OCTOBER 17, 2019

G1 – INTENDED VS IMPLEMENTED ARCHITECTURE

SPRING

SUBMITTED BY
MONALI CHANDURKAR
JEFFY POOZHITHARA
PINKI PATEL

1 CONTENTS

2 SPRING FRAMEWORK OVERVIEW

Spring frameworks provides infrastructure to developer to develop system in Java. It is very light weighted framework. It is a framework for dependency injection which allows decoupled system. The Spring framework is a layered architecture which consists of several modules. All modules are built on the top of its core container. Its modular architecture enables integration with other frameworks without much hassle.

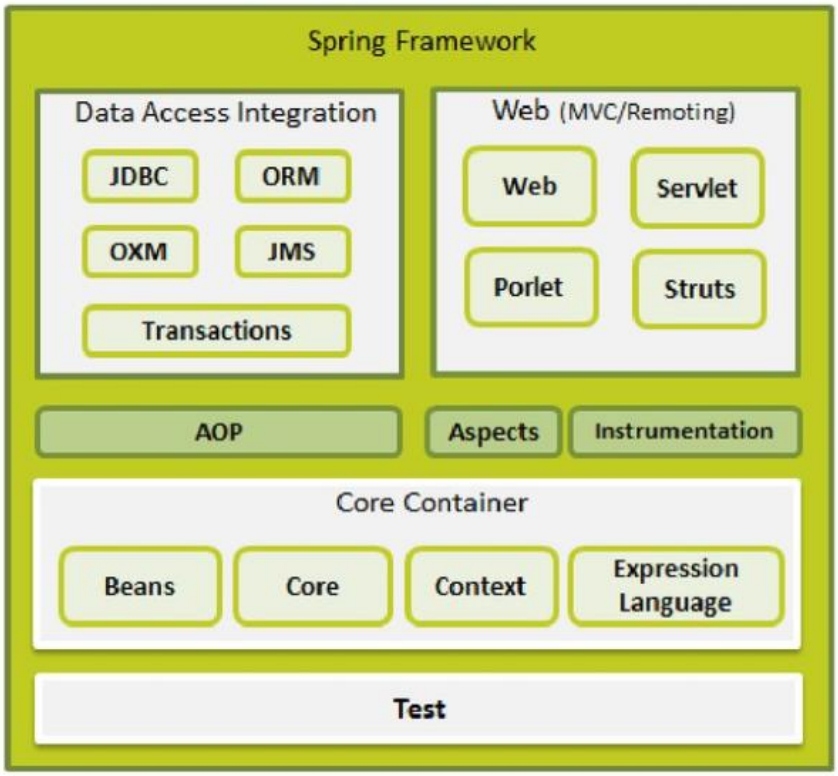


Figure 1 shows the overall as intended architecture of Spring Framework

3 ABSTRACT

In this report we have looked at three modules of spring framework namely spring-jdbc, spring-security and spring-mvc. Comparisons are made between the principal design decisions thought of when creating these modules with the current evolved implementation of these modules. In order to analyze the Implemented architecture, Star UML tool was used to generate the UML Diagrams of the modules. The intended architecture was interpreted based on the official spring documentation.

4 SPRING JDBC

Spring JDBC Framework has covered all the low-level details for using a database starting from opening the connection, prepare and execute the SQL statement, process exceptions, handle

transactions and finally close the connection. So only connection parameters and SQL statement are required from user.

4.1 INTENDED ARCHITECTURE OF SPRING JDBC

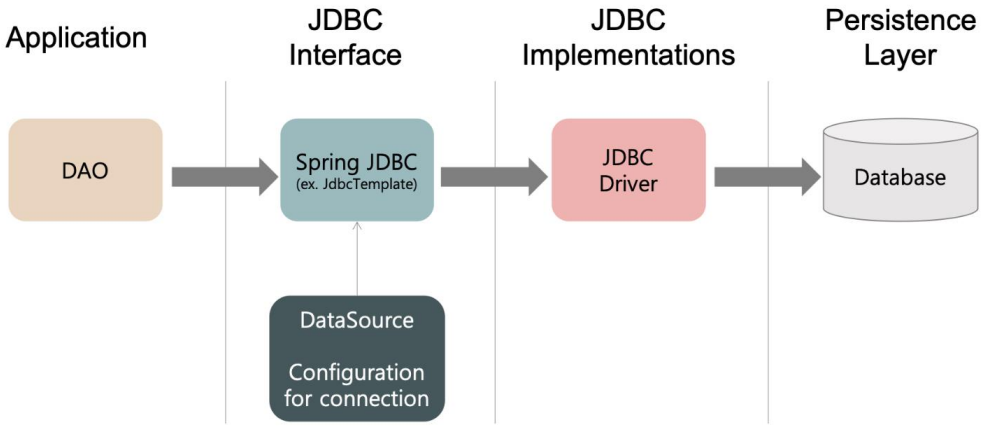


Figure 1.1 shows the architecture diagram of Spring JDBC.

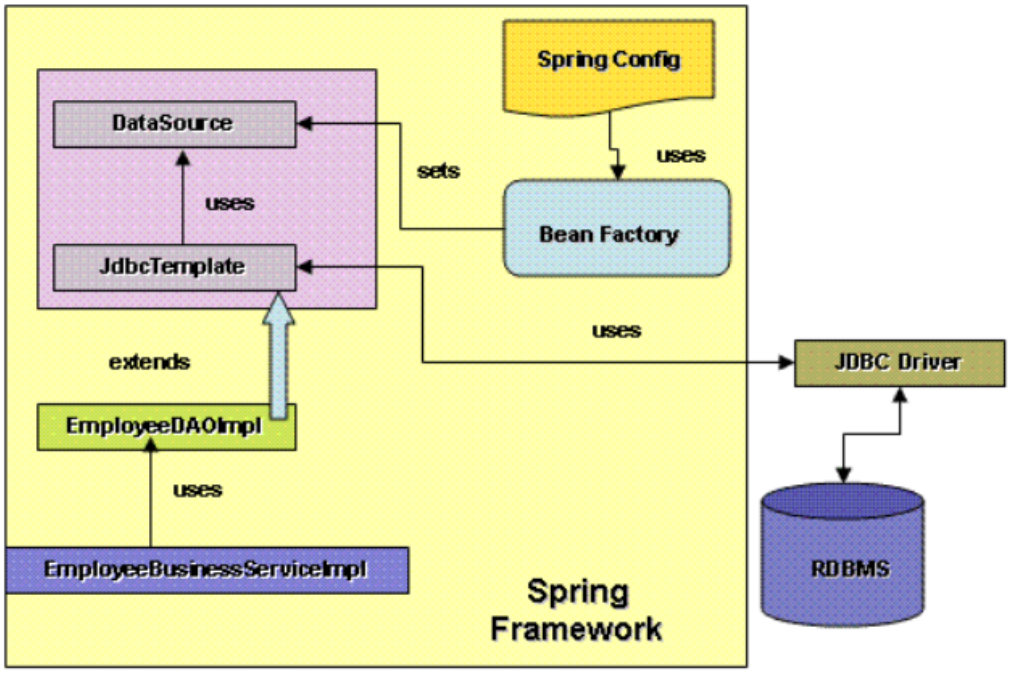


Figure 1.2 shows the major building blocks of the Spring JDBC framework.

4.2 IMPLEMENTED ARCHITECTURE OF SPRING JDBC

Package diagrams shows the arrangement and organization of Spring modules. Below figures, allow us to see both structure and dependencies between subsystems or modules.

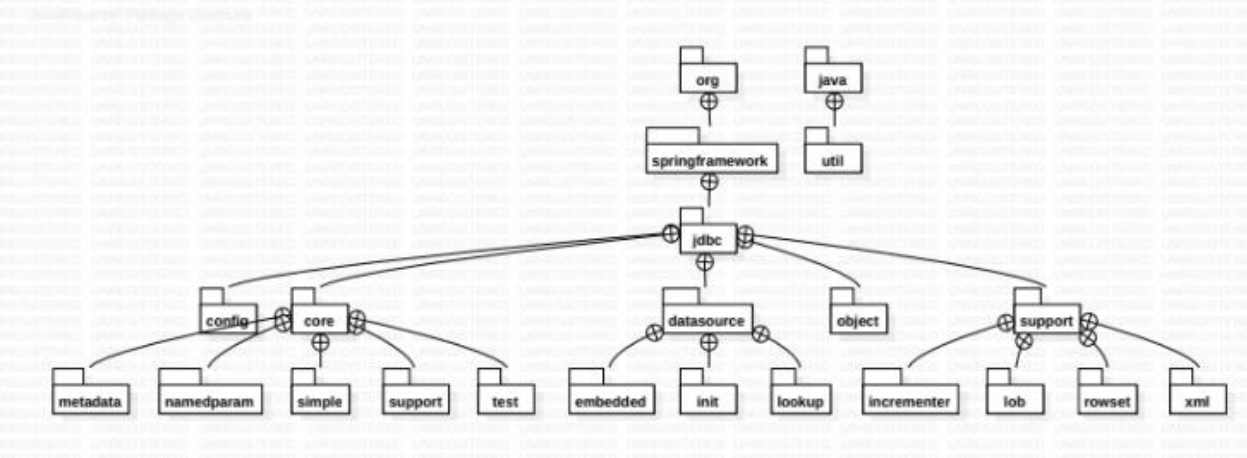


Figure 2.1 illustrates the package diagram for Spring JDBC module.

4.3 DIFFERENCE BETWEEN AS IMPLEMENTED AND AS INTENDED ARCHITECTURE OF SPRING JDBC

We found three major differences in the legacy intended architecture compared to the evolved version of the current jdbc implementation. Most of these deviations were incorporated to allow performance or security enhancements taking various use cases into consideration as more and more people started to use spring-jdbc in their enterprise level systems.

4.3.1 Providing SQL type information for parameters

One of the principal design decisions that Spring jdbc works on is the ease in integration with any Database used as repository system in a software. A key to sticking to this design decision was that Spring will assume the SQL type of the parameters based on the type of parameter passed in.

However, due to inability to create a generic implementation that supports all Database systems in practice, for example, MSSQL, MySQL, Oracle etc, the implementation has been changed to explicitly provide the SQL type to be used when setting parameter values. This is especially necessary to correctly set NULL values. For example, some databases assume NULL as a string rather than considering it NULL type. In order to make this change, the Update and Query methods of the JdbcTemplate were refactored to take an additional parameter in the form of an array which contains the SQL type for each parameter using constant values from the java.sql.Types class.

4.3.2 Object Mapping Fundamentals

A core responsibility of the Spring Data object mapping is to create instances of domain objects and populate them with store-native data structures. This means using constructors or getter setter methods for creation and population respectively. The legacy implementation was to use native java reflection-based instantiation.

In order to avoid the overhead due to Java reflection, Spring Data object creation and property value population processes currently use a factory class generated at runtime by default, which will call the domain classes constructor directly. This has enabled a performance enhancement of 10% performance boost over the legacy reflection-based entity creation. For the domain class to be eligible for such optimization, it needs to adhere to a set of constraints.

A limitation of this implementation level change is that, if a developer doesn't adhere to these specified constraints, Spring Data attempts to use generated property accessors and falls back to reflection-based process.

4.3.3 Handling BLOB and CLOB objects

When storing large chunks of binary or text data, specialized objects called CLOB and BLOB are used for character objects and binary objects respectively. Spring lets you handle these LOB data using the JdbcTemplate and LobHandler interface directly and also when using the higher abstractions provided by RDBMS Objects and the SimpleJdbc classes.

A key design decision of Spring JDBC is to be in sync with SQL standards. SQL standard requires the provision for selecting rows based on an expression that includes a variable list of values. A typical example would be "select * from SOME_TABLE where ID in (5,4,6)". However, JDBC standard does not provide any way of declaring variable number of placeholders. Developers are forced to either have a number of variations with the desired number of placeholders prepared or they have to dynamically generate the SQL string after the number placeholders in consideration is determined.

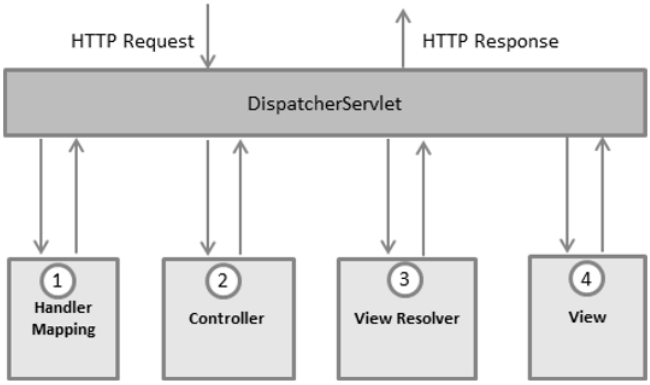
This implementation violates one more architectural design decision related to ease of integration. That is, since the SQL has to be generated instead of being able to use a template that gets interpreted based on the dialect of Database used in backend, such snippets are not agnostic to database used.

5 SPRING WEB MVC

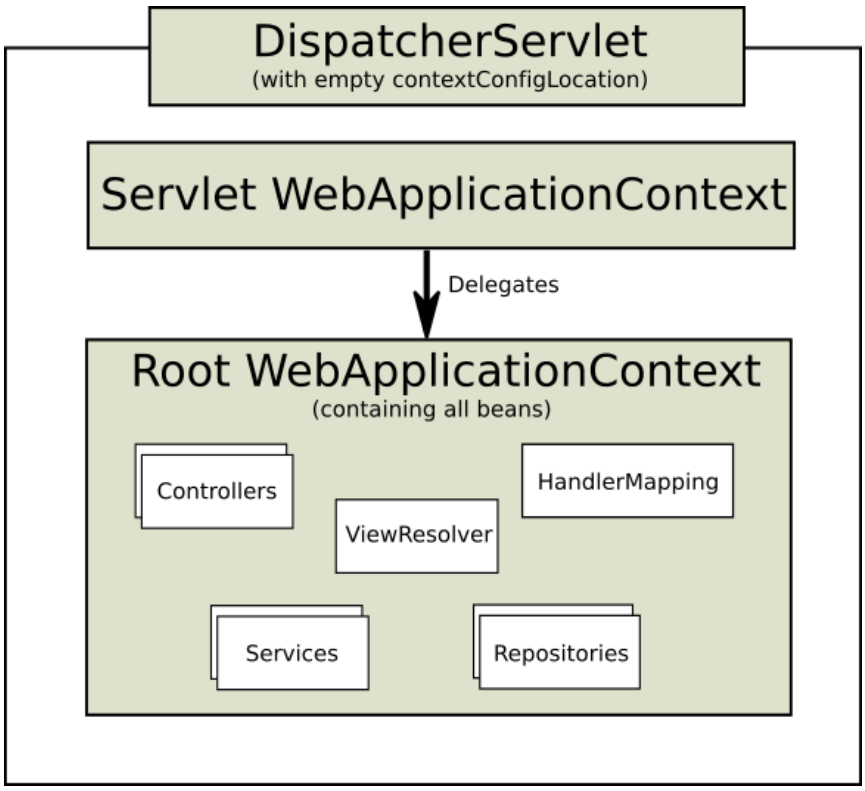
The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

1. The **Model** encapsulates the application data and in general they will consist of POJO.
2. The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
3. The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

5.1 INTENDED ARCHITECTURE OF SPRING MVC



The intended internal structure of Dispatcher Servlet is as follows:



3.2 Architecture of DispatcherServlet of Spring MVC

5.2 IMPLEMENTED ARCHITECTURE OF SPRING MVC

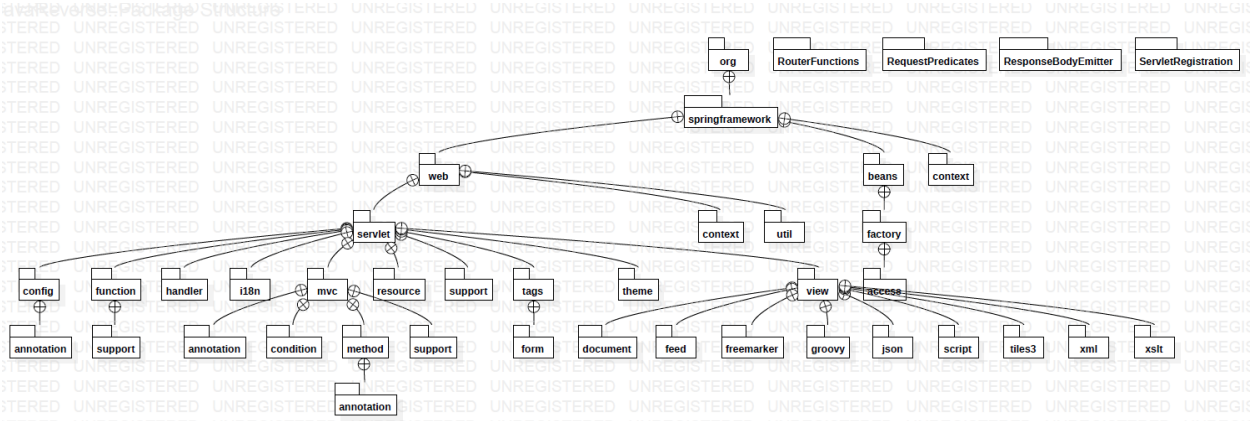


Figure 3.3 illustrates the package diagram for Spring webMVC module.

5.3 COMPARISON OF INTENDED AND IMPLEMENTED STRUCTURE OF DISPATCHER SERVLET

We could not find any deviation from the intended architecture in the implemented architecture of Spring Web MVC module. We had specifically looked into the internal structure of DispatcherServlet and the design decisions discussed on the official documentation of spring. The DispatcherServlet

intercepts the HTTP requests and interacts with HandlerMapping, ViewResolver, View and Controller as per the intended architecture diagram.

6 SPRING SECURITY

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Just like all spring module, Spring Security can be easily extended to meet custom requirements

6.1 INTENDED ARCHITECTURE OF SPRING SECURITY ACCESS INTERCEPTORS

Spring security is a large module in itself. We had looked at the specific code that focused on interceptors and AccessDecisionManger. The intended architecture as per the official documentation of spring for interceptors is as follows:

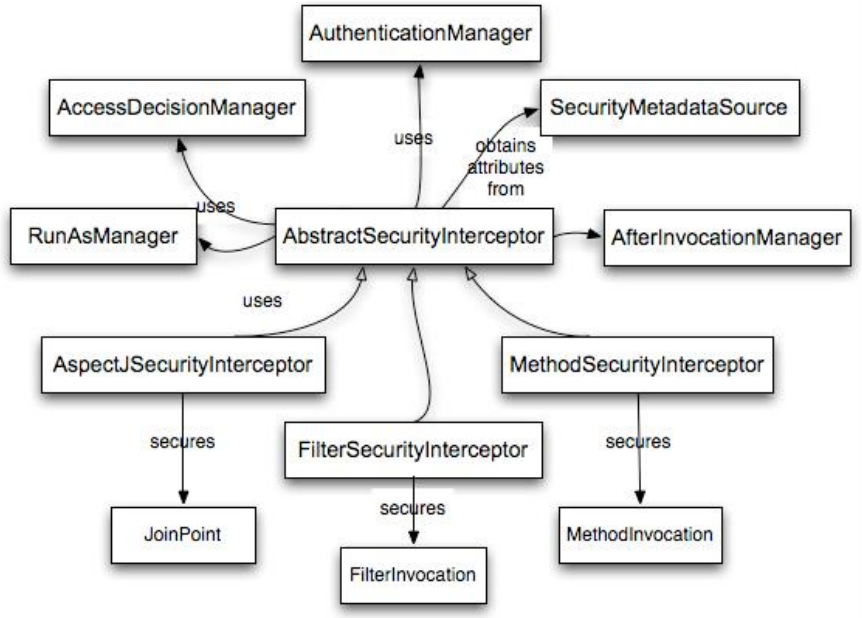


Figure 4.1 Architecture of AbstractSecurityInterceptor of Spring Security

6.2 IMPLEMENTED ARCHITECTURE OF SPRING SECURITY

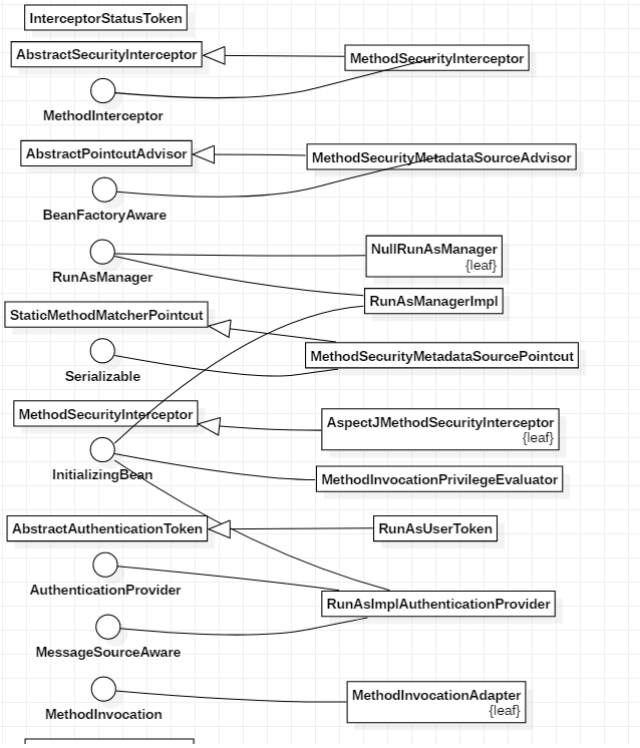


Figure 4.2 implemented architecture of AbstractSecurityInterceptor of Spring Security

6.3 INTENDED ARCHITECTURE OF SPRING SECURITY DECISION MANAGER

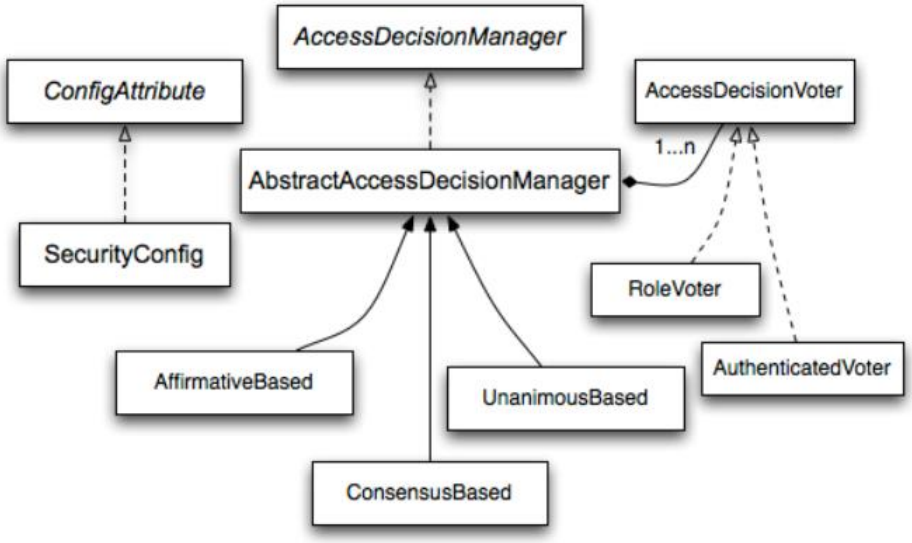


Figure 4.3 Architecture of AccessDecisionManager of Spring Security

6.4 IMPLEMENTED ARCHITECTURE OF SPRING SECURITY DECISION MANAGER

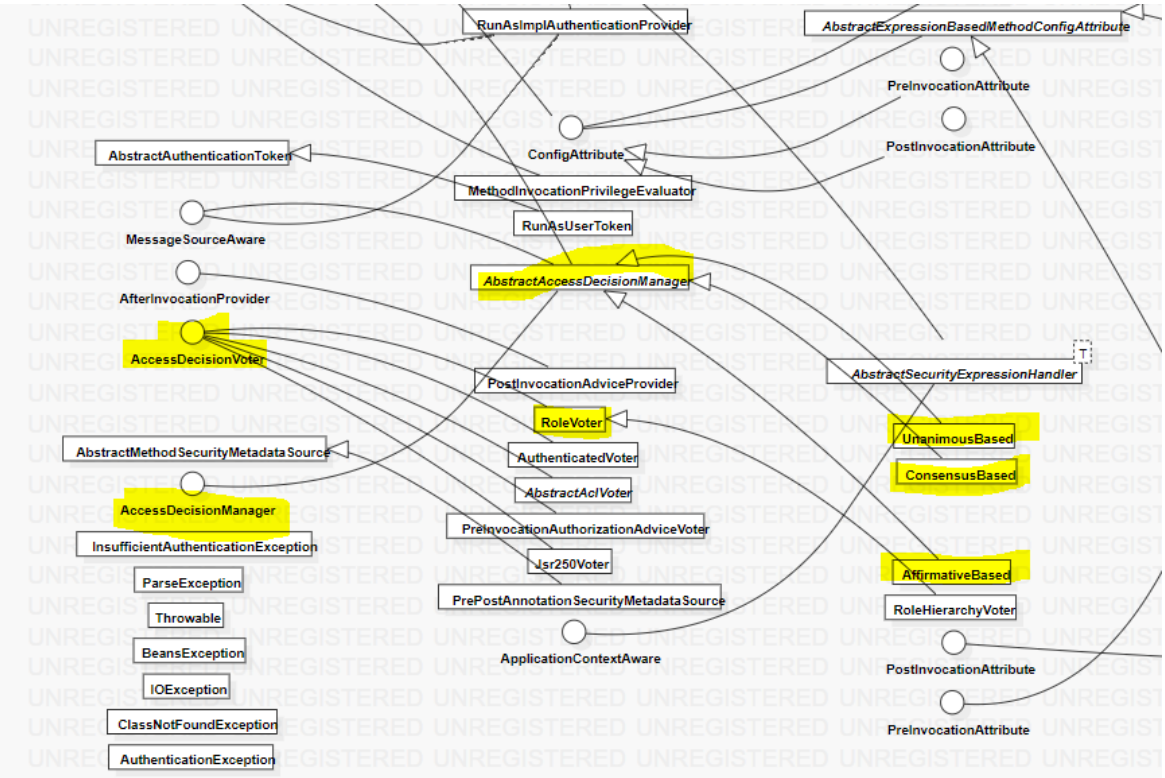


Figure 4.4 Implemented architecture of AccessDecisionManager of Spring Security

6.5 COMPARISON OF INTENDED AND IMPLEMENTED ARCHITECTURE

As evident from the fragments from the implemented architecture diagram, the system is segregated with interceptors, decision managers and voters. The AbstractSecurityInterceptor behaves as a connector between other objects with the different kinds of Interceptors like MethodSecurityInterceptor, FilterSecurityInterceptor, AspectJSecurityInterceptor etc. Similarly, the AbstractAccessDecisionManager behaves as an interface between different DecisionVoters like RoleVoter, AuthenticatedVoter etc. There is no direct communication between Voters and Interceptor except through the AbstractAccessDecisionManager.

7 ATTACHMENTS

The UML diagrams generated for the three modules of spring considered are uploaded as HTML https://drive.google.com/drive/folders/1k_izOcgw4OjjcN9ZGXoFnuik2lEHam8f?usp=sharing. In order to view the full diagram, the index.html file in the respective folder can be opened in a browser. The relevant portion of the generated diagram is already inserted in the respective sections of this report.

8 REVERSE ENGINEERING TOOL USED AND SOURCE CODE

Star UML was used to generate the package structure and Type Hierarchy diagrams of the modules analyzed as part of this assignment.

The source code was cloned from https://github.com/spring-projects/spring-framework

9 REFERENCES

1. https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#overall-architecture
2. https://docs.spring.io/spring-data/jdbc/docs/current/reference/html/#
3. https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/mvc.html
4. https://docs.spring.io/spring/docs/4.3.23.RELEASE/spring-framework-reference/htmlsingle/
5. https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch13s07.html