PROTEUS: FRIB CONFIGURATION DATABASE*

V. Vuppala, E. Berryman, S. Peng, NSCL-FRIB, USA
L. Dalesio, BNL, USA

Abstract

There is need for an integrated information system that manages the data and computational-logic used by an experimental physics facility (EPF) during its design, construction, commissioning, and operation. Such a system can be used to manage design lattices, model them, run what-if scenarios, tune the beams, troubleshoot, manage calibration data, maintenance records, alignment information and quality metrics, and generate reports for funding or regulatory agencies. A critical component of such a system is the configuration database. It manages devices, their layout, measurements, alignment, calibration, signals, and inventory. In this paper we describe development of such a component. We describe its architecture, database schema, services, and graphical and programming interfaces.

INTRODUCTION

An integrated information system is critical for the design, commissioning, operation, and maintenance of an EPF. Distributed Information Services for Control Systems (DISCS) [1][2] is a framework and implementation of such a system. It is comprised of a set of cooperating services and applications, and manages data such as machine configuration, lattice, measurements, alignment, cables, machine state, inventory, operations, calibration, and design parameters. It also includes computational services such as Online Model and Unit Conversion. DISCS is a collaborative effort of BNL, Cosylab, ESS, FRIB, and IHEP.

To enable development by multiple, dispersed, and independent teams DISCS has been divided into several domains [3]. Each domain is responsible for a portion of the system, and provides tools and services to manage the associated data and logic. One of DISCS' core service is the Configuration Domain. It is concerned with the configuration of the accelerator facility: the components, their properties, design parameters, measurements, calibration, maintenance, layout, and relationships among components. Proteus is an implementation of the Configuration Domain. It is being developed and used at the Facility for Rare Isotope Beam (FRIB).

Architecture

The basic architecture for DISCS is shown in Figure 1. It consists of three layers: Data, Service, and Application. Data Layer represents all the data sources: managed, unmanaged, structured, and unstructured. Service Layer is composed of services. A service is a reusable software component that implements a set of business functions, has a formal and documented interface, and can be located and accessed through standards-based communication mechanisms. In our case, a service can be thought of as a software process that implements controls or physics related logic, and provides high-level data structures to the user through REST-based [4] and PVaccess [5] protocols. Application Layer consists of the software tools or components that present the information to the user.

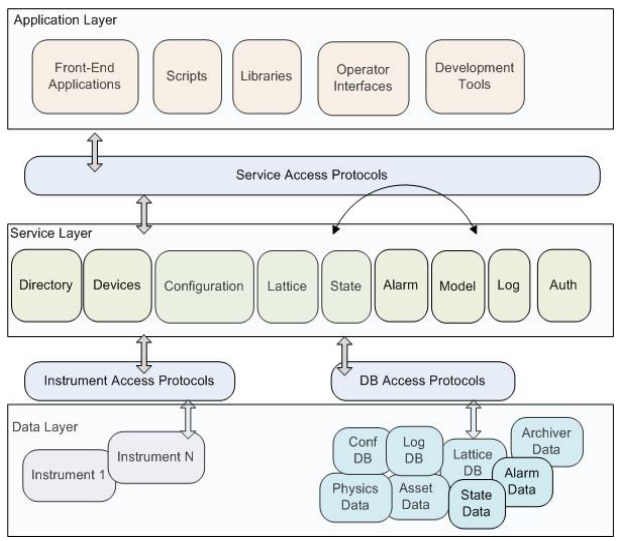

Figure 1: Applications, services, and data.

Proteus, just like all of DISCS' modules, is composed of a database, one or more services, applications to manage and load data, and an Application Programming Interface (API).

CONCEPTUAL MODEL

In this and the following two sections we describe Proteus' data model. A *component* is any entity the accelerator facility's configuration: magnet, power supply, cavity, rack, room, controller etc. Components can be looked at in different ways, and have different kinds of information associated with them: design data, measurements, test data, alignment information, physical characteristics etc. For our modelling concerns, we define two kinds of components:

1. *Physical-Component*: This represents physical entities; things that exist in the real world. A physical-component has identifiers that can identify it, and has attributes that can be measured and calibrated. For example, the cavity with part number T30802-MDE-0008 that was manufactured

* This work was supported in part by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661, the State of Michigan and Michigan State University

ISBN 978-3-95450-139-7

by Jefferson Lab is a physical-component. A physical-component has attributes such as calibration records, measurements, traveler data, manufacturer model etc

2. *Logical-Component*: It represents the entities that exist on the blueprint or configuration (layout) of the Accelerator facility. A small part of FRIB's configuration is shown in Figure 2; the various elements in the figure are Logical-Components. For example, devices LS1_CA01:CAV1_D1093 and LS1_CA01:CAV2_D1101 are Logical-Components.

Components, both physical and logical, are classified into types. A *Component Type* represents a generic component and its design. For example, there can be several horizontal electrostatic dipoles with the same characteristics i.e. they are interchangeable; in such case the common characteristics are denoted by a Component Type, 'Horizontal Electrostatic Dipole Type I'. A Component Type is a conceptual entity; Physical Components are its manifestation in the real world, and Logical Components are the manifestations on the EPF layout. Component types have a hierarchy. For example, QM1 is a quadrupole which is a magnet.

These concepts are illustrated in Figure 2. Cavity-A is a Component-Type; it has attributes such as design parameters, CAD drawings etc. The cavities LS1_CA01:CAV1_D1093 and LS1_CA01:CAV2_D1101 are Logical-Components on FRIB's LINAC Segment #1 at positions 1093 and 1101 respectively. They have attributes such as distance from the source, optical properties etc. However, they share the design parameters and CAD drawings of the Component Type, Cavity-A. At the bottom of the figure, the cavity with serial number T30802-MDE-0008 is a Physical-Component. It also shares its design attributes with the Component Type Cavity-A. It can be installed at either position 1093 or 1101 of the LINAC segment #1. Let us say it is installed at position 1101; it now represents the Logical-Component LS1_CA01:CAV2_D1101. But at a later date, it can be moved and installed at position 1093 to represent the Logical-Component LS1_CA01:CAV1_D1093.

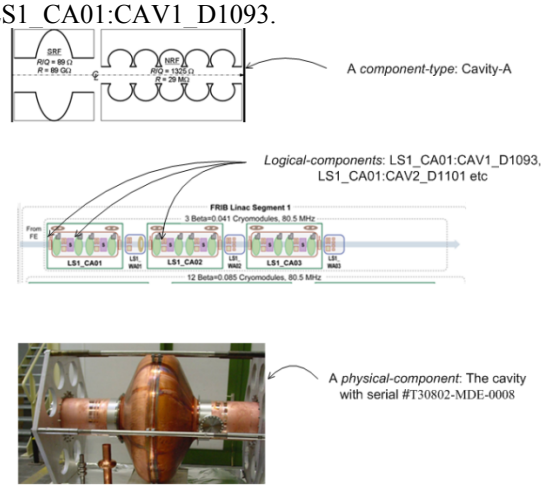

Figure 2: Logical and physical components.

Physical-Components form the inventory. Physical-components that are not installed (i.e. not linked to a Logical-Component) are the spares. A physical component is a real-world entity, a component type is its design, a logical component is its configuration entity (and an element is its simulation).

LOGICAL MODEL

There are two ways to model components:

1. A class or entity type per Component-Type. There will be several entity types: Magnet, Cavity, Power Supply, Segment etc. Each entity type will be mapped to its physical representation (tables in Relational Model).
2. An object (or entity instance) per Component-Type. There will be three entity types: one each for Logical-Components, Physical-Components, and Component-Types.

The problem with the first approach is that it will result in too many entity types and they keep on growing, which will require changes to the schema. The second approach solves this problem by having a single entity type for all Logical (Physical) components. But how to represent the different attributes of the various real-world component with one entity type? It is inefficient and sometimes impossible to associate all the attributes with one entity type. A solution to this is to have the attributes as Properties (Key-Value pairs). Based on the component type, an entity instance will have different properties associated with it. With this approach, associating entities (JOIN operation in Relational Model) based on attribute-values will be inefficient, and may lead to performance degradation. After analyzing the pros and cons of the two approaches, we have chosen the second method.

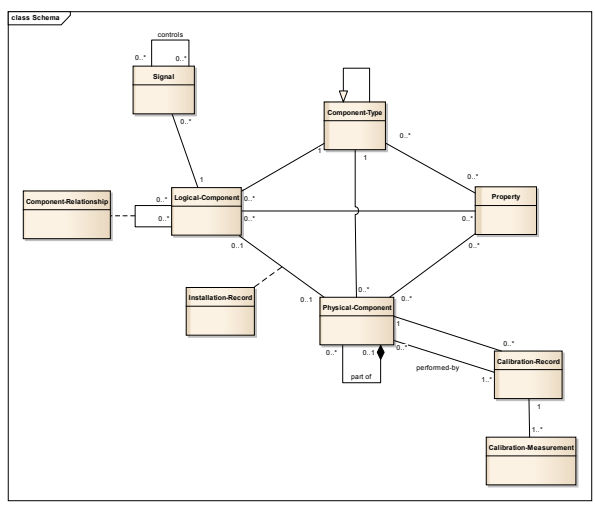

Figure 3: Logical model (partial).

Figure 3 shows the logical model of the database. A Logical-Component is based on exactly one Component-Type. Similarly, a Physical-Component is also based on exactly one Component-Type. However, there can be

ISBN 978-3-95450-139-7

many Logical-Components and Physical-Components that are implemented based on a single Component-Type. A Physical-Component is associated (during installation) at the most with one Logical-Component; it is not associated with any when it is a spare. Similarly, a Logical-Component is related to at the most one Physical-Component; it may not be related to any. When a Physical-Component is installed and gets linked to a Logical-Component, both the Physical-Component and Logical-Component must be based on the same Component-Type.

Logical-Component, Physical-Component, and Component-Type have Properties. Logical-Component has zero or more Signals.

The concept of Logical-Components can be extended to represent other entities in a configuration that may not be on the facility's blueprint. Examples of such Logical-Components are:

- Group of Physical-Components such as a segments or beamlines
- High-level virtual physics devices that are implemented in software

Such Logical-Components that do not have a physical manifestation (i.e. they are not linked to any Physical-Component) are marked as 'Abstract'

Attributes and Properties

Attributes of the Logical-Components, Physical-Components, and Component-Types keep changing. This is more so during development and commissioning of the machine, but continues into operations. Also, different accelerator facilitates would have their own attributes. Changing the database schema often results in a large effort to modify the related services and/or applications. So only the core attributes are kept in the schema, and the rest are stored as properties (key-value pairs).

Component Relationships

Two Logical-Components can be associated with each other through various relationships. Relationships are also generalized, the same way as Logical-Components are. We do not identify or represent specific relationships in the logical model. They are represented by Component-Relationship entity. The relationships are assumed to be binary.

A Physical-Component may be composed of other Physical-Components. This relationship is represented by a one-to-many association (UML Composition).

Note that the assembly relationship among components (say a cryomodule and its cavities and solenoids) is duplicated at both as-designed (among Logical-Components) and as-built (among Physical-Components) layers. During installation (uninstallation), the Logical and Physical Components have to be associated (disassociated) accordingly

PHYSICAL MODEL

Relational Model has been chosen for realization of Proteus' database. Proteus' database objects (tables, relationships, views etc) are described in [6].

Data Types

As explained in the previous section, attributes are represented by key-value pairs. A side-effect of this choice is that attributes cannot be associated with specific DBMS data types. Because attributes are represented with key-values pairs, all the values have to be of the same type. It is possible to have a fixed number of data types associated with each property but it is not efficient or workable. So it was decided to have attribute values as generic type that can accommodate large values. MySQL provides two such data types: BLOB and TEXT (with their 'larger' versions). Currently, TEXT is used but it may be changed in the future

Component Relationships

The relationships among Logical-Components are implemented through two tables: component_relation and component_pair. The 'component_relation' table stores information about the relationships: their IDs, names etc. The 'component_pair' table links two Logical-Components with a relationship. It has three (foreign-key) fields; two containing Logical-Component primary keys, and one containing relationship primary key (from component_relation).

The hierarchical composition relationship among Physical-Components is implemented by a field in the Physical-Component that points to the primary key of the parent Physical-Component.

IMPLEMENTATION

Proteus has four major components (Figure 4): Data Manager, Web Service, RESTful Service, and V4 Service. The first three run inside a Java EE application server. The V4 Service runs as a standalone service. The Data Manager has all the business logic. It collects and serves data from various sources: Proteus' Database, ChannelFinder, and Control System (EPICS). The Data Manager uses PVManager [7] to access Control System. It also manages authorization, transactions, and concurrency. The V4 Service is currently a prototype.

Technologies

Most of Proteus, excepting the V4 Service, is written using Java EE technologies. At FRIB, we have used Glassfish as the application server, MySQL as the DBMS, and Apache as the web server.

ISBN 978-3-95450-139-7

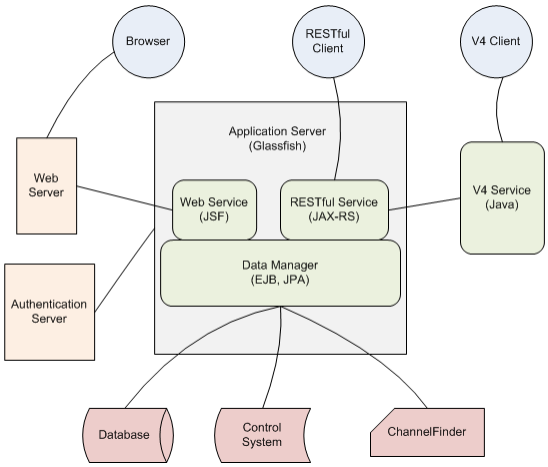

Figure 4: Proteus implementation.

Figure 5 shows a screenshot of Proteus' web interface. The component tree is on the left, and the details of the selected component on the right, including its field-curve. Figure 6 shows the control signals (PVs) associated with a component and their live values.

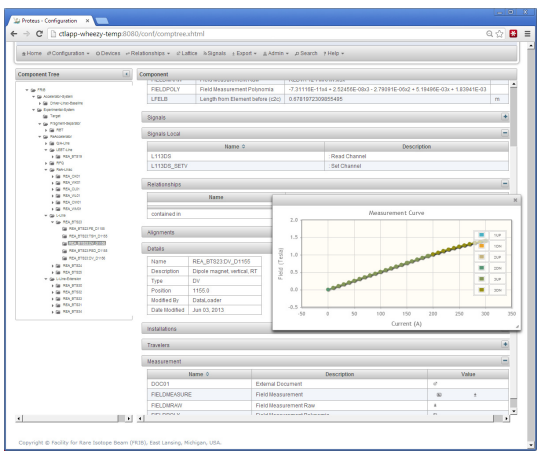

Figure 5: Components and measurements.

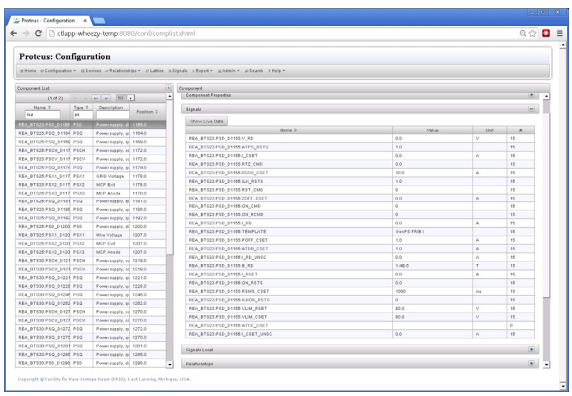

Figure 6: Component and signals.

RELATED WORK

Several systems based on an integrated database have been developed to manage the data associated with an EPF [8][9][10][11][12]. Proteus is unique in its scope, architecture, data model, services, and interfaces.

CONCLUSION

Proteus is an information service to manage the configuration data of an EPF during design, commissioning, operation, and maintenance. We have described its architecture, data model, implementation, and interfaces. It is being used at FRIB, and is available for download from [13]. Even though it is not yet ready for production use by other labs, early adaptors and developers may try it.

We are currently working on adding authorization and calibration to Proteus, expanding the RESTful interface, and improving the V4 service.

AKNOWLEDGEMENTS

We would like to thank the EPICS V4, IRMIS, ChannelFinder [14], and PVManager teams for their suggestions and support. We would like to especially thank Don Dohan for his insights into accelerator facility concepts.

REFERENCES

[1] DISCS, http://discs.openepics.org
[2] V. Vuppala et al., "Distributed Information Services for Control Systems", ICALEPCS 2013, San Francisco, 2013, in press.
[3] DISCS Handbook, http://discs.openepics.org/documentation
[4] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000.
[5] EPICS V4, http://epics-pvdata.sourceforge.net/
[6] Configuration Module Schema Design, http://discs.openepics.org/configuration
[7] PVManager, http://pvmanager.sourceforge.net
[8] IRMIS: Integrated Relational Model of Installed Systems, http://irmis.sourceforge.net
[9] J. Bobnar and K. Žagar, "BLED: A Top-Down Approach to Accelerator Control System Design", ICALEPCS 2011, Grenoble, France, 2011, pp. 537-539.
[10] Z. Zaharieva et al, "Database foundation for the configuration management of the cern accelerator controls systems", ICALEPCS 2011, Grenoble, France, pp 48-51
[11] T. Larrieu et al, "Design and implementation of the cebaf element database", ICALEPCS 2011, Grenoble, France, 2011, pp 157-159.
[12] D. Beltran et al, "ALBA control & cabling database", ICALEPCS 2009, Kobe, Japan, 2009, pp. 423-425.
[13] DISCS Configuration Module, http://discs.openepics.org/configuration
[14] ChannelFinder, http://channelfinder.sourceforge.net/

ISBN 978-3-95450-139-7