RENOVATION OF THE CERN CONTROLS CONFIGURATION SERVICE

L. Burdzanowski, C. Roderick CERN, Geneva, Switzerland

Abstract

The Controls Configuration Service (CCS) is a key component in CERN's data driven accelerator Control System. Based around a central database, the service also provides a range of client APIs and user interfaces - enabling configuration of controls for CERN's accelerator complex. The service has existed for 35 years (29 based on Oracle DBMS) [1]. To cater for changing requirements and technology advances there has been substantial evolution of the CCS over time. Inevitably this has led to increases in CCS complexity and an accumulation of technical debt. These two aspects combined have a negative impact on the flexibility and maintainability of the CCS, leading to a potential bottleneck for Control System evolution. This paper describes on-going renovation efforts (started mid-2014) to tackle the aforementioned issues, whilst ensuring overall system stability. In particular, this paper covers architectural changes, the agile development process in place - bringing users close to the development cycle, and the deterministic approach used to treat technical debt. Collectively these efforts are leading towards a successful renovation of a core element of the Control System.

INTRODUCTION

The CERN Control System is a data-driven multi-layer infrastructure including:

- *Low-level hardware and software* – e.g. timing infrastructure, equipment drivers, Front-End Computers (FEC), end-user developed C/C++ binaries representing operational "devices", etc.
- *Middleware layer* – e.g. read/write access to processes running on FECs and Role Based Access Control (RBAC).
- *High-level software* – e.g. high-level settings management, data acquisition and archiving.

The Controls Configuration Service (CCS) helps bind all of the layers together by providing them with complete and coherent configurations that are necessary for the proper functioning of the Control system [2].

The current architecture of the CCS is based on:

- An Oracle database (2-node RAC cluster)
- A set of high-level client Java APIs
- Database level client APIs (PL/SQL interfaces)
- Numerous Graphical User Interfaces based on proprietary Oracle technologies: Application Development Framework (ADF) and Oracle Application Express (APEX).

The database is implemented using a relation model, with approximately 700 domain tables and ~7GB of core domain data (excluding binary, log and history data – which collectively accounts for ~115GB).

The criticality of the service for safe operation of the accelerators chain is high (though not required for their safe shutdown): The CCS is essential for proper accelerator configuration and start-up – especially during Technical Stops when equipment and other components of the Controls System undergo maintenance and upgrades.

The CCS exists for 35 years, during which the scope, architecture, implementation technology and development methodology have kept evolving. In the middle of 2014 the first major service-wide renovation and overhaul has started – marking the beginning of a new chapter in its long history.

RENOVATION STRATEGY

The motivation behind the complete renovation of the service can be summarized as the need to increase service flexibility while lowering total cost of development, and to advance service functionality to the state required by activities planned for the next CERN Long Shutdown (LS2 – scheduled start early 2019).

The cornerstones of the renovation strategy are:

- Suppression of the technical debt accumulated over the years.
- Changes in the overall architecture
- Adaptation of the Lean software development process [3].

All of these aspects are closely related as suppression of technical debt is essential in order to advance the system architecture, while taking proper architectural and design decisions prevent further "erosion" in the system and limit existing technical debt. The adapted software development process facilitates implementing changes: enabling a lower overall cost of development and increased agility. The first two aspects are a mid–to–long–term perspective (from mid–2014 to the start of LS2). The implementation of the Lean software development process is already well advanced and can be considered finished by the end of 2015.

In order to fully understand the context of the renovation efforts it is necessary to briefly look back at the evolution of the CCS scope and technologies used.

Service Evolution

The scope of the CCS was initially limited to the PS (Proton-Synchrotron) complex controls system, meaning that the service and its database were oriented towards a concrete accelerator and its specific control system. The first relational database was introduced in 1986. Over the years the scope grew following the evolution of CERN's accelerator complex [4]. 1995 marks the introduction of graphical user interfaces (GUI) based on Oracle Forms and PL/SQL Web Toolkit (OWA). The first Java based data access API was implemented in 1999 facilitating access for

ISBN 978-3-95450-148-9

high-level applications. Starting in 2006, another Oracle based GUI solution (ADF – a Java Server Faces implementation) was put in place to replace existing OWA and Forms applications. In 2009, APEX (a subsequent framework for building database-driven GUI's) was adapted alongside ADF.

Besides the technical changes, the service role and scope has been expanding in recent years:

- Multiple additional device-property models were introduced,
- The Front-End Software Architecture (FESA) framework reached the next major version [5],
- The service incorporated configurations specific to various sub-systems like:
    - Beam Interlock System,
    - Power Converters,
    - Role Based Access Control.

Together with these changes certain functionalities of the service became partially suppressed, specifically legacy high-level settings management, for which the functionality was incorporated into LSA [6] for the majority of the accelerators.

The following patterns emerge when we look back at the aforementioned evolution: reliance on Oracle proprietary GUI technologies, and the progressive growth of the service well beyond its original system design and architecture that was tightly coupled to a specific accelerator. Both of these aspects have contributed to the current state of the service and triggered the renovation.

ADDRESSING TECHNICAL DEBT

The evolution of every complex hardware and software system inevitably results in increased technical debt and progressive erosion. The availability of new technologies and solutions, as well as the human factor of growing experience – keep changing the perception of quality and adequacy of former technical decisions [7].

In most cases, end users are not directly aware of the technical debt but as software engineers we should perceive it as negative value. It is adverse to system architecture and design, which are planned, deliberate and visionary. By clearly establishing the system boundaries and facilities to assess and measure its evolution the technical debt can be addressed whilst minimizing impact on the end users.

Accidental Complexity

At its basis the technical debt is equal to accidental complexity happening in the system. On contrary to deliberately complex solutions for equally complex problems the accidental complexity happens by itself naturally along evolution of the system.

System complexity can be described by both quantitative and qualitative factors. For the CCS, the quantitative factors include: counts of tables, views, triggers, PL/SQL packages, number of grants, grantees, accounts, lines of code, length of packages and procedures, levels of views nesting etc. All such factors can be measured and automatically classified as a potential problem based on established thresholds. The qualitative factors might be based on quantitative data but cannot nor should not be automated. Such factors are based on experience and often common sense – ultimately a human factor. For example, de-normalization of a database table may be justified by performance requirements or for the sake of clarity in a model, although in most cases it is a sign of shortcomings in the design.

The majority of complexity in a database-oriented system is caused by dependencies (direct and indirect), but rarely by algorithmic complexity. Examples of direct dependencies are relations between objects, database views referencing tables or other views, PL/SQL packages executing code based on database structures. Indirect dependencies include aspects like dynamic/background execution or exposure of the database objects to external users (grants) – an inevitable need contributing to coupling between systems.

Targeted Re-Factoring

In order to guarantee system stability during renovation – specifically when re-factoring areas with high technical debt – a methodical targeted approach with a clear strategy is essential. The CCS renovation efforts fall into two categories: targeted and ad-hoc re-factoring.

Ad-hoc refactoring is considered as a natural part of regular development and does not require extensive planning or analysis. For example: small improvements in the code base like eradication of "dead-code", addition of missing test cases, updates to stale documentation, and re-naming inadequately named objects. In general, such activities shouldn't take more than an additional 20% of the base development time needed to deliver the required functionality.

Targeted re-factoring is predefined as a concrete group of tasks based on the following criteria:

1. *Identify boundaries* – to clearly know when the activity should finish.
2. *Identify clear gains* – to justify the effort. The gains should be tangible, based on facts, and ideally quantifiable.
3. *Identify risks* – to know the impact both within – and outside the service.
4. *Define rollback / fall-back strategy* – to limit any potential negative impact, mainly in critical areas.
5. *Estimate and prioritize* – to realistically plan the effort alongside regular development activities.

The value to be gained from the re-factoring can be classified into distinct areas, and includes in descending order of importance:

1. *Consistency* – i.e. limiting the likelihood of data corruption and/or of non-deterministic states.
2. *Performance* – improving the response times for data reporting and querying for clients.
3. *Maintenance* – lowering the total cost of development, likelihood of introducing new errors, and the usage cost paid by clients (e.g. by obscurity APIs or lack of documentation).

4. *Agility* – ensuring the extendibility of the architecture and limiting the cost / time of delivering new features to clients.

By following the assessment criteria above and understanding their associated gains, it is possible to identify and prioritize renovation activities alongside regular development.

Determinism and Static Code Analysis

Static code analysis (SCA) is the analysis of computer software source code on the contrary to dynamic analysis, which is based on code execution. SCA can be applied to any type of software including databases like Oracle. In this case the analysis includes both database structures (tables, views, constraints, indices, etc.) and executable stored procedural code (PL/SQL packages and triggers).

To match our needs a custom SCA framework has been developed which enables analysis within the database engine. The framework includes a pre-defined set of analysis rules, which can be customized and extended. The analysis can be executed on demand or periodically, and generates reports summarizing the number of rule violations, severity and links to the source. These reports are used to identify areas for in-depth analysis and planning of the re-factoring. Below is an example of a metric indicating the evolution of the count of invalid objects over the past year:

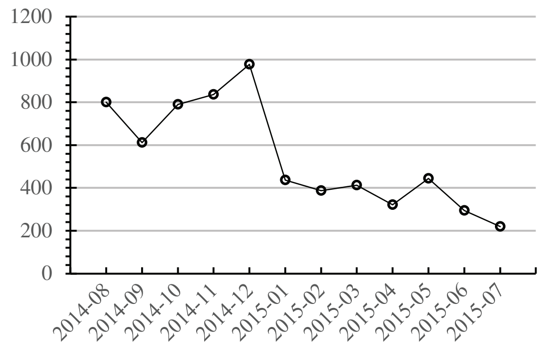

Figure 1: Count of invalid objects in development database, aggregated per months.

The analysis results, fluctuations and evolution of the metrics are the inputs to qualitative assessments and serve as a basis for future planning. The metrics cover various aspects ranging from standardisation concerns (e.g. code not adhering to naming standards, usage of disallowed or obsolete functions, etc.) to abstract complexity of the system (expressed via factors such as code size, table sizes, triggers counts, nesting of database views). With SCA in place we are able to evaluate our efforts over time, relying on factual data rather than assumptions.

ARCHITECTING FOR THE FUTURE

The renovation and supporting changes in system architecture fall into four main categories: suppression of accidental complexity and lowering overall system complexity, context based access to the data, phasing-out of proprietary GUI technologies, and applying a system wide infrastructure for tracing, auditing and monitoring. Combined, these categories form the foundations for simpler, easier and agile development, with an increased quality of service.

The accelerator controls system domain is inherently complex; the corresponding software components are therefore inevitably complex accordingly. During the process of suppressing accidental complexity / lowering overall complexity, we have started to progressively adapt an event driven architecture which has proved to increase cohesion and lowered coupling of system components. New developments and on-going re-factoring conforms to GRASP [8] (General Responsibility Assignment Software Patterns) patterns of Object-Oriented design, tailored to the world of relational databases. To support these changes we have adapted Commons4Oracle (C4O) – a set of PL/SQL libraries for Oracle database, which is actively developed in the CERN Controls group. The library assures further standardization and foundations for future development. Moreover it streamlines solutions in the CCS with other core database projects of the Controls group thus enabling transfer of knowledge and expertise.

Based on direct feedback from CCS users and domain experts we are gradually increasing the scope of context surrounding the data entreated to the system. By utilizing Commons4Oracle extensions, a set of high-level domain specific events is being implemented. Triggering of such events can be based on direct user actions, or a workflow based transition. By attaching state information to core domain entities in the system (e.g. devices), we can now automatically notify users interested in a given "domain event". For example when a computer hosting a device changes its state, the responsible for the device can be notified and take actions if necessary. It is important to add that such notifications may be filtered based on criteria specified by individual users.

The dedicated CCS GUIs are the primary means for end users to access or edit data. On average per day there are over 150 distinct user sessions (from a total of ~400 distinct registered users). Experiences of past years as well as user feedback contributed to the decision to phase-out the existing proprietary technologies in favour of widely adapted solutions of Java based RESTfull services and HTML5/JavaScript web interfaces. In addition this technology stack steadily gains popularity within the software engineering community and in turn facilitates hiring of well-trained specialists.

With tracing and auditing extensions in place, not only the time or user behind a given action is captured, but also contextual information like client IP address, database session and transaction IDs, and name of the program unit and invoked action. This information is used to augment historical data tracing, with which changes to any entity in the system are precisely tracked and can be presented back to users in a context specific manner (e.g. as a log of differences starting from a specific moment in time or by presenting dependencies between objects in a human readable way). This system-wide architectural extension considerably limits the time needed to support users in investigating suspected data problems, and potentially recovering data.

PRAGMATIC AGILITY

As a part of CERN Controls System the CCS is actively used and maintained on regular basis. Its development lifecycle is similar to software products released to end-users that have continuous support for the new extensions and improvements. Such a lifecycle requires responsiveness when addressing end-user's needs but also requires realistic planning which fits into the tight schedule of accelerators operations. The challenges of undergoing renovation and aforementioned aspects motivated the adoption of a Kanban development process [9].

The Kanban emphasizes focusing on continuous improvement, importance of human factors and bringing maximum value to the organisation. Over just a few months the new Kanban development process has been implemented, and in less than six months the efficiency of the team increased noticeably. By visualising the work on a Kanban board, bottlenecks were quickly identified (e.g. too much work in progress, too many unrelated tasks started, or too many new features waiting in quality assurance queue). By not relying on fixed development iterations / sprints – trust from end-users increased as their requested features and bug-fixes are not systematically subjected to prolonged wait times due to extensively planned ahead sprints. The agility and reactivity of the team and CCS as a whole has increased thanks to the Kanban / Lean philosophy of just-in-time delivery and constant focus on activities that bring the most value to end-users. By separating planning and reviews into short-term (weekly / monthly) and mid- term (quarterly / yearly) the CCS team balances providing new functionality within realistic time frames with reacting quickly (hours instead of days) in case of urgent problems.

Thanks to the new methodology, the overall development throughput has increased. More importantly the satisfaction of both CCS users and team members has increased. Changing the way tasks are prioritized and visualized has led to a reduction in pressure and stress on developers. CCS end-users are now much more closely involved in the development process and act as true stakeholders thanks to effective visualization of work in progress and clearly identified stages of the development cycle. These human factors are proving to be essential to the success of the on-going renovation.

In retrospective, the adaptation of a Kanban approach has already resulted in high returns on the time invested into configuring supporting tools to the CCS team needs, and regular discussions and analysis of the changes being implemented. The returns from the Kanban are that more time is spent on delivering actual value, and assuring that new features and improvements are not over-planned nor hurried to production (which would result in faults and frustration of the end-users). Regular retrospectives and critical analysis of changes applied to the working process have positively transformed the way the CCS team works.

CONCLUSIONS

The renovation of a mission critical service with many years of history is a challenge. Alongside changing requirements, growing expectations and needs to consolidate various sub-systems of the Control System, the CCS started to play an even more important role during recent years. The necessity to adapt to these changes and satisfy new requirements is the driver for the on-going CCS renovation. Progressively reducing technical debt increases overall agility, but more importantly it also helps to design a better system for the future. CCS users now have a much better understanding than previously of the value of these changes and together with their increased satisfaction the renovation and technical debt reduction is perceived as added value. The Kanban way noticeably improved the CCS team efficiency and contributed to increased end-user satisfaction. New architecture solutions lay foundations for an advanced, cohesive and agile system that embraces the context and workflows of how CCS users work. The renovation started over a year ago and marked the beginning of a new and exciting era in the long history of the Controls Configuration Service of the CERN Controls system.

REFERENCES

[1] J. Cuperus et al., ICALEPCS1997 – ID085.
[2] R. Gorbonosov, *The Control Systems of the Large Hadron Collider*, CERN Academic Training Lecture Regular Program, http://cds.cern.ch/
[3] T. Ohno, *Toyota Production System: Beyond Large-Scale Production*, ISBN 978-0-915299-14-0, Productivity Press, (1998).
[4] J. Cuperus et al., ICALEPCS2003 – WE114.
[5] M. Arruat et al., ICALEPCS2007 – WOPA04.
[6] G. Kruk et al., ICALEPCS2013 – MOCOBAB05.
[7] MM. Lehman, *Laws of Software Evolution Revisited*, EWSPT '96.
[8] C. Larman, *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.), ISBN 0-13-148906-2, Prentice Hall, (2005) [2004].
[9] H. Kniberg, *Lean from the Trenches: Managing Large-Scale Projects with Kanban* (1st ed.), ISBN 978-1934356852, Pragmatic Bookshelf, (2011).