AFECS. Multi-Agent Framework for Experiment Control Systems

V Gyurjyan, D Abbott, G Heyes, E Jastrzembski, C Timmer and E Wolin

Jefferson Lab, 12000 Jefferson Ave. MS-12B3, Newport News, VA 23606, USA
Email: gurjyan@jlab.org

Abstract. AFECS is a pure Java based software framework for designing and implementing distributed control systems. AFECS creates a control system environment as a collection of software agents behaving as finite state machines. These agents can represent real entities, such as hardware devices, software tasks, or control subsystems. A special control oriented ontology language (COOL), based on RDFS (Resource Definition Framework Schema) is provided for control system description as well as for agent communication. AFECS agents can be distributed over a variety of platforms. Agents communicate with their associated physical components using range of communication protocols, including tcl-DP, cMsg (publish-subscribe communication system developed at Jefferson Lab), SNMP (simple network management protocol), EPICS channel access protocol and JDBC.

1. Introduction

The CEBAF Online Data Acquisition (CODA) system continues to satisfy the ever growing requirements of the physics programs at JLAB [1], and demonstrate very good performance and reliability. On the other hand, the control component of CODA has not evolved much over the years partly due to the fact that run control was dependably performing its simple task of controlling the data acquisition (DAQ) state machine. However, future experiments at JLAB have new expectations from run control, namely:

1. integration of new components into the DAQ system
2. creating feedbacks between slow controls and DAQ components
3. building expert systems, with their specific control state machines
4. designing online data quality monitoring, coordinated with data production processes
5. putting together online data calibration and distribution systems
6. alarm systems, etc.

To satisfy these challenging requirements, the Jefferson Lab DAQ group has developed a framework for building complex, hierarchical control systems. The unique feature which sets this framework apart from conventional control systems is its incorporation of intelligent agent concepts. An agent is a software entity capable of acting intelligently on behalf of a component or 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.

AFECS provides a group of specialized agents for agent management and coordination. These manager agents are responsible for creating and deploying agents on a platform, educating them, (based on knowledge provided by the user), distributing them over the network, and recovering them in case of unsatisfactory behaviour. The efforts of these specialized agents ensure control system reliability and robustness.

Different types of “stem cell” agents are also provided by the framework. They can be specialized by the control system designer to become representative agents for the various real world components.

2. Design architecture

A physics experiment can be thought of a well-defined set of components, each with their specific behaviours. These behaviours can be represented by simple, finite state machines. Control of the experiment is equivalent to controlling and synchronizing the state changes of those finite state machines, and AFECS provides the facilities to do this.

Control systems designed using AFECS are composed of agents that are groups of threads in a single process or separate processes running on different machines. The architecture of the AFECS based control system can be seen as a hierarchy of agents, each with responsibility for a component of the experiment. An agent encapsulates control/monitoring algorithms, as well as external interface details of the controlled, real-world component. This provides clear separation of the control and application layers of each component, and seamless integration of legacy software components into the experiment control environment (platform). An agent’s state is a simplified external view of the current working condition of the component 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 change or monitor the state of the physical component. Agents are organized into a hierarchical tree structures that reflect the basic organization of the experiment control system itself. An agent in the control tree can have only one supervisor agent and can supervise many others. At the top of the tree is a single agent (grand supervisor), which represents the overall state of the entire experiment.

The software architecture of the system is based on the coexistence of multiple JVMs (Java Virtual Machines) communicating with each other through Java RMI (Remote Method Invocation). Software agents are grouped into virtual clusters or domains according to their specialized functionalities. Agents in the same domain may share a single JVM, which plays the role of a basic agent container. An agent container provides a complete run time environment for agent execution and allows them to concurrently run on the same host. Agents in the same domain may also be distributed over multiple containers. They are active objects, having more than one behaviour, and can engage in multiple, simultaneous activities. Agent behaviours can be added or removed at run time. A control system diagram is shown in Figure 1.

The Front-End is a special container running the normative agents, taking care of agents’ management and overall coordination of the system. It also maintains an RMI registry internally, used by other agent containers to register themselves with the Front-End and join the platform.

The platform administrator agent (NA, normative administrative agent) is responsible for the agent management and recovery. This agent is in charge of creation, recovery and removal of agents, agent clusters or entire containers. It will also repair any failed agents or containers, thus achieving platform stability and fault tolerance.

The configuration agent (NC, normative configuration agent) is the interface between a specific control system implementation and the framework. It enables the integration of non-agent software controls into an agent system. This normative agent will coordinate the creation and deployment of each representative agent (A), as well as provide them with knowledge in the form of an ontology. The ontology consists of instantiated classes generated from physical component state machine descriptions, written using the control oriented ontology language (COOL). Agents in a platform can then communicate with each other by exchanging ontology objects, and have them invoke actions on the underlying hardware or software component.

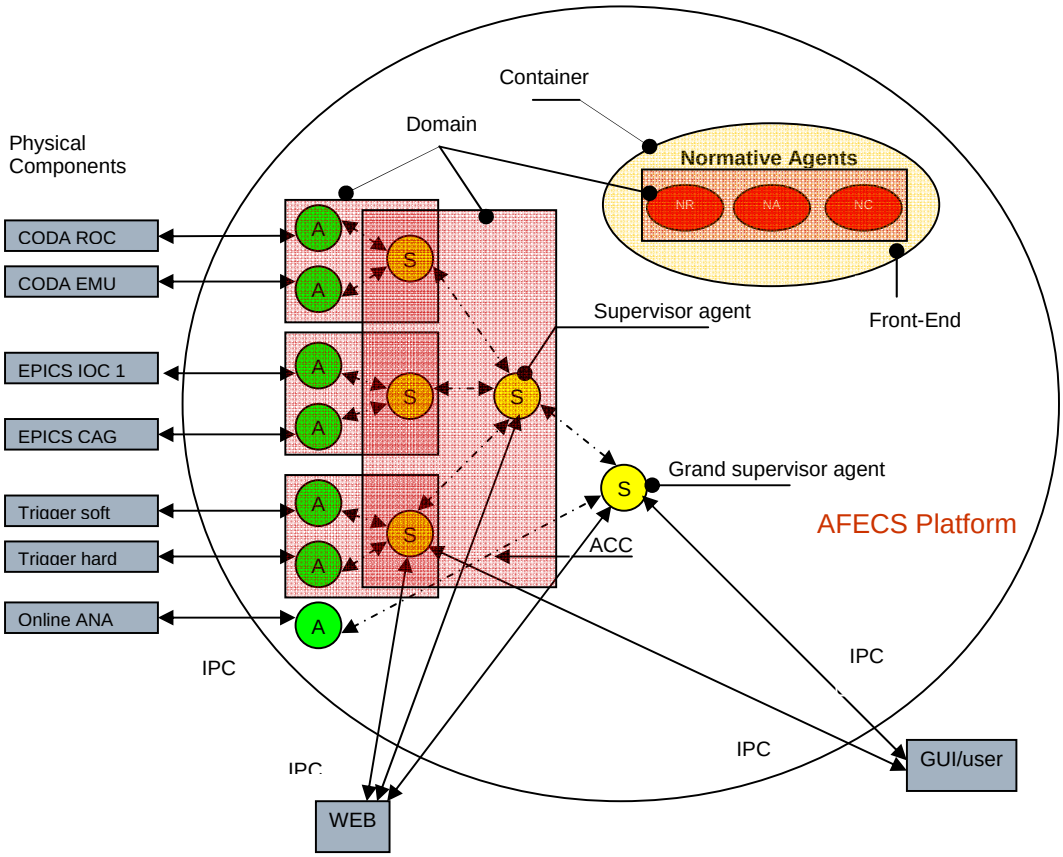

Figure 1. IPC – Inter Process Communication protocols (cMsg, DP, SNMP, JDBC, shell), used between agent and physical component. ACC – Agent Communication Channel used for intra agent communications. NR, NA, NC – normative agents for platform registration, administration and configuration respectively.

Coordination of groups of agents is accomplished by Supervisor agents (S). They ensure that partial, local solutions to the control problems of the specific domain are integrated into global experiment control. Agents in the hierarchal tree transmit messages between themselves containing commands and status information. Normally a human operator sends commands to the "grand supervisor" agent, which forwards them to supervisor agents down the tree, who in turn forward them to component agents and so on. The results of the commands 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 actions on a command or return results from a command it receives. This framework helps also to cope with the dynamic reconfiguration of the control environment in real time by auto-generating or specializing specific control agents whenever new control components are added, or control relationships are changed.

3. Inter-agent communication channel (ACC)

In a distributed, multi-agent environment, the need for standards and specifications are vital for ensuring interoperability of autonomous agents. The FIPA (Foundation for Intelligent Physical Agents) 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 standards for agent creation, registration, location, communication, migration and retirement.

All agent communications are performed through peer-to-peer message transfer. Message representation is based on the Agent Communication Language (ACL) formulated by FIPA. ACL is a language with well-defined syntax, semantics and pragmatics, based on speech act theory, containing two distinct parts:

- communicative act
- content of the message

The communicative act has a precise meaning, independent of the content of a message, and will extend any intrinsic meaning that the message content itself may have. The essential view of intra-agent messages in FIPA is that they represent a communicative act (an action that an agent can perform). For each communicative act, the FIPA standard specifies the conditions which need to be true before an agent can send the message and receive the expected effect of the action. Given the autonomous nature of agents, the receiver might simply decide to ignore the message. For this reason, instead of sending a single message, all AFECS agents initiate an interaction protocol that allows verification of the expected effect on the receiver side. Figure 2 shows the structure of the AFECS agent communicative act.

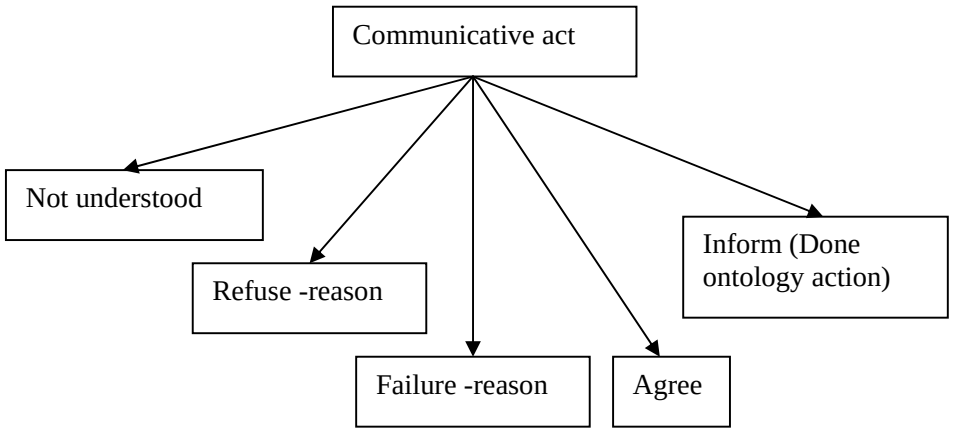

Figure 2. Agent communicative act

FIPA specifies a set of standard interaction protocols that can be used as templates to build agent conversations. A basic ACL message has the following structure:

- Sender
- Receiver
- Content (Data Object or Ontology Object)
- Language
- Ontology
- Protocol

4. Control Oriented Ontology Language (COOL)

COOL is an attempt to formulate a conceptual schema about physics experiment control in general. It is a hierarchical data structure containing all the relevant entities, their relationships, and rules specific for the control.

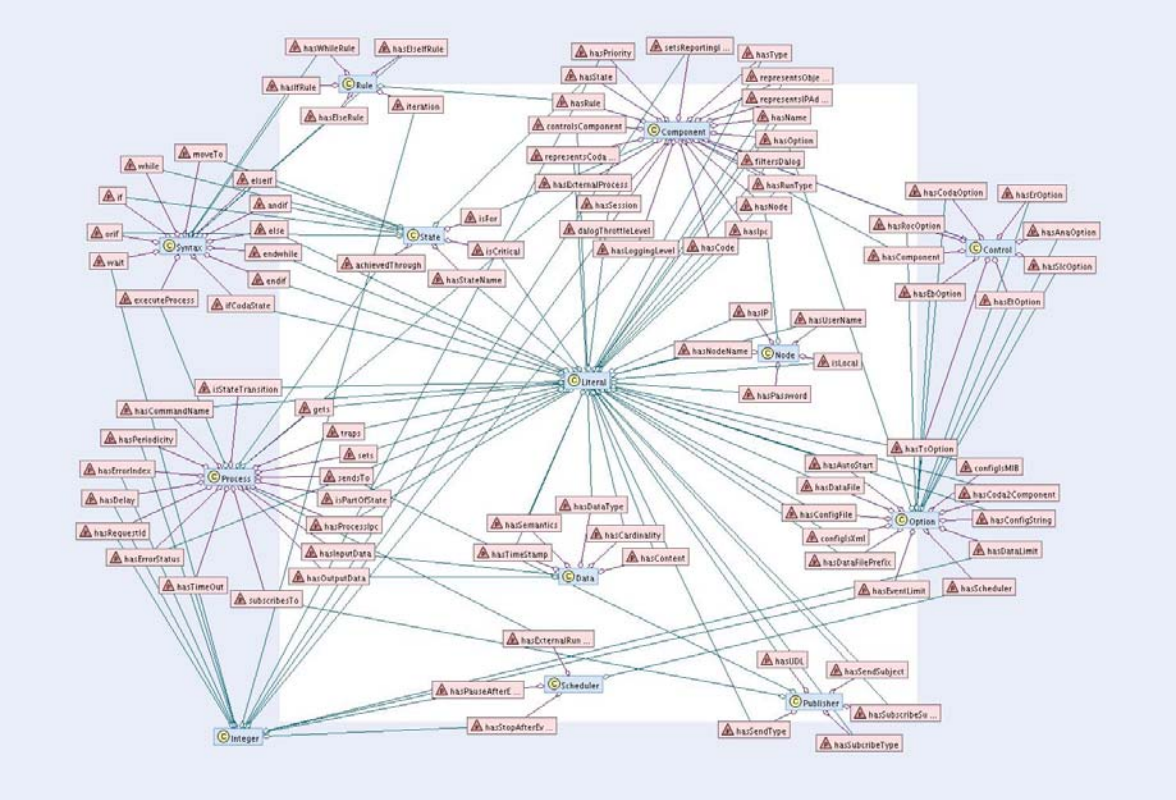

Figure 3. RDF node and arc diagram of COOL. Resources are depicted as C nodes, with surrounding P nodes as their properties.

COOL defines a common vocabulary, by means of which commands, actions, configuration information, etc. are shared among agents in the experiment control system. It includes machine-independent definitions of basic concepts and relations among them. An agent's communicative acts and their corresponding responses are described using the COOL taxonomy.

AFECS supports partitioning of the online system and is able to run with a variable set of components to control. A control system can have multiple, independent components or subsystem controls (with their hierarchical agent trees and supervisor agents) running in parallel. This implies that an AFECS based control system described in COOL is capable of creating and configuring multiple, cross-correlated hierarchical control agent trees. COOL allows detailed specification of components, their agents, their states, actions, and associated conditions. COOL was developed using RDF (Resource Definition Framework) [3]. Figure 3 shows a node and arc diagram of the COOL language, describing the language resources (subject/objects) and associated properties (predicates).

The AFECS framework provides several tools to create configuration files, written in COOL. Figure 4 is a snapshot of the graphical user interface for generating COOL configuration files that describe an experiment control system.

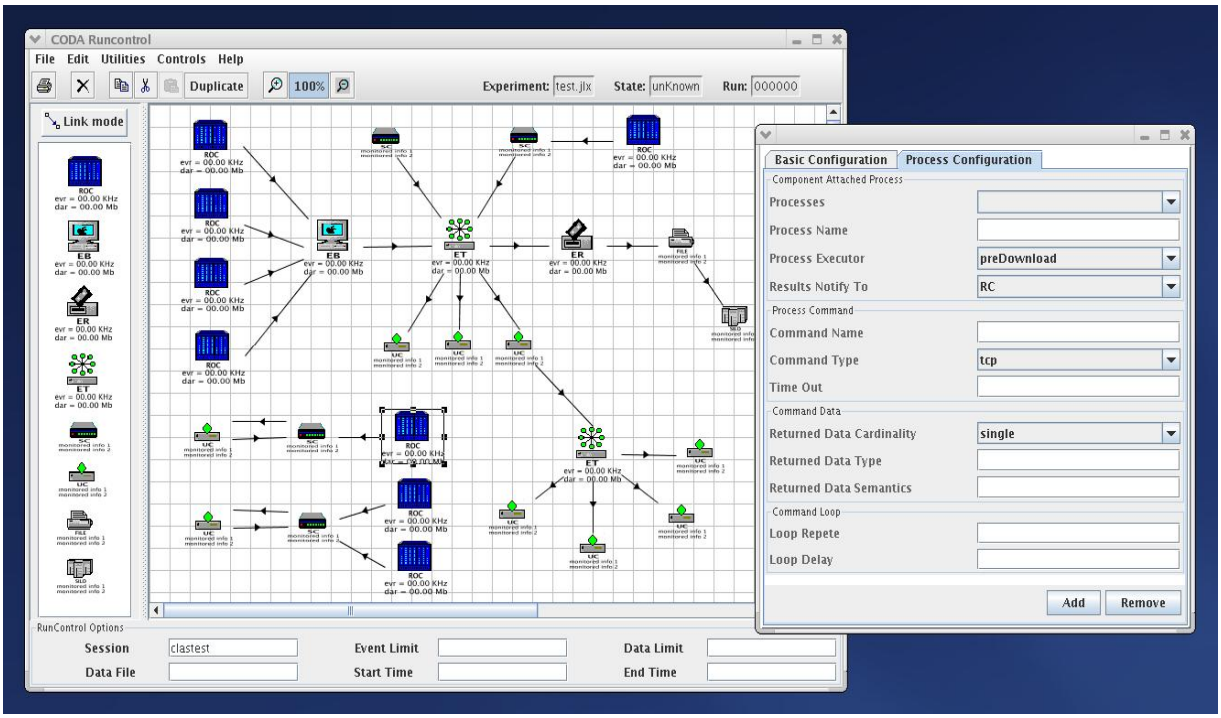

Figure 4. Experiment configuration builder GUI.

5. Implementations

The new run control system for the JLAB data acquisition system (CODA) has been developed using AFECS [4]. This run control system is designed to configure, control, and monitor Jefferson Lab experiments. It controls data-taking activities by coordinating the operation of DAQ components (readout controller, event builder, event recorder, event transfer, etc.). The graphical user interface (GUI) for the run control is intended to give a view of the status of the data acquisition system and its components 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 system. The run control system can have multiple instances of GUIs associated with a particular experiment. However, only one GUI can be a master, capable of controlling the DAQ system. Figure 5 shows a snapshot of the GUI in action.

AFECS provides support for bidirectional invocations of web services through java servlets for communicating with agents. The servlet package of the framework is an external software package much like the CODA run control GUI that interacts with the platform agents through the cMsg publish subscribe communication protocol [5]. This design approach insures clear separation between control and visualisation. The web site http://clasweb.jlab.org/clasonline/rc/hallB /e-cr.htm, dedicated to the development of the CLAS12 Experiment Control System, is an example of the AFECS monitoring and visualisation implementation.

Figure 5. CODA run control graphical user interface.

6. Summary and Conclusions

A Java based software framework for designing and implementing hierarchical, distributed control systems has been developed using intelligent agent technologies. The framework encourages abstraction, encapsulation, and overall system modularity.

The AFECS framework provides a special ontology language to describe an experiment hierarchical control structure, as well as control logic and finite state machines.

This framework has been successfully used to develop a new run control system for the JLAB data acquisition toolkit, and a CLAS experiment web-based monitoring system.

References

[1] Heyes G, et al 1994 The cebaf on-line data acquisition system *Proceedings of the CHEP 1994 Conference*

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

[3] K.Ahmed, et al. "Professional XML Meta Data". Wrox Pres Ltd.

[4] V. Gyurjyan, et al. "Jefferson Lab Data Acquisition Run Control System", *Proceedings of the CHEP conference. CERN-2005-002, Volume 1, page 151.*

[5] E. Wolin, et al. "cMsg – Publish/Subscribe Package for Real-Time and Online Control Systems", *Proceedings of the 14th IEEE-NPSS Real Time Conference, Stockholm, Sweden 2005.*