EFDA–JET–CP(04)06-43

G. De Tommasi, F. Piccolo, F. Sartori, and JET-EFDA Contributors

A Flexible and Re-useable Software for Real-Time Control Applications at JET

A Flexible and Re-useable Software for Real-Time Control Applications at JET

G. De Tommasi[1], F. Piccolo[2], F. Sartori[1,2], and JET-EFDA Contributors*

[1]Associazione EURATOM-ENEA-CREATE sulla fusione, Via Claudio 21, 80125, Napoli, Italy
[2]Euratom/UKAEA Fusion Association, Culham Science Centre, Abingdon, Oxon OX14 3DB, UK
** See annex of J. Pamela et al, "Overview of Recent JET Results and Future Perspectives", Fusion Energy 2002 (Proc. 19th IAEA Fusion Energy Conference, Lyon (2002)).*

Preprint of Paper to be submitted for publication in Proceedings of the
23rd SOFT Conference,
(Venice, Italy 20-24 September 2004)

ABSTRACT

The fast growth of the JET real-time control network and the increasing demand of new systems have been the triggers that started the development of the JETRT software framework. This new architecture is designed for maximum reuse and is particularly suited for implementation of both real-time control and data acquisition systems in complex experimental environment such as JET. The most of the software is the same in all applications independently from the platform. The varying part is the project specific algorithm, which is also compiled into a separate software component, in order to achieve a separation from the plant interface code. Thanks to this design choice reliability is maximized, development costs have been reduced and even non-specialist programmers can easily contribute to the real-time project. JETRT also provides an integrated set of debugging and testing tools, some of them well integrated with the Matlab environment. This feature besides the framework portability among different platforms allows to perform the most of test and validation phase on a desktop PC running Window, reducing significantly the commissioning time of a new real-time system.

INTRODUCTION

At the heart of the JET machine, several real-time control, measurement and protection systems collaborate in order to satisfy the most sophisticated experimental needs. A group of these systems uses the JETRT software architecture: the *Extreme Shape Controller* (XSC [1],[2]), parts of the *Vertical Stabilisation* control and data acquisition systems, the *Error Field Coil Controller* (EFCC [3]). Many others can be found among the *Real Time Data Network* (RTDN [4]) processing nodes: the one measuring plasma internal parameters, the density and q profile estimator and two different real-time equilibrium codes.

During the *JETRT* development the main design aims were to separate the algorithmic part of a real-time application (*User Application*) from the interface software (*JETRTApp*), to standardize the application development, to achieve portability among the desired computer platforms (VxWorks/Windows/RT-Linux), and, as a result, to increase the code reusability and reduce the debugging efforts. The *User Application* is the central part of the target application and contains all the sophisticated mathematical algorithms that implements the desired behaviour of the overall realtime system. In order to interface with *JETRTApp*, this software module must be organized according to a set of strict designing and programming rules. These are aimed at making the User Application both a portable and reusable module that has no references to either the hardware or the operating environment. A further benefit is that any programmer with little real-time experience and no knowledge of the platform can still implement it. Working together *JETRTApp* and *User Application* implement the desired real-time system.

The next section gives an overview of the whole framework, while section 2 introduces *JETRTApp* architecture. Section 4 deals with the User Application component. Eventually some concluding remarks are presented.

2. JETRT OVERVIEW

The *JETRT* framework is a cross-platform class-library designed with the aim of helping development of real-time applications and providing validation tools to help the testing phase. Most of the time developing a 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, for the experimental fusion applications the problem mostly lies elsewhere. The commissioning time is actually spent more on the algorithmic part of the code than in the interfaces or the real-time synchronisation, because whereas 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 normally developed separately on specialised 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 with 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. A plug-in based open-loop simulator *Application Tester* (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 closed loop systems, this method normally allows finding most of the problems in the code before testing it on the plant. A more thorough test can be performed by loading the application module into Simulink, using the *S-function* interface [5]. In this environment it is either possible to compare directly 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. A data feeder based on the ATM Real-Time Data Network [4] 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.

3. 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 off-pulse interfaces are the origin of much of the technical complexity of the real-time applications. Typically before and after every pulse the *Countdown System* sends to *JETRTApp* change of state requests in order to synchronise the evolution of the various JET subsystems. As soon as the scientists have finished pre-programming a new experiment, the Level-1, the plant management system that provides customized and automated user interface, sends a packet containing new parameters to each JET subsystem. Finally, the information collected has to be available for sending to *GAP* (General Acquisition Program) data management system. *JETRT* framework has been created to help working in this environment, answering the need for a fast and reliable deployment of new systems.

The block diagram of a generic real time application in Fig. 1 shows the *JETRTApp* connections between the JET external systems the other *JETRT* components.

The I/O Drivers plug-in system allows the customisation of the data acquisition hardware. 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* structure. Following the setting in the file, the system initialises the necessary I/O Drivers, loads the desired User Application plug-in, allocates the data collector memory and starts the interfaces with the external systems.

Figure 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.
- 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 plug-in 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.

4. 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.

After several attempts at finding a complete Application Program Interface, the present version of the program implements the functions described in Table 1. The *User Application* is implemented as a C++ dynamic loadable object. Once the module 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. 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. After the **PulseStart()** 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 standardize real-time applications 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 standardised 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 platforms, 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. Eventually the code of a new system is well tested before the deploying and this will reduce the request operational time for the commissioning.

REFERENCES

[1]. G. Ambrosino, M. Ariola, A. Pironti, F. Sartori, A New Shape Controller for Extremely Shaped Plasmas in JET, Fusion Engineering and Design **66-68** (2003) 797-802.

[2]. M. Ariola, G. De Tommasi, A. Pironti, F. Sartori, Controlling extremely shaped plasmas in the JET tokamak, 42nd IEEE Conference on Decision and Control, Hawaii, 2003.

[3]. L. Zanotto, F. Sartori, M. Bigi, F. Piccolo, M. De Benedetti, "A new controller for the JET error field correction coils", 23rd Symposium on Fusion Technology, Venice, Italy, 2004.

[4]. R. Felton et al., "Real-time plasma control at JET using ATM network," Proceedings of 11 th IEEE NPSS Real Time Conference, Santa Fe, 1999, pp. 175-181.

[5]. The Mathwors, Writing S-function, Version 5, The Mathworks Inc., 2003.

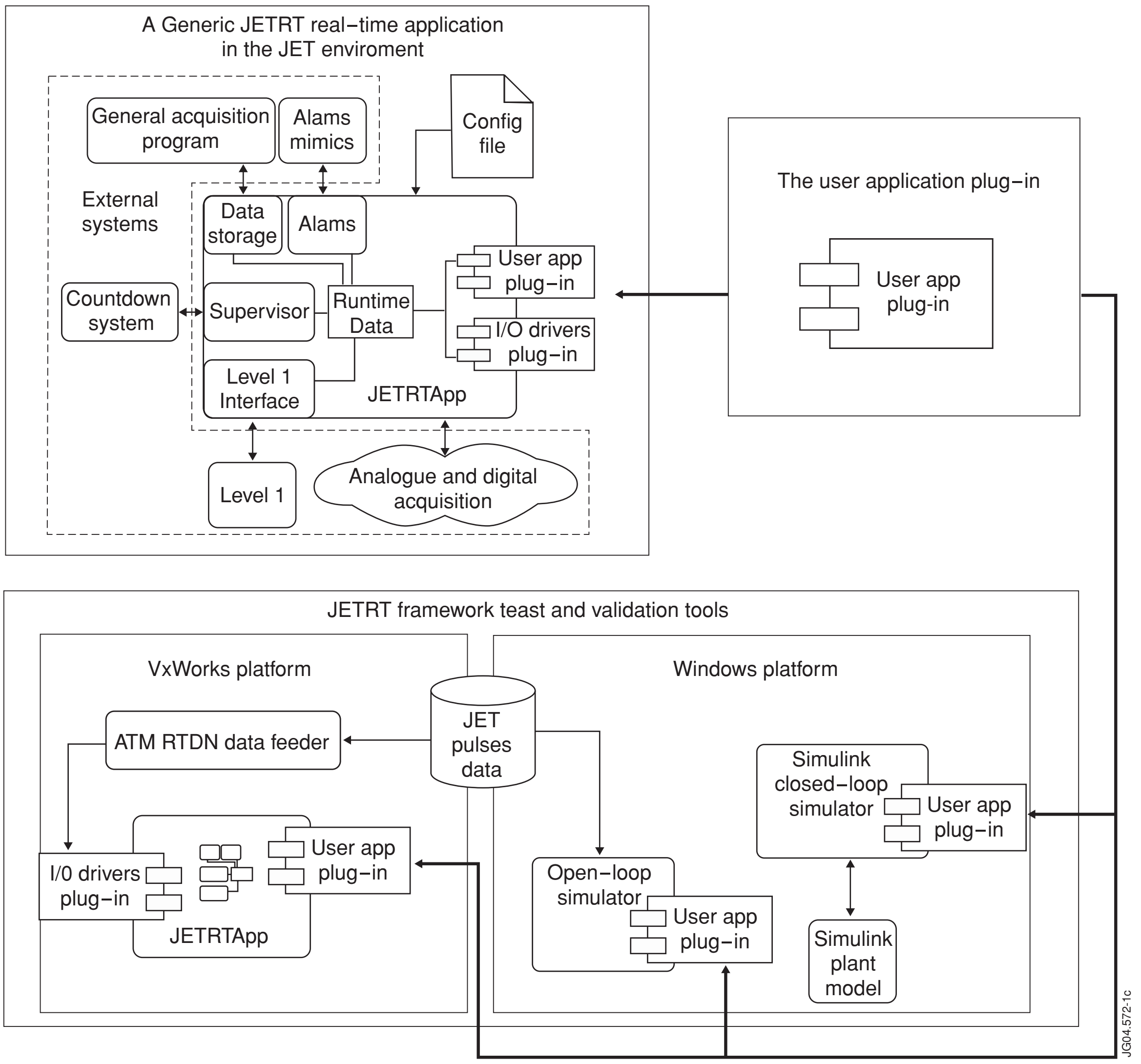

Figure 1: JETRT framework overview.

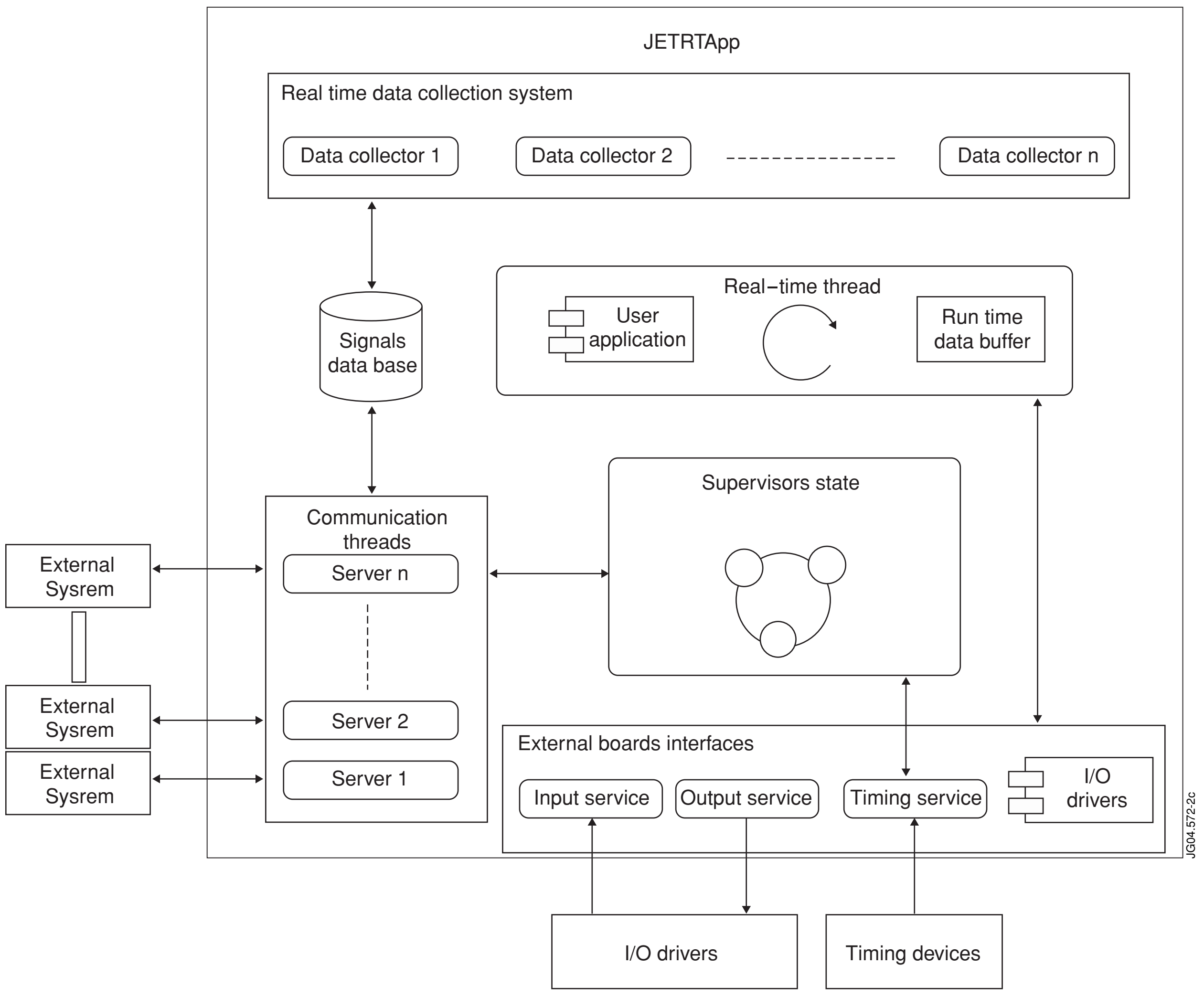

Figure 2: JETRTApp connections between the other JETRT components and the external systems.