JEFERSON LAB DATA ACQUISITION RUN CONTROL SYSTEM

V. Gyurjyan, C. Timmer, D. Abbott, G. Heyes, E. Jastrzembski, D. Lawrence, E. Wolin
TJNAF, Newport News, VA 23606, USA

Abstract

A general overview of the Jefferson Lab data acquisition run control system is presented. This run control system is designed to operate the configuration, control, and monitoring of all Jefferson Lab experiments. It controls data-taking activities by coordinating the operation of DAQ sub-systems, online software components and third-party software such as external slow control systems. The unique feature which sets this system apart from conventional systems is its incorporation of intelligent agent concepts. Intelligent agents are autonomous programs which interact with each other through certain protocols on a peer-to-peer level. In this case, the protocols and standards used come from the domain-independent Foundation for Intelligent Physical Agents (FIPA), and the implementation used is the Java Agent Development Framework (JADE). A lightweight, RDF (Resource Definition Framework) based language was developed to standardize the description of the run control system for configuration purposes. Fault tolerance and recovery issues are addressed. Key features of the system include: subsystem state management, configuration management, agent communication, multiple simultaneous run management and synchronization, and user interfaces. A user interface allowing web-wide monitoring was developed.

INTRODUCTION

In the recent years many industrial, proprietary control systems, with their specific hardware and software, were being used in high energy and nuclear physics experiments. Inevitably control and data acquisition systems for the future experiments will face problems in areas such as interoperability, scalability, and standard user interface.

The CODA (CEBAF Online Data Acquisition) system successfully satisfies the ever growing needs of the new physics programs at JLAB [1]. The run control component of the CODA was written almost seven years ago, utilizing at that time the revolutionary, self consistent object oriented programming language Eiffel. Unfortunately over the years Eiffel did not catch on the way Java did, partly because of being a commercial product and having a single source of the distribution. Current experiments at JLAB have new expectations for run control, namely integration of new control components or systems into the data acquisition (DAQ) system, organizing control feedback between slow control and DAQ components, etc. So, the old run control, being a non-distributed, graphical user interface (GUI) application, is very difficult to maintain and extend to meet the current requirements of JLAB experiments. This paper discusses relevant design and implementation aspects of the new run control component of the CODA data acquisition system.

SOFTWARE AGENTS

Software agent technology is a sub-field of the AI (Artificial Intelligence), and currently is a rapidly growing research field. An agent is a software entity capable of acting intelligently on behalf of a user, in order to accomplish a given task. A group of specialized agents cooperate and work together to solve problems that are beyond their individual capabilities. In an open and distributed, multi-agent environment, the need for standard mechanisms and specifications are vital for ensuring interoperability of the autonomous agents. The FIPA agent reference model was chosen in order to provide the normative framework within which agents can be deployed and operated [2]. The FIPA specification establishes a logical reference model for agent creation, registration, location, communication, migration and retirement.

JADE, a FIPA specification Java implementation was used to develop an agent platform for the control system [3]. Using this framework, high level mediator agents, specializing in agent platform management and system coordination were developed. These agents are responsible for creating agents on a platform, educating, (based on the knowledge, provided by the user), deploying, and recovering them in case of unsatisfactory behaviour [4]. The efforts of these agents ensure system reliability, robustness and fault tolerance.

Two different types of “stem cell” agents were also developed. These were available for specialization into either “supervisor” or “worker” agents.

RUN CONTROL DESIGN ARCHITECTURE

The run control component of the JLAB data acquisition system is designed to play the supervisory role in the overall data production environment of the experiment. The run control system is designed to reflect the structure of CODA itself, thus it is composed of separate elements (agents) that are likely to be implemented as separate processes running on different machines. This design was implemented by developing a

multi-agent control system, containing distributed control application entities that collaborate dynamically to satisfy control objectives of the DAQ system [4]. The architecture of the run control can be seen as a hierarchy of control entities called agents, each with responsibility for a well defined component or part of the DAQ system (Event Builder, Event Transfer, Event Recorder, Readout Controller, etc.).An agent encapsulates control/monitoring algorithms, as well as external interface details of the DAQ/control component. This provides significant advantages, namely clear separation of the control and application layers of the component, and seamless integration of legacy software components into the run control environment. The agent state is the simplified external view of the current working condition of the CODA component or slow control component of the data production system, under its responsibility. Each agent is capable of receiving control messages from other agents or the outside world. These messages can cause an agent to execute actions which potentially will change the visible state or monitor the state of the component. The agents are organized into a hierarchical tree structures that reflect the basic organization of the DAQ system itself. An agent in the control tree can have only one supervisor agent and can supervise many other agents. At the top of the tree is a single agent, which represents the overall state of the entire online system. A run control functional diagram of the simplest online system, containing one EB, one ER, a physics detector readout system, and detector high voltage control system is shown in Figure 1. Below the supervisor agent are a set of agents, one for

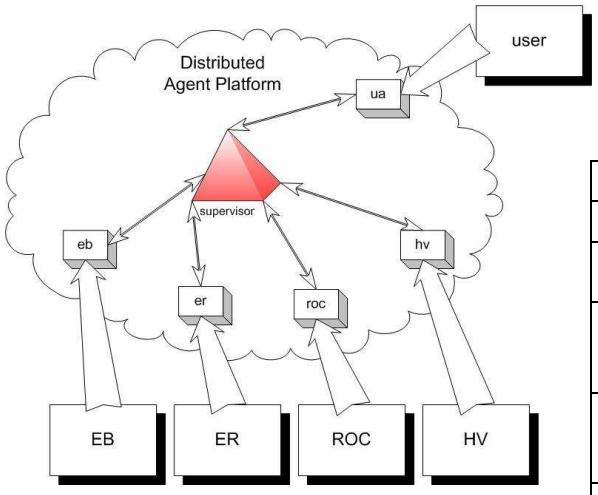

Figure 1: Example of the run control system.

each major subsystem of the online system, representing an actual DAQ component or physical detector. Obviously an agent of a sub-system itself can be a supervisor of the agents responsible for individual sub-components.

The distributed nature of the run control system allows for grouping of agents into specialized virtual clusters or domains. Even though the component agents in the agent platform interact with each other as peers, the run control system further implements a master/slave approach between the human operator and the DAQ system, where the operator is the master.

The agents in the hierarchal tree transmit messages between themselves to exchange control commands and status information. In the basic scenario a human operator sends control commands through his agent (ua, see fig. 1) to the supervisor agent, which forwards them to the sub-system agents, who in turn forward them to component agents and so on. The result of the control directives are sent back up the tree so that the human operator is made aware of any changes in the state of the system. Any agent in the hierarchy can either perform some actions on the commands or return results of the commands it receives.

RUN CONTROL STATE MACHINE

The run control system general state machine and commands were developed, having common semantics, mapped to the specific DAQ actions. Not all DAQ states or transitions are exposed to the operator. Most of the run control transitions are two or more step transitions, seen as a single state transition by the operator. For example, the configure transition is a multistep transition, during which a number of basic (stem cell) type agents are started on the agent platform [4].After each stem cell agent performs its initialization procedure (basically registration with the agent platform

Table 1: Run control state transitions

Operator		**System**
State	**Command**	**Invisible State**
Initial	Configure	Initial
		Differentiate
Configured	Download	Predownload
		Download
		Postdownload
Downloaded	Prestart	Preprestart
		Prestart
		Postprestart
Prestarted	Go	Prego
		Go
		Postgo
Active	End	Preend
		End
		Postend
Downloaded	Reset	Reset
Configured	Pause	Pause
Paused		

registration services) they will be differentiated (i.e. they will be specialized to be a representative controller for the assigned DAQ component). At the next step of the configure transition a supervisor agent is created on the platform with the full knowledge of the hierarchical control tree structure of the current DAQ system. The configure transition ends after every agent participating in the DAQ, is informed about the external processes and execution mechanisms (inner communication protocols, etc.), associated with the state transitions. When this state has been reached the run and configuration parameters can only be modified in a limited way. Table 1 provides a detailed overview of all supported run control transitions. A powerful feature of the agent framework is the mechanism of attaching software processes to the state transitions. This provides a framework that facilitates extension of the DAQ state machine by integrating user defined state machines (described as a process) in the DAQ real-time environment.

CONTROL PROCESS ABSTRACTION AND SYSTEM PARTITIONING

The DAQ and slow control components (hardware and software) can be assembled into various possible working DAQ systems. Hence, the run control system supports partitioning of the online system and is able to run with a variable set of components to control. The run control system can have multiple, independent DAQ/control systems running in parallel, with their hierarchical agent trees and supervisor agents. This implies that the run control system is capable of reading and parsing the general system configuration data and can create and configure multiple control agent trees. DAQ configuration descriptions are stored in COOL (Control Oriented Ontology Language) files. We developed this meta-language based on RDF (Resource Definition Framework) in order to achieve this control process abstraction [5].

Table 2: Overview of the COOL taxonomy

Subject/Object	Predicate
Control	hasOption, acceptsComponent , etc.
Component	hasProcess, hasNode, differentiate, etc
Process	hasCommand, hasData, isPartOf, etc.
Command	hasName, hasType,hasLoop,etc.
Data	hasSemantics, hasContent, hasType,..
Node	hasName, hasIP, isLocal,etc.
Loop	stopAt, loopRepeat, loopDelay, etc.
Option	hasDataFile, hasEventLimit, etc.

COOL defines a common vocabulary, by means of which the information (control command, control actions, configuration information, etc.) is shared among the agents in the run control system. It includes machine independent definitions of basic concepts in the control domain and relations among them. Table 2 provides an overview of the COOL taxonomy, describing the COOL resources and associated (not complete) set of the properties (predicates).

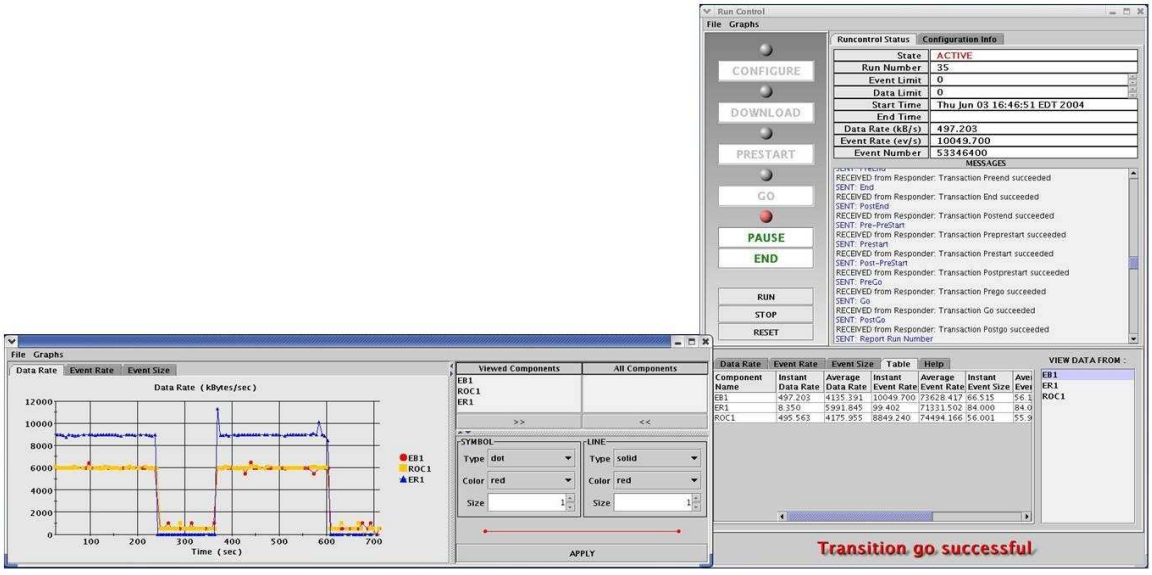

Figure 2: Run control GUI

GRAPHICAL USER INTERFACE

The graphical user interface (GUI) is intended to give a view of the status of the data acquisition system and its sub-systems (e.g. Event Transfer, Event filter, Event

builder, Back End, etc.) and to allow the user to control its operation. The GUI was developed not only for general users, such as shift operators, but also to provide DAQ experts the ability to control and debug the DAQ system. The run control system can have many GUIs associated with a particular experiment. However, only one GUI can be a master, capable of controlling the DAQ system. The rest of the GUIs will visualize the monitored information. In the run control environment the GUI is considered to be a software component and will have an associated agent in the platform to interact with the DAQ/control component agents. Figure 2 shows a snapshot of the GUI in action.

CONCLUSIONS

The new run control system for the JLAB data acquisition system has been developed using intelligent software agent technology. The run control system fault tolerance and reliability was addressed using DAQ run control platform agent recovery mechanisms. Run control process abstraction has been implemented through the control oriented ontology language. This allows description and integration of the run control specific processes, as well as control processes in general (slow control, etc.) into the general control environment. The current limitation of the run control system is that only processes having well defined external, software interfaces can be integrated into the run control system. Currently our group is developing an API and libraries for the component developers to integrate their components into the run control system without developing proprietary interfaces.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the prompt and responsive help provided by the Jade development team and community.

REFERENCES

[1] G. Heyes, et al. “The CEBAF on-line data acquisition system”, Proceedings of the CHEP conference (Apr. 1994), pp. 122-126.

[2] Foundation for Intelligent Physical Agents. Available at http://www.fipa.org

[3] Java Agent DEvelopmnet Framework. Available at http://sharon.cselt.it/projects/jade

[4] V. Gyurjyan, et al. “FIPA agent based network distributed control system”, Proceeding of the CHEP conference (March 2003), arXiv:hep-ex/0305016

[5] K.Ahmed, et al. “Professional XML Meta Data”. Wrox Pres Ltd.