ONTOLOGY LANGUAGE TO SUPPORT DESCRIPTION OF EXPERIMENT CONTROL SYSTEM SEMANTICS, COLLABORATIVE KNOWLEDGE-BASE DESIGN AND ONTOLOGY REUSE*

Vardan Gyurjyan, D. Abbott, G. Heyes, E. Jastrzembski, B. Moffit, C. Timmer, E. Wolin,
Jefferson Lab, 12000 Jefferson Ave. MS-12B3, Newport News, VA 23606, USA

Abstract

The ever growing heterogeneity of physics experiment control systems presents a real challenge to uniformly describe control system components and their operational details. Control Oriented Ontology Language (COOL) is a meta-data modelling language that provides generic means to represents physics experiment control processes and components, their relationships, rules and axioms. It provides a semantic reference frame that is useful for automating the communication of information for configuration, deployment and operation of an experiment. Additionally, COOL provides precise specification of software and hardware components. In this paper we discuss the control domain specific ontology that is built on top of the domain-neutral Resource Definition Framework (RDF). Specifically, we will discuss the relevant set of ontology concepts along with the relationships among them in order to describe experiment control components and generic event-based state machines. COOL has been successfully used to develop a complete and dynamic knowledge base for experiment control systems, developed using the AFECS[1][2] framework.

INTRODUCTION

Ontology is a term defined and used primarily in philosophy, namely in metaphysics. It is the study of reality and existence as well as of the basic categories of being, existing entities and their relations. Due to the abstract nature of the definition, the term 'ontology' has been widely used in many other scientific disciplines, including computer science, information science, library science, biomedical engineering, and software engineering. The use of the term ontology in our case is rather narrow and constitutes the working model of entities and interactions between them in the very particular domain of experimental physics. COOL, as any other ontology language, explicitly identifies a vocabulary of terms which includes definitions of specific concepts and an indication of how these concepts are inter-related.

EXPERIMENT CONTROL ENVIRONMENT

The goal of this project is to create a vocabulary describing complex, hierarchical control systems in general, and apply it to high energy and nuclear physics experiment control systems. Our intention is to capture concepts in a formal language capable of describing not only static data, but also dynamic data and control actions. A set of real world physical components, such as particle detectors, detector support devices (e.g. power supply, gas, cooling and safety systems), data readout devices, software systems, etc. comprise a real world physics experiment. These components and their effects upon each other are characterized by observables, called control channels. The collective values of these observables define the state of a physical component. Control specific processes can be applied to physical components, changing their states.

Physics data processing is a complex, multistage process, and quality data production is only possible if tools to configure, control and monitor the entire experiment control system are developed and deployed.

Components of the system have their specific behaviors. These behaviors can be represented by finite state machines. State machines at their simplest, are models of the behaviors of system components. These components have a limited number of defined states, and they transition between states according to specified rules. These rules are policies designed to achieve the control objectives of the experiment.

CONCEPTUALIZATION

Ontology language design starts by identifying the key concepts that exist in our domain of interest, their properties and relationships that hold between them. We began designing COOL by first identifying natural language terms that refer to concepts, relations and attributes, used in physics experiment control, and then structuring these terms into conceptual models. In essence, COOL is using these concepts and their properties as control system modeling primitives. Properties are applicable only to the concepts they are defined for, and value restrictions can be defined for each property. A concept provides a context for modeling one aspect of a control component (e.g. HV mainframe). An important part of the COOL language is the possibility of inheritance between concepts. The knowledge base of an experiment then consists of instances (objects) of these concepts.

In COOL, concepts interact with each other through described relations, which fall into two categories: taxonomic and associative. Taxonomic relations are used to describe primitive relationships between different types

* Notice: Authored by Jefferson Science Associates, LLC under U.S. DOE Contract No. DE-AC05-06OR23177. The U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce this manuscript for U.S. Government purposes.

of concepts (for example *Component – providesService or Service - startsStateMachine*), and for the further description of the concept (for example *State – achievedThrough–Process,*etc). Associative relations instead relate concepts across the COOL language structure, and used to specify attributes of the concepts (for example *Channel-hasName, Component-hasReportingInterval*).

COOL defines fifteen basic concepts shown in Figure 1. COOL document can be seen as a set of declarative and action statements. Declarative and action statements can be created using both categories of COOL relations. Declarative statements that are used to create instances of the COOL concepts are utilizing mostly associative relations of the language. For example the snippet of the COOL description, coded in XML, creates an instance of the *Component* concept, named EB1 (the COOL associative relations are shown in bold-italic).

```
<rdf:Description rdf:about="http://COOLHOME/Ebs/eb1#EB1">
  <cool:hasIpc>dpsh</cool:hasIpc>
  <cool:representsCoda2Client>true</cool:representsCoda2Client>
  <cool:hasType>EB</cool:hasType>
  <cool:hasName>EB1</cool:hasName>
  <cool:hasCode>{CODA}{CODA}</cool:hasCode>
  <cool:hasPriority>33</cool:hasPriority>
  <cool:hasReportingInterval>3</cool:setsReportingInterval>
<cool:hasStaterdf:resource="http://COOLHOME/State/TState#TState1"/>
</rdf:Description>
```

COOL MODEL AND SYNTAX

COOL is designed to describe control specific data and processes that are used to handle the data, retrieve information about the data, or share the data or its description with other control systems. The use of common meta-data for the control components and sub-systems of the experiment allows straightforward integration of various control systems in a unified, hierarchical environment.

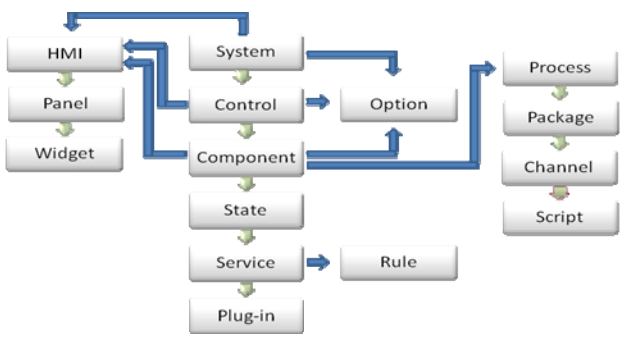

Figure 1: COOL basic concepts.

XML[3] is a widely accepted general-purpose specification for creating meta-data elements, and is the most popular text-based mark-up language. However XML, with its well known flexibility, remains just a data format. XML places no semantic restriction on element or attribute names, meaning that a person or software reading the XML document must have prior knowledge to be able to comprehend it. The newly emerged specification by W3C known as the Resource Definition Framework (RDF)[3][4], is more suited for meta-data modeling. The most appealing aspect of RDF is its ability to organize, interrelate, classify, and annotate knowledge. RDF is capable of conveying semantics (i.e. formally describing meaning in a simple text format). One can understand the meaning of an RDF document without having prior knowledge about it. Statements in RDF have a triplet structure: *subject-predicate-object*. This structure mimics the basic sentence structure (*subject-verb- object*) of declarative and explanatory sentences of the English language. This makes RDF documents extremely powerful and easy to read and understand by humans and machines. COOL uses RDF specification as a metadata model.

The subject of a COOL statement is a defined control system concept (or instance of a concept class). The predicate is a concept as well (COOL action-concept), representing a relationship. The object is a concept or a primitive type (RDF literal, integer, etc.). Here are a few triplet structure statements, describing a hypothetical control system, written using COOL taxonomy.

In these statements, the concept classes or the terminal nodes (RDF primitive types) are shown in bold letters. Language concept instances are shown in bold italic, and predicates of the statements are shown in italic letters.

Control	*hasComponent*	***HVMainframe***.
HVMainframe	*hasState*	***HVState1***.
HVState1	*achievedThrough*	***HVProcess1***.
HVProcess1	*hasCommandName*	**Shell-executable-string**.
HVMainframe	*hasState*	***HVState2***.
Control	*hasService*	***Service1***.

RDF models are often represented graphically as *node and arc* diagrams, where nodes are RDF subjects or objects, and relationships or properties are arcs/arrows.

GRAPHICAL INTERFACE

Even though the COOL language learning curve is minimal, and an RDF based COOL document is easily readable and understandable, a graphical user interface (GUI) has been developed to hide language details from the experiment control system designer. In addition, a set of tools has been developed to translate existing experiment control knowledge at Jefferson Lab into the COOL language.

In this program, instances of *Component* concepts are created by simply dragging and dropping component type specific icons into the design board. For each of these components COOL statements are automatically created and reflected in the GUI. State machine statements (COOL action statements) are generated based on user input and are described in the next section. All of these inputs are compiled into a COOL document. This program can be used to edit and modify existing experiment control system design documents as well.

STATE MACHINE DESCRIPTION LANGUAGE

The COOL *Rule* concept has the *hasCode* property that accepts state machine algorithm description code as its

value. This terminal node of the COOL statement uses C++/Java like syntax and some SML[5] language constructs to describe finite state machines of the experiment control system. The following table 1 shows all the defined COOL state machine language keywords.

Table 1: COOL Keywords

if	**elseif**	**else**	**do**	**while**
in_state	**not_in_state**	**move_to**	**true**	**false**
group	**supervisor**	**All**	**sleep**	//

The COOL state machine language also uses *&&* and || to denote logical-AND and logical-OR operators. Using the provided text editor as a part of the GUI, a state machine can be generated using COOL concept instances, COOL state machine language keywords, and operators. The semicolon is used as a statement terminator. The COOL ontology language, as well as COOL states machine description language is case sensitive. Below is an example of a COOL *Rule* description.

```
If  ( ( EB1 in_state EBState1 ) &&
 ( HVMainFrame not_in_state HVState1 ) ) {
     EB1 move_to EBState2;
     do externalProcess1;
}  elseif  ( HVMainFrame in_state HVState1 ){
    move_to HVState2;
}
```

A statement or a group of statements are combined together using curly brackets, associating an action statement block with a conditional statement. White space is ignored allowing the user to spread COOL state machine language statements across any number of lines, or to group a number of statements together on a single line (as long as they are inside of curly brackets). In order to simplify the process of state machine coding, all the COOL *Component* and *State* instances in a specific control system will be shown in the GUI.

SUMMARY AND CONCLUSIONS

The control oriented ontology language has been developed to describe hierarchical control system structures, as well as to describe the control logic and finite state machines. By using RDF instead of XML as a basis for the language, we hope to increase the descriptive power of the language, and achieve more complete depiction of the experiment control system. Moreover, COOL helps to make individual development tools more collaborative and to design a unified experiment control environment that enables common understanding of the information from different tools. This approach guaranties loose coupling between system components, and will help the experiment control system designer to easily build a hierarchical system when using heterogeneous system components. A graphical user interface has been developed to simplify experiment control knowledge design. COOL is currently used to design and deploy a knowledge base for AFECS-based experiment control systems.

REFERENCES

[1] V. Gyurjyan, et al., “Jefferson Lab Data Acquisition Run Control System”, Proceeding of the CHEP Conference, CERN-2005-002, Volume 1, page 151.

[2] V. Gyurjyan, et al., “AFECS. Multi-agent Framework for Experiment Control Systems”, J. Phys. Conf. Ser. Volume 119 (2008) 022025.

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

[4] Shekky Powers, “Practical RDF”, O’Reilly & Accociates, Inc.

[5] B. Franek, C. Gaspar, “SMI++ Object Oriented Framework for Designing and Implementing Distributed Control Systems”, Nuclear Science Symposium Conference Record, 2004 IEEE Volume 3 Issue, 16-22 Oct. 2004 Page(s):1831 - 1835 Vol. 3.