ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA041

CONCEPTION AND REALIZATION OF THE VERSIONING OF DATABASES BETWEEN TWO RESEARCH INSTITUTES

S. Mueller, R. Mueller, GSI, Darmstadt, Germany

Abstract

This paper describes the version control of oracle databases across different environments. The basis of this paper is the collaboration between the GSI Helmholtz Centre for Heavy Ion Research (GSI) and the European Organization for Nuclear Research (CERN) on several aspects of the control system.
The goal is to provide a sufficient and practical concept to improve database synchronization and version control for the database landscape of the two research facilities.

First, the relevant requirements for both research facilities were identified and compared, leading to the creation of a shared catalog of requirements. In the process database tools, such as Liquibase and Flyway, were used and integrated as prototypes into the Oracle database landscape.

During the implementation of prototypes several issues were identified, which arose from the established situation of two collaborating departments of the research facilities. Requirements on the prototype were, to be flexible enough to adapt to the given conditions of the database landscape and an easy integration without too many changes into the existing development environment.

The creation of a flexible and adjustable versioning system enables the two research facilities to use, synchronize and update the shared database landscape.

INITIAL SITUATION

CERN and GSI collaborate since 2008 on the settings management framework LSA (LSA - LHC software architecture [1]) and the device controller software framework FESA (FESA - frontend software architecture [2]). During the collaboration so far, the main focus was on the software and software versioning and not on the database. The LSA and FESA frameworks have grown between the two institutes and the software is versioned in Subversion [3]. The databases grows without a system that allows to trace or track changes. Although, databases can be restored by backups, it is not possible to determine when which changes were applied and which database change belongs to which software feature. In addition, there is a lack of tools for the collaboration environments that allow the interactions on database level between the institutes, e.g. the easy exchange of features. In the past, both institutes saved the database schema changes in separate versioned text files and the order inside the file represented the execution ordering of the database schema changes.If the changes were executed on every database environment (development - testing - production), they were removed from the file. Therefore, the traceability of the changes was not provided and only existed if one compared two revisions of this file. This had, among other things, the disadvantage that software developers could not determine on which version of the database the software is currently running. This is a disadvantage, because the different database environments like development, production etc. may have different versions of the same schema.

METHODS

First, a requirements catalog of the two institutes was created and taken as basis for the choice of the best method to collaborate and later on for evaluation of a suitable tool. The second step was to make a model of each possible method and test these models in a simulation environment. This approach helped to discover possible issues, which, based on the concept only, could not be detected. To find out the institutes requirements, it was necessary to gain an impression of the actual environment and then ask the developers about the current state and functional requirements, which would be needed in the future.

Collecting of Requirements

To get an overview of each institutes requirements, it was helpful to figure out what the technical backgrounds were. CERN's LSA and FESA databases were built without any system of traceability, due to historical reasons. In the past, database versioning was uncommon. Because of this, many databases cannot be setup from scratch. They can be modified only by applying delta changes, while no global script for setup exists. Therefore, the versioning method of choice must be applicable on an existing, as well as on an empty database. In case of the empty one the scripts should setup the whole schemas from scratch, also for different environments like development and production.

Another important point was to analyze, how the existing traceability system for software versions works. One early requirement was, that the method for the database should be integrated into the existing software versioning system. For example, if the software is versioned in SVN, then the database scripts should also be stored in SVN. In this case the developer does not have to use two separate systems, but instead he can rely on a seamless workflow.

An issue, related specifically to the LSA-framework, was the availability of a lot of Java-code for interacting with the database, used to change the configuration of the LSA system. To prevent having to maintain the same SQLs in several locations, it was necessary for the tool to call Java-code for executing some of the tasks. Another point for importing data was, that it must be possible to run database imports scripts after a database migration step.This can be used, among other things, to repeatably update or import configuration data like calibration curves, granting database rights, or for compiling invalid objects. The tool for the database

Table 1: Requirements

Requirements	CERN	GSI
Java-Migration	X	X
Separate Database	X	X
Traceability	X	X
Repeatable Jobs	X	X
Existing Database	X	
Expanding	X	
Open source and free	X	X
Cherry-picking		X
Data import	X	X

migration must also evolve together with the database. If some features are added to the database, the tool will have to support these features too. Hence, the tool should we well supported and used and some kind of plug-in concept would be helpful.

A requirement of GSI was the separation of the schema changes from the ones at CERN. Compared to CERN, GSI is a smaller institute and has fewer developers. The institutes differ in some development aspects. Since features developed in the collaboration might have different priorities for each institute, one GSI requirement was "cherry-picking" of features from CERN development branch at times, when it fits GSI's time schedule. That means GSI must be able to select the changes, their order and migration time. If GSI decides that a schema change is needed at a later point in time or is not needed at all, then the executing order of the migrations might be changed to fit GSI's priorities.

Finally, it was required, that the tool has to be open source and should be free of charge. A summary of the specifications from both institutes is showed in Table 1 requirements.

Selection Process

After collecting the requirements, the next step was to select tools which will fit theoretically. As it had to be an open source tool, there are two popular migration tools on the market - Flyway and Liquibase. Each of these tools fits theoretically to the requirements catalog. For a better exclusion process, an environment was created, which is similar to the production environments of both departments. The goal was to simulate the database landscape of both institutes, to be able to test the chosen migration tools in an environment close to the real-world. First of all, the database was converted to a set of Flyway and Liquibase scripts. After this step, the tools were tested and analyzed with several predefined use cases derived from the requirements catalog.

Implementation

A DDL export of the recent production databases was used as a blueprint to setup the scripts for a test environment, which was important for further evaluation. In our case it was as close as possible to the productive environment and therefore allowed to compare the different products and how they fit with the requirements in near real-life conditions. During the migration the following issues have been identified:

- **Circular dependency:** The two schemas LSA and FESA have internal dependencies into the respective other schema. While both tools will start to migrate and thus execute the scripts, this will cause an error at some point. Both tools works on one specific schema at a time and the error occurs when a dependent object from another schema is not available. This situation occurs only when the two schemas are empty, and when both are setup from scratch.
- **Commons4Oracle:** The library Commons4Oracle [4] - c4o is a collection of database tools and is heavily used by the LSA and FESA database. C4o can be installed in its own schema, or inside an existing schema. At CERN, the c4o database is installed into the LSA and FESA schemas due to access rights. In this case the database migration tool has basically to version two database schemas in one. The c4o schema has an independent product lifecycle and therefore can not be easily versioned together with LSA or FESA.
- **Different database levels:** At both institutes - CERN and GSI - the database landscape is composed of a development, test and production database. Every level of database needs a different migration script and possibly a different set of configuration files.

During implementation those three points mentioned above were to be the most difficult to solve. The circular dependency is relatively simple to avoid, but there is no easy way to solve it. Either the database structure will have to be adapted - this is the best solution - or the script runs two times at the beginning. At first, the script will stop just before the dependency would get executed with an error, after this the other script migrates the database schema with the missing dependency up to the point were the missing dependency is created. Finally, the first database script can finish completely.

The issue where the c4o database schemas is installed into an existing schema has the difficulty, that the tool Flyway orders only on the basis of filenames and has no additional configuration. Liquibase uses an XML-file in which every change is configured. Because Flyway only relies on filenames managing c4o inside another schema is difficult, Liquibase does a better job because this tool can be configured through XML-files. The way how Flyway works is, it looks in the folder for the migrations files and generates a ordered list depending on the filenames. The list will then be compare with a log table in the schema to determine which files Flyway has already executed. If new files that are ot of order are found, like the changes for the c4o, then Flyway might throw an error because it conflicts with the already applied executing order. For this issue the c4o can be installed in a separate schema, but this scenario is not desired. What also might be possible but was not practically tested, is to try two Flyway installations, with slightly different configuration, working on the same schema, one for c4o and one for the product using it.

The last issue is, how to set-up different database with a specific version of the schemas. On the production database only code is executed which has already been tested on the test- and development-database. This problem can be solved in different ways. It was decided to work with database specific maven projects which will include a different set of migrations depending on the corresponding database that has to be setup. For example the development database will include folders, containing scripts also for future planned releases (e.g. R9, R9.1, R10) and the production database only already applied releases (e.g. R9, R9.1). Another possible solution is to use the Liquibase feature to apply scripts only in a specific 'context' and to configure in that way the environment in which a migration should execute.

RESULTS

If both institutes would stick to an agreement, that every change will be compatible to the other tool, in principle every institute could handle the change with the tool of his choice. For example Liquibase users have to set-up the filenames in a way that Flyway recognizes a sensible execution order. This is not a common or easy way but it is possible. But because of the desire for "cherry-picking" and the versioning of two schemas (c4o and the product) in one, both tools reach their limits. Liquibase performs better, it gains flexibility and has more advantages through the XML-configuration approach than Flyway a common versioning in Flyway is not possible because of the difference in execution order when "cherry-picking".

Both tools are suitable for continuous delivery. And help developers to roll-out the software and the database simultaneously with a matching set of features.

Another advantage for Liquibase is, that the tool has an XML parser which can read the XML files and translate into a SQL query for a respective database type. In this case the same XML file can be executed for a test on an Oracle database and in the next step into for example a DB2 database. Since the change is described through XML it is translated into the appropriate database dialect. Liquibase also provides the possibility to specify a rollback script for a migration step.

Flyway's biggest advantage is also its biggest disadvantage. Flyway uses convention over configuration, so has basically no configuration files and generates its list of schema changes based on the filename ordering. Generating sensible version numbers for these schema change scripts in two different institutes, must be considered when defining a versioning scheme for the filenames. For example if two people in each institute generate a new file, and the current version number is 100, it is not possible that both create a version 101. Flyway would not execute both migrations, because the version number needs to be unique. To solve this issue creation date and time of the file were included into the version number, like "V00_01_10102017_1500.sql". This file was created on 10.10.2017 at 3 pm (15:00).

STATUS

At the moment at GSI the accelerator controls group uses Liquibase for database migration. Since GSI was starting with empty schemas from scratch for the LSA and FESA products they did not have to care about existing data, contrary to that CERN is investigating at the moment how to handle the already existing content and looking into baselining its database.

The administrator can execute the migration for the one of the databases. The import process includes the repeatables (like some access rights). Figure 1 shows on the right side the different databases at the accelerator controls department (AccDBU: development, AccDBT: testing, AccDBP: productive).

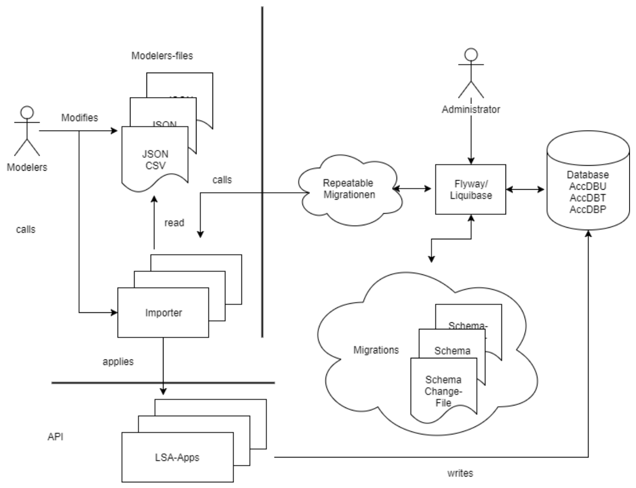

Figure 1: GSI's accelerator controls group migration overview.

SUMMARY/OUTLOOK

The biggest effort is, to introduce the tool in the daily workflow of the developers. At the beginning the usage of a database migration tool is a big step. When introducing a database versioning tool there is no transition, the change is instant. SQL (or DDL and DCL) [5] cannot be simply executed on the database anymore, basically all the changes have to go through the versioning tool. But in long term the tool has more advantage than disadvantage. Like the ability to setup a new database from scratch for testing, to easy setup a database off site, adding in a specific environment data for Unit-tests, or execute an automatic roll out of the already tested changes on the production database. These are just some examples of the advantages that database migration tools can have. Another point is the possibility that Liquibase is expandable through plug-ins. The opportunity to write extensions for Liquibase provides a lot of possibilities. Another topic currently investigated is the possibility to managed a part of the content, mainly more or

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA041

less static configuration, through Liquibase. This is shown on the left side of Figure 1. The information is kept in configuration files that can also be modified by people that are not experts on database migrations and imported through Liquibase ”custom changes”. Which are basically a custom Liquibase extension [6]. Figure 2 shows the future outlook of the versioning from both institutes. This status will be accomplished when the migration of Liquibase is complete.

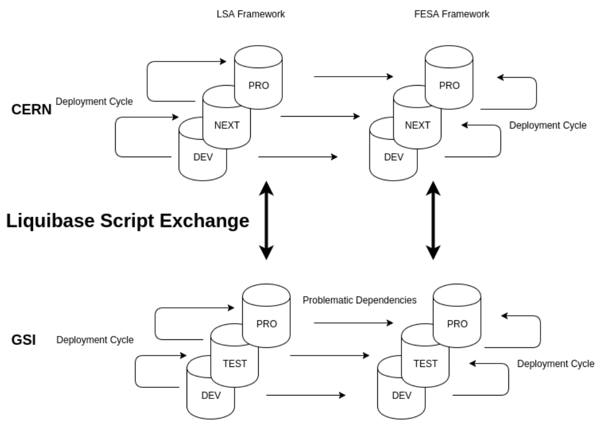

Figure 2: Overview of the future migration from the accelerator controls group from both institutes.

REFERENCES

[1] G. Kruk *et al.*, “LHC Software Architecture (LSA) - Evolution toward LHC Beam Commissioning”, in *Proc. ICALEPCS’07*, Knoxville, TN, USA, paper WOPA03.

[2] Z. Zaharieva, M. Martin Marquez, and M. Peryt, ”Database Foundation for the Configuration Management of the CERN Accelerator Controls Systems”, in *Proc. ICALEPCS’11*, Grenoble, France, paper MOMAU004.

[3] M. Clemencic and H. Degaudenzi, “Migration of the Gaudi and LHCb Software Repositories from CVS to Subversion”, in *Proc. CHEP ’10*, Taipei, Taiwan, LHCb-TALK-2010-158.

[4] L. Burdzanowski and C. Roderick, “Renovation Of The CERN Controls Configuration Service”, in *Proc. ICALEPCS’15*, Melbourne, Australia, paper MOPGF006.

[5] Types of SQL Statements, https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_1001.htm

[6] Liquibase | Database Refactoring | Change customChange, http://www.liquibase.org/documentation/changes/custom_change.html