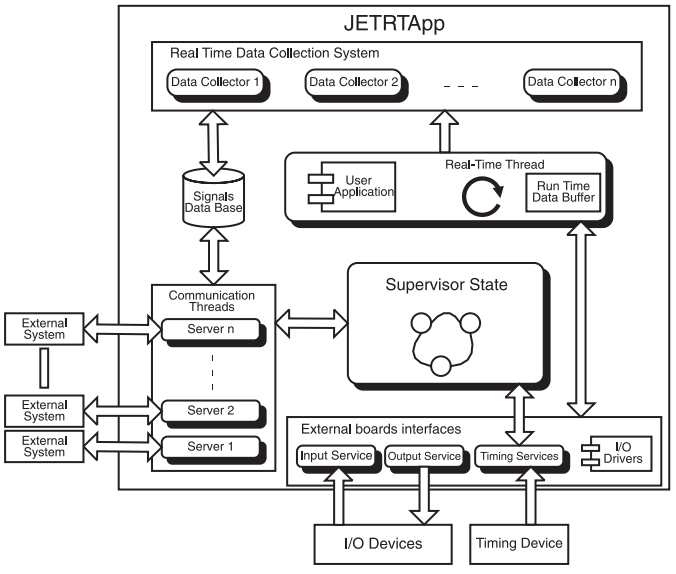

Fig. 3. Schematic of JETRTApp.

Moreover, when an error occurs during the real-time calculation, the Supervisor State Machine changes its state and sets *JETRTApp* in *SAFETY* mode. In this modality all the control outputs are set at safety levels until the end of the pulse.

The *Real-Time Thread* is responsible to call the *User Application* plug-in during the JET pulse. Meanwhile the *Real-Time Data Collector System* stores all the requested data and sends them to GAP at the end of the experiment.

The *External Boards Interface* manages all the I/O boards by means of the I/O drivers plug-in common interface.

The *Communication Threads* handles all the communications between *JETRTApp* and the other JET systems. It starts a thread for each system it is communicating with, and tries to keep the socket open until the remote system shuts it down. This means that there is a thread handling the communication with each external system. It is worth to outline that communications are possible only when the application is OFF-LINE.

The different subsystems are actually parts of the *JETRTApp* object, as shown in Fig. 4.

The *JETSupervisorStateMachine* object implements the Supervisor State Machine and *JETStatus* is its state, while *CODASMessageReceiver* manages the communication threads.

The *SignalDataBase* and *RTDataCollectorGroup* objects implement the *Real-Time Data Collector System*. *SignalsDataBase* stores the information about all the signals of interest for the external data collection. During the ON-LINE phase several data collectors, contained in the *RTDataCollectorGroup* object, store part of the information flowing into the system. Each collector is configured to store data at varying rate during different acquisition time windows or after certain events.

Eventually *RTApplicationThread* is the object which runs and manages the *User Application*.

6. JETRTApp timing

In order to synchronize the activities of the many real-time systems, the time information and the real-time triggers are broadcast from a central server via optical fibres.

Since this system provides the timing with only 1 ms resolution, it was necessary to find a method to measure time at higher resolution. The solution was to use any high-resolution time measurement available in the specific platform, whether it was a CPU internal counting register, or a chipset timer.

The time information was then composed of two parts: the 1 ms precision time at the start of the real-time

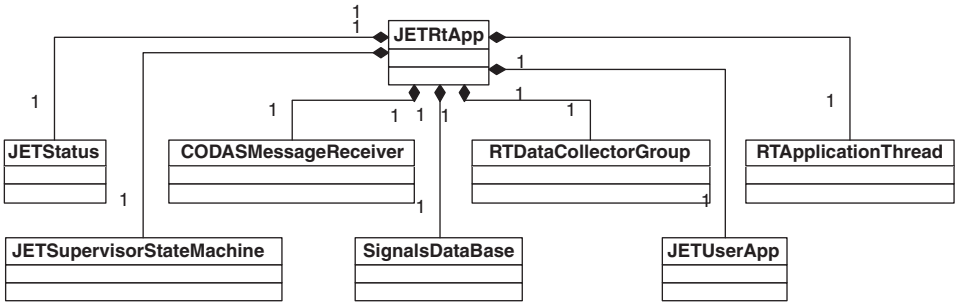

Fig. 4. UML diagram of the JETRTApp object.

computation cycle, and the higher resolution time within the cycle.

The *ExternalTimeTriggeringService* object manages the external timing triggering synchronization hardware, while the *InternalCPUTimingService* object provides the high-resolution information.

ExternalTimeTriggeringService is also a container of activities which need to be scheduled at a precise JET time.

One of the most difficult tasks in designing *JETRTApp* was to find a method which allowed the simultaneous use of many different input devices, while keeping the real-time process synchronized to JET. Two main families of input hardware have been identified. The first produces data with a periodic timing synchronous to JET. The second can provide data either some time after a software command has been issued, or in a non-synchronous way.

The solution to the problem was to start the real-time cycle at a fixed delay from the arrival of any JET synchronous information, whether it was a data packet or a timing trigger.

When the acquisition routine is called, the processor waits for a new arrival on the synchronizing channel within a specific time windows. Once the synchronizing input arrives, the processor gets the latest data on the remaining cards (providing all the needed commands).

If new data on the synchronizing channel arrive after the end of the acquisition window, a non-valid acquisition is marked. In this case the managing of the faulty data are left to the *User Application*.

Hard real-time constraints are given in terms of a maximum allowed number of consecutive faulty data. In case of acquisition error the behaviour is left to the *User Application*, which can allow the continuation of the real-time calculation. Therefore, *JETRTApp* can be also used to develop non-critical systems with soft real-time requirements.

7. The drivers

The *GenericAcquisitionModule* abstract class is the representation of the I/O module used by *JETRTApp*. According to this model, an I/O module can perform any combination among three types of activities: it can work as data source, as data sink, and it can provide timing services. The actual driver inherits the interface of the above abstract class, and implements the available features.

It is the system installer responsibility to use the right driver for a specific task. The choice of the drivers and their setting is controlled via the Configuration File.

The drivers are compiled into a separate dynamic loadable library. They are linked at run-time only if necessary. When a new hardware has to be used, only this plug-in has to be modified, without affecting the rest of *JETRTApp* code and allowing the implementation of more I/O drivers according to specific needs.

8. The user application

The *User Application* is a highly sophisticated mathematical code, which implements a measurement, a control or a diagnostic system.

Most of the time, the scientists writing the program do not want to know the technical details of the external world interfaces. From their point of view the algorithm is simply a function reading some data and producing results. With the proposed framework all the details are hidden in the *JETRTApp* component.

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* abstract interface is described in Table 1.

The *User Application* component has been compiled as dynamic loadable library, as it has been done with the drivers. Thanks to this separation only this plug-in has to be changed if a new real-time application is developed, without affecting the rest of the *JETRTApp* code.

Once the plug-in is loaded, it is initialized by calling *Init*(). The *Level*-1 parameters processing is completely left to the application, which should provide the *MessageProcessing*() function. The data acquisition is active during the OFF-LINE phase. Data are collected at a lower rate and passed to the *User Application* using the *OfflineProcessing*() call. This call can be used either to check that the system outputs are set to safe values, or to keep simply monitoring the inputs. Before entering the ON-LINE phase, *Check*() is called in order to verify if the *User Application* is ready to

Table 1
User application abstract interface

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

begin the real-time action. If any parameter is missing or invalid, or if some plant readings are wrong, the application can abort the experiment.

After *PulseStart*() is called and the experiment starts, *JETRTApp* performs the following cyclical operations: when the data acquisition is completed, the *MainRealTimeStep*() function is called and the data are written to the outputs. Then *SecondaryRealTimeStep*() is called to perform ancillary computations and, finally, the data are stored on the data collectors.

If during the ON-LINE phase the *User Application* - generates a non-recoverable internal error, the system stops executing the standard sequence and enters in the SAFETY mode. In this case the *SafetyRealTimeStep*() function is called between the data input and output.

9. Internal debugger

The introduction of a simple debugger facility in the *JETRT* framework has proven to be very useful in commissioning the real-time systems. Thanks to this feature the application is aware of its own data structures. Therefore, it is possible to interrogate a program about the value of a variable by simply specifying its name. Consequently, it was very easy to provide a simple browser to interactively explore the internal values of both the *JETRTApp* and the *User Application*.

This feature has been developed by using CINT (The CINT C/C++ Interpreter, 2002), a very powerful C++ interpreter developed at CERN. A simple script is then used with CINT to create a C++ source file containing all the desired information about the data structures.

10. Conclusion

A software architecture designed to improve the development of real-time control systems at JET 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.

The current version of the *JETRT* framework is available on three different platforms by simply recompiling it: Motorola68k/VxWorks, PowerPC/VxWorks INTEL/WinNT4. The first two versions are used to develop safety-critical systems with hard real-time constraints. The last one is used to implement both measurements and control systems, which are non-safety-critical for the experiment. Moreover, the INTEL/WinNT4 platform can be 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 deployment, and this reduce the operational time requested for commissioning on the real plant.

In 2003, *JETRT* has been successfully used to develop the eXtreme Shape Controller 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 successfully tested. This system uses off-the-shelf tools (Real-Time Workshop User's Guide, 2001) to automatically generate the *User Application* from a Simulink diagram.

Linux port is very interesting when looking after cheaper hardware solutions, based on standard PCs rather than on expensive industrial computer boards. The automatic generation of the application plug-in is a step further into the reduction of development time.

References

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

Ariola, M., De Tommasi, G., Pironti, A., & Sartori, F. (2003). Controlling extremely shaped plasmas in the jet tokamak. In *42nd conference on decision and control*. Maui, Hawaii.

Behler, K., et al. (1999). Real-time information exchange between data acquisition and control systems at asdex upgrade. In *Proceedings of 11th IEEE NPSS real time conference* (pp. 264–267). Santa Fe, USA.

Cuatto, T., et al. (2000). A case study in embedded system design: an engine control unit. *Design Automation for Embedded Systems*(6), 71–88.

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

Lennholm, M., et al. (1997). Plasma vertical stabilisation at jet using adaptive gain control. In *Proceedings of the 17th SOFE conference* (Vol. 1) (pp. 539–542).

Lennholm, M., et al. (1999). Plasma control at jet. In *2nd IAEA technical committee meeting*. Lisbon.

Liu, J. W. S. (2000). *Real time systems*. Englewood Cliffs, NJ: Prentice-Hall.

Luchetta, A., & Manduchi, G. (1999). General purpose architecture for real-time feedback control in nuclear fusion experiments. In *5th IEEE real time technology and applications symposium*. Vancouver, Canada.

Moulin, D., et al. (1998). Real time data acquisition system for control and long pulse operation in tore supra. *IEEE Transaction on Nuclear Science*, *45*(4), 2033–2038.

Puppin, S., et al. (1996). Real-time control of the plasma boundary at jet. In *Proceedings of the 16th SOFT*. Lisbon.

Real-time workshop user's guide (2001). The mathworks inc. ed.

Saoud, S., Ben, D., Gajski, D., & Gerstlauer, A. (2002). Co-design of embedded controllers for power electronics and electric systems. In *Proceedings of the 2002 IEEE international symposium on intelligent control* (pp. 379–383). Vancouver.

Sartori, F., et al. (2004). The system architecture of the new jet shape controller. In *23rd SOFT*. Venice, Italy.

The CINT C/C++ Interpreter (2002). <http://root.cern.ch/root/Cint.html>.

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

Zanotto, L., et al. (2004). A new controller for the jet error field correction coils. In *23rd SOFT*. Venice, Italy.