DISTRIBUTED INFORMATION SERVICES FOR CONTROL SYSTEMS

L. Dalesio, D. Dohan, G. Shen, K. Shroff, R. Buono, BNL, USA*
M. Vitorovic, Cosylab, Slovenia
K. Zagar, COBIK, Slovenia
S. Gysin, L. Fernandez, K. Rathsman, G. Trahern, ESS, Sweden
F. Guo, H. Lv, C. Wang, Z. Zhao, IHEP, China**
E. Berryman, P. Chu, D. Liu, S. Peng, V. Vuppala, NSCL-FRIB, USA***

Abstract

During the design and construction of an experimental physics facility (EPF), a heterogeneous set of engineering disciplines, methods, and tools is used, making subsequent exploitation of data difficult. In this paper, we describe a framework (DISCS) for building high-level applications for commissioning, operation, and maintenance of an EPF that provides programmatic as well as graphical interfaces to its data and services. DISCS is a collaborative effort of BNL, FRIB, Cosylab, IHEP, and ESS. 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. The services/applications include Channel Finder, Logbook, Traveler, Unit Conversion, Online Model, and Save-Restore. Each component of the system has a database, an Application Programming Interface (API), and a set of applications. The services are accessed through REST and EPICS V4. We also discuss the challenges to developing software in an environment where requirements continue to evolve and developers are distributed among different laboratories with different technology platforms.

INTRODUCTION

There is need for an integrated information system that manages the data and computation 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.

Distributed Information Systems for Control Systems (DISCS) [1], [2] is a framework for integrating, managing, and accessing this information base, and other necessary computation. Its scope covers these data:

- Machine Configuration: Components and their configuration; design, measurement, alignment, maintenance, inventory, and calibration data.
- Lattice: Elements and their settings.
- Logbooks: Electronic logbook entries.
- Traveler Information: Device fabrication, test, and measurement data.
- Save/Restore: Save and restore state of the machine and its segments.
- Online Model: Simulation of the machine. Inputs and outputs of the simulations.
- Physics: Data related to physics applications.
- Cables: Information related to cables, trays, routing, etc. to help with pulling and maintenance of cables.
- Security: User authentication, authorization, device and data access control.
- Control Signals: Control System (EPICS) related information such as process variables and input/output controllers.
- Alarm: Supervision of alarm states in the EPF, assistance to operators for diagnostics and resolution of alarm conditions.
- Operations: Beam statistics, run hours, beam on target, shift summary, downtime, bypass records.

Vision

DISCS' vision is to develop collaborating database-driven services that any experimental physics facility can easily configure, use, and extend for its design, commissioning, operation, and maintenance. In the process, a set of APIs will be defined that will form the framework of such an integrated system.

Collaboration

Due to the large scope of this endeavour and limited resources, it was felt that this effort can be realized only through collaboration among various labs. However, collaboration among labs brings unique challenges to the project. Each lab has conflicting and evolving requirements. They have differing schedules, technologies, and development methodologies.

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

* This work is done by Brookhaven Science Associates, LLC under Contract No DE-AC02-98CH10886 with the U.S. Department of Energy.
** This work is supported by the CSNS Project.
*** 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

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 [3] and PVaccess [4] protocols. Application Layer consists of the software tools or components that present the information to the user. Application Layer accesses the databases through Service Layer.

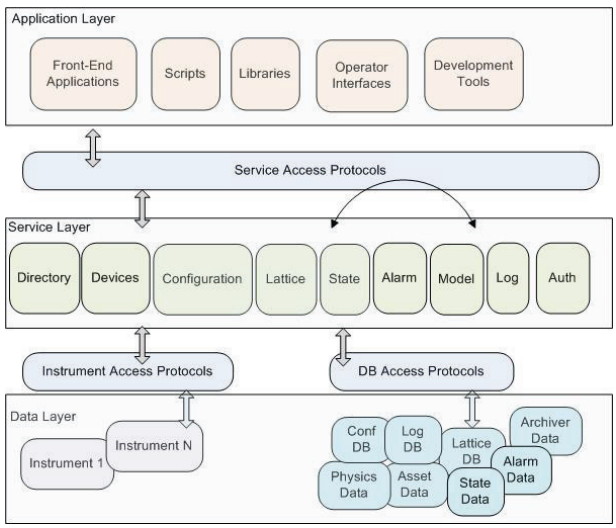

Figure 1: Applications, services, and data.

EPICS V4

DISCS is part of the EPICS V4 ecosystem [4]. It focuses on middle-layer services and end-user applications primarily driven by databases of structured or unstructured data. This is illustrated in Figure 2. Applications access the DISCS services (middle layer) through Channel Access, PVaccess, or RESTful protocols. Note that many DISCS services may be used with control systems other than EPICS.

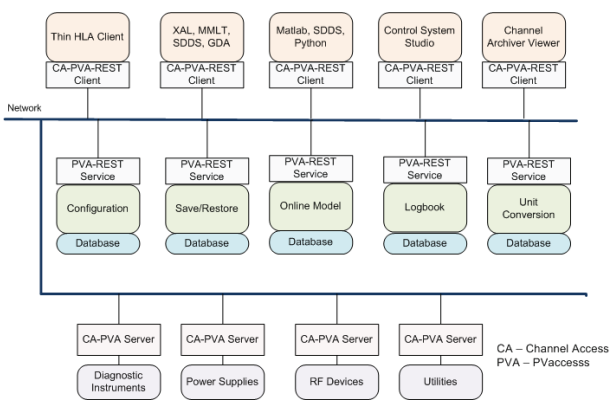

Figure 2: EPICS V4 and DISCS.

DEVELOPMENT METHODOLOGY

DISCS is an ambitious undertaking. It is being developed by multiple teams from different labs at dispersed locations. Each lab has its own priorities, requirements, schedules, constraints, and processes. The traditional approaches of database design, software development and project management do not work well in such environment. It demands varying degrees of redundancy, agility, and uncertainty in its development.

Development in such collaborative environment is a challenge. Traditional database application development methodology, where schema is developed first and managed centrally, was initially tried. It was abandoned due to slow pace of development. Pros and cons of the development techniques that were considered are described in [2]. It was decided to develop DISCS in a decentralized approach as described below.

The entire development process is broken into phases:

- Phase I: The entire system is broken down into smaller loosely-coupled parts or domains. Two domains are loosely coupled if they do not share a lot of data. The division is also based on functionality, domain expertise, and ease of development.
- Phase II: In this phase each domain is implemented independently. A *module* is an implementation of a domain. There can be several implementations (modules) for a domain. Each module is developed by a team which is responsible for the entire development: requirements, analysis, design, database schema, services, API, and applications. The goal of this phase is to understand requirements, refine the database schema, and release usable applications. Agile techniques are recommended during this phase, however, each team is free to follow any methodology.
- Phase III: Form the API for each domain. Even though formation of APIs gets initiated during Phase II, the priority is to get the applications to the users. During Phase III APIs are made complete, including standardizing of data-structures for data exchange.
- Phase IV: Integrate the domains. This involves:
  - Enhance functionality by accessing data from other modules
  - Manage data redundancy among modules
  - Evaluate performance bottlenecks
  - Develop domain-level APIs from module APIs. If there are multiple modules for a domain, then come up with a comprehensive standard API for the domain.

Integration may occur at various levels. It is encouraged to integrate through APIs at service level. However, due to performance reasons, it may be necessary to have certain integration at database level. See *Challenges* section for details.

MODULES

A module is an implementation of a domain. It consists of a database, one or more services, an API, and zero or more applications to manage the data/service. Modules interact with one-another through their service or database interfaces.

ISBN 978-3-95450-139-7

A module is defined as a composition of:

- A database
- Zero or more module applications. The applications of a module are for managing the data or services of the module. They are generally written by the Module Team. They are distinct from the applications written by users.
- The Module API. It is a specification for the module's functionality and data. It is a contract between the Module Team and module's users. A module's API is not restricted to V4 or REST interface only; it may have database interfaces: join columns or primary keys.

The structure of components (database, API, services, and module applications) within a module is left to the discretion of the Module Team. A simple approach is to have the services and module applications access the database directly. Module Team manages the changes to schema and its effects (on applications and services). Another approach is to have a layer (stored procedures or data library) over database; services and applications access the data only through this layer. The third approach is to have applications access the data only through the services. Note that these options are for the structuring of the components within a module. User applications must access a module's data only through its service API.

The entire system is made up of a collection of collaborating modules as shown in Figure 3. User applications access module services using the module's service APIs through REST or V4 services. However, modules may access other modules using the database interfaces as well. Some of the DISCS modules are described in [[5]-[8]].

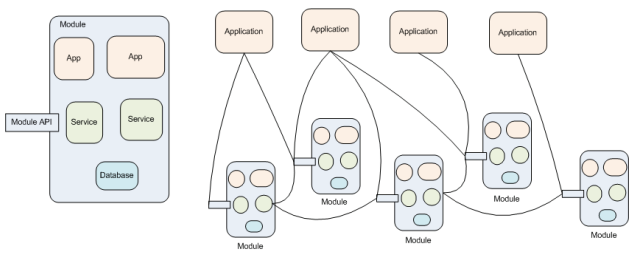

Figure 3: Collaborating modules.

CHALLENGES

Many modules have limited common data among them. They integrate with one another easily. They just need the APIs to access data from one another. However, for some modules that are somewhat tightly coupled (high data dependency), such as Configuration and Lattice or Cable modules, integration can become problematic. Some of the issues in such cases are:

- Performance: Accessing data through API is slow especially if queries need joins across modules. This can be mitigated in several ways:
  - The service can access tables of another module directly. The two teams must agree upon the interface (table name, column names and types) and make it part of the module's API.
  - One module may cache (read-only) another module's data, and manage the coherency through a service or federated tables.
  - If the data cohesion between the modules is very high, then it is better to combine the modules into one.
- Data Integrity: A module may need to maintain referential integrity through data in other module i.e. foreign keys to tables in another module. This can be achieved through including the referenced primary key in that module's API (table name, column names and types). Another way is to have an application check for integrity periodically. Triggers may also be used but are discouraged as they are DBMS specific.
- Data Redundancy: Two modules may have the same data. This redundancy may be eliminated from one module and replaced with joins and/or foreign keys as mentioned above. In some cases the data in one module can be treated as cache and kept coherent through a service or a federated table.

As discussed above, a module's API is not restricted to V4 or REST interface only; it may have database interfaces: join columns or primary keys corresponding to certain foreign keys. Since surrogate key for an entity may change over time, natural keys must be used for any database interfaces (unless surrogate keys are guaranteed to remain unchanged).

Another challenge is of data migration. When a schema changes, due to a new version of the module or due to integration, it is sometimes difficult to automate migration of old data to the new schema. This depends on the type of schema changes.

Different teams may use different technologies for development based on a lab's policies and expertise. This can cause huge problems in deployment, maintenance, and customization of the module at another lab. To limit this problem, the collaboration recommends a set of technologies for development. For example, MySQL is the recommended DBMS, V4 and REST the recommended service layer protocols, and Java and Python the recommended languages.

It is important for teams to understand one another's schema. However, different teams may use different conventions for naming their database objects. Hence, the collaboration recommends a schema naming convention [2].

Project Management

DISCS Collaboration has no control over the software lifecycle development and project management methodologies of individual teams as these are mostly dictated by the labs. Each lab uses different techniques and tools. For example, FRIB uses a mix of agile and traditional project management processes, and ESS uses Scrum-based processes. Hence project management activities are kept to a minimal. A lot of emphasis is put on communication. Quarterly goals are set for each

ISBN 978-3-95450-139-7

domain. The group meets every week using web conferencing. The group meets in-person every quarter. It also meets with the rest of EPICS V4 community during the in-person meeting. As the modules mature, a more rigorous system of requirements, schedule, and performance monitoring is being planned.

CURRENT STATUS

Different DISCS modules are at different stages of development as shown in Table 1. Multiple implementations of Lattice-Model domain are being developed at different labs. Module teams working on coming up with domain-level APIs for each domain.

Table 1: Domain Status

Domain	Status
Logbook	In production at FRIB & BNL
Etraveler	In production at FRIB
Configuration	In production at FRIB. Developed by FRIB, ESS, and Cosylab
Cables	Under development at FRIB
Lattice-Model	Under development at FRIB, BNL, and ESS
Operations	Under development at FRIB
Naming Convention	In production at FRIB. Developed by FRIB, ESS, and BNL
Unit Conversion	Under development at BNL
Control Signals (PV)	In production at BNL and FRIB (ChannelFinder)
Inventory & Maintenance	Under development at FRIB, ESS, and Cosylab
Security	Under development at ESS
Machine state dumps for MPS	Not initiated
Interlock hierarchy	Not initiated
Save/Restore	In production at BNL

RELATED WORK

Several systems based on an integrated database have been developed to manage the data associated with an EPF [[9]-[13]]. Most of them are limited in their scope, mostly to information related to Control System. Their development effort is limited to one lab, and hence their database and applications can be designed using the traditional methods. Our scope and collaborative structure bring unique challenges to the design and development of the system. Our goals of developing generic modules and APIs that anyone can use also make DISCS a unique collaborative endeavour.

CONCLUSION

There is a need for an integrated information system that provides access to all the data of an accelerator facility. DISCS is an effort to implement such a system using distributed and collaborating services. It is part of the EPICS V4 ecosystem. DISCS is being developed collaboratively by several organizations. This presents a unique set of technical and developmental challenges. We addressed these challenges using suitable architecture, software development processes, conventions, and database implementation techniques.

ACKNOWLEDGEMENTS

We would like to thank the Control System Studio [14], EPICS V4, IRMIS, and PVManager [15] teams for their suggestions and support.

The Centre of Excellence for Biosensors, Instrumentation and Process Control is an operation financed by the European Union, European Regional Development Fund and Republic of Slovenia, Ministry of Higher Education, Science and Technology.

REFERENCES

[1] DISCS, http://discs.openepics.org
[2] DISCS Handbook, http://discs.openepics.org/
[3] R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. Dissertation, University of California, Irvine, 2000.
[4] EPICS V4, http://epics-pvdata.sourceforge.net/
[5] V. Vuppala et al., “Proteus: FRIB Configuration Database,” TUPPC031, ICALEPCS 2013, San Francisco, CA, to be published; www.JACoW.org
[6] Olog, http://olog.sourceforge.net/olog/
[7] P. Chu et al., “Accelerator Lattice and Model Services,” MOPPC152, ICALEPCS 2013, San Francisco, CA, to be published; www.JACoW.org
[8] G. Shen, “NSLS II Middlelayer Services”, MOPPC152, ICALEPCS 2013, San Francisco, CA, to be published; www.JACoW.org
[9] IRMIS: Integrated Relational Model of Installed Systems, http://irmis.sourceforge.net
[10] J. Bobnar and K. Žagar, “BLED: A Top-Down Approach to Accelerator Control System Design”, TUAAULT03, ICALEPCS 2011, Grenoble, France, pp. 537-539; www.JACoW.org
[11] Z. Zaharieva et al., “Database foundation for the configuration management of the CERN accelerator controls systems,” MOMAU004, ICALEPCS 2011, Grenoble, France, pp. 48-51; www.JACoW.org
[12] T. Larrieu et al., “Design and implementation of the CEBAF element database,” MOPKN029, ICALEPCS 2011, Grenoble, France, 2011, pp. 157-159; www.JACoW.org
[13] D. Beltran et al., “ALBA control & cabling database,” MOPMN003, ICALEPCS 2009, Kobe, Japan, 2009, pp. 423-425; www.JACoW.org
[14] CS Studio, http://controlsystemstudio.github.io/
[15] PVManager, http://pvmanager.sourceforge.net

ISBN 978-3-95450-139-7