Service-oriented architecture of adaptive, intelligent data acquisition and processing systems for long-pulse fusion experiments

J. González[a], M. Ruiz[a,*], E. Barrera[a], J.M. López[a], G. de Arcas[a], J. Vega[b]

[a] Grupo de Investigación en Instrumentación y Acústica Aplicada. Universidad Politécnica de Madrid, Crta. Valencia Km-7 Madrid 28031 Spain
[b] Asociación EURATOM/CIEMAT para Fusión, Madrid, Spain

ARTICLE INFO

Keywords:
Long-pulse experiments
Intelligent data acquisition
PXI
Real time data processing
Service-oriented architecture
JINI
SCXML

ABSTRACT

The data acquisition systems used in long-pulse fusion experiments need to implement data reduction and pattern recognition algorithms in real time. In order to accomplish these operations, it is essential to employ software tools that allow for hot swap capabilities throughout the temporal evolution of the experiments. This is very important because processing needs are not equal during different phases of the experiment. The intelligent test and measurement system (ITMS) developed by UPM and CIEMAT is an example of a technology for implementing scalable data acquisition and processing systems based on PXI and CompactPCI hardware. In the ITMS platform, a set of software tools allows the user to define the processing algorithms associated with the different experimental phases using state machines driven by software events. These state machines are specified using the State Chart XML (SCXML) language. The software tools are developed using JAVA, JINI, an SCXML engine and several LabVIEW applications. Within this schema, it is possible to execute data acquisition and processing applications in an adaptive way. The power of SCXML semantics and the ability to work with XML user-defined data types allow for very easy programming of the ITMS platform. With this approach, the ITMS platform is a suitable solution for implementing scalable data acquisition and processing systems based on a service-oriented model with the ability to easily implement remote participation applications.

1. Introduction

Data acquisition systems for long-pulse fusion devices must include processing capabilities in order to be able to run data reduction and/or pattern recognition algorithms to detect known behaviours in the acquired signals. In addition, the type of algorithms to be used might change during the experiment depending on its evolution. The use of intelligent data acquisition systems (iDAQ) simplifies the setup and definition of these complex experiments. A data acquisition and processing system can be considered intelligent when it is capable of running autonomously and adapting its behaviour to the changes in the environment. A traditional data acquisition and processing system acquires all input signals and processes them immediately, repeating this cycle continuously. On the other hand, an iDAQ decides which signals must be acquired and which processing algorithms must be run for every cycle. This paper presents a model of such a system with adaptive processing capabilities that are based on two crucial characteristics of the platform: its ability to change processing elements dynamically during acquisition (software hot swapping) and the ability – introduced by the use of the SCXML language [1] – to define state machines that organise the sequence of operations depending upon the experiment's evolution over time.

In addition, DAQ systems must be easy to integrate into distributed systems, and therefore must include specific tools to manage and use them efficiently. The use of a service-oriented architecture (SOA), as proposed by JINI [2], simplifies and standardises the development, integration, deployment and maintenance of such systems [3]. All described elements have been included in a platform known as the Intelligent Test and Measurement System (ITMS) developed by Universidad Politécnica de Madrid and CIEMAT.

Fig. 1 shows the components of the ITMS platform. The figure shows the deployment of a N nodes distributed system for data acquisition and processing. Each node has a PXI or PXIe including DAQ cards and CPUs, and an additional computer. This computer commands the hot swap of processing elements according to the temporal evolution of the experiment that is defined using a state diagram. These features are available to network clients as a remote service. The deployment of services in the model proposed by JINI requires the presence of a Lookup Server. Its mission is to publish the services that are on the N nodes and subscribe the remote clients. Thus, the remote client (represented in the figure by a PC and a user) can be listener of the events occurring in the system

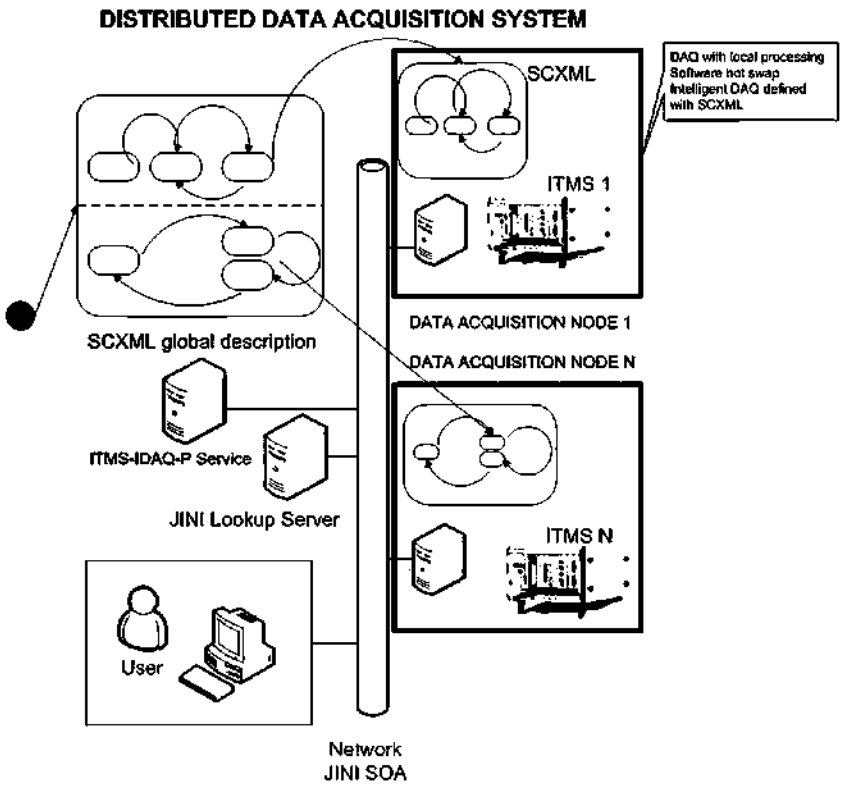

Fig. 1. DDAQ implementation with ITMS.

and can set the behaviour (state machine) of each node. The figure also represents the possibility of implementing a service capable of command and synchronises the N nodes state machines.

2. ITMS architecture: basic description

The ITMS platform consists of a set of CompactPCI hardware elements and specific software applications primarily developed in LabVIEW and JAVA. Its main goal is to simplify the development of data acquisition applications that require processing capabilities during acquisition (online processing) [4,5]. The development of online processing techniques based on pattern recognition and data reduction are considered to be essential for long-pulse fusion devices. The need for simple tools that make it possible to adapt the selection of processing algorithms depending on the temporal evolution of the experiment is a great challenge for such systems. A key aspect of the ITMS platform for the development of such systems is its local distributed processing capability, which links with its global distributed capability through the deployment of all functionalities as remote services.

The ITMS platform consists of the following hardware elements:

- A 6U or PXI 3U CompacPCI chassis that supports both PCI and PCIe standards.
- A system controller known as the system CPU (SCPU).
- Additional CPU cards in peripheral slots known as processing CPUs (PCPUs).
- Data acquisition cards that depend on the applications.
- An external host for the deployment of the remote services needed in case the operating systems used in the SCPU and PCPUs do not support JAVA virtual machines.

The ITMS software architecture is based on a set of software applications (see Fig. 2) that run in both the SCPU and the PCPUs, namely Setup&DAQ, DDPS (Dynamic Data Processing System) and IDAQ&P (Intelligent data acquisition and processing). By using the PCPUs, the SCPU is freed from processing tasks and can focus on data acquisition and distribution to PCPUs.

Setup&DAQ running in the SCPU deals with the DAQ setup, the data acquisition process and its distribution among the different CPUs. Setup&DAQ running in PCPU receives data and sends them to DPPS. Setup&DAQ can be run under Linux (with the Real Time Application Interface extension for Linux–RTAI), Windows and LabVIEW/Real Time. In Linux, data acquisition and distribution have been implemented by means of kernel modules running in RTAI. These modules use the COMEDI [6] data acquisition driver to manage data acquisition cards. In Windows and LabVIEW/Real Time, the application has been developed using National Instrument's DAQmx driver. In all platforms, the acquired data are transmitted to the PCPUs and stored in FIFO buffers to make it available for the data processing algorithms (hereafter "processing algorithms") that run in the DDPS module. The management of Setup&DAQ is based on a set of XML commands to setup, start and stop this module.

DDPS runs the lists of processing algorithms defined by the user through Setup&DAQ. These processing algorithms, which must be developed in LabVIEW, must be downloaded to the ITMS platform before launching DDPS. DDPS uses a software manager module–DDPSmanager–that allows for the addition and removal of processing algorithms using XML commands without stopping the system. This capability is the origin of the software hot swapping feature. When ITMS modules are starting, the software applications running in the PCPUs communicate with those running in the SCPU to confirm their correct operational state, i.e., to confirm that they are able and ready to acquire and process data, as defined by the user requirements. The communication between counterpart software modules that run in different hardware machines is done through XML messages sent over TCP/IP to the DDPS software manager. The final start-up state is saved in a global variable that stores the input channels that are being acquired, the processing algorithms that are running and any error that might have occurred in

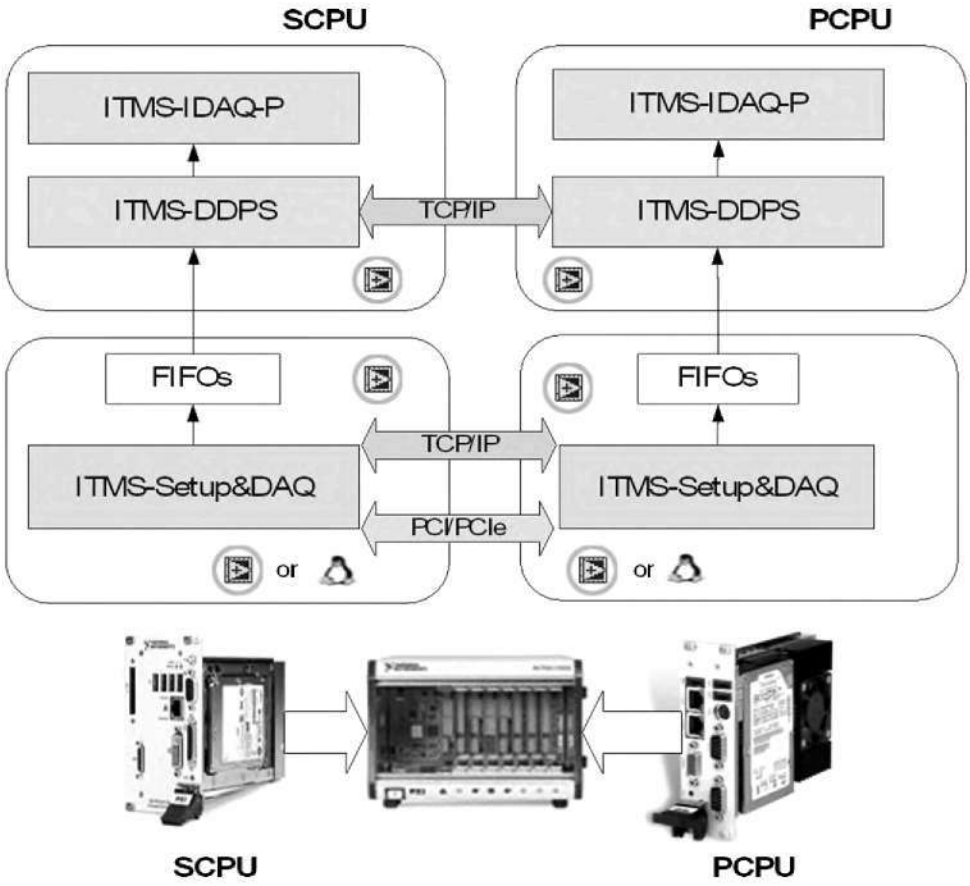

Fig. 2. Diagram showing the software applications running in the PCPUs and SCPU.

the system during the process. The content of this global variable can be accessed remotely for supervision purposes.

The ITMS-IDAQ-P application consists of a LabVIEW module and a JINI service. If the operating system of the SCPU/PCPU running the module does not support JAVA virtual machines, an external host must be used to deploy the JINI service. The LabVIEW module is responsible for receiving the events generated by the algorithms running in the DDPS, which are the processing units defined by the user for every phase. Events are produced in response to a property change in the signals that are being acquired, and once received they are transmitted to the JINI service. If they produce a transition in the state machine that defines the experiment dynamics, the service will react by sending a notification to all system components indicating the new state. Notifications result in commands to the DDPS manager to remove, upload and run the corresponding processing algorithms in response to the updated system state.

ITMS-IDAQ-P's JINI service conforms to the SOA paradigm, so it must be published in at least one Lookup Server, thus allowing authorised network users to subscribe to it in order to have remote access to the functionalities it offers. The functionalities and implementation details are described in Section 4. JINI service deployment of ITMS-IDAQ-P is shown in Fig. 3.

3. Experiment modeling with the ITMS platform

In general, a data acquisition and processing experiment consists of a set of phases during which the signals that must be acquired and the processing algorithms that must be run might be different. Each phase has an associated profile that holds the configuration of the acquisition and processing parameters. Modeling an experiment in ITMS consists of defining the static knowledge (which input channels to acquire and which algorithms to use) and the dynamic behaviour (which events have to be detected, what system transitions they produce and which algorithms must be changed in response). The static behaviour is described through profiles in a file using XML. Each profile defines the processing algorithms (Virtual Instrument) that must be run for a set of input channels.

The rules for the order in which to use each profile are defined using SCXML language, making it very simple to determine the logic of the adaptive acquisition and processing depending on the experiment's evolution with time.

In the aforementioned circumstances, two different scenarios can be defined: individual behaviour, where every ITMS node acquires and processes data independently following the behaviour defined in a state machine, and collective behaviour, where several nodes acquire and process data in parallel but following the logic established by one state machine.

Fig. 4 shows an example of an individual ITMS system where two profiles with two different processing algorithms (processing A and processing B) are defined. In the first state of the state machine (Initial state), the profiles are read from the Profiles.xml file and loaded in the profiles list (profiles[] variable). Immediately afterward, the state machine goes to state 01 and loads the first profile, activating the A processing algorithm. The state machine will remain in this state until the event defined as "Event x" is detected, which will produce a transition to state 02. At this point, the second profile will be loaded so that the processing algorithm will be changed to processing algorithm. The state machine will remain in this state until the event defined as "Event y" is detected, which will produce a transition back to state 01, and so on.

The versatility of the SCXML language allows for the definition of different state transitions depending on variable values.

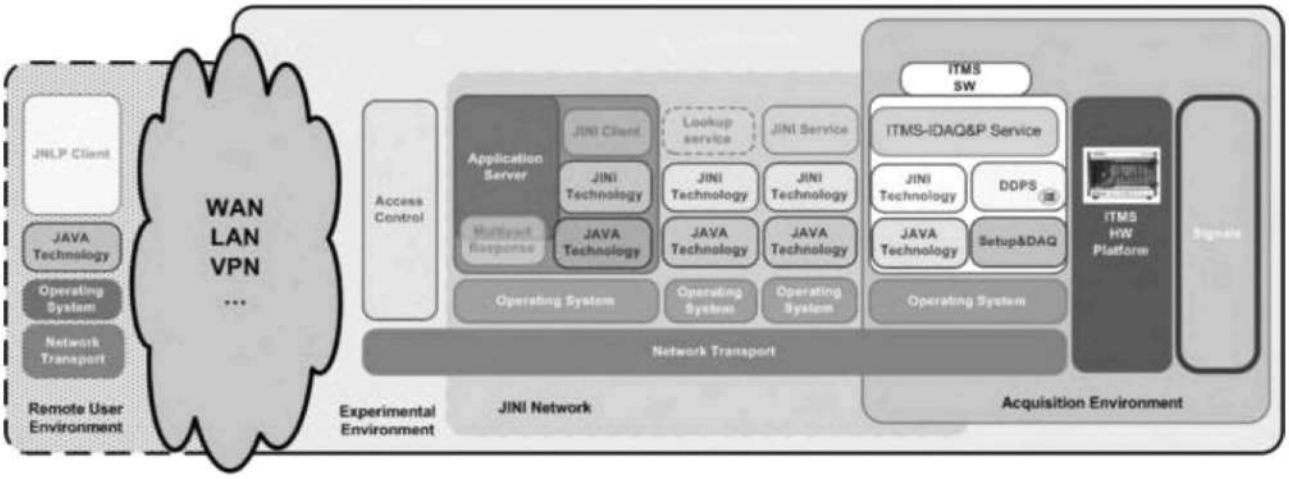

Fig. 3. Deployment scheme of the JINI service of ITMS-IDAQ-P.

The basic elements that provide the adaptive characteristics of the system are as follows: the DDPS Manager, responsible at any time for changing the processing algorithms being executed by using two operations named "removing" and "adding", and an SCXML engine based on the Apache Commons SCXML [7] implementation that is capable of running a state machine defined in SCXML. This engine runs on any CPU in the system that supports a JVM. Two additional objects, a JEXLContext and a JEXLEvaluator, have been included in order to be able to evaluate advanced syntax expressions.

From the user's point of view, the following tasks must be performed to set up the system. First, the different phases of the experiment, the states, and the events that will produce the transitions between them must be identified. A profile must be defined for each of them specifying the acquisition and processing parameters (which input channels to acquire, with which configuration and which processing algorithms must be run with the acquired data). This process generates an XML file with a set of programs and profiles. The user must then develop the programs that will process the acquired data for each profile. These programs will include the detection of the events of interest previously defined, and they will use the communication structures provided by ITMS-IDAQ-P to notify the system of the need for a state change.

Finally, the system dynamics must be defined through a state machine, which will typically include an initial state that loads all the profiles from the configuration file into an indexed variable. The system will change automatically to the first phase of the experiment. Then, for every phase, the events and transitions must be defined. In the definition, the options parameter must be used if a transition requires the system to unload a processing algorithm (remove option). Finally, for every state, the set of operations to execute must be defined from the following set: open_profiles (reads the profiles from the XML file and loads them into an indexed variable); run_profile (runs the profile specified by the indexed variable); and run_profile_options (runs the profile specified by the indexed variable, but using the options specified in the state transition).

During execution, when an event is detected in a processing algorithm, it will be transmitted to the SCXML engine to produce a change of state. This requires the list of possible events to be included in the definition of the state machine. The versatility of the SCXML language allows for this transition to be defined

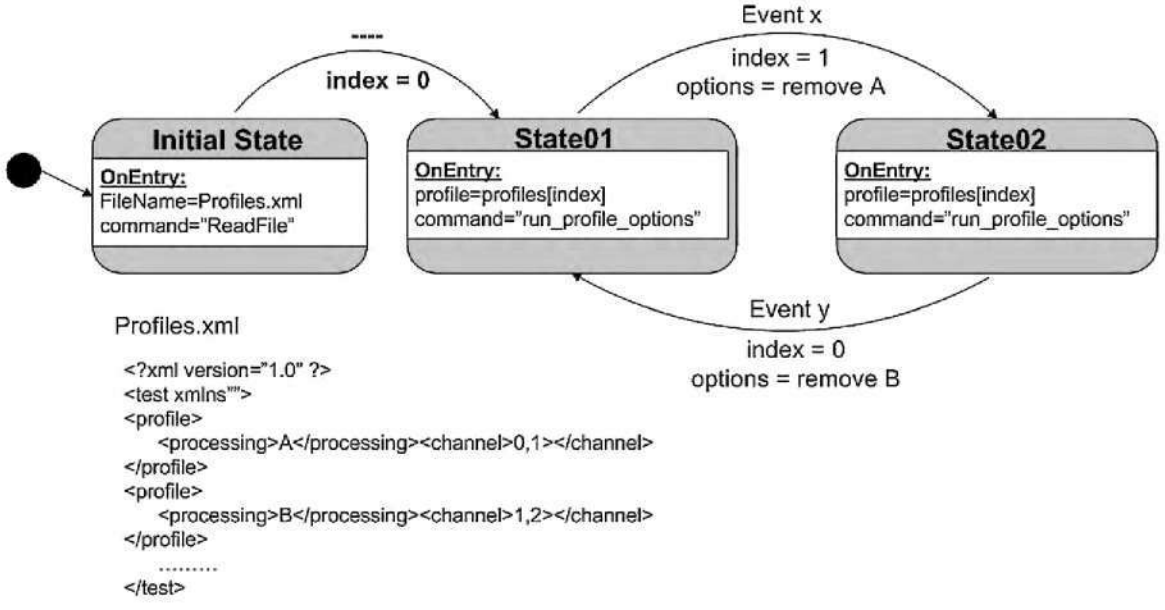

Fig. 4. Experiment model with state machine.

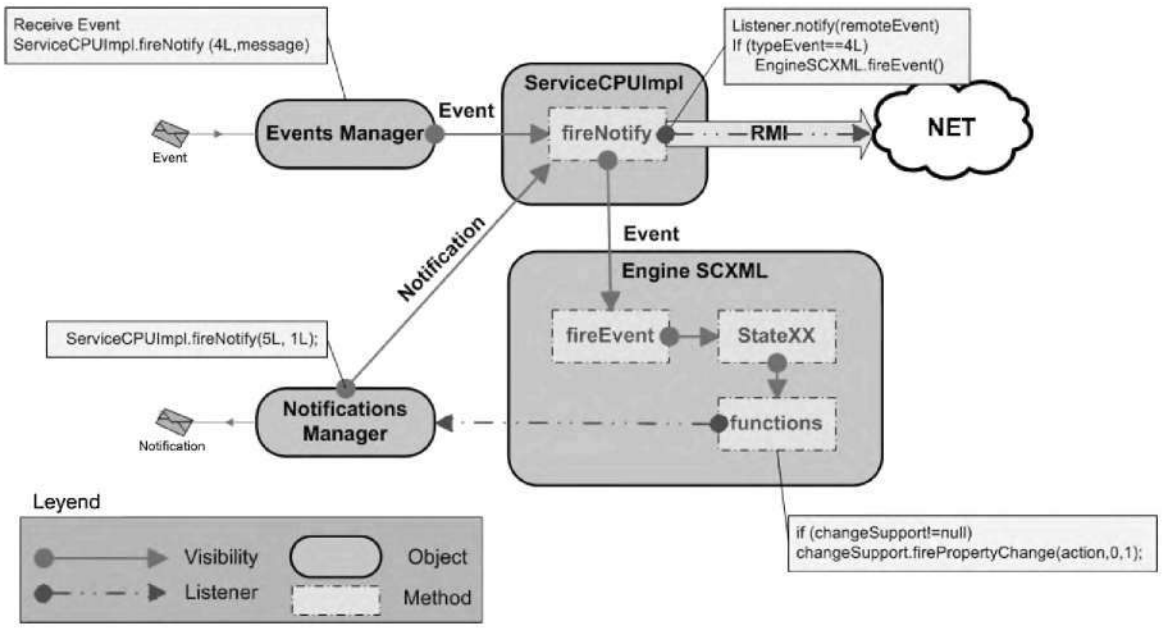

Fig. 5. Behaviour of the ITMS-IDAQ-P service.

using conditional variables. Other options include: definition of sub-states, use of parallelism, synchronisation, and use of onEntry and onExit actions associated with every state. SCXML provides a set of advanced programming resources that are easy-to-use.

4. Functionalities of JINI Service in ITMS platform

The JINI service that deploys the ITMS functionalities is the base for the concept of the service-oriented architecture. In the current version, it is implemented with a JAVA object named ITMS-IDAQ-P that is published through a lookup service to make it available to remote clients [3,8]. The following methods are available in this object:

- Obtain/update the configuration of the hardware elements present in the system. Any modification in the system's hardware can be detected automatically by the user.
- Upload the processing algorithms developed as LabVIEW Virtual Instruments (Vis) so they can be run by the DDPS. All processing elements that are developed in LabVIEW can be uploaded in any node.
- Send the XML file containing the SCXML description of the state machine that defines the system dynamics.
- Run the SCXML engine that governs the behaviour of the system by running the state machine previously defined.
- Supervise the system state through the reception of monitoring events that are automatically transmitted to all listeners that have subscribed to this service. These events include the current state of the state machine.

The internal behaviour of the ITMS-IDAQ-P module is diagrammed in Fig. 5. The event, generated by the running VIs in the DDPS, triggers the execution of the fireNotify method. A notification is sent to the subscribed listeners and produces a call of the fireEvent method of the SCXML engine. If the event produces any transition in the state machine, the method associated with the new state will be executed (there must be correspondence between the state name and the method name), after that a call to "function" method is executed to perform the corresponding actions depending on the context values. Finally the invocation of fireProperty Change method of the subscribed changeSupport. The latter will send a notification through TCP/IP to the DDPS Manager to execute the desired operations.

5. Discussion

One of the main goals of the ITMS platform is to reduce the effort and complexity required to define complex acquisition and processing tasks for long-pulse experiments. As a result, the user can focus on developing the processing algorithms in a high level programming language and on the definition of the system behaviour using familiar, easy-to-use standard state machines. This requires determining the experiment's states, or phases, and the events and transitions between them and expressing them in SCXML. These features define the adaptive and intelligent functionality of the ITMS platform.

The evaluation of this model (using qualitative measures) shows simplicity and flexibility The simplicity is due to the fact that the processing units do not need to include additional code to define the sequencing of the operations (system transitions) or their time synchronisation. Flexibility is understood as the capacity to adapt the system and to reuse its components for different scenarios. From a quantitative point of view, it is necessary to determine the latencies introduced by the software modules responsible for defining and managing the system's behaviour, i.e., the additional processing load.

Several tests have been performed to obtain quantitative measures using a PXIe-1062Q chassis with the following hardware elements:

- A PXIe-8130 embedded controller (2.3 GHz Dual-Core PXI Express) as SCPU running RT_LABVIEW, responsible for data acquisition and its distribution to the PCPUs.
- An ICP-P4 2 GHz processing card running Windows XP, acting as PCPU to run the processing algorithms.
- Several data acquisition cards, model PXI 6070E.

The system has been monitored with a Standard PC (3 GHz z Core 2 Duo running Windows XP Professional) through an Ethernet network using TCP/IP with all devices in the same subnet.

The most relevant time parameters obtained for several trials are:

- Time needed to remove an active processing algorithm (remove): 4 ms
- Time needed to add a processing algorithm (add): 1 ms
- Time needed to run the processing algorithm with the input channel configurations (launch): 18 ms
- Time needed to transmit an event, to produce a transition of the SCXML engine and to transmit the corresponding notifications: 15 ms
- Time required for the rest of the code: 17 ms
- Average mean time: 55 ms

The 55 ms total average time corresponds to the time an event is detected in a processing algorithm until the new processing algorithm is running, according to the state machine definition. The FIFO memories present at the input of the DDPS module guarantee that no data is lost during this 55 ms. In order to show in detail a real application of this system, an example is described in [9].

6. Conclusions

The ITMS platform is a service-oriented architecture of adaptive and intelligent data acquisition and processing systems for long-pulse fusion experiments. It is a service-oriented architecture because all functionalities are deployed as remote services that can be located and used transparently by the user through lookup services. It is adaptive and intelligent because the definition of the experiment includes the specification of the system dynamics, where it is possible to add and remove processing units in response to events without the need to stop the acquisition system and without any loss of information.

ITMS proposes an advanced DAQ model based on the local processing capabilities of CPU nodes (SCPU and PCPUs) and on the capacity to build a DDAQ solution based on the node's capacity to export functionalities using services. The ability to define the DDAQ's intelligent based on the SCXML behaviour of the different nodes coordinated by a supervisor host is an important challenge.

The model proposed has many advantages, but also some limitations that reduce the performance of the system. ITMS uses a SCXML engine, so we need to have JVM installed in the SCPU and PCPUs. This situation limits the performance of the system. DAQ and distribution tasks run in real time, but JAVA, JINI service and the SCXML Engine do not. Also because we use an Ethernet network, we have an additional problem with latencies of communications events

There are other technologies like JINI providing ubiquitous services such as UPnP. The differences, advantages and disadvantages between UPnP and JINI are reflected in Refs. [10–11]. Ref. [12] shows the evaluation of both technologies in order to develop architectures for ubiquitous environments.

Acknowledgements

This work is funded by the Spanish Ministry of Science and Technology under the Projects No. DPI2006-06624, ENE2009-10280 and No. ENE2004-07335.

References

[1] http://www.w3.org/TR/scxml/.
[2] http://www.jini.org/wiki/Main_Page.
[3] J. González, M. Ruiz, E. Barrera, J.M. López, G. de Arcas, J. Vega, Configuration and supervision of advanced distributed data acquisition and processing systems for long pulse experiments using JINI technology, Fusion Engineering and Design 84 (2009) 832–836.
[4] M. Ruiz, J.M. López, G. de Arcas, E. Barrera, R. Meléndez, J. Vega, Data reduction in the ITMS system through a data acquisition model with self-adaptive sampling rate, Fusion Engineering and Design 83 (2008) 358–362.
[5] M. Ruiz, E. Barrera, S. López, D. Machón, J. Vega, E. Sánchez, Distributed real time data processing architecture for the TJ-II data acquisition system, Review of Scientific Instruments 75 (10) (2004) 4261–4264.
[6] http://www.comedi.org/.
[7] http://commons.apache.org/scxml/.
[8] E. De Las Heras, D. Lastra, J. Vega, M. Ruiz, R. Castro, E. Barrera, Web Based System Architecture for Long Pulse Remote Experimentation, Fusion Engineering and Design 85 (3–4) (2010) 292–297.
[9] M. Ruiz, J. Vega, E. Barrera, J. González, A. Murari, R. Meléndez, G. Ratta, S. González, and JET EFDA contributors, Test-bed of a real time detection system for L/H and H/L transitions implemented with the ITMS platform, Fusion Engineering and Design 85 (3–4) (2010) 360–366.
[10] http://www.jini.org/files/meetings/eighth/presentations/Newmarch/upnp.pdf.
[11] http://www.psinaptic.com/link_files/jini_and_plugandplay.pdf.
[12] Z. Salvador, R. Jimeno, A. Lafuente, M. Larrea, J. Abascal, Architectures for ubiquitous environments, Wireless And Mobile Computing, Networking And Communications, in: IEEE International Conference, Vol. 4, 22–24 August 2005 (WiMob'2005), 2005, pp. 90–97.