JAVA BASED SUPERVISION OF DIGITAL FEEDBACK SYSTEMS IN THE RFX NUCLEAR FUSION EXPERIMENT

A. Luchetta, G. Manduchi, C. Taliercio, Consorzio RFX, Padova, Italy

Abstract

RFX is a large toroidal machine in the European Programme for nuclear fusion. Digital active feedback systems are used in RFX for the control of the axisymmetric magnetic configuration and for the correction of local magnetic field errors. A general scalable distributed architecture has been realised for the RFX feedback systems. It is based on processor networks using Digital Signal Processors (Texas Instruments TMS320C40) mounted on VME racks and connected either directly or via dedicated fibre optic links. The software architecture defines a general framework in which software components can be defined and configured. Each component is associated with a processor and is responsible for the overall data flow management. A transfer function is then associated with each component in order to define the algorithms performed by the associated processor.

A graphical tool written in Java provides the overall handling of the control systems. For each system component it allows the data flow configuration and the retrieval of both status and set-up information.

1 INTRODUCTION

RFX uses active digital feedback for the control of machine axisymmetric electromagnetic parameters, such as the vertical field required for plasma equilibrium and the reversed toroidal magnetic field at the wall, and for the control of local magnetic errors in critical parts of the magnetic structure. New feedback systems are also under development in order to enhance the magnetic configuration and to control the position of wall locked MHD modes. These various feedback control systems have different timing requirements, ranging from a required throughput of 1 to 20 kHz. Moreover they may have to deal with a large number of input/output signals. The control of the local field errors, for example, acquires about 50 analogue input signals and produces 22 output reference waveforms. In addition, input and output signals are usually acquired and emitted in different experiment halls, located at distances up to hundred meters.

As a consequence of the above facts, the architecture of the feedback systems for RFX has been developed to fulfil the following requirements [1]:

- Scalability in performance, to allow the implementation of systems with different timing and I/O requirements;
- Support for distributed multiprocessor systems providing data communication over relatively long distances (hundreds of meters);
- Configurability in both system topology and algorithm definition.

Since high reliability is required, the decision of using commercial of the shelf hardware components (COTS) has been taken in the early stage of the project.

The architecture adopted is modular: a hardware platform has been selected and a general software framework has been developed, which allow the definition of systems as sets of co-operating modules, performing computation and exchanging data. For each module, the characteristics of the data flow are specified, along with the relevant data processing. The configuration of the data flow is specified in a set of include files, whilst the definition of the computation is achieved by means of a dedicate language, which is then parsed in order to produce optimised C code. Besides making the development of new systems easier, the use of the same framework for different applications increases the overall reliability since new systems re-use tested code.

Though no new C code is written when developing new systems, their configuration implies however the definition of many parameters and requires deep understanding of the framework internal organisation. In order to make the composition of modules easier, a graphical tool is currently under way for assisting developers in the configuration of new systems. The tool, written in Java, allows the definition of the system distributed architecture, the data flow and the computation, producing then the required include files and generating the C code for computation. The tool also provides a certain level of supervision by checking modules against each other in order to highlight possible inconsistencies in the overall system configuration.

2 ARCHITECTURE OVERVIEW

The modular architecture of the framework influenced both the choice of the hardware and the development of the software.

2.1 Hardware Configuration

The main factors that influenced the choice of the hardware have been performance and distribution. Consequently, a powerful CPU was needed with the ability of supporting high-speed communication with other CPUs. Moreover, it was important to de-couple as far as possible data flow from computation in order to achieve performance. For these reasons, we selected the TMS320C40 floating point Digital Signal Processor (DSP), which is equipped with 6 independent communication ports (comports) and 6 DMA controllers [2]. The CPU boards are mounted on VMEbus racks [3]. A different CPU, acting also as VMEbus controller, handles the communication with the central control system of RFX via Local Area Network. It is also used to load set-up parameters before the experiment pulse and to collect diagnostic data after the pulse.

Data communication between CPUs mounted on the same VME rack is performed via comports, while CPUs mounted on different racks need dedicated high-speed links. This has been achieved by using reflective memories connected to fibre optic links, which provide a memory area in the VME addressing range and ensure that updates in memory locations are reflected into all other reflective memories [4].

Comports are also used to move raw data from A/D Converters to CPUs [5]. The VMEbus is used for communication between CPUs and reflective memory boards and also between CPUs and D/A boards. Communication over the VMEbus represents a possible bottleneck in the system performance because of the possible contentions both in the VMEbus and in the VMEbus Interface chip. Care is therefore required in the definition of the topology for new systems in order to minimise data transfer over the VME buses.

2.2 Software Architecture

The framework defines two kinds of software modules: the *pre-elaboration* and the *control* modules. Pre-elaboration modules acquire raw data from A/Ds, perform some kind of data pre-processing, and send the results to one or more control modules. Control modules receive data from one or more pre-elaboration modules, perform control computation, and send the results to D/A converters. Control modules can also send elaborated data to other control modules, thus allowing the split of the required computation in a pipeline of communicating modules. Modules can be freely assigned to CPUs with the only constraint that one CPU can serve only one module. Communication between modules is achieved either through comports or by means of reflective memory.

By connecting pre-elaboration and control modules it is possible to build distributed control systems. The simplest system that can be implemented is composed of one pre-elaboration module connected with one control module, but more complex configurations can be defined. For example, figure 1 illustrates the system used for the control of local field errors, which defines two pre-elaboration modules and one control module, mounted on two VMEbus racks. The first pre-elaboration module runs on a rack installed in the diagnostics control room: it

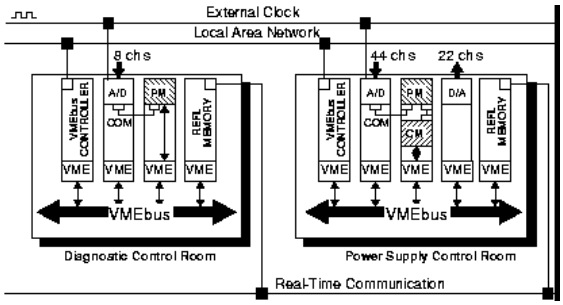

Figure 1: Example of hardware and software system architecture

acquires 8 input signals, performs data format conversions and spatial analysis. It then sends the results through the reflective memory to a control module mounted on a rack installed in the power supply room. This module receives also pre-processed data from another pre-elaboration module mounted on the same board, which acquires and elaborates 44 input signals produced in the power supply control room.

The framework carries out all the functions related to the data movement, but does not make any assumption on the kind of computation being performed, which is carried out by a separate piece of code. As control algorithms are often changed in order to investigate better plasma configurations, a straightforward implementation of the computation as a C routine would require the assistance of software specialists each time the control algorithms are changed. Moreover, the development of new routines may introduce programming errors that are usually very hard to diagnose. For these reasons, we decided to develop a code generation tool which can accept a high-level description of the control algorithm, directly expressed in the Z or Laplace transform domain, and produces optimised C code, which is then integrated in the framework at build time.

3 THE CONFIGURATION TOOL

The configuration of the pre-elaboration and control modules requires the definition of several (10-15) parameters for each module, such as the comports involved, the amount of the data being exchanged and the location of data in the reflective memory. Due to our experience, the manual definition of these parameters in a set of include files is not immediate and configuration errors are common, especially when defining complex systems composed of several modules.

For this reason, a graphical tool for system configuration is currently being developed. Its main tasks are the following:

- Overall organisation of the system topology: definition of pre-elaboration and control modules, and description of the associated data flow. A consistency check will then validate the configuration;
- Computation description: generation of the input/output parameters of the transfer functions, based on the current topology of the system, and editor support for the detection of syntactical errors in the descriptive language;
- Run-time system monitoring in order to report the status of every CPU involved in the control chain during experiment sessions.

We decided to use Java for the development of the configuration tool. Java in fact provides a powerful and flexible environment in which graphical applications can be quickly developed and, due to their object-oriented organisation, easily modified. Moreover, the thread and network supports provided by the Java environment are useful for the integration of the run-time system monitoring in the graphical tool.

The configuration tool is essentially a configuration data repository, and its interface presents information in a hierarchical fashion as shown in figure 2. At the top level is the definition of the control systems (e.g. Local Field Error in fig.1), each involving a set of VME racks (e.g. Signal Server, Poloidal, Equatorial in fig.1), each mounting a set of CPUs (CPU_A, CPU_B). At each level the corresponding information is presented. For example, at the VME rack level, the IP address of the VME controller is defined, and this information is shared by all the underlying CPU configurations, in order to let the system download the code and acquire CPU parameters. Most information is defined at CPU level, mainly to define how data are exchanged between the corresponding module and the rest of the system.

Once the configuration has been entered, the tool performs a consistency check in order to detect wrong configurations, such as different data sizes for two communicating modules. A further check is done to detect wrong timing configurations. Pre-elaboration modules, in fact, acquire and elaborate raw data at the sampling speed of the corresponding A/D converters. Since the inter-module data communication protocol requires produced data to be consumed before a new sample is sent, the throughput of the control modules depends on the throughput of their input modules. If, for example, a control module is configured to receive data from two modules with different throughputs, the system will inevitably loose data. Frequency division can be defined for the output data in each module, making thus possible the definition of systems with modules working at different speeds.

Based on the current system topology, the tool provides also support for the development of the computation description, by generating templates defining the input and output parameters for each module.

Once the configuration has been entered and validated, and the computations have been described in the high level language, the tool generates the source code and the include files for each module, builds the executables and downloads them to the corresponding CPUs.

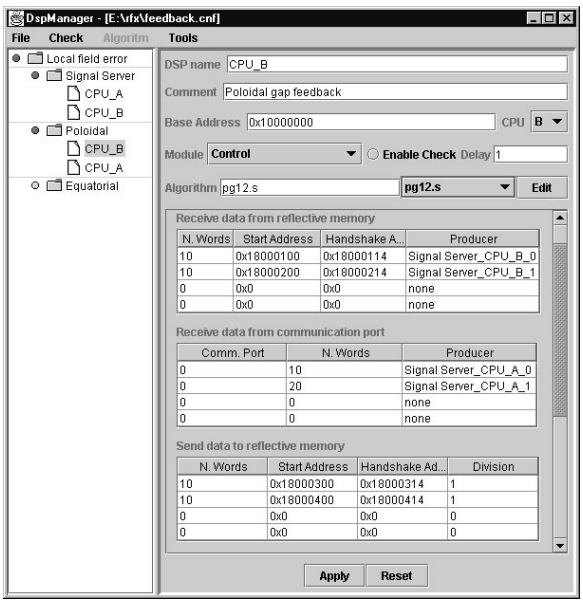

Figure 2: Example of Graphical Configuration Tool

4 CONCLUSIONS

The development of a graphical configuration tool represents the last step towards the final implementation of a powerful and versatile framework for high demanding control systems. The framework is currently being used at the RFX experiment for a variety of feedback controls.

The initial effort in the development of a general-purpose architecture, rather than providing straightforward implementations of every single control system, proved to be a right choice, since it allowed a dramatic reduction in development and test time for new control systems.

REFERENCES

[1] A. Luchetta, G. Manduchi, "General Purpose Architecture for Real-Time Feedback Control in Nuclear Fusion Experiments", proceedings 5th IEEE RTAS, Vancouver, Canada, June 2-4, 1999

[2] Texas Instruments, TMS320C4X User Manual

[3] Blue Wave Systems Ltd., DBV46 TMS320C4X Carrier Board, Technical Reference Manual

[4] VMIC, VMIVME-5588DMA Reflective Memory Board Product Manual

[5] Pentland Ltd., VGX User's Guide