DATABASE AND INTERFACE MODIFICATIONS: CHANGE MANAGEMENT WITHOUT AFFECTING THE CLIENTS

M. Peryt, R. Billen, M. Martin Marquez, Z. Zaharieva, CERN, Geneva, Switzerland

Abstract

The first Oracle®-based Controls Configuration Database (CCDB) was developed in 1986, by which the controls system of CERN's Proton Synchrotron became data-driven. Since then, this mission-critical system has evolved tremendously going through several generational changes in terms of the increasing complexity of the control system, software technologies and data models. Today, the CCDB covers the whole CERN accelerator complex and satisfies a much wider range of functional requirements. Despite its online usage, everyday operations of the machines must not be disrupted.

This paper describes our approach with respect to dealing with change while ensuring continuity. How do we manage the database schema changes? How do we take advantage of the latest web deployed application development frameworks without alienating the users? How do we minimize impact on the dependent systems connected to databases through various APIs? In this paper we will provide our answers to these questions, and to many more.

INTRODUCTION

The Controls Configuration Database (CCDB) provides the Configuration Management services for the Control System of all CERN accelerators [1]. It is a multifaceted software infrastructure composed of many interrelated components at the heart of which lies an instance of Oracle® database. CCDB relies on web deployed tools for data browsing and editing, and is accessible through Application Programming Interfaces (APIs) written in different programming languages. It is a component of a distributed database environment for the CERN accelerator complex and as such, it is linked to several other Oracle instances.

The CCDB forms the data foundation of the accelerator Control System and is used online for all controls operations. Consequently, full availability and reliability needs to be guaranteed to ensure accelerator operations and thus the CERN physics program.

No Downtime Allowed

Fortunately, the occurrences for which the CCDB was the cause of accelerator downtime have been extremely rare. However, in order to maintain its high quality, the CCDB has to evolve over time, similarly as all equipment that needs to be maintained and upgraded.

The accelerator operations follow a precise schedule per accelerator with technical stops with different frequencies and varying lengths. As for any hardware or software intervention that affects machine operations, deployments of CCDB changes need to be carefully planned in order to minimize the risk of disruptions during physics exploitation. Typically, we are granted an hour of database unavailability, every two months, to be scheduled on a single target day. Interventions requiring a longer downtime have to be scheduled during shutdown periods, i.e. outside the physics program.

Preparing the Interventions

Due to the severe constraints on deployment time and full proof quality of the intended modifications, every change undergoes prior multistage testing and comes bundled with a clear procedure for rolling back to the previous state. The sections that follow go into details of how these changes in the CCDB are dealt with. In order to better understand what type of changes are concerned, it is useful to recall the major milestones in the Controls Configuration Database.

CCDB HISTORY

The list below sketches the major modifications that were introduced in the CCDB over its 25 years of existence [2]. The important database schema modifications, interfacing technologies and functionality leaps are indicated.

- 1980 – Creating a centralized file-based data storage for some components of the Controls System
- 1986 – Introduction of Oracle RDBMS: CCDB birth
- 1987 – Data extraction scripts (embedded SQL in Pro*Fortran, re-implemented in Pro*C later on)
- 1995 – User interfaces based on Oracle Forms and PL/SQL Web Toolkit (OWA)
- 1999 – Java Directory Services
- 2003 – Introducing FESA Device-Property model
  2004 – Migration of data browsing interface to Oracle APEX technology
- 2005 – Introducing authentication and authorization mechanisms
- 2005 – Big-bang refactoring of the database schema
- 2005 – Introducing the Session Auditing and History Recording Framework
- 2006 – Redesign of the data editing interfaces using Oracle ADF technology
- 2007 – Introducing Hardware device-property model
- 2010 – Introducing Virtual device-property model
- 2011 – Introducing Configuration Change Management and Status Accounting in the CCDB [1]

The original database in 1986 was based on Oracle version 5, which was migrated through each major version up to the current Oracle 10*g*. The upgrade to

Oracle 11*g* is scheduled in January 2012 during the next winter shutdown.

The CCDB is now under the responsibility of the second generation of database engineers. Over the 25 years of the lifetime of the CCDB, programming languages and technologies change, applications and software come and go, but the data remains.

REASONS FOR CHANGE

A look at the timeline above makes it clear that the driving forces for change can be classified into the following categories:

- Changing user requirements.
- Changing software technologies.
- Internal refactoring to improve software quality and reduce maintenance burden.

Changing User Requirements

Changes in the user requirements constitute the principal reason for CCDB modifications. New functionalities are requested, existing ones change, obsolete ones are dropped, new components of the Control System are in need of data driven configuration, and users' expectations towards the IT systems evolve. Although most frequent, these modifications benefit from the highest rate of acceptance because their rationale is well understood by the users.

Changing Software Technologies

The second category of changes is forced by the evolution of the technology stack that CCDB depends upon. Since the early days of CCDB we have been using Oracle products and now that Java is also part of Oracle we are nearly 100% dependent on this vendor's products. The potential vendor lock-in is not the main concern, but as Oracle technologies evolves so needs the CCDB.

Database upgrades are necessary to ensure the long-term continuity and support, and to take advantage of new features, functionality and efficiency. These upgrades are never carefree, but have been consistently programmed towards the stable, terminal release of a major version.

The data-driven client APIs have evolved from Pro*C precompiled code to Java (although some Pro*C legacy has not yet been phased out).

For interactive user interfaces, Oracle Forms have been upgraded from version 4.5 up to 9*i* and finally replaced with J2EE-compliant Oracle Application Development Framework (ADF).

Web-deployed interfaces and reports, originally generated by the PL/SQL OWA module, left its place to Oracle HTMLDB which evolved into Oracle Application Express (APEX).

All these technological changes are imposed upon the developers as well as upon the end-users. Often this requires a change in the users' habits and requires well devised communication campaigns and careful deployment. The human factor must not be neglected here as no software product can be successful unless its users are happy.

Software Refactoring

The last – but not least – driver for change comes from our own quality assurance standards. We are aware that in order to maintain the high quality of the systems we provide, there is a need to regularly revisit existing production code and evaluate following considerations:

- Improving the performance of interfaces.
- Streamlining data propagation.
- Simplifying workflows.
- Improving data quality and integrity.
- Getting the most out of new technologies.
- Complying with our own coding and data management standards.

Modifications that result from these considerations do not necessarily give any added value or visible impact for the end user, but is purely for reasons of code maintenance and efficiency of the development team. In addition, these changes are not always transparent to the end user. For example, the introduction of referential integrity constraints across the database – which were not present in the original design – resulted in error messages seen by the users. This modification greatly enhanced the quality of data, as opposed to recording incoherent, erroneous information prior to the refactoring.

It has to be noted that all three categories of changes have an impact on the users of the services provided by the database. These users are not only human actors, but also the dependent computer systems that are linked to the CCDB. The following sections outline our time-tested strategies for managing this impact.

STABLE INTERFACES

By having well defined interfaces towards the external systems, the CCDB can afford to change underlying implementation quite freely and transparently. Data is exposed through database views. Unless a very profound refactoring takes place, the structure of views can remain identical even if underlying tables are altered. Backwards compatibility after data model changes is a primary concern. If this cannot be ensured, the interfaces have to be renegotiated with the users, and the whole deployment process becomes much more complicated. For example, hundreds of front end computers may need to be rebooted to force the correct runtime deployment. Depending on the functionality of the front end, this is something that can only be scheduled during a technical stop.

We have separate dedicated database accounts with different sets of privileges corresponding to different usages of data. This way we control in a very precise manner the data access – for reading or for editing – by a particular set of users. For distinct clients, specific views are provided, which all point to the same underlying data.

An additional layer of isolation is provided by the APIs that are exposed to the users. Nowadays, these are written in Java or PL/SQL with XML being the representation of

choice for data transfer. These APIs have proven to be very stable . One example is Java Directory Services [3] which dates back to 1999, but has been recently refactored to improve the performance and benefit from the many useful features provided by the Spring framework [4]. Even if its internals changed to a very large extent, and some new interfaces were added, the existing ones remained fully backwards compatible.

STAGED TESTING AND DEPLOYMENT

Over the years, we have worked out a staged environment that provides a complete and efficient framework for testing and deployment. We have provided dedicated instances of the CCDB in four distinct environments: DEV, TEST, NEXT, and PRO as shown in Fig. 1.

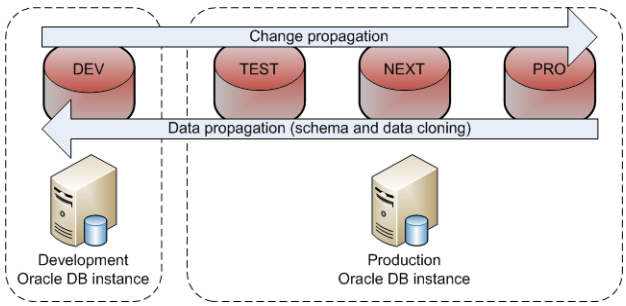

Figure 1: Overview of the four CCDB environments.

DEV is the development environment in which the database schema is installed at a development database instance. It is our "try-it-out" environment where no stability is required. DEV is used for proofs of concepts, general development and introducing new code.

TEST is the environment for functional testing. Tests are started with the database being a clean 1:1 copy of the production schema. Consequently, new code from DEV is copied across and all necessary tests are carried out in isolation from other systems that depend on CCDB. The following tests are against all existing applications that are affected by the newly introduced changes.

NEXT is a relatively recent environment (created in 2008) that has been introduced for integration testing. It is a part of the Controls Testbed since 2010 [5], a fully functional vertical slice of the Accelerator Controls System. As such it has every component from the whole control system present. The NEXT environment represents the CCDB as it will be after the following deployment in PRO.

PRO is the production environment used by the Controls System. It is guaranteed to be stable, as no changes are performed outside of scheduled interventions.

TEST, NEXT and PRO environments are hosted on the same database server, which allows us to test the performance and scalability in real life conditions. For resource demanding testing against TEST and NEXT, precautions are taken in order to avoid performance degradation of the production environment.

INCREMENTAL APPROACH

Historically some major change sets were applied to CCDB as "big-bangs" but with the LHC in full operation, this type of revolution is excluded in order to avoid any disturbance noticeable by the user community. Therefore, deployments are rolled out adiabatically and step-wise over relatively long periods of time. This method is applied to the vast refactoring of the data management services for the CERN Front-End Software Architecture (FESA) framework [6]. The objective is to fully rationalize the database schema and to eradicate inefficient XML objects inside database in favour of a relational data model. The FESA workflow is complex and consists of controls device class modelling, deployment and instantiation phases. These workflow phases are tackled one at a time, passing through all four environments. Passing to the next one is only done when the expected behaviour is ensured. This way, an effective rollback strategy is in place in case of failure and the impact on users is limited. Moreover, even when new database schema and APIs are deployed, the previous ones are maintained in parallel for a few weeks to guarantee redundancy in case of problems. The same is valid when new user applications are put in place – the old ones are supported, if possible, for a certain period (sometimes for months) in order to ensure that the users have accepted the new functionality.

The incremental approach has also been applied for the replacement of PL/SQL OWA based CCDB Data Browser with the new one built using Oracle APEX [7]. The full functionality of the existing set of pages needed to be reproduced, while adding support for new sets of data. Navigation and overall usability was to be improved, providing a new, contemporary look and feel. The major reports were developed first – i.e. roughly 80% of all content – and deployed for public use, followed by continued adding of the remaining ones. However, to gain user acceptance, a proactive communication campain and feedback channels were put in place. In parallel, the outdated data browser was kept running for one year without any development effort. Eventually access to the obsolete service was blocked, in line with the announced planning. A very similar strategy has been followed when replacing more than 150 Oracle Forms with their Oracle ADF successors.

We are convinced that incremental approach to change is a valid one. This conviction is further strengthened by looking at some examples from the "real world", especially in the domain of Web applications. One notable example of incremental functionality enhancements is Google Mail where new features are added one at a time and there is a possibility for the users to switch them off, at least for a while.

CAREFUL PLANNING

With all CERN accelerators dependent on CCDB for their everyday operations, there is no room for

improvisation when it comes to rolling out new features. Every deployment requires very careful planning.

The impact of every change to be deployed is assessed right from the start. Deployment scripts are prepared with rollback paths at every step of the process. An exact copy of the PRO environment in NEXT is created and the timed execution of the scripts is carried, indicating the time to be allocated for the final deployment in PRO. The interventions are planned in line with the accelerator schedule in order to have zero impact on the machine operations. Announcements are prepared and sent to mailing lists of the user community.

With this type of preparation, the only action on D-day is to execute the plan. In case of any mishap, a clear exit path is available and possible at every stage. In addition, every action is recorded and stored in a secure file system in a dated folder for future reference.

This scenario ensures full transparency and traceability of each software deployment session, essential for the overall quality assurance process.

DEALING WITH HUMAN FACTOR

Dealing with the human factor is probably the most delicate part of the whole software change process in the context of CCDB. The accelerator community is large and diverse from control room operators to equipment specialists and accelerator physicists. Their expectations related to the provided tools are high, but also vary for each accelerator, due to operational habits which have not yet fully converged. As designers and developers of those tools, we have to find the balance between generic implementations and specific functionality.

Twenty-five years ago, the CCDB was deployed in the scope of the CERN Proton Synchrotron complex, without covering the larger machines. Data entry was centralized and done by a team of two dedicated data management engineers, working in close collaboration with operators and equipment experts. Since 2003, with a second generation of database engineers, the CCDB scope has been extended to cover the complete accelerator complex. Focussing on the data model, interfaces and business logic, the ownership of configuration data has been transferred to the responsible people of the hardware or software, which needs to be configured through CCDB. To this end, data entry tools have been adapted, authentication and authorization mechanisms introduced and the people trained. For reasons of traceability, every single data manipulation, executed by a user, is recorded together with time stamp and session information. This mechanism also enables the possibility of reverting to a previous data situation. Currently, more than 300 unique users are active across all applications of the CCDB.

Whenever an end user tool is to be refurbished, the key users are involved very early in the development process. They are invited to test the iterative versions of the software and to provide feedback. Prior to a deployment in PRO, the changes and their possible impact are communicated to all concerned parties via the relevant announcement channels.

CONCLUSIONS

The Controls Configuration Database is an evolutive system and as such it is subject to change. However, considering that it lies at the heart of the CERN accelerator complex, there is no room for improvisation when deploying new features. The successful strategy for dealing with change that has been put in place is based on the following guidelines:

- Involve end-users right from the start, throughout the design and development process
- Provide four separate environments for development, unit and functional testing, integration testing (TestBed), production
- Analyze the impact of a change and try to apply only backward compatible changes
- Communicate timely, clearly and transparently on scheduled intervention and their impact
- Coordinate the upgrades with impacted clients

By respecting these guidelines, we have been able to perform and manage changes and have them accepted by the user community.

REFERENCES

[1] Z. Zaharieva et al., “Database Foundation for the Configuration Management of the CERN Accelerator Control Systems”,ICALEPCS’11, Grenoble, France, Oct-2011

[2] J. Cuperus, R. Billen, M. Lelaizant, “The Configuration Database for the CERN Accelerator Control System”, Icalepcs2003, Geongeoeng, Korea

[3] J. Cuperus et al., “A Directory Service for the CERN PS/SL Java Programming Interface”, Icalepcs’99, Trieste, Italy

[4] http://www.springsource.org/

[5] J.Nguyen Xuan, V.Baggiolini, “Testbed for validating the LHC controls system core before deployment”, Icalepcs2011, Grenoble, France, Oct-2011

[6] M. Arruat et al., “FESA 3.0”, ICALEPCS’11, Grenoble, France, Oct-2011

[7] Z. Zaharieva, R. Billen, “Rapid Development of Database Interfaces with Oracle APEX, used for the Controls Systems at CERN”, ICALEPCS’09, Kobe, Japan, Oct-2009, THP108