NEW WAVE OF COMPONENT REUSE WITH SPRING FRAMEWORK AP CASE STUDY*

J. Wozniak, G. Kruk, S. Deghaye, CERN, Geneva, Switzerland

Abstract

The myth of component reuse has always been the "holy grail" of software engineering. The motivation varies from less time, effort and money expenditure to higher system quality and reliability which is especially important in the domain of high energy physics and accelerator controls. Identified as an issue by D. McIlroy in 1968 [1], it has been generally addressed in many ways with various success rates. But only recently with the advent of fresh ideas like the Spring Framework with its powerful yet simple "Inversion of Control" paradigm the solution to the problem has started to be surprisingly uncomplicated. Gathered over years of experience this document explains best practices and lessons learned applied at CERN for the design of the operational software used to control the accelerator complex and focuses on features of the Spring Framework that render the component reuse achievable in practice. It also provides real life use cases of mission-critical control systems developed by the Application Section like the LHC Software Architecture (LSA), the Injector Control Architecture (InCA) or the Software Interlock System (SIS) that have built their own success mostly upon a stack of reusable software components.

SOFTWARE REUSE

As first proposed by Douglas McIlroy from Bell Laboratories in 1968, the idea of software reuse has evolved over time from simple subroutines and libraries through the object oriented programming with reusable classes into the modern software components based on architectures, frameworks and design patterns.

EXISTING FRAMEWORKS

There are many existing software frameworks that address to certain extend the reuse principle. One can easily find those built on CORBA or Microsoft OLE DCOM model. For Java systems the standards have been drawn by SUN with its Java Platform Enterprise Edition (J2EE) specification where the components are called Enterprise Java Beans (EJB). Although based on solid grounds the J2EE EJB framework was criticised for its heaviness and complexity which opened door for development of more lightweight solutions like the Spring Framework.

SPRING APPROACH

The original motivation for the new framework came from problems with existing heavyweight J2EE solutions where the EJB business components were increasingly coupled with the surrounding environment making developers more focused on the 'plumbing' code than on the actual business functionality. The Spring's main aim is to make enterprise Java easier to use and promote good programming practice [2]. It addresses many areas which seem forgotten by other frameworks. Its comprehensive and modular architecture eases the use of any part in isolation yet rendering the global picture internally consistent. Finally it is designed to facilitate the use of plain old java objects (POJOs) containing only business logic with no or little reference to the surrounding framework [2]. Although the coverage of different architectural problems is quite wide in scope this paper will focus on the key points that make this particular framework successful at CERN.

Dependency Injection and Inversion of Controls

As stated before the core of Spring is designed to work with POJO objects which by convention are called **beans** (but not necessarily Java Beans). The important higher layer of abstraction is **a bean factory**. The Spring bean factory enables configured objects to be stored in a container and to be retrieved **by name [2].** It also **manages relationships** between them transparently with the paradigm of Dependency Injection (DI) and Inversion of Control (IoC) with no specific API involved. The DI principle itself refers to supplying an external dependency object to a software component or object. This is typically realized with a setter method or a constructor argument. Dependencies are either injected explicitly where one bean refers to another or with automatic injection where the necessary services are discovered either by name or by type. The second case is particularly interesting when the dependency is optional. The IoC principle is best explained by a "Hollywood Principle" - "don't call us, we will call you". The IoC container injects all the dependencies and the business objects do not have to worry about the instantiation and origin of the dependent services they use.

Lightweight Remoting Support

The remoting support eases the development of remote-enabled services, implemented by simple (Spring) POJOs. The implemented protocols like RMI, HTTP, JAX-RPC or JMS cover most of the needs of a typical 3-tier control system used at CERN. Exposing a service via RMI now takes no more than few lines of the XML code with no need for tedious *rmic* compilation at all. The same applies for other protocols.

*BE/CO/AP - Application Section, Beams Department, CERN, Geneva

Aspect Oriented Programming (AOP)

The framework comes with good support for this popular paradigm in an elegant way. It enables for a cross cutting concerns like caching, security or transaction support to be added with no time.

JDBC Database Support

The complex and low-level details of the JDBC programming are hidden behind the different flavours of the JdbcTemplate class. All error prone code like connection or error handling is now addressed by the framework. The declarative transactional support is offered with the AOP primitives. This feature is a base for data access in many CERN projects.

Integration of Components by File Inclusion

The modularity of the subsystems and particular components is enforced with the inclusion mechanism where parts of the system are declared in XML in different files. They can be assembled together by including them in a global application context file. This mechanism complements the automatic dependency injection and it very important in a system integration process.

Testing

Spring provides a generic and extensible infrastructure for integration and unit tests with support for mock classes. Those are commonly used to replace parts of the system with fake components to test certain functionalities in isolation.

PRINCIPLES OF GOOD DESIGN

Among popular methods that are considered as good practices such as agile programming in a development process or application of the design patterns, there are others which are often forgotten. Those when applied carefully also lead to good results and render software reuse much simpler in nature. They specifically apply well to the AP environment where the control system as a whole is represented by a large number of individual products sometimes forced to work together in a final stage of their lifecycle.

Domain Object Base

Whereas many different projects coexist in the control system it is very important to define a common layer of domain specific classes that is used by all the projects as a common language. If necessary the different systems can now easily talk to each other and the subsystems can be reused since they are based on the same common grounds.

Programming to (narrow) Interfaces

The abstraction should be represented with the use of interfaces that specify what should be done without giving any details of the underlying implementation. Those interfaces should be kept as narrow as possible. Such approach promotes testability and allows for eventual replacement or mocking certain parts of the subsystems without the need to adapt the surrounding components. It enables designing a system that is composed of a set of pluggable components with customizable implementation.

Generic Libraries and Specific Solutions

Care should be put to avoid mixing generic libraries with specific solutions. The generic code should stay well separated which guarantees portability among different projects. The custom adaptations should be placed apart, forming the extension libraries.

Similar Problems Similar Solutions

Better portability of components is achieved when similar problems are addressed in a common way between different projects. This rule seems to be pretty obvious but it is surprisingly hard to apply in practice due to knowledge transfer issues. It leads again to building a common foundation of libraries that address different aspects of the domain in a homogenous way.

HIGH LEVEL CONTROL SOFTWARE AT CERN

The CERN control system as viewed by AP with its 10 million lines of Java code places itself among those which are relatively complex. It is composed of multiple, individual high level systems dealing with particular control and monitoring sub-domains. The following sections will focus on the reuse from the smallest components up to the whole subsystems.

Base Components and Libraries

The accelerator devices are accessed with a stack of middleware services and their properties are logically represented in a structural device/property model. The primitive types (domain objects) used for communication are grouped in two libraries called **accsoft-commons-value** and **japc-value**. Those are used by ***Java API for Parameter Control (JAPC)*** *[5]* that is applied for device access from the high level services. The JAPC library follows the principle of programming to interfaces and has multiple extensions for different flavours of equipment being either real or virtual. This is also a good example of generic library with many specific solutions as described earlier. Additional services like device subscription definition and management, parameter value conversions and buffering or alarms handling are provided in a library called **japc-monitoring** which is used in several high level projects. It introduces a concept of high-level business modules (as Spring beans) that perform some accelerator domain operations being based on values received from the equipment. The way the data is obtained is completely hidden and transparent to the module itself allowing for better separation of concerns. The library constitutes a framework where different modules can be plugged in independently forming completely different applications, yet preserving the common structure of the project. Commonly used solutions like XML processing, process logging and monitoring, data base

access and many others are located in the set of libraries called by convention accsoft-commons. Also all GUI applications are based on common frame components that provide functionalities used by all user applications. This set of components is used later to build high level services and applications.

LHC Software Architecture (LSA)

The LSA system covers all of the most important aspects of accelerator controls: the optics, the parameters space, the settings generation and management, the trim and operational exploitation, hardware exploitation and beam based measurements [3]. One of the main goals of LSA is to provide a clean and generic API to all core functionality, to be used by all operational applications. In principle all LSA libraries are split between the core of the system and the additional extensions plus the generic applications. This project uses almost all foundation libraries mentioned in the previous subsection. The device access is standardized with the JAPC and the GUI applications use the common frame components. Being inherently a 3-tier system with well separated layers, it can also be deployed in 2-tier mode where all the server code is executed on the client side. This functionality is achieved with Spring and interfaces where the GUI client is not aware if it talks to the local or remote controller. The remoting is fully implemented with Spring using the RMI and JMS calls. To avoid unnecessary round-trips to the database the server side caching is introduced also with the previously mentioned AOP Spring services in a fully declarative way.

The LSA project also standardizes the way the database is accessed with the concept of DB finder and persister classes that is also used in other projects like SIS or InCA.

Software Interlock System (SIS)

The SIS system protects the machine through surveillance and by analyzing the state of various key devices and dumping or inhibiting the beam if a potentially dangerous situation occurs [4]. Being a part of the machine protection it plays a vital role in the overall control system. As designed with extensibility in mind the core architecture is based on pluggable interfaces and the main controller is simply a japc-monitoring module. Extension points allow developers to provide their own implementation of components by leveraging the Spring Framework. They cover the areas of system configuration, data transformation, trigger events and exporters of the machine state calculations. Again as in LSA, SIS uses AOP to declare the security schemas or decouple itself from the underlying database. The remote communication to the operational GUIs is done with the RMI and JMS protocols.

Injector Control Architecture (InCA)

The InCA project is an effort to renovate the existing high level control software used in the Proton Synchrotron (PS) complex. Aiming at the homogenisation of the control systems across CERN accelerators, InCA is based on existing components and systems developed for the LHC but also new components required to fulfil the specific operation needs of the PS [6]. The architecture of the server is composed of three main parts: the Control Core, the Acquisition Core and the Configuration Service. Having the reuse principle in mind the control core is based fully on the LSA project and all components that were used previously for LHC or SPS are used here without major changes. The Acquisition Core utilizes the japc-monitoring framework for data acquisition with specialized customizations needed for advanced data calculations. Those will be again reused in the SIS project during its next extension phase. A good example of conceptual reuse is the application of the principle of the MakeRules [3] taken from LSA to calculate high level virtual acquisition parameters. The LSA database infrastructure for parameter relationships as well as the LSA software components that deal with it is completely reused. The Configuration Service itself is also based on LSA components and uses the LSA caching features to reduce the load on the database.

CONCLUSIONS

The art of software development at CERN turned out to be not an easy task given the complexity of the environment and its relative uniqueness. Some inevitable mistakes have been made at the very beginning such as lack of clear domain object base or too much separation between different projects that lead to the code duplication. Those were discovered over time and a unification process has been started to extract the common functionalities and nomenclature. Reuse of software is now a strongly promoted approach that can bring only benefits improving the overall robustness of software control systems used at CERN. Spring as an enterprise Java framework proved itself in practice, helped greatly in the unification and became to be widely used in most of the projects.

REFERENCES

[1] M.D. Mcilroy, “Mass Produced Software Components”, Garmisch, Germany, 7th to 11th October 1968", Scientific Affairs Division, NATO, Brussels, 1969, 138-155.

[2] Rod Johnson et al., Spring Framework 2.5 Documentation, http://www.springsource.org.

[3] G. Kruk et al., “LHC Software Architecture (LSA) – Evolution toward LHC beam commissioning”, ICALEPCS’07, Knoxville, Tennessee, U.S.A.

[4] J. Wozniak et al., “Software Interlocks System”, ICALEPCS’07, Knoxville, Tennessee, U.S.A.

[5] V. Baggiolini et al., “JAPC - the Java API for Parameter Control”, ICALEPCS’2005, Geneva, Switzerland.

[6] S. Deghaye et al., “Cern Proton Synchrotron Complex High-Level Controls Renovation” ICALEPCS’09, Kobe, Japan.