Reference Documentation

Architecture Principles, Patterns and Best Practices

1.0

Appverse Web is a multi-presentation and multi-integration technology web application framework build over the best-of-breed open source framework stack using it applying well proven arquitectural solutions, patterns and best practices to meet the challenges of design, build, test, integrate and deliver secure, robust, extensible, flexible and maintenaible software. It aims to make web developer's life easier and provide risk contention to deliver first enterprise class innovative JEE projects.

MULTI-PRESENTATION

It allows choosing the frontend that better fits on your project requirements. Due to its layered architecture allow the adoption of future technologies in a cost-effective way, giving out-of-the-box extensibility and mantenability to your project. Designed to provide the adoption of multi-channel easity.

MULTI-INTEGRATION

Support multiples data source types and solve out-of.the-box in a simple way multi data source type scenerario. Again due to its layered architecture allow datasource changes and the adaptation of future datasources, in a simple, elegat way, minimizing the impact of change and providing both extensibility and mantenability.

BASED ON OPEN SOURCE STACK

Based on well proven open source frameworks and third party libraries with an extensive documentation and support over the net thanks to the community, gives a short learning curve to a great number of developers with skills in de facto standard open source java frameworks.

PROVIDES ARCHITECTURAL SOLUTIONS

Use and enforce to use well proven architectural solutions, patterns, and best practices. With 'Don't Reinvent the Wheel' concept in mind, framework architects and developers continuily improves the overall framework simplifying their usage at the same time that the framework growns.

INNOVATIVE, MODULAR, SECURE, ROBUST AND PERFORMANCE ORIENTED

Use only needed framework modules based on innovatives technologies. Provides proactive and reactive security out-of-the-box and patterned solutions to several common attacks. Performarce matters, so a wide set of solutions to improve the performance is given.

STACK COMPLETELY BASED ON JAVA

Whole stack based on proven enterprise Java technologies, minimizing the need of multi-disciplinary developers.

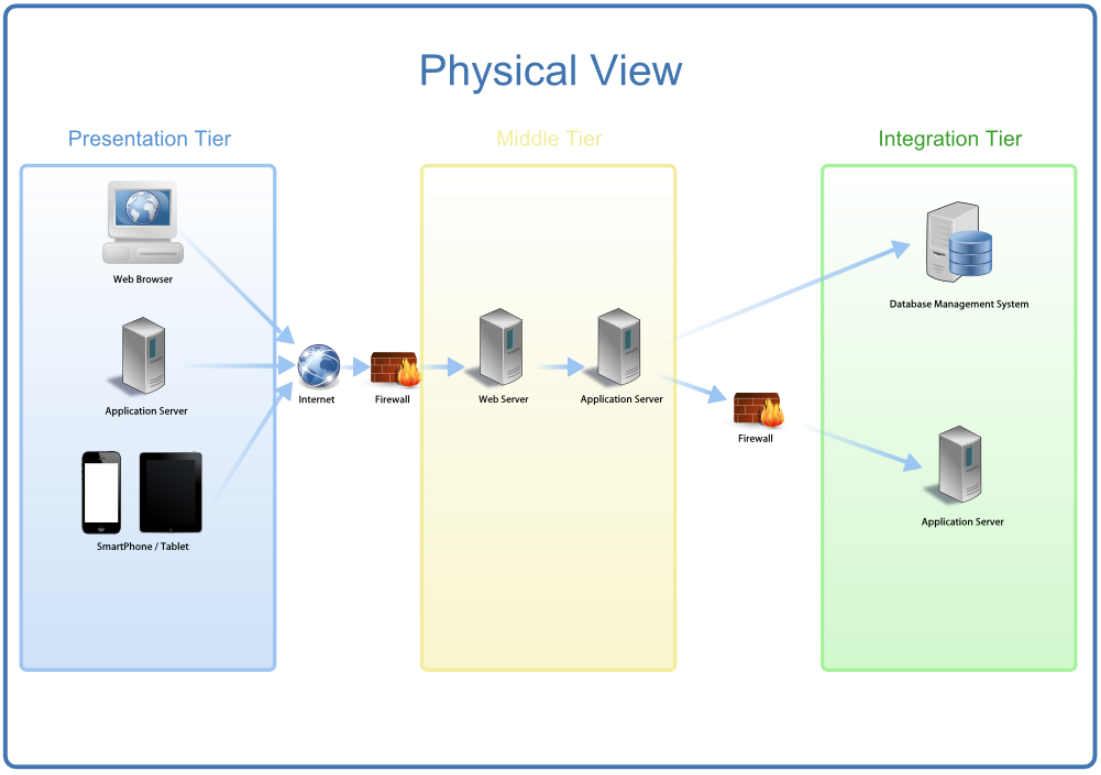
2.1. Architectures and Tiers

Appverse Web is built using a multi-tier architecture pattern, either using a three-tier architecture or being part of a more complex N-tier architecture

2.1.1. Multitier Architecture[1]

N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific tier, instead of reworking the entire application.
Three-tier architectures typically comprise a Presentation Tier, a Middle Tier and a Integration Tier.

2.1.2. Three Tier Architecture



Presentation Tier could be implemented as a set of web pages or a RIA application deployed in a web browser, as a mobile application running in a smartphone/ tablet device or as another enterprise application deployed external application server. Future new implementations could be considered. Even the Presentation Tier could combine a set of these implementations in a multi-channel system.

Middle Tier is usually implemented as enterprise application deployed in an application server that often is behind a web server within an intranet protected by a firewall.

Integration Tier could be a database deployed at a database management system or in another enterprise application deployed at intranet application server, probably behind a firewall. As Pres-

1 http://en.wikipedia.org/wiki/Multitier_architecture

entation Tier future implementations could be considered and a set of these implementations cold be used.

2.1.3. N-Tier Architecture

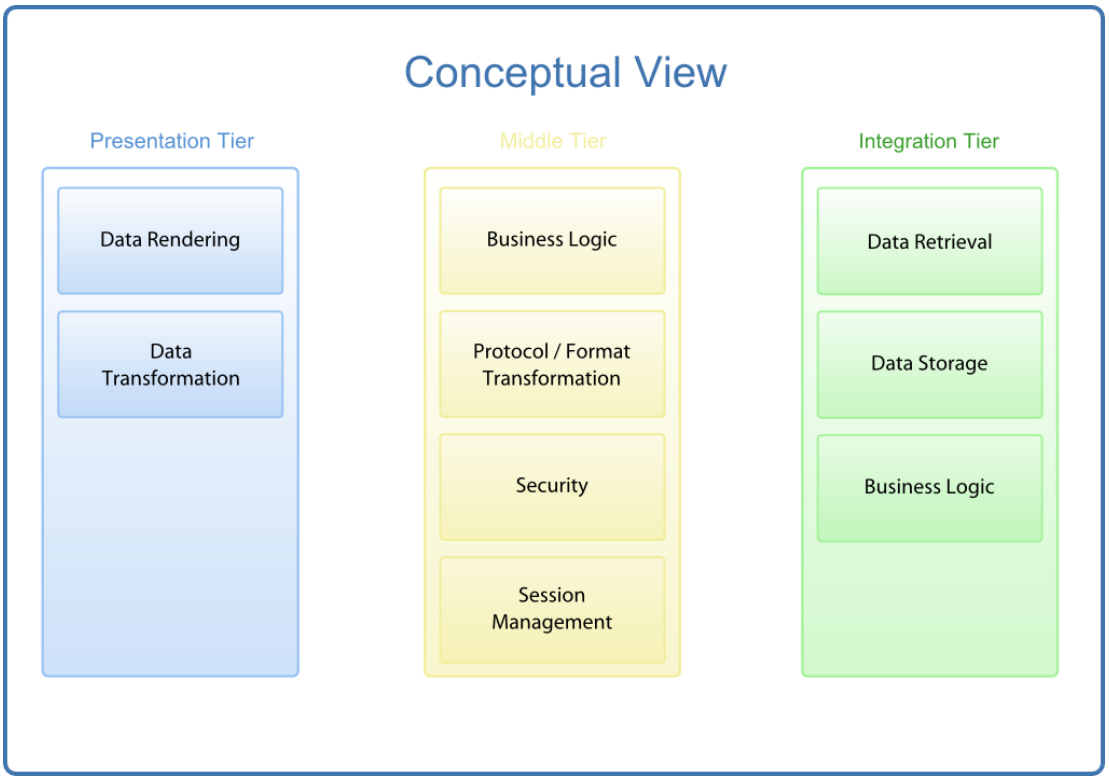
In a more complex N-Tier architecture more tiers could be placed before the Presentation Tier and behind the Integration Tier, especially if these Tiers are Enterprise Applications.

2.2. Tiers, Concerns and Flavours

Each tier is responsible of a set of concerns or responsibilities. The way the tier resolves its concerns is the flavour of the tier, Presentation and Integration Tiers could have several flavours, Business Tier its unflavoured by itself but have adapters to Presentation and Integration Tiers flavours.

Separation of Concerns[2]

> Separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A program that embodies SoC well is called a modular program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well defined interface.



2.2.1. Presentation Tier

Presentation Tier is responsible of data rendering, providing a visual representation of the data feed by the middle tier into presentation flavour specifics and reacts to the user interaction calling to middle tier in order to perform transactions and build a response.

A second flavour could be considered, in this scenario the Presentation Tier performs data transformation action for the data fed by the Middle Tier. This is the kind of structure followed by a N-Tier architecture. In this scenario Presentation Tier could no be the first layer at the architecture but could have more tiers before it.

---

2 http://en.wikipedia.org/wiki/Separation_of_concerns

The flavours of the Presentation are mainly related to the kind of device where this tier are deployed and to the technology use to render the presentation of the data.

2.2.2. Middle Tier

Middle Tier is responsible, in a high level, for business logic, protocol/format transformation, security including authentication and authorization and session management for stateful implementations. It Receives calls from the Presentation Tier and delegates into the Integration Tier to retrieve and save data or perform business logic.

As can be seen at the Conceptual View diagram, the business logic could be placed at Middle Tier, which is the more usual configuration taking part of a Three Tier Architecture and use the Integration Tier mainly for data retrieval and storage concerns.

There is a alternative configuration focused to a N-Tier Architecture where the Business logic is situated at the Integration Tier and Middle Tier acts performing Protocol/Format Transformation

In both scenarios Middle Tier could be responsible of security, session management and other concerns as caching or transactionality. There is no obligation of implement each of these responsibilities. In N-Tier scenario some of the resposabilities could be delegated to other Tiers.

2.2.3. Integration Tier

Integration Tier is responsible for data storage and data retrieval or the business logic concerns.

In a Three Tier Architecture, the most common scenario, the resposibility for Integration Tier is data retrieval and storage.

In N-Tier Architecture, Integration Tier is used to perform the business logic. In this scenario the Integration Layer could no be the last layer in the architecture but have more layers involved behind.

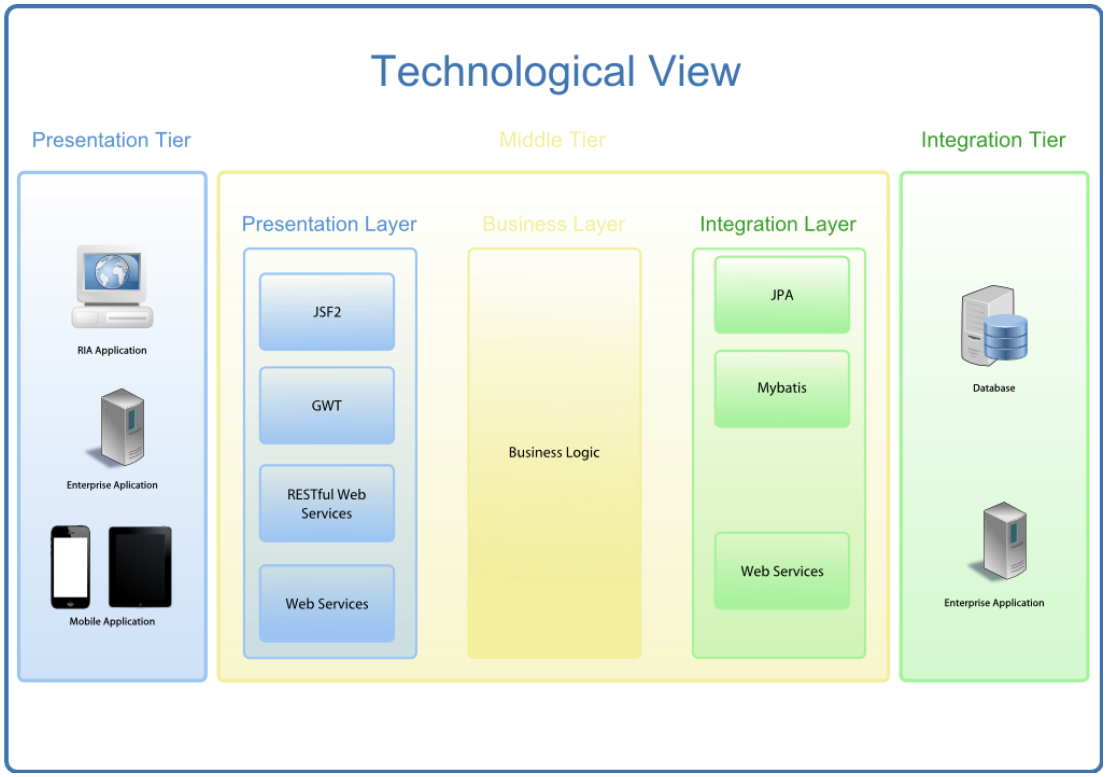
As happens with Presentation Tier it could be flavoured. The flavours of the Integration Tier are related with the way of the information is saved and retrieved and the type of the systems that are in charge of these operations.

2.3. Layers and Technologies

In order to fullfil its purpose, Middle Tier is layered and seveveral technologies are used. Middle Tier could be broken down in two or three layers; Presentation Layer, Business Layer, usually not used in a N-Tier scenario, and Integration Layer.

Tiers and Layers[3]

> Logical layers are merely a way of organizing your code. Typical layers include Presentation, Business and Integration. When we're talking about layers, we're only talking about logical organization of code. In no way is it implied that these layers might run on different computers or in different processes on a single computer or even in a single process on a single computer. All we are doing is discussing a way of organizing a code into a set of layers defined by specific function.
>
> Physical tiers however, are only about where the code runs. Specifically, tiers are places where layers are deployed and where layers run. In other words, tiers are the physical deployment of layers.



3 http://stackoverflow.com/questions/120438/whats-the-difference-between-layers-and-tiers

2.3.1. Presentation Layer

Presentation Layer is responsible of isolate the rest of layers of the Middle Tier of the specifics of the connection between the Middle Tier and the Presentation Tier.

If a flavour of Integration Tier is changed or a new flavour is added to it just the Presentation Layer of the Middle Tier will be affected.

2.3.2. Presentation Layer Flavours

Middle Tier can use any the flavours of Presentation Layer, or even a set of them in response of the Presentation Tier flavours. More flavours can be added when new needs will be detected. This capability opens the possibility of build multichannel projects.

2.3.2.1. GWT

When Middle Tier is exposed to a Web Browser in Presentation Tier, a GWT Flavoured Presentation Layer is recommended. GWT allows writing Rich Internet Applications (RIA) including widgets wth intensive interactions completely in java.

Sencha GXT[4] is the recommended application framework in order to build a GWT Presentation Tier and GWT Flavoured Presentation Layer of MiddleTier deliverables.

2.3.2.2. Java Server Faces 2

When Middle Tier is exposed to a Web Browser in Presentation Tier, a Java Server Faces 2 (JSF2) Flavoured Presentation Layer is recommended. Java Server Faces 2 allows writing a Rich Internet Applications (RIA) as classical server-response applications in a simple and cost effective way

JBoss RichFaces[5] is the recommended application framework in order to build a JSF2 Presentation Tier and JSF2 Flavoured Presentation Layer of Middle Tier deliverables.

2.3.2.3. When to use GWT and JSF2 at Presentation Layer

At Presentation Layer our recommendation to build a user interface into a Presentation Tier are both GWT and JSF2, so when use each? What about other presentation frameworks?

- GWT is more adequate for web applications containing RIA features that require intensive client-side interactions. Since your server-side is just a service layer you can be fully stateless on the server side.
- JSF is more adequate for applications that are better suited for component-oriented stuff. Allows both write RIA applications and classic ones giving structure and simplicity.
- We are open to study other presentation frameworks. GWT and JSF2 covers a significant set of needs present at current innovative web based application landscape fulfilling this requeriment with a reduced set of developer skills. A special attention is being taken following new iniciatives in presentation frameworks including no web based ones in order to get a rich-featured yet simple solution.

2.3.2.4. Web Services

If Middle Tier is exposed to another Enterprise Application at Presentation Tier and requeriments as interoperativity or security are key factors a Web Services Flavored Presentation is recommended.

---

4 http://www.sencha.com/products/gxt
5 http://www.jboss.org/richfaces

As Web Services we mean SOAP Web Services. SOAP Web Services are the facto standard protocol to ensure the maximum Business to Business interoperabiliity. The Web Services flavoured Presentation Layer takes the server role in the web services client-server pair

Apache Axis2[6] is the recommended engine in order to render SOAP Web Services.

2.3.2.4.1. RESTful web services

When a light communition between Middle and Presentation Tiers is derired in order to get improvements at requierements like performance or Middle Tier is exposed to a Smartphone, Tablet Application at PresentationTier a a RESTful flavoured Presentation Layer is recommended. A special mention has to be taken talking about RIA Applications as RESTful web services could be used to transport the data between and to the browser at the Presentation Tier.

For RESTful web services we recommend use JavaScript Object Notation (JSON[7]), a lightweight data-interchange notation. A Human friendly format and easy to generate and parse by machines.

Jackson[8]. Is the recommended library in order to convert java bean from and to JSON notation.

2.3.2.4.2. When to use SOAP and RESTFul Web Services at Presentation Layer

- Use SOAP Web Services when interoperability is the key or some of the services built over the SOAP Web Services are needed. Usually here the Presentation Tier is deployed as an enterprise application in a application server.
- Use RESTFul Web Services when the performance of the marshaling and unmarshaling is the key factor. Usually here the Presentation Tier is deployed as a mobile application in a smartphone or tablet or a RIA application in a web browser.

2.3.3. Business Layer

When Three Tier Application Architecture is used, the main concern of Business Layer is to implement the business logic. It uses the Integration Layer to retrieve and save data from and to each of the Integration Tier Flavours and it is used by the Presentation Layer to feed data to every Presentation Tier Flavours.

The main concept of the Business Layer is that it's agnostic of the specifics of using the flavours of Integration Tier through the Integration Layer and it's also agnostic in the same way of the specifics of using the flavours of the Presentation Tier through the Presentation Layer. That is, contains no details of the technologies used to connect Middle Tier with Presentation neither Integration Tier.

This kind of decupling allows easily adding or changing to new Presentation or Integration Layer flavours for connecting to new Presentation or Integration Tier Flavours. That is the kind of extensibility that enables to build multichannel applications or ensure that a technological migration will be feasible and cost-effective.

When an N-Tier Application Architecture is used, the main purpose of the Middle Tier is data protocol and format transformation. In this case the Business Layer and is dropped and Presentation Layer to the Integration Layer are diretly connected.

---

6 http://axis.apache.org/axis2/java/core/
7 http://www.json.org/
8 http://wiki.fasterxml.com/JacksonHome

2.3.4. Integration Layer

Integration Layer is responsible of isolate the rest of layers of the Middle Tier of the specifics of the connection between the Middle Tier and the Integration Tier.

If a flavour of Integration Tier is changed or a new flavour is added to it just the Integration Layer of the Middle Tier will be affected.

2.3.4.1. Integration Layer Flavours

Middle Tier can use any flavours of Integration Layer, or even a set of them in response of the Integration Tier flavours. More flavours can be added when new needs are detected.

2.3.4.1.1. JPA[9]

When Middle Tier interacts with a Relational Database Management System deployed at Integration Tier, the use of JPA flavored Integration Layer is recommended.

JPA is an Object Relational Mapping specification to ease CRUD operations and query building with JPQL Language. The way this mapping is governed is highly customizable allowing finding the proper configuration to get appropriate non functional aspects such as performance.

JPA is part of the JEE specification, concretely JPA2, the current release at times of writing, is defined by the JSR 317. JPA specification allows choosing of the concrete implementation.

EclipseLink[10], the Reference Implementation of JPA2, is the recommended implementation of JPA2 specification

2.3.4.1.2. MyBatis[11]

If the Middle Tier interacts with a Relational Database Management System deployed at Integration Tier, the use of MyBatis flavored Integration Layer is recommended.

MyBatis is a SQLMapper formerly known as iBatis that can perform a fine grained SQL queries in a declarative way. MyBatis is specially indicated to perform searches into a database and for queries where the performance is especially critical.

2.3.4.1.3. When to use JPA, Mybatis and plain JDBC at Integration Layer

At Integration Layer our recommendation to interact to a Relational Database Management System deployed at Integration Tier are both JPA and Mybatis, so when use each? How about plain JDBC?

- Use JPA for maintenance methods (CRUD[12]) and simple, non-performance critical queries. Consideer the use of JPA implementation performance hints to get a performance improvement.
- Use Mybatis for dynamic, complex or performance critical queries. Are you creating a search function depending of open subset of a set of fields? Then use Mybatis.
- About plain JDBC, it's easy: Never.

---

9 http://en.wikipedia.org/wiki/Java_Persistence_API
10 http://www.eclipse.org/eclipselink/
11 http://www.mybatis.org/core/
12 http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

2.3.4.1.4. Web Services

When Middle Tier connects to another Enterprise Application at Integration Tier, a Web Services flavored Presentation Layer is recommended.

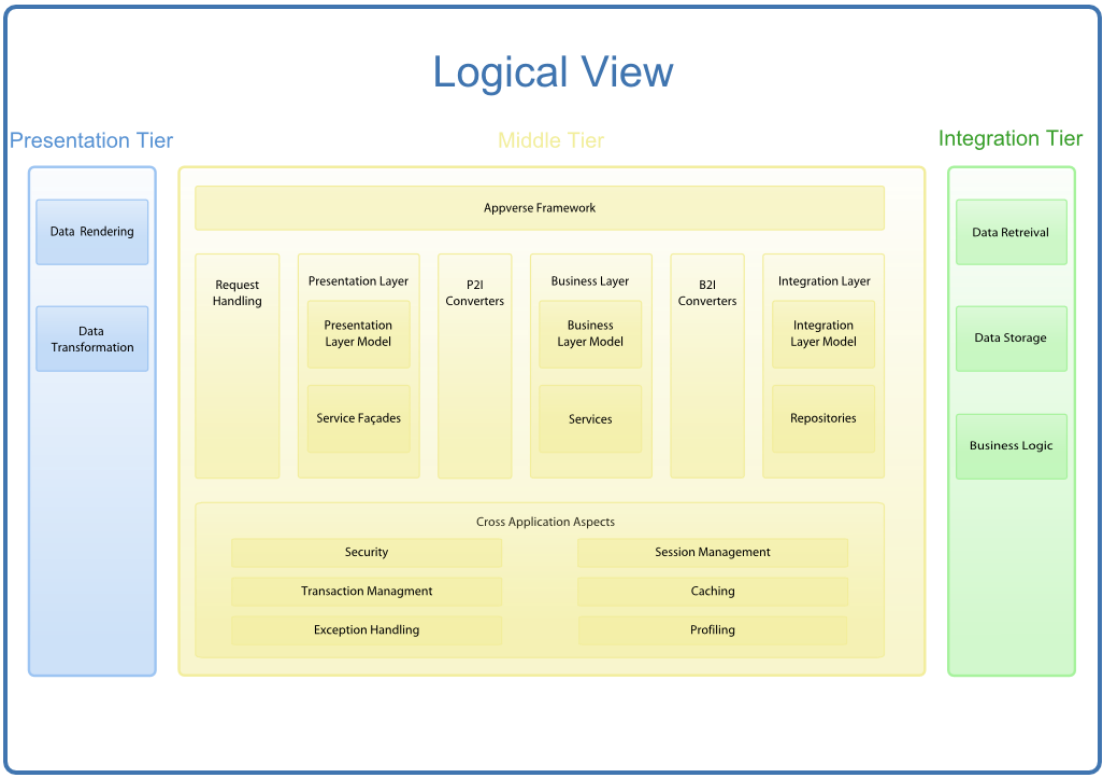
As Web Services we mean SOAP Web Services. SOAP Web Services are the facto standard protocol to ensure the maximum Business to Business interoperabiliity. The Web Services flavoured Presentation Layer takes the server role in the web services client-server pair

Apache Axis2[13] is the recommended engine in order to render SOAP Web Services.

13 http://axis.apache.org/axis2/java/core/

3.1. Overview

The following diagram illustrates the logical building blocks of the architecture:



3.1.1. Presentation Tier

Presentation Tier is responsible of data rendering, and data transformation, this was explained on section 2.2.1 on this document.

3.1.2. Architecture Components

The architecture adheres to the Service Façade Pattern - the approach is explained in more detail on section 3.2 on this document. Components will be broken down into Service Façade, Service and Repository services keeping a strict separation of concerns between layers to ensure ease of maintenance and reusability – a more detailed explanation of the principles applied to these layers within the architecture are detailed on section 3.2.1 on this document

3.1.3. Domain Objects

Domain objects represent the data/values that the architecture will operate on. There are 3 models contemplated in the architecture, this is explained in more detail on section 3.2.2 on this document

3.1.4. Inversion of Control

An IoC container will control the architecture. The container will inject the relationship between architecture components and implementations to specific architecture interfaces at runtime. The architecture will leverage all the functions of the IoC container – in this case, the Spring Framework – to keep architecture and application code to a minimum and avoid functional overlap. The following features will be extensively used by the architecture;

- Injection of dependencies/implementation

- Declarative security
- Declarative transactions
- Control of data source connections
- Look-up and resolution of components and resources
- Aspect Oriented Programming for exception handling, profiling, etc.

The rationale and benefits of using an IoC container are detailed on section 3.3 on this document.

3.1.5. Integration Tier

Integration Tier is responsible for data storage and data retrieval or the business logic concerns, this was explained on section 2.2.3 on this document.

3.2. Service Façade Pattern

The architecture for Appverse Web follows the Service Façade Pattern.

Service Façades, Services and Repositories are coupled to and implement specific interfaces. Service façades represent the presentation tier facing interfaces, services are the discreet functions that can be called from service façades to compose a specific functionality and repositories represent the finer grained low-level access to data and services of provide from the integration tier. These interfaces vary in granularity as we work down the stack from service façades to repositories; coarse grained at the service façade level to fine grained at the repository level.

This design principle allows for interfaces to be used throughout to loosely couple the different building blocks of the application implementation. In specific, service façade logic will be implemented and coupled to a specific service interface and the actual service logic provided via configuration at runtime – the same principle applying to repositories. Coupling refers to a connection or relationship between two things; the measure of coupling is comparable to a level of dependency. This principle advocates the creation of a specific type of relationship between two interfaces within and outside of service boundaries; the emphasis is on reducing ("loosening") dependencies between the service contract (it's interface), its implementation, and service consumers.

The principle of Service Loose Coupling promotes the independent design and evolution of a service's logic and implementation while still guaranteeing baseline interoperability with consumers that have come to rely on the service's interface. There are numerous types of coupling involved in the design of a service, each of which can impact the content and granularity of its contract.

Service façades constitutes the architectural public interface and entry point and provides the nature and quantity of content that will be published to the presentation tier. Services and repositories are used internally and are not published to the presentation tier or directly accessible; as a design choice, services and repositories may call other services or repositories in order to avoid code duplication and maintain separation of concerns but service façades may not call other service façades – each service façade should compose the functionality required independently from other service façades. Service façades are not technically complex but very important when deciding the granularity and quantity of data that will be published to the consuming presentation tier; service façades provide the technical point cut for transactions.

3.2.1. Layers and Separation of Concerns

As a design choice, the following principles should be adhered to for the design and implementation of services:

3.2.1.1. Service Façades

Service Façades should provide coarse-grained functions to perform the actions required by a specific group of functional requirements. It should provide the services needed to perform related functional tasks. For example, search functions throughout the application would be provided for by a single Façade Service that groups together all search functionality. Similarly, if the application needs static data in order to populate selection lists, a single Façade Service should provide access to all the static data that the application requires.

The following rules should be adhered to when designing and implementing Service Façades:

- Should compose their functionality by calling one or more Services and not implement the functionality itself.
- Can use both Presentation Object Model and Business Object Model domain objects.

- Should not invoke other Façade Services due to separation of concerns and transactional demarcation.
- Should reference only the Service interface and not the implementation. Service interfaces should be used throughout the façade - actual implementations are injected at runtime by the IoC container.
- Should not handle transactions – the architecture will use declarative transactions managed by the IoC container.
- From the transactional perspective, each function/method is considered to execute atomically irrespective of how many underlying Services (and Repositories) are involved.

3.2.1.2. Services

These services provide the granular building blocks and business logic of the application. The functions provided by a Service should be sufficiently granular to allow for composition and use by many Service Façades; the granularity required depends on each service and the functional richness of the application – special consideration should be given to the reusability of the service during the detail design phase. For example, a service providing the logic for search functionality should contemplate the individual elements that may be used during a search – search by id, by name, by value, etc. – a Service Façade will use the different functions as needed depending on the search criteria provided by the consuming client.

The following rules should be adhered to when designing and implementing Services:

- Should implement the business logic to execute a specific granular functional requirement.
- For requests, it should perform get value operations on the Business Object Model, execute rules, calculations and/or formatting as required and perform set value operations on the Integration Object Model required to obtain the required data from the underlying Repositories.
- To compose a response, it should perform get value operations on the Integration Object Model that contains the data returned by the underlying Repositories, execute rules, calculations and/or formatting as required and perform set value operations on the Business Object Model in order to return the result to the Service Façade invoking the function.
- Can use both Business Object Model and Integration Object Model domain objects.
- Must never invoke a Service Façade but may invoke other Services in order to avoid code duplication. However, special care should be taken during the detailed design phase to avoid dependencies between Services
- Should reference only the Repository interface and not the implementation. Repository interfaces should be used throughout the Service – actual implementations are injected at runtime by the IoC container.
- Should compose their data retrieval functions by calling one or more Repositories and not implement the functionality itself. Data must never be retrieved directly – Services should always use a Repository in order to access data providers. No data persistence code should be found within a Service.
- Should not handle transactions – the architecture will use declarative transactions managed by the IoC container.

3.2.1.3. Repositories

Repositories provide very granular data operation functions for a specific Integration Model entity such as insert, delete, update, select, send, receive, etc. These represent the fine-grained operations that are performed with a specific type of data. As a design choice, the Appverse Web architecture contemplates two distinct groups of Repositories; Repoitories that handle relational data using an Object Relational Mapper (ORM) or a SQL Mapper and Repositories that invoke services (web services, queues, etc.) from the integration tier. For example, a Repository would provide a specific search (select) function for the entity type it manages and return a list of data matching the search criteria (based on contains, like, equals, etc. conditions as needed).

The following rules should be adhered to when designing and implementing Repositories:

- Should implement the data operation logic to execute a specific and atomic operation on data.
- Should only use objects pertaining to the Integration Model Object domain.
- Must never invoke a Façade Service or Business Service but may invoke other Repository Services in order to avoid code duplication. However, special care should be taken during the detailed design phase to avoid dependencies between Repository Services wherever possible.
- Relational Repository Services should always use the architecture defined ORM or SQLMap frameworks provided for database operations – JDBC code should never be present within a Repository implementation.
- Repository Services that encapsulate calls to services of the Integration Tier should follow the same behaviour as Relational Repositories Services albeit that implementation details will depend on the underlying services being invoked

3.2.2. Domain Object Models

The service façade pattern adopted for the Appverse Web architecture contemplates 3 data models – one used by service façades that will be part of the middle tier interface to exchange data with presentation tier, other used by and service façades and services that will represent the business domain model, and a third used by services and repositories that will be part of the middle tier interface to exchange data with integration tier.

3.2.2.1. Presentation Object Model

Representing the objects that are used by the presentation tier. From an interface design perspective, this model may be used by function/method declarations of service façade interfaces.

3.2.2.2. Business Object Model

Representing the business object model domain. From an interface design perspective, this model may be used by function/method declarations of services interfaces.

3.2.2.3. Integration Object Model

Representing data model in relational databases and services of the integration tier. From an interface design perspective, this model is used exclusively by function/method declarations of repositories interfaces.

The individual service façades, services and repositories implementations are bound together and composed using an IoC container.

3.3. Inversion of Control (IoC) Container

In traditional programming the flow is controlled by a central piece of code. Using Inversion of Control this central control as a design principle is left behind. Although the caller will eventually get its answer, how and when is out of control of the caller. It is the callee who decides to answer how and when. Inversion of Control as a design guideline serves the following purposes:

- There is a decoupling of the execution of a certain task from implementation.
- Every component can focus on what it is designed for based on its interface.
- Every component does not make assumptions about what other components do or should do – there are strict contracts in place that adhere to a specific interface.
- Replacing component implementations will have no side effect on other components.

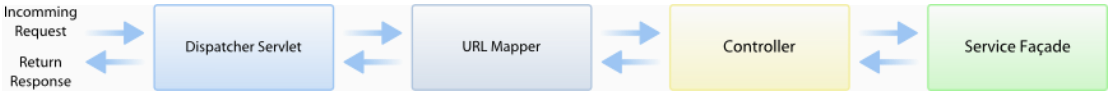
In Appverse Web, the Spring Framework is the chosen IoC container and forms the basis for the composition of the application from the diverse services it provides; service façades, services and repositories. Additionally, the Spring Framework also provides support for AOP (Aspect Oriented Programming) and allows for specific code to be injected into the application based on the configuration of "pointcuts" (conditions that apply for the code to be executed – before entering a method, after leaving a method, on encountering an exception in a given package, class or method).

The Spring Framework will be used to inject the dependencies implementations into the different architecture components, will govern security and transactionality (using declarative configuration), and will control data source connections and perform the look-up and resolution of components and resources in addition to other services such as exception handling or profiling that will be provided as AOP advices to the architecture.

3.4. Other Components

3.4.1. Request Handling

Request mapping will be addressed using Spring MVC Dispactcher Servlet and Controllers components. A example of Spring MVC configuration on section 5.3.2.2 on this document.



3.4.1.1. GWT

Spring MVC and a unique framework controller component are needed in order to address request handling for GWT-RPC protocol request.

3.4.1.2. Web Services

The implementation of Axis2 Web service skeletons and service definitions are needed in order to address Web Services request handing

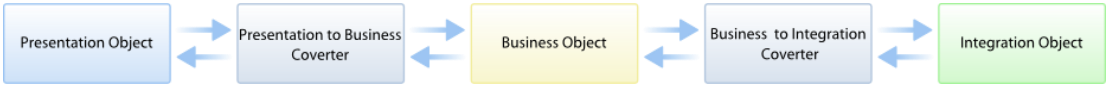
3.4.1.3. RESTFul Web Services

Spring MVC and a unique framework controller component are needed in order to address request handling for RESTFul JSON formated request.

3.4.2. Converters

Converters are necessary to transfer data between layers. Nevertheless, as Appverse Web Middle tier relies on a two or three layered architecture uses objects belonging to each layer to model their data. For instance, use presentation beans that could be dependent on the presentation tier details, optionally business beans in case of two layered middle tier or integration beans that could be dependent of the integration tier details. This implies that conversions must be done every time is needed to transfer data between layers.

In a three layered middle tier, used in common three tier architectures two set of converters are needed:



- Presentation to Business (P2B): In charge to transform objects from Presentation Layer to Business Layer and Business Layer to Presentation Layer.
- Business to Integration (B2I): In charge to transform objects from Business Layer to Integration Layer and Integration Layer to Business Layer.

In case of two layered middle tier, used for N-tier architectures, only a set of converters is needed.

- Presentation To Integration (P2I): In charge to ransform objects from Presentation Layer to Integration Layer and Integration Layer to Presentation Layer.

Converters can be highly automated. Taking this into account, the framework provides architectural structure so that developers do not have to write so much boilerplate source code.

Dozer data mapper is used in order to copy properties from plain objects (POJO) from different layers. However, the framework provides abstract classes structure so that a different solution can be implemented or even custom converters are allowed.

3.5. Cross Application Aspects

3.5.1. Security

3.5.1.1. Disclosure

All server elements should hide their identities; no product names or versions should be disclosed in HTTP headers or error messages (apache, tomcat, etc.). Anonymising server components is important to prevent malicious users from exploiting known issues with specific versions of a server product.

3.5.1.2. Communication

All communication will be redirected from HTTP to HTTPS – SSL3/TLS.

A strong encryption certificate with a valid certification chain must be configured by the web server attending user requests.

Under no condition should any Appverse Web pages or application logic be accessible over a non HTTPS channel; the only exceptions are resources that may be used by the application – css, images, etc – these may be served over HTTP by the web server to reduce processing overhead serving static content.

3.5.1.3. Authentication

A plain HTML page will be used for authentication – form elements should be posted to the server for authenticating the user principal and credentials. The login page must be protected from JavaScript code injection.

No cookies must be explicitly set to a user session until a user has authenticated with the system.

3.5.1.4. Authorization

Frontend application will provides proactive security to hide/enable certain page elements based on a user's role; these roles must be populated at runtime and must not be hard-coded into the application.

IsUserInRole type checks must be performed proactively by the front-end to decide whether a given element is to be made available to the authenticated user based on their role.

IsUserInRole type checks must be performed reactively by the server-side components to decide whether a given function is available to the authenticated user based on their role.

3.5.1.5. Cross-site Scriptiong (XSS) Protection

In order to protect the Appverse Web Applications from Cross-Site Scripting (CSS) attacks, the technical architecture provides a FilterServlet that rejects URLs that contain certain characters in either the base URL or the query string parameters.

Applications must not use any of the following characters in URL or query string parameters, whether escaped or unescaped:

- single quote ('), character %27

- left angle bracket (<), character %3c
- right angle bracket (>), character %%3e

Note that escaping of the characters does not mask them from detection. Thus, all of the following sequences will be rejected as they all resolve to a single left angle bracket (<): %3c, %253c, %25%33%43, %25%32%35%33%43, etc.

3.5.1.6. Cross-site Request Forgery (XSRF) Protection

As a protection for XSRF attacks a random seed will be generated in the middle tier and stored in user session, the same seed will be sent to the presentation tier with first response and given from the request header at the subsequent request and checked with the seed stored at the middle tier user session. The use of Cross-site Request Forgery (XSRF) protection is mandatory when using Appverse Web Framework for develop sensitive applications.

3.5.1.7. Session Fixation Protection

The session ID needs to be changed upon a successfully authentication in order to fix a 'session fixation' vulnerability. The use of session fixation protection is mandatory when using Appverse Web Framework for develop sensitive applications

3.5.1.8. Cookie Protection

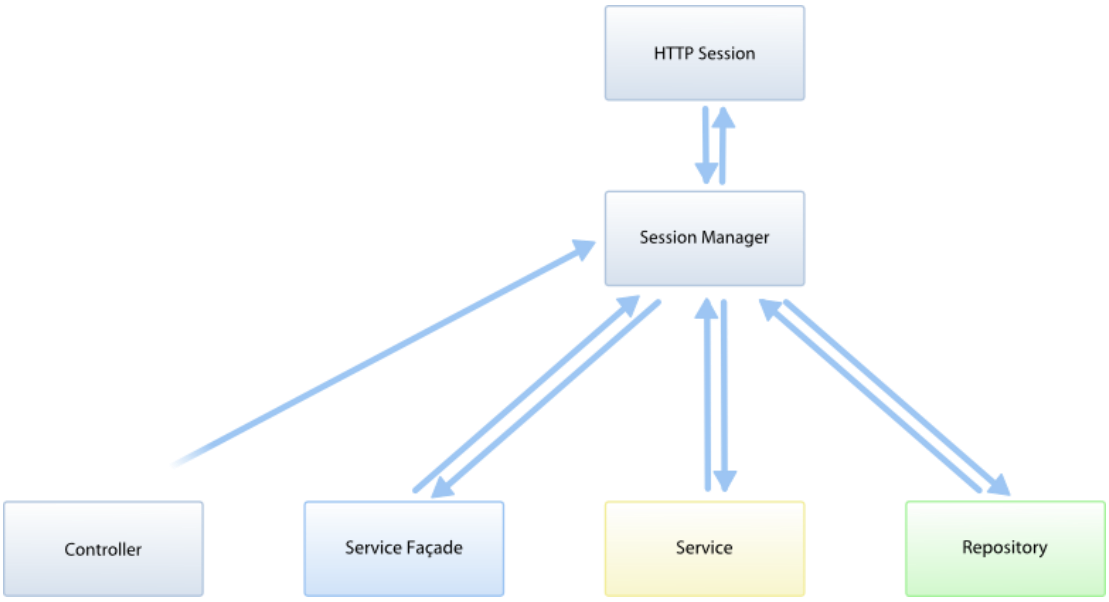
The use of Secure and HTTP only flags are mandatory in order to protect the cookies sent to client. The use of cookie protection is mandatory when using Appverse Web Framework for develop sensitive applications.

3.5.1.9. Use of GET

All the services will be exposed exclusively with POST. The use of GET to expose services is forbidden when using Appverse Web Framework for develop sensitive applications.

3.5.2. Session Management

As a rule of thumb all components used at Appverse Web Framework middle tier are stateless. This pattern rules for applications without state or other ones where the state is managed by another tier. In some scenarios middle tier need to have state, in these cases the Appverse Web choose is use a session aware controller to create a session manager at session creation time. The session manager will be unique and shared between all the other stateless components.



The session manager will be a wrapper with the httpsession component acting as a proxy between the application components and the httpsession. The session manager will be session scoped and its life will be the same that the httpsession life. An example ot the implementation of this pattern can be found on section 4.2.3.2 on this document.

3.5.3. Transaction Management

The transactional boundary in the Appverse Web architecture is deemed to be at the services façade level as it represents the coarsest grained operations a user can perform on the application; services implement more finely grained operations to build-up functionality and repositories, in turn, implement very fine grained access to data.

The IoC container, using the AOP txAdvice, uses declarative transactions – service façades that require transaction management are configured and not coded. As a design choice, the transaction commences before invoking any method of a configured service façade method and concludes when the method returns or an error condition is encountered. If no conditions occur that require the transaction to be rolled-back, the transaction is committed prior to returning the result to the consuming client.

There are pre-conditions that apply for a transaction to be committed. If any of the pre-conditions are not met, the transaction is rolled-back and an exception is propagated to the consuming client as soon as a failed condition is encountered. The following conditions (or rules) will provoke the transaction to be aborted and rolled-back:

- Optimistic Locking – an attempt to update data that is considered stale (see section 3.5.4).
- Unchecked Exceptions – an unchecked exception is thrown anywhere in the underlying code path.
- Checked Exceptions – a checked exception that is declared as a cause for rollback is encountered.

Each service façade method is considered to be an atomic transaction; the entire batch of data operations carried out by Repository will either be committed or rolled-back when their underline systems are attached to spring transaction system.

There are two main approaches for implementing transactions within the Spring IoC container – programmatic and declarative transaction management:

- The programmatic approach relies on developers to explicitly code the checks to ensure all pre-conditions are met in order to commit or roll-back a transaction; this approach is error prone and cum-

bersome, requiring virtually the same code to be present in most service façade method implementations. This approach is not recommended and is not adopted by the Appverse Web architecture.

- The declarative approach has the least impact on application code, is the most flexible and allows for transactional behaviour to be configured on a method by method basis or for complete packages or interfaces; the configuration of pointcuts are very flexible. Additionally, it allows for additional pre-conditions/roll-back rules to be introduced without modifying application code. This is the recommended approach and is the adopted approach of the Appverse Web architecture.

The declarative transaction approach using AOP txAdvice ensures the integrity of data and at the same time reduces the risk of inconsistent rules being applied to transactions – these being managed exclusively by the Appverse Web architecture and expressed via configuration files

3.5.4. Optimistic Locking

The Appverse Web architecture could use an optimistic locking approach to afford long user transactions; transactions often span hours from the moment the data is first retrieved to the moment the data is persisted to the data source. Potentially, multiple users trying to update the same data would lead to previous updates being over-written by the last update operation committed if this condition is unchecked.

The architecture manages these long running transactions by using mandatory version attributes within all data model objects (refer futher on this document) supported. These mandatory version attributes are managed in the top-level classes of the Business and Integration Object Models.

The following use-cases apply in optimistic locking:

- On inserting a new record - If in the data source no prior version of the specific entity exists, then the entity is created with a value of 1 to its "version" attribute.
- On updating an existing record - The version attribute is checked prior to performing any update attempt of the data. The "version" attributes of the object to be persisted and the value held in the data source MUST BE the same for the data to be considered valid for update.

If on update the object.version == datasource.version then the data is considered of the same version and valid for update; the transaction can proceed.

If on update the object.version != datasource.version then the data is considered to be stale and not valid for update; the transaction is aborted and the client application notified to obtain a refreshed object containing the new data.

There is a need to check version attributes in a consistent manner, this will, therefore, be exclusively handled by the Appverse Web architecture and never delegated to functional sub-classes of the build applications. Presentation tier must handle the display of the error and the presentation of the refreshed data. Special consideration should be given to avoid losing changes a user may have carried out locally prior to being forced to refresh the stale data.

In addition to checking the version attribute the super classes that manage the "version" attribute of an object, and underlying entity, must also manage the when the version attribute is incremented – this will happen after every successful update operation. Incrementing the version attribute value will be managed by the architecture rather than by delegating this task to a database managed sequence. This approach avoids having to create separate sequences for each database entity.

3.5.5. Proactive and Reacive Security

There are two security modes in operation within the Appvese Web architecture. One that executes in the front-end and one on the server; proactive and reactive security respectively.

Proactive security is tasked with enabling/disabling certain buttons in the front-end in response to the functions a given user is able to execute; this approach ensures that only users with the appropriate entitlements are shown enabled/visible buttons for all the operations they can execute.

Reactive security is the server-side counterpart of proactive security. On receiving a request, a service will check the user entitlements and the associated functions. An operation is only performed if the user originating the request has the required function to perform the operation.

This approach provides a double safety net to ensure only entitled users perform certain operations; even if the front-end incorrectly enables an action for an operation, the server would refuse to execute the operation if the function is not available to the user.

The Appverse Web architecture supports both proactive and reactive security:

- **Proactive** – based on a user's profile and permissions, actions can be hidden to the user to prevent the user from executing a given action. This allows for certain controls to be either hidden or deactivated. The proactive security model is mainly implemented at the front-end to customize the presentation the user perceives based on their profile.

- **Reactive** – based on a user's profile and permissions, the server can check whether a user can perform a given action. Irrespective of whether the front-end hides/deactivates certain actions a user can perform, the architecture always checks whether the user's profile and permissions allow for a certain function to be performed. If a user without permission attempts to execute an action, the architecture intercepts the call and returns an error to the user.

3.5.6. Checked & Unchecked Exceptions

Checked exceptions are those that are explicitly contemplated by a service and declared in their respective interfaces. Unchecked exceptions are those that are not declared in any interface and encountered at runtime.

- **Checked Exceptions** – All checked exceptions that are encountered should have a registered error code. The architecture manages these exceptions and populates relevant error messages to be propagated to the consuming client application. By default, checked applications do not provoke a transaction to be rolled back. These exceptions only provoke a transaction to be rolled back if they are explicitly configured to do so.

- **Unchecked Exceptions** – All unchecked exceptions provoke a transaction to be rolled back. Unchecked exceptions cannot be left unmanaged or configured to be ignored by the AOP txAdvice transaction management. Unchecked Exceptions are still intercepted by the dbSyndicate architecture and managed to avoid technical error messages from being propagated to the consuming application.

3.5.7. Auditing of Changes

A common requirement of the architecture is to keep track of all the changes made during the lifetime of a data record.

3.5.7.1. Complete auditing in One Separate Table

The Appverse Web architecture recommends a single database table to store and keep track of all the alphas made to all entities. The following information is explicitly kept:

This kind of audit functionality is managed in order to implement change history views for any given record.

- User making the change.
- Timestamp of change.
- Values that are changed.

3.5.7.2. Automatic Partial Auditing in the Same Data Row

Appverse Web has an JPA entityListener that can be used for any entity that subclases a framework class to save automatically the following data:

- User creating the data
- Timestamp of creating
- User making the change.
- Timestamp of change.
- Version for optimistic locking

3.5.8. Logging

Logging and its proper management is vital in the resolution of production environment issues nevertheless its inappropriate use could end damaging the whole application performance.

The basic guidelines include **never** use direct write to the standard output (System.out.print statements). Instead of these statements these guidelines enforces to use a logging framework.

Appverse Web Framework uses for logging the Simple Logging Facade for Java (SLF4J) and Log4j as its default implementation. SLF4J acts as a logging façade allowing choosing the actual logging implementation and helps to standardize the potential several loggers that could be used by the third party libraries. A example of third party library logging standarizationg could be found on section 4.2.1.3.2 of this document

These logging framework permits write on a several types of appenders depending of a declarative configuration that could be externalized from the application code and change depending of the environment and necessities on a specific point of the project. Our recommendation consist in the use of filerolling appenders for different kinds of serverity and not writing to standard output in productions environtments. A development configuration file for log4j could be found on section 5.3.2.13 of this document.

Loggers use commonly the category (or logger) concept that consists in give to every piece of code a category to have fine control of the severity of the logs that would be send to a specific appender. Sadly, a very common antipattern is used for many developers that 'copy and paste' the category configuration from one class to another, then invalidating that kind of fine control. A solution to this antipattern could be found on section 4.2.1.3.1

3.5.9. Caching

Using caching a application can increase in a sensible way its performance. Caching is a cross-application service that have no connection with the implementation of the service that is modifying. For this reason EHCache is the recommended library, because provides a declarative configuration and a set of annotations that can control the behabiour of caching and the when cache an evict a response.

3.5.10. Exception Handling

When a proper exception handling is strongly recommended, there are lots of methods in the services that only can log the exceptions and throw them up. There are expected to do in a clean and consistence way. To have the ability to perform the exception handling to every service in the architecture and to be able to enable and disable this ability declaratively a exception handling aspect is suggested, taking into account that a spring container is the design solution to apply AOP an Spring Aspect is the recommended implementation. The details of the technical solution adopted by Appverse Web could be found on section 4.2.1.1.1 later on this document.

3.5.11. Profiling

When a performance problem arise is important to isolate it into the more specific component possible. The measure of the time consumed by every architecture component is a major help to have a diagnostic of the problem. To have the ability to perform the profiling to every service in the architecture and to be able to enable and disable this ability declaratively a profile aspect is suggested, taking into account that a spring container is the design solution to apply AOP an Spring Aspect is the recommended implementation. The details of the technical solution adopted by Appverse Web could be found on section 4.2.1.1.1 later on this document.

3.6. Integration Tier/Layer Aspects

3.6.1. JPA Flavour

Appverse Web recommends the use of JPA[14] and eclipselink[15] to interact with Relational Database Management Systems to manage the CRUD[16] operations and and non-complex queries using the JPQL[17] Language.

3.6.1.1. JPA

The Java Persistence API, sometimes referred to as JPA, is a Java programming language framework managing relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

The Java Persistence API originated as part of the work of the JSR 220 Expert Group. JPA 2.0 is the work of the JSR 317 Expert Group.

The focus of JPA 2.0 was to address features that were present in some of the popular ORM vendors but couldn't gain consensus approval for JPA 1.0.

JPA covers three main areas:

- The API itself, defined in the javax.persistence package
- The Java Persistence Query Language (JPQL)
- Object/relational metadata

JPA specification at time of writing is implemented by:

- BatooJPA
- DataNucleus (formerly JPOX)
- EclipseLink (formerly Oracle TopLink)
- JBoss Hibernate
- ObjectDB
- OpenJPA
- IBM, via its OpenJPA-based Feature Pack for OSGi Applications and JPA 2.0 for WebSphere Application Server
- Versant JPA (not relational, object database)

FInally remark that JPA is the standard taken by Java as Object Relational Mapping specification over other specifications like Java Data Objects API (JDO) due its specification to Relational Database Management Systems.

3.6.1.2. JPQL

The JPA Query Language (JPQL) can be considered as an object oriented version of SQL. Users familiar with SQL should find JPQL very easy to learn and use.

---

14 http://en.wikipedia.org/wiki/Java_Persistence_API
15 http://www.eclipse.org/eclipselink/
16 http://en.wikipedia.org/wiki/CRUD
17 http://en.wikipedia.org/wiki/Java_Persistence_Query_Language

JPQL is based on the Hibernate Query Language (HQL), an earlier non-standard query language included in the Hibernate object-relational mapping library.

An example of a JPQL Query is:

```
SELECT a FROM Author a ORDER BY a.firstName, a.lastName
```

3.6.1.3. JPA Performance[18]

The Java Persistence API (JPA) provides a rich persistence architecture. JPA hides much of the low level dull-drum of database access, freeing the application developer from worrying about the database, and allowing them to concentrate on developing the application. However, this abstraction can lead to poor performance, if the application programmer does not consider how their implementation affects database usage.

3.6.1.4. EclipseLink

EclipseLink is the open source Eclipse Persistence Services Project from the Eclipse Foundation. The software provides an extensible framework that allows Java developers to interact with various data services, including databases, web services, Object XML mapping (OXM), and Enterprise Information Systems (EIS). EclipseLink supports a number of persistence standards including:

- Java Persistence API (JPA)
- Java Architecture for XML Binding (JAXB)
- Java Connector Architecture (JCA)
- Service Data Objects (SDO).

EclipseLink is based on the TopLink product from which Oracle contributed the source code to create the EclipseLink project. The original contribution was from TopLink's 11g code base, and the entire code-base/feature set was contributed, with only EJB 2 Container-Managed Persistence (CMP) and some minor Oracle Application Server specific integration removed. This differs from the TopLink Essentials GlassFish contribution, which did not include some key enterprise features. The package names were changed and some of the code and configuration was moved around.

Sun Microsystems selected the EclipseLink project to be the reference implementation for JPA2[19]

3.6.1.5. JPA Code Generation

There are two ways in order to generate java and database code with JPA. Direct engineenering, that is, go from java classes to database tables or reverse engieenering or generate java classes to database tables. Dali[20]

3.6.1.6. FAQ & Bad Practices

18 http://java-persistence-performance.blogspot.com.es/2011/06/how-to-improve-jpa-performance-by-1825.html
19 http://www.eclipse.org/org/press-release/20080317_Eclipselink.php
20 http://www.eclipse.org/webtools/dali/

3.6.1.7. JPA Lifecycle Events

Callback methods are user defined methods that are attached to entity lifecycle events and are invoked automatically by JPA when these events occur.

3.6.1.7.1. Internal Callback Methods

Internal callback methods are methods that are defined within an entity class. For example, the following entity class defines all the supported callback methods with empty implementations:

```
@Entity
public static class MyEntityWithCallbacks {
    @PrePersist void onPrePersist() {}
    @PostPersist void onPostPersist() {}
    @PostLoad void onPostLoad() {}
    @PreUpdate void onPreUpdate() {}
     @PostUpdate void onPostUpdate() {}
    @PreRemove void onPreRemove() {}
    @PostRemove void onPostRemove() {}
}
```

Internal callback methods should always return `void` and take no arguments. They can have any name and any access level (`public`, `protected`, `package` and `private`) but should not be `static`.

The annotation specifies when the callback method is invoked:

- @PrePersist - before a new entity is persisted (added to the EntityManager).
- @PostPersist - after storing a new entity in the database (during commit or flush).
- @PostLoad - after an entity has been retrieved from the database.
- @PreUpdate - when an entity is identified as modified by the EntityManager.
- @PostUpdate - after updating an entity in the database (during commit or flush).
- @PreRemove - when an entity is marked for removal in the EntityManager.
- @PostRemove - after deleting an entity from the database (during commit or flush).

An `entity` class may include callback methods for any subset or combination of lifecycle events but no more than one callback method for the same event. However, the same method may be used for multiple callback events by marking it with more than one annotation.

By default, a callback method in a super `entity` class is also invoked for `entity` objects of the subclasses unless that callback method is overridden by the subclass.

3.6.1.7.2. Implementation Restrictions

To avoid conflicts with the original database operation that fires the `entity` lifecycle event (which is still in progress) callback methods should not call `EntityManager` or `Query` methods and should not access any other `entity` objects.

If a callback method throws an exception within an active transaction, the transaction is marked for rollback and no more callback methods are invoked for that operation.

3.6.1.7.3. Listeners and External Callback Methods

External callback methods are defined outside `entity` classes in a special `listener` class:

```
public class MyListener {
    @PrePersist void onPrePersist(Object o) {}
    @PostPersist void onPostPersist(Object o) {}
    @PostLoad void onPostLoad(Object o) {}
    @PreUpdate void onPreUpdate(Object o) {}
    @PostUpdate void onPostUpdate(Object o) {}
    @PreRemove void onPreRemove(Object o) {}
    @PostRemove void onPostRemove(Object o) {}
}
```

External callback methods (in a `listener` class) should always return `void` and take one argument that specifies the `entity` which is the source of the lifecycle event. The argument can have any type that matches the actual value (e.g. in the code above, `Object` can be replaced by a more specific type). The `listener` class should be stateless and should have a public no-arg constructor (or no constructor at all) to enable automatic instantiation.

The listener class is attached to the entity class using the @EntityListeners annotation:

```
@Entity @EntityListeners(MyListener.class)
public class MyEntityWithListener {
}
```

Multiple listener classes can also be attached to one entity class:

```
@Entity @EntityListeners({MyListener1.class, MyListener2.class})
public class MyEntityWithTwoListeners {
}
```

Listeners that are attached to an entity class are inherited by its subclasses unless the subclass excludes inheritance explicitly using the @ExcludeSuperclassListeners annotation:

```
@Entity @ExcludeSuperclassListeners
public class EntityWithNoListener extends EntityWithListener {
}
```

3.6.1.7.4. Default Entity Listeners

Default entity listeners are listeners that should be applied by default to all the entity classes. Currently, default listeners can only be specified in a mapping XML file because there is no equivalent annotation:

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
 http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version="1.0">
  <persistence-unit-metadata>
    <persistence-unit-defaults>
      <entity-listeners>
        <entity-listener class="samples.MyDefaultListener1" />
        <entity-listener class="samples.MyDefaultListener2" />
      </entity-listeners>
    </persistence-unit-defaults>
  </persistence-unit-metadata>
</entity-mappings>
```

The mapping file has to be located either in the default location, META-INF/orm.xml, or in another location that is specified explicitly in the persistence unit definition (in persistence.xml).

Default listeners are applied by default to all the entity classes. The `@ExcludeDefaultListeners` annotation can be used to exclude an entity class and all its descendant classes from using the default listeners:

```
@Entity @ExcludeDefaultListeners
public class NoDefaultListenersForThisEntity {
}
@Entity
public class NoDefaultListenersForThisEntityEither
  extends NoDefaultListenersForThisEntity {
}
```

3.6.1.7.5. Callback Invocation Order

If more than one callback method has to be invoked for a lifecycle event (e.g. from multiple listeners) the invocation order is based on the following rules:

- All the external callback methods (which are defined in listeners) are invoked before the internal callback methods (which are defined in entity classes).
- Default listeners are handled first, then listeners of the top level entity class, and then down the hierarchy until listeners of the actual entity class. If there is more than one defaultlistener or more than one listener at the same level in the hierarchy, the invocation order follows the definition order.
- Internal callback methods are invoked starting at the top level entity class and then down the hierarchy until the callback methods in the actual entity class are invoked.

3.6.2. Mybatis Flavour

Appverse Web recommends the use of Mybatis[21] to interact with Relational Database Management Systems when when a high control of the SQL sentences is required, two representative cases that need this kind of control are performace-critical queries and complex queries involving open sets of input data as searchers.

The MyBatis data mapper framework makes it easier to use a relational database with object-oriented applications. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations. Simplicity is the biggest advantage of the MyBatis data mapper over object relational mapping tools.

To use the MyBatis data mapper, you rely o existing objects, XML, and SQL. There is little to learn that you don't already know. With theMyBatis data mapper, you have the full power of both SQL and stored procedures at your fingertips.

The MyBatis project is developed and maintained by a team that includes the original creators of the "iBATIS" data mapper.

3.6.2.1. Mybatis Spring

MyBatis-Spring[22] integrates MyBatis seamlessly with Spring. This library allows MyBatis to participate in Spring transactions, takes care of building MyBatis mappers and SqlSessions and inject them into other beans, translates MyBatis exceptions into Spring DataAccessExceptions, and finally, it lets you build your application code free of dependencies on MyBatis, Spring or MyBatis-Spring.

Using Mybatis Spring along with Spring 3 and Mybatis, gererate an mybatis based repository Spring bean is as easy as write the following code:

21 https://code.google.com/p/mybatis/
22 http://mybatis.github.io/spring/index.html

```
public interface UserMapper {
  @Select("SELECT * FROM users WHERE id = #{userId}")
  User getUser(@Param("userId") String userId);
}
```

3.6.3. Web Services Flavour

Appverse Web recommends the use of Mybatis[23] to interact with Relational Database Management Systems when when a high control of the SQL sentences is required, two representative cases that need this kind of control are performace-critical queries and complex queries involving open sets of input data as searchers.

The MyBatis data mapper framework makes it easier to use a relational database with object-oriented applications. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations. Simplicity is the biggest advantage of the MyBatis data mapper over object relational mapping tools.

To use the MyBatis data mapper, you rely o existing objects, XML, and SQL. There is little to learn that you don't already know. With theMyBatis data mapper, you have the full power of both SQL and stored procedures at your fingertips.

The MyBatis project is developed and maintained by a team that includes the original creators of the "iBATIS" data mapper.

23 https://code.google.com/p/mybatis/

3.7. Presentation Tier/Layer Aspects

3.7.1. Common Presentation Tier Concepts

3.7.1.1. Separated Presentation[24]

Ensure that any code that manipulates presentation only manipulates presentation, pushing all domain and data source logic into clearly separated areas of the program.

This pattern is a form of layering, where we keep presentation code and domain code in separate layers with the domain code unaware of presentation code. This style came into vogue with Model-View-Controller architecture and is widely used.

To use it you begin by looking at all the data and behavior in a system and looking to see if that code is involved with the presentation. Presentation code would manipulate GUI widgets and structures in a rich client application, HTTP headers and HTML in a web application, or command line arguments and print statements in a command line application. We then divide the application into two logical modules with all the presentation code in one module and the rest in another module.

Further layering is often used to separate data source code from domain (business logic), and to separate the domain using a Service Layer. For the purposes of Separated Presentation we can ignore these further layers, just referring to all of this as 'the domain layer'. Just bear in mind that further layering of the domain layer is likely.

The layers are a logical and not a physical construct. Certainly you may find a physical separation into different tiers but this is not required (and a bad idea if not necessary). You also may see a separation into different physical packaging units (eg Java jars or .NET assemblies), but again this isn't required. Certainly it is good to use any logical packaging mechanisms (Java packages, .NET namespaces) to separate the layers.

As well as a separation, there is also a strict visibility rule. The presentation is able to call the domain but not vice-versa. This can be checked as part of a build with dependency checking tools. The point here is that the domain should be utterly unaware of what presentations may be used with it. This both helps keep the concerns separate and also support using multiple presentations with the same domain code.

Although the domain cannot call the presentation it's often necessary for the domain to notify the presentation if any changes occur. Observer is the usual solution to this problem. The domain fires an event which is observed the by presentation, the presentation then re-reads data from the domain as needed.

A good mental test to use to check you are using Separated Presentation is to imagine a completely different user interface. If you are writing a GUI imagine writing a command line interface for the same application. Ask yourself if anything would be duplicated between the GUI and command line presentation code - if it is then it's a good candidate for moving to the domain.

---

24 http://martinfowler.com/eaaDev/SeparatedPresentation.html

3.7.2. GWT Flavour

3.7.2.1. Common GWT Flavour Presentation Tier Concepts

3.7.2.1.1. Mediated Syncronization[25]

Synchronize multiple screens by having them all be observers to a shared area of domain data.

In some applications you have multiple screens available that display presentations of a common area of data. If a change is made to the data through one of these screens, you want all the other screens to update correctly. However you don't want each screen to know about the others, because that would increase the complexity of the screens and make it harder to add new ones.

Observer Synchronization uses a single area of domain oriented data and has each screen be an observer of that data. Any change in one screen propagates to that domain oriented data and thence to the other screens. This approach was a large part of the Model View Controller approach.

The essence of this approach is that each screen, with its associated screen state, acts as an Observer on a common area of session data. All changes to the session data result in events which the screens listen to and respond by reloading from the session data. Once you have this mechanism set up, then you can ensure synchronization simply by coding each screen to update the session data. Even the screen that makes the change doesn't need to refresh itself explicitly, since the observer mechanism will trigger the refresh in the same way as if another screen made the change.

Perhaps the biggest issue in this design is deciding what granularity of events to use and how to set up the propagating and observer relationships. At the very fine-grained level each bit of domain data can have separate events to indicate exactly what changed. Each screen registers for only the events that may invalidate that screen's data. The coarsest grained alternative is to use an Event Aggregator to funnel all events into a single channel. That way each screen does a reload if any piece of domain data changes, whether or not it would have affected the screen.

As usual the trade-off is between complexity and performance. The coarse-grained approach is much simpler to set up and less likely to breed bugs. However a coarse grained approach leads to lots of unnecessary refreshes of the screen, which might impact performance. As usual my advice is to start coarse grained and introduce appropriate fine-grained mechanisms only when needed after measuring an actual performance problem.

Events tend to be hard to debug since you can't see the chain of invocations by looking at the code. As a result it's important to keep the event propagation mechanism as simple as you can, which is why I favor coarse-grained mechanisms. A good rule of thumb is to think of your objects as layered and to only allow observer relationships between layers. So one domain object should not observe another domain object, only presentation objects should observe domain objects.

Another thing to be very wary of is chains of events, where one event causes another event to fire. Such event chains can quickly get very hard to follow because you can't understand the behavior by looking at the code. As a result I tend to discourage events within a layer and prefer either a single line of events, or going through an Event Aggregator.

---

25 http://martinfowler.com/eaaDev/MediatedSynchronization.html

When an observer registers for an event, you get a reference from the subject to the observer. If the observer doesn't remove itself from the subject when you get rid of the screen, you have a zombie reference and a memory leak. If you destroy and create domain data with each session and your session is short, this may not lead to a problem. However a long lived domain object could lead to a serious leak.

3.7.2.1.2. GUI Arquitectures[26]

Graphical user interfaces have become a familiar part of our software landscape, both as users and as developers. Looking at it from a design perspective they represent a particular set of problems in system design - problems that have led to a number of different but similar solutions.

3.7.2.1.2.1. Forms and Controls

This architecture that is both simple and familiar. It doesn't have a common name, so for the purposes of document I shall call it "Forms and Controls". It's a familiar architecture because it was the one encouraged by client-server development environments in the 90's - tools like Visual Basic, Delphi, and Powerbuilder. It continues to be commonly used, although also often vilified by design geeks.

Most GUI environments come with a hefty bunch of common controls that we can just use in our application. We can build new controls ourselves, and often it's a good idea to do so, but there is still a distinction between generic reusable controls and specific forms. Even specially written controls can be reused across multiple forms.

The form contains two main responsibilities:

- Screen layout: defining the arrangement of the controls on the screen, together with their hierarchic structure with other.
- Form Logic: behavior that cannot be easily programmed into the controls themselves.

Most GUI development environments allow the developer to define screen layout with a graphical editor that allows you to drag and drop the controls onto a space for the form. This pretty much handles the form layout. This way it's easy to setup a pleasing layout of controls on the form (although it isn't always the best way to do it)

The controls display data. This data will pretty much always come from somewhere else; in this case let's assume a SQL database as that's the environment that most of these client server tools assume. In most situations there are three copies of the data involved:

- One copy of data lies in the database itself. This copy is the lasting record of the data, so I call it the record state. The record state is usually shared and visible to multiple people via various mechanisms.
- A further copy lies inside in-memory Record Sets27 within the application. Most client-server environments provided tools which made this easy to do. This data was only relevant for one particular session between the application and the database, so I call it session state. Essentially this provides a temporary local version of the data that the user works on until they save, or commit it, back to the database - at which point it merges with the record state
- The final copy lies inside the GUI components themselves. This, strictly, is the data they see on the screen; hence I call it the screen state. It is important to the UI how screen state and session state are kept synchronized.

---

26 http://martinfowler.com/eaaDev/uiArchs.html
27 http://martinfowler.com/eaaCatalog/recordSet.html

Keeping screen state and session state synchronized is an important task. A tool that helped make this easier was Data Binding[28]. The idea was that any change to either the control data, or the underlying record set was immediately propagated to the other. So if I alter the actual reading on the screen, the text field control effectively updates the correct column in the underlying record set.

Data Binding handles much of the functionality of a client sever application pretty nicely. If I change the actual value the column is updated, even changing the selected station alters the currently selected row in the record set, which causes the other controls to refresh.

Much of this behavior is built in by the framework builders, who look at common needs and make it easy to satisfy them. In particular this is done by setting values, usually called properties, on the controls. The control binds to a particular column in a record set by having its column name set through a simple property editor.

Using data binding, with the right kind of parameterization, can take you a long way. However it can't take you all the way - there's almost always some logic that won't fit with the parameterization options. In this case calculating the variance is an example of something that doesn't fit in this built in behavior - since it's application specific it usually lies in the form.

In order for this to work the form needs to be alerted whenever the value of the actual field changes, which requires the generic text field to call some specific behavior on the form. This is a bit more involved than taking a class library and using it through calling it as Inversion of Control is involved.

There are various ways of getting this kind of thing to work - the common one for client-server toolkits was the notion of events. Each control had a list of events it could raise. Any external object could tell a control that it was interested in an event - in which case the control would call that external object when the event was raised. Essentially this is just a rephrasing of theObserver pattern where the form is observing the control. The framework usually provided some mechanism where the developer of the form could write code in a subroutine that would be invoked when the event occurred. Exactly how the link was made between event and routine varied between platform and is unimportant for this discussion - the point is that some mechanism existed to make it happen.

Once the routine in the form has control, it can then do whatever it needed. It can carry out the specific behavior and then modify the controls as necessary, relying on data binding to propagate any of these changes back to the session state.

This is also necessary because data binding isn't always present. There is a large market for windows controls, not all of them do data binding. If data binding isn't present then it's up to the form to carry out the synchronization. This could work by pulling data out of the record set into the widgets initially, and copying the changed data back to the record set when the save button was pressed.

We can summarize the architecture with a few soundbites:

- Developers write application specific forms that use generic controls.
- The form describes the layout of controls on it.
- The form observes the controls and has handler methods to react to interesting events raised by the controls.
- Simple data edits are handled through data binding.

---

28 http://martinfowler.com/eaaDev/DataBinding.html

- Complex changes are done in the form's event handling methods.

3.7.2.1.2.2. Model View Controller

Probably the widest quoted pattern in UI development is Model View Controller (MVC) - it's also the most misquoted. I've lost count of the times I've seen something described as MVC which turned out to be nothing like it. Frankly a lot of the reason for this is that parts of classic MVC don't really make sense for rich clients these days. But for the moment we'll take a look at its origins.

As we look at MVC it's important to remember that this was one of the first attempts to do serious UI work on any kind of scale. Graphical User Interfaces were not exactly common in the 70's. The Forms and Controls model I've just described came after MVC - I've described it first because it's simpler, not always in a good way.

At the heart of MVC, and the idea that was the most influential to later frameworks, is what I call Separated Presentation. The idea behind Separated Presentation is to make a clear division between domain objects that model our perception of the real world, and presentation objects that are the GUI elements we see on the screen. Domain objects should be completely self contained and work without reference to the presentation; they should also be able to support multiple presentations, possibly simultaneously. This approach was also an important part of the Unix culture, and continues today allowing many applications to be manipulated through both a graphical and command-line interface.

In MVC, the domain element is referred to as the model. Model objects are completely ignorant of the UI. To begin discussing our assessment UI example we'll take the model as a reading, with fields for all the interesting data upon it.

In MVC I'm assuming a Domain Model[29] of regular objects, rather than the Record Set notion that I had in Forms and Controls. This reflects the general assumption behind the design. Forms and Controls assumed that most people wanted to easily manipulate data from a relational database; MVC assumes we are manipulating regular Smalltalk objects.

The presentation part of MVC is made of the two remaining elements: view and controller. The controller's job is to take the user's input and figure out what to do with it.

At this point I should stress that there's not just one view and controller, you have a view-controller pair for each element of the screen, each of the controls and the screen as a whole. So the first part of reacting to the user's input is the various controllers collaborating to see who got edited. In the case that's it's the actuals text field so that text field controller would now handle what happens next.

Like later environments, Smalltalk figured out that you wanted generic UI components that could be reused. In this case the component would be the view-controller pair. Both were generic classes, so needed to be plugged into the application specific behavior. There would be an assessment view that would represent the whole screen and define the layout of the lower level controls, in that sense similar to a form in Forms and Controllers. Unlike the form, however, MVC has no event handlers on the assessment controller for the lower level components.

So there is no overall object observing low level widgets, instead the low level widgets observe the model, which itself handles many of the decision that would be made by the form. In this case, when it comes to figuring out the variance, the reading object itself is the natural place to do that.

---

29 http://martinfowler.com/eaaCatalog/domainModel.html

Observers do occur in MVC, indeed it's one of the ideas credited to MVC. In this case all the views and controllers observe the model. When the model changes, the views react. In this case the actual text field view is notified that the reading object has changed, and invokes the method defined as the aspect for that text field and sets its value to the result.

You'll notice that the text field controller didn't set the value in the view itself, it updated the model and then just let the observer mechanism take care of the updates. This is quite different to the forms and controls approach where the form updates the control and relies on data binding to update the underlying record-set. These two styles I describe as patterns:Flow tion[30] and Observer Synchronization[31]. These two patterns describe alternative ways of handling the triggering of synchronization between screen state and session state. Forms and Controls do it through the flow of the application manipulating the various controls that need to be updated directly. MVC does it by making updates on the model and then relying of the observer relationship to update the views that are observing that model.

Flow Synchronization is even more apparent when data binding isn't present. If the application needs to do synchronization itself, then it was typically done at important point in the application flow - such as when opening a screen or hitting the save button.

One of the consequences of Observer Synchronization is that the controller is very ignorant of what other widgets need to change when the user manipulates a particular widget. While the form needs to keep tabs on things and make sure the overall screen state is consistent on a change, which can get pretty involved with complex screens, the controller in Observer Synchronization can ignore all this.

This useful ignorance becomes particularly handy if there are multiple screens open viewing the same model objects. The classic MVC example was a spreadsheet like screen of data with a couple of different graphs of that data in separate windows. The spreadsheet window didn't need to be aware of what other windows were open, it just changed the model and Observer Synchronization took care of the rest. With Flow Synchronization it would need some way of knowing which other windows were open so it tell them to refresh.

While Observer Synchronization is nice it does have a downside. The problem with Observer Synchronization is the core problem of the observer pattern itself - you can't tell what is happening by reading the code. I was reminded of this very forcefully when trying to figure out how some Smalltalk 80 screens worked. I could get so far by reading the code, but once the observer mechanism kicked in the only way I could see what was going on was via a debugger and trace statements. Observer behavior is hard to understand and debug because it's implicit behavior.

While the different approaches to synchronization are particularly noticeable from looking at the sequence diagram, the most important, and most influential, difference is MVC's use of Separated Presentation. Calculating the variance between actual and target is domain behavior, it is nothing to do with the UI. As a result following Separated Presentation says we should place this in the domain layer of the system - which is exactly what the reading object represents. When we look at the reading object, the variance feature makes complete sense without any notion of the user interface.

At this point, however, we can begin to look at some complications. There's two areas where I've skipped over some awkward points that get in the way of MVC theory. The first problem area is to deal with setting the color of the variance. This shouldn't really fit into a domain object, as the color by which we display a value isn't part of the domain. The first step in dealing with this is to realize

---

30 http://martinfowler.com/eaaDev/FlowSynchronization.html
31 http://martinfowler.com/eaaDev/MediatedSynchronization.html

that part of the logic is domain logic. What we are doing here is making a qualitative statement about the variance, which we could term as good (over by more than 5%), bad (under by more than 10%), and normal (the rest). Making that assessment is certainly domain language, mapping that to colors and altering the variance field is view logic. The problem lies in where we put this view logic - it's not part of our standard text field.

This kind of problem was faced by early smalltalkers and they came up with some solutions. The solution I've shown above is the dirty one - compromise some of the purity of the domain in order to make things work. I'll admit to the occasional impure act - but I try not to make a habit of it.

We could do pretty much what Forms and Controls does - have the assessment screen view observe the variance field view, when the variance field changes the assessment screen could react and set the variance field's text color. Problems here include yet more use of the observer mechanism - which gets exponentially more complicated the more you use it - and extra coupling between the various views.

A way I would prefer is to build a new type of the UI control. Essentially what we need is a UI control that asks the domain for a qualitative value, compares it to some internal table of values and colors, and sets the font color accordingly. Both the table and message to ask the domain object would be set by the assessment view as it's assembling itself, just as it sets the aspect for the field to monitor. This approach could work very well if I can easily subclass text field to just add the extra behavior. This obvious depends on how well the components are designed to enable sub-classing - Smalltalk made it very easy - other environments can make it more difficult.

The final route is to make a new kind of model object, one that's oriented around around the screen, but is still independent of the widgets. It would be the model for the screen. Methods that were the same as those on the reading object would just be delegated to the reading, but it would add methods that supported behavior relevant only to the UI, such as the text color.

This last option works well for a number of cases and, as we'll see, became a common route for Smalltalkers to follow - I call this a Presentation Model because it's a model that is really designed for and thus part of the presentation layer.

The Presentation Model32 works well also for another presentation logic problem - presentation state. The basic MVC notion assumes that all the state of the view can be derived from the state of the model. In this case how do we figure out which station is selected in the list box? The Presentation Model solves this for us by giving us a place to put this kind of state. A similar problem occurs if we have save buttons that are only enabled if data has changed - again that's state about our interaction with the model, not the model itself.

So now I think it's time for some soundbites on MVC.

- Make a strong separation between presentation (view & controller) and domain (model) -Separated Presentation.
- Divide GUI widgets into a controller (for reacting to user stimulus) and view (for displaying the state of the model). Controller and view should (mostly) not communicate directly but through the model.
- Have views (and controllers) observe the model to allow multiple widgets to update without needed to communicate directly - Observer Synchronization.

---

32 http://martinfowler.com/eaaDev/PresentationModel.html

3.7.2.1.2.3. VisualWorks Application Model

As I've discussed above, Smalltalk 80's MVC was very influential and had some excellent features, but also some faults. As Smalltalk developed in the 80's and 90's this led to some significant variations on the classic MVC model. Indeed one could almost say that MVC disappeared, if you consider the view/controller separation to be an essential part of MVC - which the name does imply.

The things that clearly worked from MVC were Separated Presentation and Observer Synchronization. So these stayed as Smalltalk developed - indeed for many people they were the key element of MVC.

Smalltalk also fragmented in these years. The basic ideas of Smalltalk, including the (minimal) language definition remained the same, but we saw multiple Smalltalks develop with different libraries. From a UI perspective this became important as several libraries started using native widgets, the controls used by the Forms and Controls style.

Smalltalk was originally developed by Xerox Parc labs and they span off a separate company, ParcPlace, to market and develop Smalltalk. ParcPlace Smalltalk was called VisualWorks and made a point of being a cross-platform system. Long before Java you could take a Smalltalk program written in Windows and run it right away on Solaris. As a result VisualWorks didn't use native widgets and kept the GUI completely within Smalltalk.

In my discussion of MVC I finished with some problems of MVC - particularly how to deal with view logic and view state. VisualWorks refined its framework to deal with this by coming up with a construct called the Application Model - a construct that moves towards Presentation Model. The idea of using something like a Presentation Model wasn't new to VisualWorks - the original Smalltalk 80 code browser was very similar, but the VisualWorks Application Model baked it fully into the framework.

A key element of this kind of smalltalk was the idea of turning properties into objects. In our usual notion of objects with properties we think of a Person object having properties for name and address. These properties may be fields, but could be something else. There is usually a standard convention for accessing the properties: in Java we would see temp = aPerson.getName() and aPerson.setName("martin"), in C# it would temp = aPerson.name and aPerson.name = "martin".

A Property Object changes this by having the property return an object that wraps the actual value. So in VisualWorks when we ask for a name we get back a wrapping object. We then get the actual value by asking the wrapping object for its value. So accessing a person's name would use temp = aPerson name value and aPerson name value: 'martin'

Property objects make the mapping between widgets and model a little easier. We just have to tell the widget what message to send to get the corresponding property, and the widget knows to access the proper value using value and value:. VisualWorks's property objects also allow you to set up observers with the message onChangeSend: aMessage to: anObserver.

You won't actually find a class called property object in Visual Works. Instead there were a number of classes that followed the value/value:/onChangeSend: protocol. The simplest is the ValueHolder - which just contains its value. More relevant to this discussion is the AspectAdaptor. The AspectAdaptor allowed a property object to wrap a property of another object completely. This way you could define a property object on a PersonUI class that wrapped a property on a Person object by code like

The main difference between using an application model and classic MVC is that we now have an intermediate class between the domain model class (Reader) and the widget - this is the application model class. The widgets don't access the domain objects directly - their model is the applica-

tion model. Widgets are still broken down into views and controllers, but unless you're building new widgets that distinction isn't important.

At this point the observer relationships kick in. We need to set things up so that updating the actual value causes the reading to indicate that it has changed. We do this by putting a call in the modifier for actual to indicate that the reading object has changed - in particular that the variance aspect has changed. When setting up the aspect adaptor for variance it's easy to tell it to observe the reader, so it picks up the update message which it then forwards to its text field. The text field then initiates getting a new value, again through the aspect adaptor.

Using the application model and property objects like this helps us wire up the updates without having to write much code. It also supports fine-grained synchronization (which I don't think is a good thing).

Application models allow us to separate behavior and state that's particular to the UI from real domain logic. So one of the problems I mentioned earlier, holding the currently selected item in a list, can be solved by using a particular kind of aspect adaptor that wraps the domain model's list and also stores the currently selected item.

The limitation of all this, however, is that for more complex behavior you need to construct special widgets and property objects. As an example the provided set of objects don't provide a way to link the text color of the variance to the degree of variance. Separating the application and domain models does allow us to separate the decision making in the right way, but then to use widgets observing aspect adapters we need to make some new classes.

Directly updating the widgets like this is not part of Presentation Model[33], which is why the visual works application model isn't truly a Presentation Model. This need to manipulate the widgets directly was seen by many as a bit of dirty work-around and helped develop the Model-View-Presenter approach.

So now the soundbites on Application Model

- Followed MVC in using Separated Presentation and Observer Synchronization.
- Introduced an intermediate application model as a home for presentation logic and state - a partial development of Presentation Model.
- Widgets do not observe domain objects directly, instead they observe the application model.
- Made extensive use of Property Objects to help connect the various layers and to support the fine grained synchronization using observers.
- It wasn't the default behavior for the application model to manipulate widgets, but it was commonly done for complicated cases.

3.7.2.1.2.4. Model-View-Presenter (MVP)

See a complete discussion of this pattern on section 3.7.2.2 on this document

3.7.2.1.2.5. Humble View

In the past few years there's been a strong fashion for writing self-testing code. When people talk about self-testing code user-interfaces quickly raise their head as a problem. Many people find that testing GUIs to be somewhere between tough and impossible. This is largely because UIs are tightly coupled into the overall UI environment and difficult to tease apart and test in pieces.

---

33 http://martinfowler.com/eaaDev/PresentationModel.html

Sometimes this test difficulty is over-stated. You can often get surprisingly far by creating widgets and manipulating them in test code. But there are occasions where this is impossible, you miss important interactions, there are threading issues, and the tests are too slow to run.

As a result there's been a steady movement to design UIs in such a way that minimizes the behavior in objects that are awkward to test. Michael Feathers crisply summed up this approach in The Humble Dialog Box[34].Gerard Meszaros[35] generalized this notion to idea of a Humble Object - any object that is difficult to test should have minimal behavior. That way if we are unable to include it in our test suites we minimize the chances of an undetected failure.

The Humble Dialog Box paper uses a presenter, but in a much deeper way than the original MVP. Not just does the presenter decide how to react to user events, it also handles the population of data in the UI widgets themselves. As a result the widgets no longer have, nor need, visibility to the model; they form a Passive View, manipulated by the presenter.

This isn't the only way to make the UI humble. Another approach is to use Presentation Model, although then you do need a bit more behavior in the widgets, enough for the widgets to know how to map themselves to the Presentation Model.

The key to both approaches is that by testing the presenter or by testing the presentation model, you test most of the risk of the UI without having to touch the hard-to-test widgets.

With Presentation Model you do this by having all the actual decision making made by thePresentation Model. All user events and display logic is routed to the Presentation Model, so that all the widgets have to do is map themselves to properties of the Presentation Model. You can then test most of the behavior of the Presentation Model without any widgets being present - the only remaining risk lies in the widget mapping. Provided that this is simple you can live with not testing it. In this case the screen isn't quite as humble as with the Passive View approach, but the difference is small.

Since Passive View makes the widgets entirely humble, without even a mapping present,Passive View eliminates even the small risk present with Presentation Model. The cost however is that you need a Test Double to mimic the screen during your test runs - which is extra machinery you need to build.

A similar trade-off exists with Supervising Controller. Having the view do simple mappings introduces some risk but with the benefit (as with Presentation Model) of being able to specify simple mapping declaratively. Mappings will tend to be smaller for Supervising Controller than for Presentation Model as even complex updates will be determined by thePresentation Model and mapped, while a Supervising Controller will manipulate the widgets for complex cases without any mapping involved.

3.7.2.1.3. Ray Ryan Presentation

Ray Ryan performed a excellent presentation at Google IO about how to arquitecture a GWT Application.

---

34 http://www.objectmentor.com/resources/articles/TheHumbleDialogBox.pdf
35 http://xunitpatterns.com/

3.7.2.1.3.1. Abstract

Google Web Toolkit provides the infrastructure you need to build a high performance web application and leaves the architecture open to fit your needs. Learn from others who have gone before. In this session we'll discuss best practices that real web applications are using to achieve high performance event handling, UI creation, and more.

In that presentation Ray Ryan examplain and recommends the use of the following patterns and best practices:

- Model View Presenter
- Event Bus
- Dependency Injection
- Command Pattern

3.7.2.1.3.2. Ray Rian Presentation Video[36]

3.7.2.1.3.3. Ray Ryan Presentation Transcript [37]

3.7.2.2. Reverse Model-View-Presenter

3.7.2.2.1. Model – View – Presenter[38]

MVP is an architecture that first appeared in IBM and more visibly at Taligent the 1990's. It's most commonly referred via the Potel paper. The idea was further popularized and described by the developers of Dolphin Smalltalk. As we'll see the two descriptions don't entirely mesh but the basic idea underneath it has become popular.

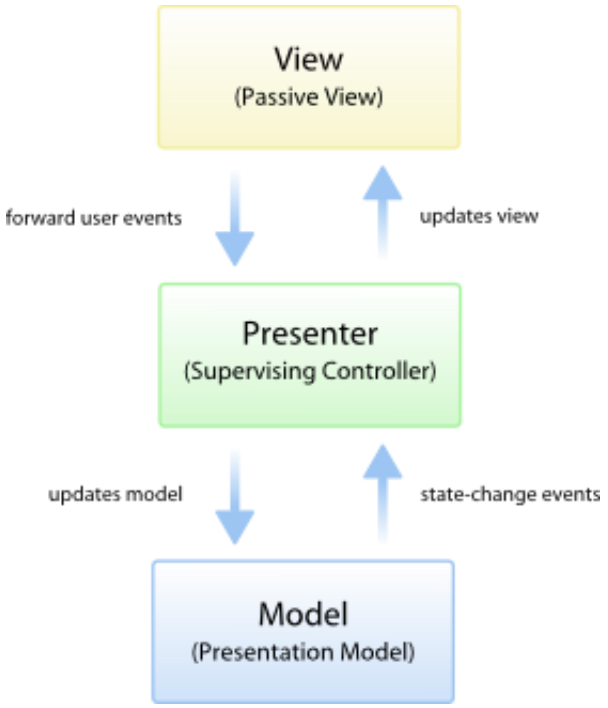
To approach MVP I find it helpful to think about a significant mismatch between two strands of UI thinking. On the one hand is the Forms and Controller architecture which was mainstream approach to UI design, on the other is MVC and its derivatives. The Forms and Controls model provides a design that is easy to understand and makes a good separation between reusable widgets and application specific code. What it lacks, and MVC has so strongly, is Separated Presentation and indeed the context of programming using a Domain Model. I see MVP as a step towards uniting these streams, trying to take the best from each.

The first element of Potel is to treat the view as structure of widgets, widgets that correspond to the controls of the Forms and Controls model and remove any view/controller separation. The view of MVP is a structure of these widgets. It doesn't contain any behavior that describes how the widgets react to user interaction.

The active reaction to user acts lives in a separate presenter object. The fundamental handlers for user gestures still exist in the widgets, but these handlers merely pass control to the presenter.

---

36 http://www.google.com/intl/ca/events/io/2009/sessions/GoogleWebToolkitBestPractices.html
37http://extgwt-mvp4g-gae.blogspot.com.es/2009/10/gwt-app-architecture-best-practices.html
38 http://martinfowler.com/eaaDev/ModelViewPresenter.html



The presenter then decides how to react to the event. Potel discusses this interaction primarily in terms of actions on the model, which it does by a system of commands and selections. A useful thing to highlight here is the approach of packaging all the edits to the model in a command - this provides a good foundation for providing undo/redo behavior.

As the Presenter updates the model, the view is updated through the same Observer Synchronization approach that MVC uses.

The Dolphin description is similar. Again the main similarity is the presence of the presenter. In the Dolphin description there isn't the structure of the presenter acting on the model through commands and selections. There is also explicit discussion of the presenter manipulating the view directly. Potel doesn't talk about whether presenters should do this or not, but for Dolphin this ability was essential to overcoming the kind of flaw in Application Model that made it awkward for me to color the text in the variation field.

One of the variations in thinking about MVP is the degree to which the presenter controls the widgets in the view. On one hand there is the case where all view logic is left in the view and the presenter doesn't get involved in deciding how to render the model. This style is the one implied by Potel. The direction behind Bower and McGlashan was what I'm callingSupervising Controller, where the view handles a good deal of the view logic that can be describe declaratively and the presenter then comes in to handle more complex cases.

You can also move all the way to having the presenter do all the manipulation of the widgets. This style, which I call Passive View isn't part of the original descriptions of MVP but got developed as people explored testability issues. I'm going to talk about that style later, but that style is one of the flavors of MVP.

Before I contrast MVP with what I've discussed before I should mention that both MVP papers here do this too - but not quite with the same interpretation I have. Potel implies that MVC controllers were overall coordinators - which isn't how I see them. Dolphin talks a lot about issues in MVC, but by MVC they mean the VisualWorks Application Model design rather than classic MVC that I've described (I don't blame them for that - trying to get information on classic MVC isn't easy now let alone then.)

So now it's time for some contrasts:

- Forms and Controls: MVP has a model and the presenter is expected to manipulate this model with Observer Synchronization then updating the view. Although direct access to the widgets is allowed, this should be in addition to using the model not the first choice.
- MVC: MVP uses a Supervising Controller to manipulate the model. Widgets hand off user gestures to the Supervising Controller. Widgets aren't separated into views and controllers. You can think of presenters as being like controllers but without the initial handling of the user gesture. However it's also important to note that presenters are typically at the form level, rather than the widget level - this is perhaps an even bigger difference.
- Application Model: Views hand off events to the presenter as they do to the application model. However the view may update itself directly from the domain model, the presenter doesn't act as a Presentation Model. Furthermore the presenter is welcome to directly access widgets for behaviors that don't fit into the Observer Synchronization.

There are obvious similarities between MVP presenters and MVC controllers, and presenters are a loose form of MVC controller. As a result a lot of designs will follow the MVP style but use 'controller' as a synonym for presenter. There's a reasonable argument for using controller generally when we are talking about handling user input.

Here are the MVP soundbites:

- User gestures are handed off by the widgets to a Supervising Controller.
- The presenter coordinates changes in a domain model.
- Different variants of MVP handle view updates differently. These vary from using Observer Synchronization to having the presenter doing all the updates with a lot of ground in-between.

3.7.2.2.2. Passive view[39]

A screen and components with all application specific behavior extracted into a controller so that the widgets have their state controlled entirely by controller.

A perennial problem with building rich client systems is the complication of testing them. Most rich client frameworks were not built with automated testing in mind. Controlling these frameworks programmatically is often very difficult.

A Passive View handles this by reducing the behavior of the UI components to the absolute minimum by using a controller that not just handles responses to user events, but also does all the updating of the view. This allows testing to be focused on the controller with little risk of problems in the view.

This pattern is yet another variation on model-view-controller and model-view-presenter. As with these the UI is split between a view that handles display and a controller that responds to user gestures. The significant change with Passive View is that the view is made completely passive and is no longer responsible for updating itself from the model. As a result all of the view logic is in the controller. As a result, there are no dependencies in either direction between the view and the model.

The primary driver for Passive View is testing, as a result it's often valuable to use of Testble[40] for the view so that the controller can be tested without needing any interaction with the UI framework.

---

39 http://martinfowler.com/eaaDev/PassiveScreen.html

3.7.2.2.3. Supervising Controller[41]

Factor the UI into a view and controller where the view handles simple mapping to the underlying model and the the controller handles input response and complex view logic.

Many UI frameworks provide the ability to easily map between the view and model, often using some kind of Data Binding[42]. These approaches are very effective in allowing you to declaratively set up a relationship between elements in the view and model. Usually, however, there are more complex relationships which require you to have more complex view logic. This logic can be hard to manage, and in particular hard to test, while embedded in the view.

Supervising Controller uses a controller both to handle input response but also to manipulate the view to handle more complex view logic. It leaves simple view behavior to the declarative system, intervening only when effects are needed that are beyond what can be achieved declaratively.

Supervising Controller decomposes presentation functionality into two parts: a controller (often called presenter) and view. The domain data that needs to be displayed is separate, and following rough MVC terminology I'll refer to it as a model, although it need not be a Domain Model[43]. The basic division of responsibilities echoes the Model-View-Presenter architecture in its Dolphin form, as described by Bower and McGlashan.

A Supervising Controller has two primary responsibilities: input response and partial view/model synchronization.

For input response the controller operates in the presenter style. The user gestures are handled initially by the screen widgets, however all they do in response is to hand these events off to the presenter, which handles all further logic.

For view/model synchronization the controller defers as much of this as reasonable to the view. The view typically uses some form of Data Binding to populate much of the information for its fields. Where Data Binding isn't up to more complex interactions then the controller steps in.

One of the prime reasons to use Supervising Controller is for testability. Assuming the view is hard to test, by moving any complex logic into the controller, we put the logic in a place that's easier to test. In order to run tests on the controller, however, we do need some form of view, so a Test Double is often in order. With the double in place, we don't have any need for UI framework objects in order to test the more awkward parts of the UI behavior.

3.7.2.2.4. Presentation Model[44]

Represent the state and behavior of the presentation independently of the GUI controls used in the interface
Also Known As: Application Model

GUIs consist of widgets that contain the state of the GUI screen. Leaving the state of the GUI in widgets makes it harder to get at this state, since that involves manipulating widget APIs, and also encourages putting presentation behavior in the view class.

---

40 http://xunitpatterns.com/Test%20Double.html
41 http://martinfowler.com/eaaDev/SupervisingPresenter.html
42 http://martinfowler.com/eaaDev/DataBinding.html
43 http://martinfowler.com/eaaCatalog/domainModel.html
44 http://martinfowler.com/eaaDev/PresentationModel.html

Presentation Model pulls the state and behavior of the view out into a model class that is part of the presentation. The Presentation Model coordinates with the domain layer and provides an interface to the view that minimizes decision making in the view. The view either stores all its state in the Presentation Model or synchronizes its state with Presentation Model frequently

Presentation Model may interact with several domain objects, but Presentation Model is not a GUI friendly facade to a specific domain object. Instead it is easier to consider Presentation Model as an abstract of the view that is not dependent on a specific GUI framework. While several views can utilize the same Presentation Model, each view should require only one Presentation Model. In the case of composition a Presentation Model may contain one or many child Presentation Model instances, but each child control will also have only one Presentation Model.

Presentation Model is known to users of Visual Works Smalltalk as Application Model

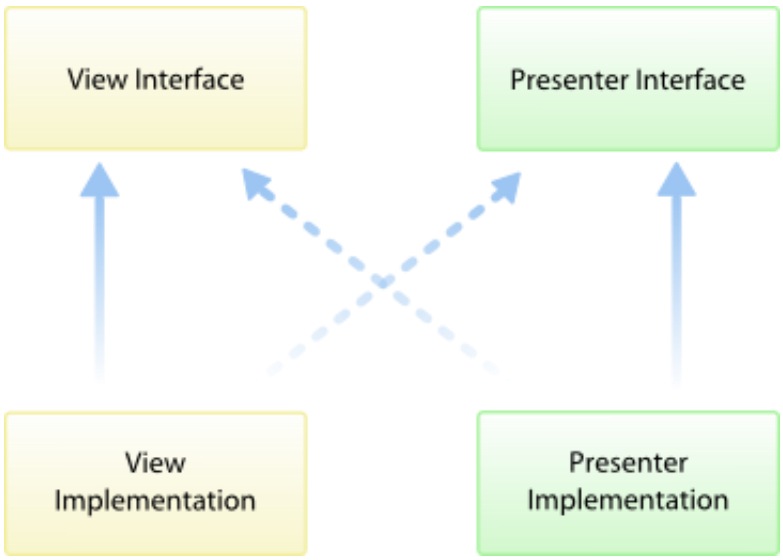
The essence of a Presentation Model is of a fully self-contained class that represents all the data and behavior of the UI window, but without any of the controls used to render that UI on the screen. A view then simply projects the state of the presentation model onto the glass.

To do this the Presentation Model will have data fields for all the dynamic information of the view. This won't just include the contents of controls, but also things like whether or not they are enabled. In general the Presentation Model does not need to hold all of this control state (which would be lot) but any state that may change during the interaction of the user. So if a field is always enabled, there won't be extra data for its state in the Presentation Model.

Since the Presentation Model contains data that the view needs to display the controls you need to synchronize the Presentation Model with the view. This synchronization usually needs to be tighter than synchronization with the domain - screen synchronization is not sufficient, you'll need field or key synchronization.

3.7.2.2.5. Reverse Model – View - Presenter or View Delegate[45]

In View Delegate or Reverse Model – View - Controller, the view handles the events, but delegates them to the presenter. This is of particular merit if you're using GWT UiBinder and its support for @UiHandler to reduce boilerplate code, and can be seen in Large-Scale Application Development and MVP. In this model, the view will respond to the events of its controls, but elegate the processing of that event to the Presenter.



45 http://blog.codiform.com/2011/03/view-presenter-interaction-patterns-in.html

Reverse Model – View - Presenter

This event delegate style fits most naturally as an extension of the *Interaction View*.

3.7.2.3. Event Bus or Event Aggregator[46]

Channel events from multiple objects into a single object to simplify registration for clients.

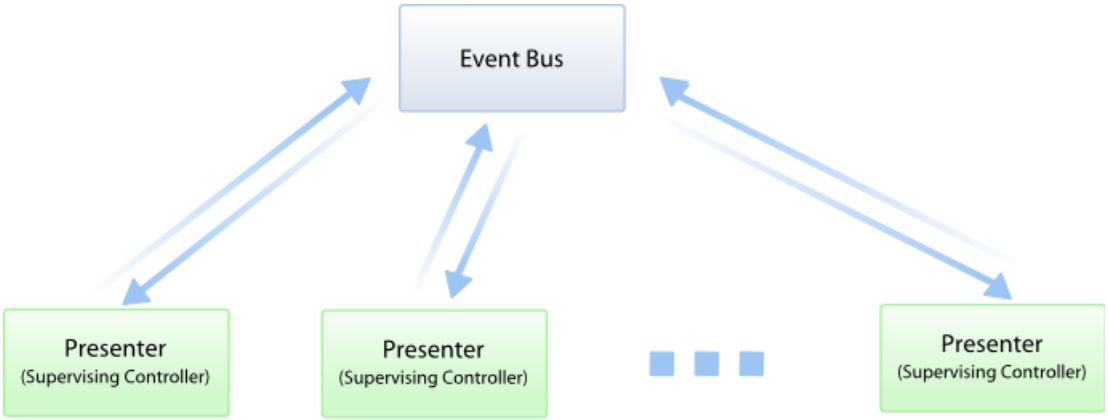
A system with lots of objects can lead to complexities when a client wants to subscribe to events. The client has to find and register for each object individually; if each object has multiple events then each event requires a separate subscription.

An Event Aggregator acts as a single source of events for many objects. It registers for all the events of the many objects allowing clients to register with just the aggregator.

An Event Aggregator is a simple element of indirection. In its simplest form you have it register with all the source objects you are interested in, and have all target objects register with the Event Aggregator. The Event Aggregator responds to any event from a source object by propagating that event to the target objects.

The simplest Event Aggregator aggregates events from multiple objects into itself, passing that same event onto its observers. An Event Aggregator can also generalize the event, converting events that are specific to a source object into a more generic event. That way the observers of the aggregators don't need to register for as many individual event types. This simplifies the registration process for observers, at the cost of being notified of events that may not have any material effect on the observer.

Since an Event Aggregator is based around observer, it's important to take into account all of the danger areas with observer.



3.7.2.4. Dependency Injection[47]

GIN (GWT INjection) brings automatic dependency injection to Google Web Toolkit client-side code. GIN is built on top of Guice[48] and uses (a subset of) Guice's binding language.

46 http://martinfowler.com/eaaDev/EventAggregator.html
47 https://code.google.com/p/google-gin/
48 https://code.google.com/p/google-guice/

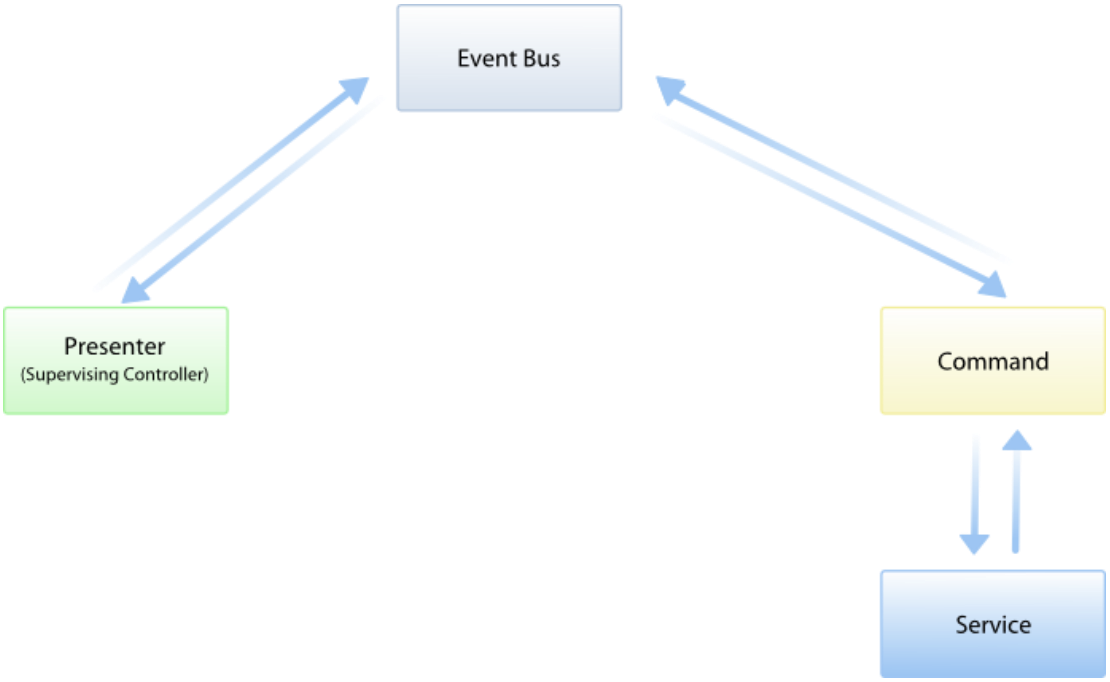
(See GuiceCompatibility for details.) By using GWT's compile-time Generator support, GIN has little-to-no runtime overhead compared to manual DI.

3.7.2.5. Command Pattern[49]

The Command pattern is known as a behavioural pattern, as it's used to manage algorithms, relationships and responsibilities between objects. The definition of Command provided in the original Gang of Four book on Design Patterns states:

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations

Command declares an interface for all commands, providing a simple method which asks the Receiver of the command to carry out an operation. The Receiver has the knowledge of what to do to carry out the request. The Invoker holds a command and can get the Command to execute a request by calling the execute method. The Client creates ConcreteCommands and sets a Receiver for the command. The ConcreteCommand defines a binding between the action and the receiver. When the Invoker calls execute the ConcreteCommand will run one or more actions on the Receiver.



The Command Pattern is useful when:

- You need callback functionality
- The invoker should be decoupled from the object handling the invocation.

The command component could Implement:

- Caching
- Batching

49 http://java.dzone.com/articles/design-patterns-command

- Security

3.7.2.6. History Management[50]

Ajax applications sometimes fail to meet user's expectations because they do not interact with the browser in the same way as static web pages. This is often apparent — and frustrating for users — when an Ajax application does not integrate with browser history. For example, users expect browsers to be able to navigate back to previous pages visited using back and forward actions. Because an Ajax application is a usually single page running JavaScript logic and not a series of pages, the browser history needs help from the application to support this use case. Thankfully, GWT's history mechanism makes history support fairly straightforward.

GWT's History mechanism has a lot in common with other Ajax history implementations, such as RSH (Really Simple History)[51] The basic premise is to keep track of the application's "internal state" in the url fragment identifier. This works because updating the fragment doesn't typically cause the page to be reloaded.

This approach has several benefits:

- It's about the only way to control the browser's history reliably.
- It provides good feedback to the user.
- It's "bookmarkable". I.e., the user can create a bookmark to the current state and save it, email it, et cetera.

GWT includes a mechanism to help Ajax developers activate browser history. For each page that is to be navigable in the history, the application should generate a unique history token. A token is simply a string that the application can parse to return to a particular state. This token will be saved in browser history as a URL fragment (in the location bar, after the "#"), and this fragment is passed back to the application when the user goes back or forward in history, or follows a link.

For example, a history token named "page1" would be added to a URL as follows:

```
http://www.example.com/com.example.gwt.HistoryExample/HistoryExample.html#page1
```

When the application wants to push a placeholder onto the browser's history stack, it simply invokes History.newItem(token). When the user uses the back button, a call will be made to any object that was added as a handler with History.addValueChangeHandler(). It is up to the application to restore the state according to the value of the new token.

3.7.2.7. MVP4G [52]

GWT is a very powerful framework that allows you to build efficient applications, especially if you follow the best practices described by Ray Ryan at Google IO 2009:

- **Model View Presenter**
- **Event Bus**
- **Dependency Injection**

Place Service

50 https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsHistory
51 https://code.google.com/p/reallysimplehistory/
52 https://code.google.com/p/mvp4g/

However, following these best practices is not always easy and you can end up with a project with a lot of boilerplate code that is hard to manage.

That's why Mvp4g offers a solution to following these best practices using simple mechanisms that only need a few lines of code and a few annotations.

For example, this is all you need to create an event bus with four events:

```
@Events( startView = CompanyListView.class, module = CompanyModule.class
)
public interface CompanyEventBus extends EventBus {
        @Event( handlers = CompanyEditPresenter.class )
        public void goToEdit( CompanyBean company );

        @Event( handlers = CompanyDisplayPresenter.class )
        public void goToDisplay( CompanyBean company );

        @Event( handlers = { CompanyListPresenter.class,
CompanyDisplayPresenter.class } )
        public void companyCreated( CompanyBean newBean );

        @Event( handlers = CompanyListPresenter.class )
        public void companyDeleted( CompanyBean newBean );
}
```

3.7.2.7.1. MVP4G MVP Features

- Create a presenter and inject a view with one annotation
- Inject anything you want to your presenters/views thanks to GIN
- Support for multiple instances of the same presenter
- Easily implement the Reverse MVP (or View Delegate) pattern thanks to Reverse View feature
- Easily control your presenter thanks to onBeforeEvent, onLoad and onUnload methods (thanks to the Cycle Presenter feature)

3.7.2.7.2. MVP4G Event Bus Features

- Create an event bus using a few annotations and one centralized interface where you can easily manage your events
- Control your event flow thanks to event filtering, event logs, event broadcast, passive event
- Have the same control of user's navigation as the GWT Activities/Place architecture thanks to Navigation Event

3.7.2.7.3. MVP4G History Management/Place Service Features

- Convert any event to history token thanks to simple history converters
- Support for crawlable urls
- Easily customize your place service
- Support for hyperlink token

3.7.2.7.4. MVP4G Other Features

Not only does Mvp4g help you follow the best practices, it also provides mechanisms to build fast applications:

- Support for GWT code splitting feature: Easily divide your applications into smaller modules thanks to Multi-Modules feature or split one or a few presenters thanks to Splitter.
- Support for lazy loading: Build your presenters/views only when you need them. Useless presenters/views are also automatically removed.

3.7.2.8. UI Binder[53]

At heart, a GWT application is a web page. And when you're laying out a web page, writing HTML and CSS is the most natural way to get the job done. The UiBinder framework allows you to do exactly that: build your apps as HTML pages with GWT widgets sprinkled throughout them.

Besides being a more natural and concise way to build your UI than doing it through code, UiBinder can also make your app more efficient. Browsers are better at building DOM structures by cramming big strings of HTML into innerHTML attributes than by a bunch of API calls. UiBinder naturally takes advantage of this, and the result is that the most pleasant way to build your app is also the best way to build it.

```
<!-- HelloWorld.ui.xml -->

<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'>
  <div>
    Hello, <span ui:field='nameSpan'/>.
  </div>
</ui:UiBinder>
```

```
public class HelloWorld extends UIObject {
  interface MyUiBinder extends UiBinder<DivElement, HelloWorld> {}
  private static MyUiBinder uiBinder = GWT.create(MyUiBinder.class);

  @UiField SpanElement nameSpan;

  public HelloWorld() {
    setElement(uiBinder.createAndBindUi(this));
  }

  public void setName(String name) { nameSpan.setInnerText(name); }
}
```

UiBinder:

- Helps productivity and maintainability — It's easy to create UI from scratch or copy/paste across templates;
- Makes it easier to collaborate with UI designers who are more comfortable with XML, HTML and CSS than Java source code;
- Provides a gradual transition during development from HTML mocks to real, interactive UI;
- Encourages a clean separation of the aesthetics of your UI (a declarative XML template) from its programmatic behavior (a Java class);
- Performs thorough compile-time checking of cross-references from Java source to XML and vice-versa;
- Offers direct support for internationalization that works well with GWT's i18n facility; and
- Encourages more efficient use of browser resources by making it convenient to use lightweight HTML elements rather than heavier-weight widgets and panels.

But as you learn what UiBinder is, you should also understand what it is not. It is not a renderer, or at any rate that is not its focus. There are no loops, no conditionals, no if statements in its markup,

53 https://developers.google.com/web-toolkit/doc/latest/DevGuideUiBinder

and only a very limited expression language. UiBinder allows you to lay out your user interface. It's still up to the widgets or other controllers themselves to convert rows of data into rows of HTML.

3.7.2.9. GWT Editor Framework[54]

- The GWT Editor framework allows data stored in an object graph to be mapped onto a graph of Editors. The typical scenario is wiring objects returned from an RPC mechanism into a UI.
- GWT Editor Framework:
- Decrease the amount of glue code necessary to move data from an object graph into a UI and back.
- Be compatible with any object that looks like a bean, regardless of its implementation mechanism (POJO, JSO, RPC, RequestFactory).
- Support arbitrary composition of Editors.
- For post-GWT 2.1 release, establish the following trajectories:
- Create an API that can be used by UiBinder
- Support client-side JSR 303 Validation when it's available

3.7.2.10. Validations[55]

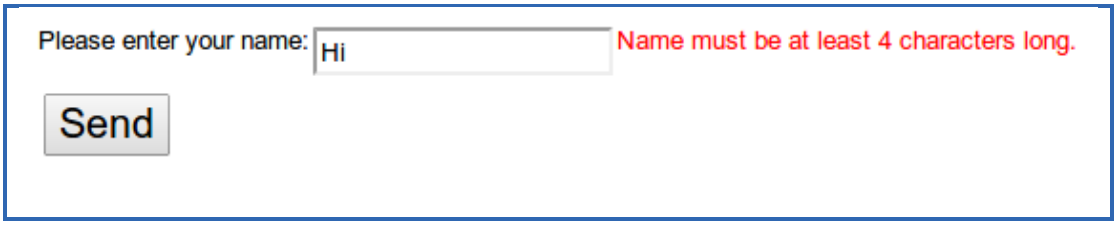
An important part of any application that accepts input is the process of validating that input to ensure it is correct and valid. This validation can be done in several ways, including manually inserting conditionals every time any input is received, but this process can be tedious and difficult to maintain.

To make data validation simpler, GWT supports JSR-303 Bean Validation[56] which provides a framework for specifying "constraints" that define what data is valid or allowed for your application.

For example, by specifying the following annotations on your class:

```
public class Person implements Serializable {
  @NotNull
  @Size(min = 4, message = "Name must be at least 4 characters long.")
  private String name;
}
```

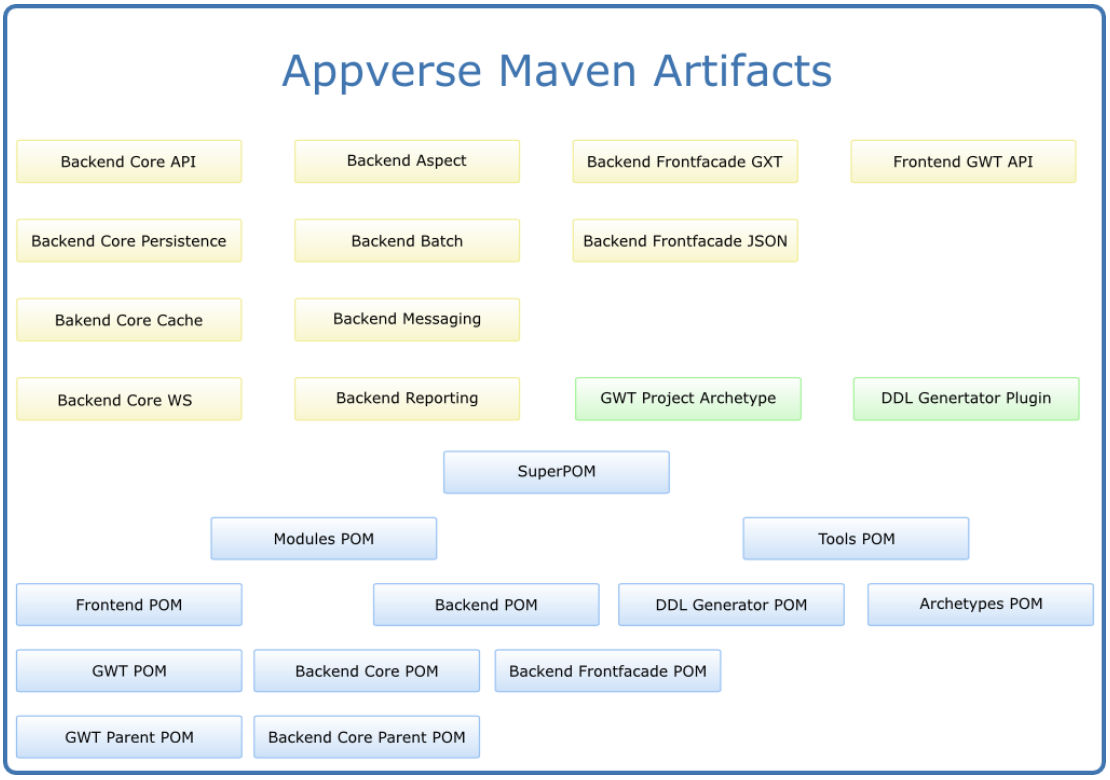
You can ensure these conditions are met, as well as generate meaningful errors when they are not:



At compile time GWT validation creates a Validator for all your objects that can be used to do client-side validation. If you are using a Java-based server the very same constraints can be used to do server-side data validation as well.

54 https://developers.google.com/web-toolkit/doc/latest/DevGuideUiEditors
55 https://developers.google.com/web-toolkit/doc/latest/DevGuideValidation
56 http://jcp.org/en/jsr/detail?id=303

4.1. Overview


Appverse Maven Artifacts
Backend Core API
Backend Aspect
Backend Frontfacade GXT
Frontend GWT API
Backend Core Persistence
Backend Batch
Backend Frontfacade JSON
Bakend Core Cache
Backend Messaging
Backend Core WS
Backend Reporting
GWT Project Archetype
DDL Genertator Plugin
SuperPOM
Modules POM
Tools POM
Frontend POM
Backend POM
DDL Generator POM
Archetypes POM
GWT POM
Backend Core POM
Backend Frontfacade POM
GWT Parent POM
Backend Core Parent POM

4.2. Backend Core API

4.2.1. AOP

4.2.1.1. Advices

org.appverse.web.framework.backend.api.aop.advices

org.appverse.web.framework.backend.api.aop.advices.technical

4.2.1.1.1. Profiling

Profiling -> ProfilingAdvice

4.2.1.1.2. Exception

Exception Handling -> ExceptionAdvice

4.2.1.2. Managers

org.appverse.web.framework.backend.api.aop.managers

org.appverse.web.framework.backend.api.aop.managers.impl.live

4.2.1.2.1. Profiling

4.2.1.2.2. Exception

4.2.1.3. Logging

4.2.1.3.1. Autowired Logger

In order to have the proper logger or category for every class to allow a correct configuration for the logger implementation, Appverse Web has the `@AutowiredLogger` Runtime Field annotation.

```
package org.appverse.web.framework.backend.api.helpers.log;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface AutowiredLogger {
}
```

In a Spring bean that use slf4j `Logger` to use AutowiredLogger feature annotate the logger field with `@AutowiredLogger` annotation. Don't initialize the logger with the class object.

```
@Service("mailIntegrationService")
public class MailIntegrationServiceImpl extends AbstractIntegrationService
        implements MailIntegrationService {

    @AutowiredLogger
    private static Logger logger;
…
```

In Spring configuration add the `AutowiredLoggerBeanPostProcessor` bean:

```
<bean
class="org.appverse.web.framework.backend.api.helpers.log.AutowiredLoggerBeanPostProcessor" />
```

The `AutowiredLoggerBeanPostProcessor` implements the Spring BeanPostProcessor interface; All the beans included to the Spring Application Context after the it will be processed inicialitating the fields with `@AutowiredLogger` with a Logger containing the proper bean class.

```
package org.appverse.web.framework.backend.api.helpers.log;
import java.lang.reflect.Field;
import org.slf4j.LoggerFactory;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.util.ReflectionUtils;
public class AutowiredLoggerBeanPostProcessor implements BeanPostProcessor {
    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName)
            throws BeansException {
        return bean;
    }
    @Override
    public Object postProcessBeforeInitialization(final Object bean,
            String beanName) throws BeansException {
        ReflectionUtils.doWithFields(bean.getClass(),
                new ReflectionUtils.FieldCallback() {
                    @Override
                    public void doWith(Field field)
                            throws IllegalArgumentException,
                            IllegalAccessException {
                        if (field.isAnnotationPresent(AutowiredLogger.class)) {
                            field.setAccessible(true);
                            field.set(bean,
                                    LoggerFactory.getLogger(bean.getClass()));
                        }

                    }
                });
        return bean;
    }
}
```

4.2.1.3.2. Eclipselink Logger

Eclipselink uses java.util.logging logger and Appserse Web as a desing chooise use a log4j logger using the slf4j façade. To redirect the logs writted by eclipselink to log4j logger along the use of slf4j and the set of depenciencies to do so, (already indluded as transitive dependencies) is required to include the follow property at JPA persistence.xml:

```
<property name="eclipselink.logging.logger" val-
ue="org.appverse.web.framework.backend.persistence.helpers.log.EclipselinkLogger"/>
```

With this property eclipselink will use the Appverse Web `EclipselinkLogger` bridge between java.util.logging and log4j.

```
package org.appverse.web.framework.backend.persistence.helpers.log;
import org.eclipse.persistence.logging.AbstractSessionLog;
import org.eclipse.persistence.logging.SessionLog;
import org.eclipse.persistence.logging.SessionLogEntry;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class EclipselinkLogger extends AbstractSessionLog implements
SessionLog {
    public static final Logger LOG = LoggerFactory
            .getLogger(EclipselinkLogger.class);
```

```
    @Override
    public void log(SessionLogEntry sessionLogEntry) {
        switch (sessionLogEntry.getLevel()) {
        case SEVERE:
            LOG.error(sessionLogEntry.getMessage());
            break;
        case WARNING:
            LOG.warn(sessionLogEntry.getMessage());
            break;
        case INFO:
            LOG.info(sessionLogEntry.getMessage());
            break;
        default:
            LOG.debug(sessionLogEntry.getMessage());
        }
    }
}
```

Then at log4j.properties (or log4j.xml) is possible to control the verbosity of eclipselink with the following category:

```
log4j.logger.org.appverse.web.framework.backend.persistence.helpers.log.EclipselinkLogger=deb
ug
```

4.2.1.3.3. Log4j Spring Initialization Listener

If Spring is used and the application is not deployed in a contaniner that don't expand the war or ear archive then the Log4j Spring Initialization Listener is the prefered approach to initializate log4j. The follow snippet has to be included at web.xml file before the Spring Application Context initialization context listener and context-param

```
<listener>
    <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
</listener>
<context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>classpath:log4j/log4j.xml</param-value>
</context-param>
```

One of the adventages of this approach is that enables the use of `=${webapp.root}` variable at log4j configuration referring the classpath of the current application. Please consult the online documentation if you want to use this approach for more than one application in a web container to find the alternative approach for these cases.

```
log4j.appender.RE = org.apache.log4j.DailyRollingFileAppender
log4j.appender.RE.Threshold=error
log4j.appender.RE.File = ${webapp.root}/WEB-INF/logs/gwtshowcase.error.log
log4j.appender.RE.Append = true
log4j.appender.RE.DatePattern = '.'yyy-MM-dd
log4j.appender.RE.layout = org.apache.log4j.PatternLayout
log4j.appender.RE.layout.ConversionPattern = [GWT Showcase] [%p] %d{yyyy-MM-dd
HH:mm:ss,SSS} [%c{1}.%M(%L)]  %m%n
```

4.2.1.3.4. Log4j Initialization Servlet

There are cases with the log4j spring managed configuration for log4j is not possible, as when the application is deployed inside a container that no expands the war or ear achive, then the servlet initialization approach is recommended to be used. The follow snippet has to be included at web.xml.

```
<servlet>
    <servlet-name>log4j-init</servlet-name>
    <servlet-class>org.appverse.web.framework.backend.api.helpers.log.Log4jInit</servlet-
class>
    <init-param>
        <param-name>log4j-init-file</param-name>
        <param-value>classpath:/log4j/log4j.properties</param-value>
     </init-param>
     <load-on-startup>1</load-on-startup>
</servlet>
```

Appverse Web framework the servlet implementation to configure log4j properly.

```
package org.appverse.web.framework.backend.api.helpers.log;
import java.io.InputStream;
import java.util.Properties;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.PropertyConfigurator;
public class Log4jInit extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @Override
    public void init(ServletConfig config) throws ServletException {
        String log4jLocation = config.getInitParameter("log4j-init-file");
        if (log4jLocation == null) {
            BasicConfigurator.configure();
        } else {
            InputStream is = this.getClass().getResourceAsStream(
                    log4jLocation.substring(log4jLocation.indexOf(':') + 1));
            if (is != null) {
                Properties properties = new Properties();
                try {
                    properties.load(is);
                } catch (Exception e) {
                    BasicConfigurator.configure();
                }
                PropertyConfigurator.configure(properties);
            } else {
                BasicConfigurator.configure();
            }
        }
        super.init(config);
    }
}
```

4.2.1.4. JPA

4.2.1.4.1. JPAPersistenceService

Class `JPAPersistenceService` defines the basic methods needed to interact to a JPA Flavoured Tier in a JPA Flavored Layer. The following table contains the complete list of `JPAPersistenceService` methods grouped by their pourpose and type of arguments:

	.	Pk	Beans	Query String	Query String + Parameters	Query String+ Parameters + pagination	IntegrationDataFiler	QueryName	QueryJPACallBack
Retrieve		X	X	X	X		X	X	X
Retrieve List	X			X	X	X	X	X	X
Count	X			X	X		X	X	X
Retrieve Result Type List				X	X	X		X	X
Count Result Type				X	X			X	X
Retrieve Result Object Array List				X	X	X		X	X
Count Result Object Array				X	X			X	X
Update				X	X			X	X
Persist			X						
Delete			X						
Delete All	X								
Detach			X						
Contains			X						
Flush	X								

4.2.1.4.1.1. Retrieve Methods

The retrieve methods are used for retrieving a entity from the underlying DBMS using the JPA EntityManager. These methods are thinked to retrieve no more than one element. Remember that entities recovered using retrieving methods are in managed state and if are modificated and not detached those changes will be persisted when the active transaction commits.

- T retrieve(long pk) throws Exception;
- T retrieve(T bean) throws Exception;
- T retrieve(final String queryString) throws Exception;
- T retrieve(final String queryString, final Map<String, Object> parameters) throws Exception;
- T retrieve(final IntegrationDataFilter filter) throws Exception;
- T retrieve(final String queryName, final Object... values) throws Exception;

- T retrieve(QueryJpaCallback<T> query) throws Exception;

4.2.1.4.1.2. Retrieve List Methods

RetriveList metods returns a list of the entity where are defined, the result could be only one element without raise any exception.

- List<T> retrieveList() throws Exception;
- List<T> retrieveList(String queryString) throws Exception;
- List<T> retrieveList(String queryString, Map<String, Object> parameters)throws Exception;
- List<T> retrieveList(final String queryString, final Map<String, Object> parameters, final int maxRecords, final int firstResult) throws Exception;
- List<T> retrieveList(IntegrationDataFilter filter) throws Exception
- List<T> retrieveList(final String queryName, final Object... values) throws Exception;
- List<T> retrieveList(QueryJpaCallback<T> query) throws Exception;

4.2.1.4.1.3. Count Methods

Counts methods returns the number of elements that will return a query to the DBMS without returning the actual elements and without the subsequent latency

- int count() throws Exception;
- int count(String queryString) throws Exception;
- int count(final String queryString, final Map<String, Object> parameters) throws Exception;
- int count(final IntegrationDataFilter filter) throws Exception;
- int count(final String queryName, final Object... values) throws Exception;
- int count(QueryJpaCallback<T> query) throws Exception;

4.2.1.4.1.4. Retrieve Result Type List Methods

RetriveResultTypeList metods returns a list of a ResultType created with a new JPQL operator. The resulting bean has to extend ResultIntegrationBean class.The result could be only one element without raise any exception.

- <U extends ResultIntegrationBean> List<U> retrieveResultTypeList (String queryString) throws Exception;
- <U extends ResultIntegrationBean> List<U> retrieveResultTypeList(String queryString, Map<String, Object> parameters) throws Exception;
- <U extends ResultIntegrationBean> List<U> retrieveResultTypeList(final String queryString, final Map<String, Object> parameters,final int maxRecords, final int firstResult) throws Exception;
- <U extends ResultIntegrationBean> List<U> retrieveResultTypeList(final String queryName, final Object... values) throws Exception;
- <U extends ResultIntegrationBean> List<U> retrieveResultTypeList (QueryJpaCallback<U> query) throws Exception;

4.2.1.4.1.5. Count Result Type Methods

- CountResultType methods return the number of elements that will return a query to the DBMS without returning the actual elements and without the subsequent latency.
- <U extends ResultIntegrationBean> int countResultType(final String queryString) throws Exception;
- <U extends ResultIntegrationBean> int countResultType(final String queryString, final Map<String, Object> parameters) throws Exception;

- public <U extends ResultIntegrationBean> int countResultType(String queryString) throws Exception {
- <U extends ResultIntegrationBean> int countResultType(final String queryName, final Object... values) throws Exception;
- <U extends ResultIntegrationBean> int countResultType(QueryJpaCallback<U> query) throws Exception;

4.2.1.4.1.6. Retrieve Result Object Array List Methods

RetriveResultTypeList metods returns a list tuples of a convination of JPA entities, Result Types and java simple types.The result could be only one element without raise any exception.

- List<Object[]> retrieveObjectArrayList(final String queryString) throws Exception;
- List<Object[]> retrieveObjectArrayList(final String queryString, final Map<String, Object> parameters) throws Exception;
- List<Object[]> retrieveObjectArrayList(final String queryString, final Map<String, Object> parameters, final int maxRecords, final int firstResult) throws Exception;
- List<Object[]> retrieveObjectArrayList(final String queryName, final Object... values) throws Exception;
- List<Object[]> retrieveObjectArrayList(QueryJpaCallback<Object[]> query) throws Exception

4.2.1.4.1.7. Count Result Object Array Methods

CountObjectArray methods return the number of elements that will return a query to the DBMS without returning the actual elements and without the subsequent latency.

- int countObjectArray(String queryString) throws Exception;
- int countObjectArray(final String queryString, final Map<String, Object> parameters) throws Exception;
- int countObjectArray(final String queryName, final Object... values) throws Exception;
- int countObjectArray(QueryJpaCallback<Object[]> query) throws Exception;

4.2.1.4.1.8. Update Methods

Update methods are used to perform queries that have a void return type and modifies the entity where are defined.

- void update(String queryString) throws Exception;
- void update(String queryString, Map<String, Object> parameters) throws Exception;
- void update(final String queryName, final Object... values) throws Exception;
- void update(UpdateJpaCallback<T> query) throws Exception;

4.2.1.4.1.9. Other Methods

The last group of methods are intended for several pourposes:

Persist method change the status of a entity to managed if the bean status is not already managed.

- long persist(T bean) throws Exception;

Detach method change the status of a entity to detached if the bean status is not already detached.

- void detach(final T beanP) throws Exception;

Contains returns if the entity belongs to current persistence context.

- boolean contains(final T beanP) throws Exception;

Delete removes the entity from database in the next commit or flush operation.

- void delete(T bean) throws Exception;

Delete all removes the entities of the entity type from database in the next commit or flush operation.

- void deleteAll() throws Exception;

Refresh the state of the instance from the database, overwriting changes made to the entity, if any.

- void refresh(final T beanP) throws Exception;

JPA providers can cache the SQL instructions they are supposed to send to the database, often until you actually commit the transaction. For example, you call em.persist(), the provider remembers it has to make a database INSERT, but does not actually execute the instruction until you commit the transaction. Afaik, this is mainly done for performance reasons.

In some cases anyway you want the SQL instructions to be executed immediately; generally when you need the result of some side effects, like an autogenerated key, or a database trigger.

What flush() does is to empty the internal SQL instructions cache, and execute it immediately to the database.

Bottom line: no harm is done, only you could have a performance hit since you are overriding the JPA provider decisions as regards the best timing to send SQL instructions to the database.

- void flush() throws Exception;

4.2.1.4.2. EntityListener

Class `EntityListener` use JPA Default Listener explained on sectuon 3.6.1.7.4 on this document defining two listener methods `prePersist` and `preUpdate`. To apply these listener methods to all JPA entities the `EntityListener` class the approach used is define it at META-NF/orm.xml file:

```
log4j.logger.org.appverse.web.framework.backend.persistence.helpers.log.EclipselinkLogger=deb
```

```
<entity-mappings  xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_2_0.xsd"
                 version="2.0">
    <persistence-unit-metadata>
        <persistence-unit-defaults>
            <entity-listeners>
                <entity-listener
class="org.appverse.web.framework.backend.persistence.services.integration.helpers.EntityLi
stener">
                    <pre-persist method-name="prePersist"/>
                    <pre-update method-name="preUpdate"/>
                </entity-listener>
            </entity-listeners>
        </persistence-unit-defaults>
    </persistence-unit-metadata>
</entity-mappings>
```

Next snippet show the `EntityListener` class defined at framework level as explained on section 3.5.7.2 on this document this class is configured to store a partial auditing of changes in the same row storing the user and timestamp of row creation and modification

```
package org.appverse.web.framework.backend.persistence.services.integration.helpers;

import javax.persistence.PrePersist;
import javax.persistence.PreUpdate;
import org.appverse.web.framework.backend.api.helpers.util.GMTTimeHelper;
import
org.appverse.web.framework.backend.persistence.model.integration.AbstractIntegrationAudited
JPABean;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
public class EntityListener {
    public String getUsername() {
        Authentication authentication = SecurityContextHolder.getContext()
                .getAuthentication();
        String username = null;
        if (authentication == null) {
            username = "batch_process";
        } else {
            User user = (User) authentication.getPrincipal();
            username = user.getUsername();
        }
        return username;
    }
    @PrePersist
    public void prePersist(AbstractIntegrationAuditedJPABean integrationBean) {
        Date timestamp = GMTTimeHelper.toGMTDate(new Date());
        integrationBean.setCreatedBy(getUsername());
        integrationBean.setCreated(timestamp);
        integrationBean.setUpdatedBy(getUsername());
        integrationBean.setUpdated(timestamp);
        integrationBean.setVersion(1);
        if (integrationBean.getStatus() == null) {
            integrationBean
                    .setStatus(AbstractIntegrationAuditedJPABean.STATUS_ACTIVE);
        }
    }

    @PreUpdate
    public void preUpdate(AbstractIntegrationAuditedJPABean integrationBean) {
        Date timestamp = GMTTimeHelper.toGMTDate(new Date());
        integrationBean.setUpdatedBy(getUsername());
        integrationBean.setUpdated(timestamp);
    }
}
```

The listener methods only applies to entities that subclass AstractIntegrationAuditedJPABean. A JPA MappedSupperclass that ensures that the fields and mapping database columns required to perform the partial auditing exists.

```
package org.appverse.web.framework.backend.persistence.model.integration;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.MappedSuperclass;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import
org.appverse.web.framework.backend.api.model.integration.AbstractIntegrationAuditedBean;
@MappedSuperclass
public abstract class AbstractIntegrationAuditedJPABean extends
        AbstractIntegrationAuditedBean {
    private static final long serialVersionUID = -2070164067618480118L;
    protected long id;
    // Always add condition "if (id == 0 || id != other.id)" so in case that
    // Dozer non-cummulative collections and remove-orphans in mappings works
    // fine
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        AbstractIntegrationAuditedJPABean other = (AbstractIntegrationAuditedJPABean) obj;
        if (id == 0 || id != other.id) {
            return false;
        }
        return true;
    }
    @Override
    public void finalize() throws Throwable {
    }

    @Override
    @Temporal(TemporalType.TIMESTAMP)
    @Column(nullable = false)
    public Date getCreated() {
        return created;
    }

    @Override
    @Column(name = "CREATED_BY", nullable = false, length = 100)
    public String getCreatedBy() {
        return createdBy;
    }

    @Override
    @Column(nullable = false)
    public String getStatus() {
        return status;
    }

    @Override
    @Temporal(TemporalType.TIMESTAMP)
    public Date getUpdated() {
        return updated;
    }

    @Override
    @Column(name = "UPDATED_BY", length = 100)
    public String getUpdatedBy() {
        return updatedBy;
    }
```

```
    @Override
    @Column(nullable = false)
    public long getVersion() {
        return version;
    }
    // Required so that Dozer non-cummulative collections and remove-orphans in
    // mappings works fine
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + (int) (id ^ id >>> 32);
        return result;
    }

    public void setId(long id) {
        this.id = id;
    }
}
```

4.2.2. Request Handling

4.2.3. Controllers

4.2.3.1. GWTRPCController

4.2.3.2. Session Manager

4.3. Famework Converters

The follow list descrives the framework classes related with conversion:

Framework abstract classes / interfaces	Description
IBeanConverter	Top hierarchy converter interface common to all sorts of converters
IP2BBeanConverter	Interface for Business to Integration (B2I) converters
AbstractDozerP2IBeanConverter	Abstract class that any Presentation to Integration (P2I) converter must extend. Implements IP2IBeanConverter interface. This class is key to allow multiple solutions: Dozer converters, custom converters, other solutions
ConversionType	Allows a converter to define different scopes or conversion levels (full object, specifying which dependencies to convert, for example)
ConvertionException	Custom exception triggered by converters
IBeanConverter	Top hierarchy converter interface common to all sorts of converters

Table 1 Framework Converters

4.3.1. Common

4.3.2. Controllers

4.3.3. Coverters

4.3.4. Helpers

4.3.5. Model

5.1. Presentation Tier Building Blocks

5.1.1. Package Names

Package	Description
[base_package].gwtfrontend.[module].commands	Commands Interfaces
[base_package].gwtfrontend.[module].commands.impl.live	Commands Live Implementations
[base_package].gwtfrontend.[module].commands.impl.mock	Commands Mock Implementations

Table 2 Presentation Tier Package Names

5.1.2. Commands

5.1.2.1. Interface

Conventions

Project [project]-gwtfrontend

Package [base_package].gwtfrontend.[module].commands

Interface [CommandName]RpcCommand

Example

```
package [base_package].gwtfrontend.[module].commands;
import org.appverse.web.framework.frontend.gwt.rpc.ApplicationAsyncCallback;
import [base_package].backend.model.presentation.[BusinessObject]VO;
public interface [CommandName]RpcCommand {
        void load[BusinessObject]VO (long id, ApplicationAsyncCallback<[BusinessObject]VO>
applicationAsyncCallback);
}
```

Remarks

5.1.2.2. Implementation

Conventions

Project	[project]-gwtfrontend
Package	[base_package].gwtfrontend.[module].commands.impl.live
Class	[CommandName]RpcCommandImpl
Extends	org.appverse.web.framework.frontend.gwt.commands.AbstractRpcCommand<EventBus>
Implements	[CommandName]RpcCommand

Example

```
package [base_package].gwtfrontend.[module].commands.impl.live;

import org.appverse.web.framework.frontend.gwt.commands.AbstractRpcCommand;
import org.appverse.web.framework.frontend.gwt.rpc.ApplicationAsyncCallback;

import [base_package].backend.model.presentation.[BusinessObject]VO;
import [base_package].backend.services.presentation.[BusinessObject]ServiceFacade;
import [base_package].backend.services.presentation.[BusinessObject]ServiceFacadeAsync;
import [base_package].gwtfrontend.[module].[EventBusName]EventBus;
import [base_package].gwtfrontend.[module].commands.[CommandName]RpcCommand;
import com.google.gwt.core.client.GWT;

public class [CommandName]RpcCommandImpl extends AbstractRpcCommand<[EventBusName]EventBus>
                implements [CommandName]RpcCommand {

        private final [BusinessObject]ServiceFacadeAsync service = ([BusinessObject] Service-
FacadeAsync) GWT.create([BusinessObject]ServiceFacade.class);

        protected [BusinessObject]ServiceFacadeAsync getService() {
                return super.getService(service);
        }

        @Override
        public void load[BusinessObject]VO (long id,
                        ApplicationAsyncCallback<[BusinessObject]VO>
applicationAsyncCallback){
                getService().load[BusinessObject]VO(id, applicationAsyncCallback);
        }
}
```

5.2. Middle Tier Building Blocks

5.2.1. Package Names

Package	Description
[base_package].backend.model.presentation	Presentation Object Model
[base_package].backend.model.business	Business Object Model
[base_package].backend.model.integration	Integration Object Model
[base_package].backend.services.presentation	Service Façade Interfaces
[base_package].backend.services.presentation.impl.live	Service Façade Live Implementations
[base_package].backend.services.presentation.impl.mock	Service Façade Mock Implementations
[base_package].backend.services.presentation.impl.test	Service Façade Test Implementations
[base_package].backend.services.business	Service Interfaces
[base_package].backend.services.business.impl.live	Service Live Implementations
[base_package].backend.services.business.impl.mock	Service Mock Implementations
[base_package].backend.services.business.impl.test	Service Test Implementations
[base_package].backend.services.integration	Repository Interfaces
[base_package].backend.services.integration.impl.live	Repository Live Implementations
[base_package].backend.services.integration.impl.mock	Repository Mock Implementations
[base_package].backend.services.integration.impl.test	Repository Test Implementations
[base_package].backend.converters.p2b	Presentation to Business Converters
[base_package].backend.converters.b2i	Business to Integration Converters

Table 3 Middle Tier Package Names

5.2.2. Model

5.2.2.1. Presentation

5.2.2.1.1. GWT

Conventions

Project [project]-gwtsharedmodel

Package [base_package].backend.model.presentation

Class [BusinessObject]VO

Extends org.appverse.web.framework.backend.api.model.presentation.AbstractPresentationBean

Example

```
package [base_package].project.backend.model.presentation;
import org.appverse.web.framework.backend.api.model.presentation.AbstractPresentationBean;
public class [BussinesObject]VO extends AbstractPresentationBean {
    // Custom object properties and setters/getters should be defined here
 }
```

Remarks

- It must be a POJO, can contain integration tier specifics details
- The presentation model should be as simple as possible, avoiding complex objects. It must be this way in order to not overload presentation with all the we will not need/use. We should try only retrieving the necessary data.
- This model is shared with frontend project
- See section 3.2.2.1 on this document for more information on presentation object model

5.2.2.1.2. Non-GWT

Conventions

Project	[project]-backend
Package	[base_package].backend.model.presentation
Class	[BusinessObject]VO
Extends	org.appverse.web.framework.backend.api.model.presentation.AbstractPresentationBean

Example

```
package [base_package].project.backend.model.presentation;
import org.appverse.web.framework.backend.api.model.presentation.AbstractPresentationBean;
public class [BussinesObject]VO extends AbstractPresentationBean {
    // Custom object properties and setters/getters should be defined here
 }
```

Remarks

- It must be a POJO, can contain integration tier specifics details
- The presentation model should be as simple as possible, avoiding complex objects. It must be this way in order to not overload presentation with all the we will not need/use. We should try only retrieving the necessary data.
- This model is shared with frontend project
- See section 3.2.2.1 on this document for more information on presentation object model

5.2.2.2. Business

Conventions

Project	[project]-backend
Package	[base_package].backend.model.business
Class	[BusinessObject]
Extends	org.appverse.web.framework.backend.api.model.business.AbstractBusinessBean

Example

```
package [base_package].project.backend.model.business;
import org.appverse.web.framework.backend.api.model.business.AbstractBusinessBean;
public class [BussinesObject] extends AbstractBusinessBean {
    // Custom object properties and setters/getters should be defined here
}
```

Remarks

- It must be a POJO avoiding any technical details
- See section 3.2.2.2 on this document for more information on business object model.

5.2.2.3. Integration

Conventions

Project	[project]-backend
Package	[base_package].backend.model.integration
Class	[BusinessObject]DTO
Extends	org.appverse.web.framework.backend.api.model.integration.AbstractIntegrationBean

Example

```
package [base_package].project.backend.model.integration;
import org.appverse.web.framework.backend.api.model.integration.AbstractIntegrationBean;
public class [BussinesObject]DTO extends AbstractIntegrationBean {
    // Custom object properties and setters/getters should be defined here
}
```

Remarks

- It must be a POJO, can contain integration tier specific details
- If are JPA flavored can contain JPA annotations; the ORM flavored are not allowed to contain specific annotations.
- See section 3.2.2.3 on this document for information on integration model objects.

5.2.3. Services

5.2.3.1. Presentation

5.2.3.1.1. GWT

5.2.3.1.1.1. Interface

Conventions

Project	[project]-gwtsharedmodel
Package	[base_package].backend.services.presentation

Class [BusinessObject]ServiceFacade

Extends org.appverse.web.framework.backend.api.services.presentation.IPresentationService

Example

```
package [base_package].backend.services.presentation;
import org.appverse.web.framework.backend.api.services.presentation.IPresentationService;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;
@RemoteServiceRelativePath("services/[businessObject]ServiceFacade.rpc")
public interface [BusinessObject]ServiceFacade extends IPresentationService, RemoteService
{
// Service facade interface methods should be defined here
}
```

Remarks

- GWT remote service requires a service façade RPC declaration

5.2.3.1.1.2. Implementation

Conventions

Project [project]-backend

Package [base_package].backend.services.presentation.impl.[implementation_type]

Class [BusinessObject]ServiceFacade[ImplementationType]

Extends org.appverse.web.framework.backend.api.services.presentation

.AbstractPresentationService

Implements [base_package].backend.services.presentation.[BusinessObject]ServiceFacade

Stererotype org.springframework.stereotype.Service

Example

```
package [base_package].backend.services.presentation.impl.[implementation_type];
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import [base_package].backend.converters.b2i.[BusinessObject]B2IBeanConverter;
import [base_package].backend.model.business.[BusinessObject];
import [base_package].backend.model.presentation.[BusinessObject]VO;
import [base_package].backend.services.business.[BusinessObject]Service;
import [base_package].backend.services.presentation.[BusinessObject]ServiceFacade;
import
org.appverse.web.framework.backend.api.services.presentation.AbstractPresentationService;

@Service("[businessObject]ServiceFacade")
public class [BusinessObject]ServiceFacadeImpl  extends AbstractPresentationService
       implements [BusinessObject]ServiceFacade {

    @Autowired
    private [BusinessObject]Service [businessObject]Service;

    @Autowired
    private [BusinessObject]P2IBeanConverter [businessObject]P2IBeanConverter;

    // Service interface methods should be implemented here
```

```
}
```

Remarks

- Use of other BusinessServices should be avoided
- Should compose their functionality by calling Integration Services.
- Should only use objects from Business and Integration Model Object domain.
- Should implement atomic business logic.
- Business Service uses B2I (business to integration) converters in order to transform objects between layers.
- Spring configuration defines the current implementation type ("live" by default), through Spring auto scanning components mechanism.
- See section 3.2.1.1 on this document for information on service façades

5.2.3.2. Business

5.2.3.2.1.1. Interface

Conventions

Project	[project]-backend
Package	[base_package].backend.services.business
Class	[BusinessObject]Service

Example

```
package [base_package].backend.services.business;
public interface [BusinessObject]Service {
    // Service interface methods should be defined here
}
```

5.2.3.2.1.2. Implementation

Conventions

Project	[project]-backend
Package	[base_package].backend.services.business.impl.[implementation_type]
Class	[BusinessObject]Service[ImplementationType]
Extends	org.appverse.web.framework.backend.api.services.business.AbstractBusinessService
Implements	[base_package].backend.services.business.[BusinessObject]Service
Stererotype	org.springframework.stereotype.Service

Example

```
package [base_package].backend.services.business.impl.[implementation_type];
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import [base_package].backend.converters.b2i.[BusinessObject]B2IBeanConverter;
import [base_package].backend.model.business.[BusinessObject];
import [base_package].backend.model.integration.[BusinessObject]DTO;
import [base_package].backend.services.business.[BusinessObject]Service;
import [base_package].backend.services.integration.[BusinessObject]Repository;
import [base_package].backend.services.integration.[BusinessObject]Mapper;
import org.appverse.web.framework.backend.api.services.business.AbstractBusinessService;

@Service("[businessObject]Service")
public class [BusinessObject]ServiceImpl extends AbstractBusinessService
       implements [BusinessObject]Service {

  @Autowired
  private [BusinessObject]Repository [businessObject]Repository;
  @Autowired
  private [BusinessObject]DataMapper [businessObject]DataMapper;
  @Autowired
  private [BusinessObject]B2IBeanConverter [businessObject]B2IBeanConverter;
  // Service interface methods should be implemented here
}
```

Remarks

- Use of other BusinessServices should be avoided
- Should compose their functionality by calling Integration Services.
- Should only use objects from Business and Integration Model Object domain.
- Should implement atomic business logic.
- Business Service uses B2I (business to integration) converters in order to transform objects between layers.
- Spring configuration defines the current implementation type ("live" by default), through Spring auto scanning components mechanism.
- See section 3.2.1.2 on this document for information on services

5.2.3.3. Integration

5.2.3.3.1. JPA

5.2.3.3.1.1. Interface

Conventions

Project	[project]-backend
Package	[base_package].backend.services.integration
Class	[BusinessObject]Repository
Extends	org.appverse.web.framework.backend.persistence.services.integration.IJPAPersistenceService

Example

```
package [base_package].backend.services.integration;
import
org.appverse.web.framework.backend.persistence.services.integration.IJPAPersistenceService;
import [base_package].backend.model.integration.[BusinessObject]DTO;
public interface [BusinessObject]Repository extends IJPAPersistenceSer-
vice<[BusinessObject]DTO> {
     // Custom interface methods should be defined here
}
```

Remarks

- Repository inherits from IJPAPersistenceService interface the default CRUD operations.
- It's possible to declare custom methods to achieve complex operations.

5.2.3.3.1.2. Implementation

Conventions

Project	[project]-backend
Package	[base_package].backend.services.integration.impl.[implementation_type]
Class	[BusinessObject]Repository[ImplementationType]
Extends	org.appverse.web.framework.backend.persistence.services.integration.impl.live. JPAPersistenceService
Implements	[base_package].backend.services.integration.[BusinessObject]Repository
Stererotype	org.springframework.stereotype.Repository

Example

```
package [base_package].backend.services.integration.impl.[implementation_type]
import org.springframework.stereotype.Repository;
import [base_package].backend.model.integration[BusinessObject]DTO;
import [base_package].backend.services.integration.impl.[BusinessObject]Repository;
import org.appverse.web.framework.backend.persistence.services.integration.impl.live.
JPAPersistenceService;
@Repository("[businessObject]Repository")
public class [BusinessObject]RepositoryImpl extends JPAPersistence-
Sevice<[BusinessObject]DTO>
          implements [BusinessObject]Repository {
    // Custom interface methods should be implemented here
}
```

Remarks

- DAO Service inherits from JPAPersistenceService implementation the default CRUD operations over integration bean (JPA entity).
- It's possible to implement custom methods to achieve complex operations with JPQL.
- Spring configuration defines the current implementation type ("live" by default), through Spring auto scanning components mechanism.
- See section 3.2.1.3 , 2.3.4.1.1 and 2.3.4.1.3 on this document on this document for information on repositories, JPA and when user JPA, mybatis or plain JDBC

5.2.3.3.2. myBatis

5.2.3.3.2.1. Interface

Conventions

Project	[project]-backend
Package	[base_package].backend.services.integration
Class	[BusinessObject]DataMapper
Stererotype	org.appverse.web.framework.backend.api.services.integration.helpers.Mapper

Example

```
package [base_package].backend.services.integration;
import org.appverse.web.framework.backend.api.services.integration.helpers;
@Mapper
public interface [BusinessObject]DataMapper {
  // Mapper methods should be defined here
  // For example:
  String getDataValue(String id);
}
```

Remarks

- Only write the interface is needed. Don't write a implementation.
- Use the Appverse Web Mapper annotation at type level.
- The signature of the methods defined here must adhere to the ones declared at the DataMapper file .

5.2.3.3.2.2. DataMapper file location

Conventions

Project	[project]-backend
Folder	src\main\resources\mybatis\[BusinessObject]DataMapper.xml

Example

```
<mapper namespace="[base_package].backend.services.integration.[BusinessObject]Mapper">
        <select id="getDataValue" resultType="String" parameterType="String">
                select value from data_table where id=#{id}
        </select>
</mapper>
```

Remarks

- There have to be a file for every interface
- There have to be a mapper tab for every interface method and the signature have to coincide.
- The queries defined could be static or dynamic ones.
- See sections 3.2.1.3, 2.3.4.1.2 and 2.3.4.1.3 on this document for information on repositories, myBatis and when user JPA, mybatis or plain JDBCConverters

5.2.3.4. Presentation to Business (P2B)

Conventions

Project	[project]-backend
Package	[base_package].backend.converters.p2b
Class	BusinessObject]P2BBeanConverter
Extends	org.appverse.web.framework.backend.api.converters.AbstractDozerP2BBeanConverter
Stererotype	org.springframework.stereotype.Component

Example

```
package [base_package].backend.converters.p2b;
import org.appverse.web.framework.backend.api.converters.AbstractDozerP2BBeanConverter;
import [base_package].model.presentation.[BusinessObject]VO;
import [base_package].model.business.[BusinessObject];
import org.springframework.stereotype.Component;
@Component("[businessObject]P2BBeanConverter")
public class [BusinessObject]P2BBeanConverter
                extends
                AbstractDozerP2BBeanConverter<[BusinessObject]VO, [BusinessObject] > {

        public [BusinessObject]P2BBeanConverter() {
                setScopes("[businessObject]-p2b");
                setBeanClasses([BusinessObject]VO.class, [BusinessObject].class);
        }
}
```

Conventions

Project	[project]-backend
Folder	src\main\resources\dozer\p2b-bean-mappings.xml

Example

```
<mapping map-id="[BusinessObject]-p2b-[scope]">
   <class-a>[BusinessObject].backend.model.presentation.[BusinessObject]VO</class-a>
   <class-b>[BusinessObject].backend.model.business.[BusinessObject]</class-b>
</mapping>
```

Remarks

- For every converter class a set of mappers could be created for every custom usage of the converter.
- All the presentation to business bean mappers will be placed in s a single file following the conventions
- See sections 3.4.2 on this document for more information **on** converters.

5.2.3.5. Business to Integration (B2I)

Conventions

Project	[project]-backend
Package	[base_package].backend.converters.b2i

Class [BusinessObject]B2IBeanConverter

Extends org.appverse.web.framework.backend.api.converters.AbstractDozerB2IBeanConverter

Stererotype org.springframework.stereotype.Component

Example

```
package [base_package].backend.converters.b2i;

import org.appverse.web.framework.backend.api.converters.AbstractDozerB2IBeanConverter;
import [base_package].model.business.[BusinessObject];
import [base_package].model.integration.[BusinessObject]DTO;
import org.springframework.stereotype.Component;

@Component("[businessObject]B2IBeanConverter")
public class [BusinessObject]B2IBeanConverter
                extends
                AbstractDozerB2IBeanConverter<[BusinessObject], [BusinessObject]DTO> {

        public [BusinessObject]B2IBeanConverter() {
                setScopes("[businessObject]-b2i");
                setBeanClasses([BusinessObject].class, [BusinessObject]DTO.class);
        }
}
```

Conventions

Project [project]-backend

Folder src\main\resources\dozer\b2i-bean-mappings.xml

Example

```
<mapping map-id="[BusinessObject]-b2i-[scope]">
   <class-a>[BusinessObject].backend.model.business.[BusinessObject]</class-a>
   <class-b>[BusinessObject].backend.model.integration.[BusinessObject]DTO</class-b>
</mapping>
```

Remarks

- For every converter class a set of mappers could be created for every custom usage of the converter.
- All the business to integration bean mappers will be placed in s a single file following the conventions.
- See sections 3.4.2 on this document for more information on converters

5.3. Configuration Files

5.3.1. Configuration Files Overview

The next table shows an overview about configuration files used at the project.

Path	File name	Description
src/main/web-app/WEB-INF	web.xml	Web Application Deployment Descriptor
	servlet-displatcher.xml	Spring MVC Configuration
src/main/resources/spring	application-config.xml	Spring Application Context Configuration
	security-config.xml	Spring Security Configuration
	database-config.xml	Spring Database Configuration
src/test/resources/spring	database-config.xml	Inmemory Database Configuration
src/main/resources /META-INF	persistence.xml	JPA Persistence Configuration
	persistence-ddl.xml	SQL Generator JPA Persistence Configuration
	orm.xml	JPA ORM Configuration
src/main/resources /mybatis	*DataMapper.xml	myBatis DataMappers
src/main/resources /dozer	*-bean-mappings.xml	Dozer Mappers
src/main/resources /log4j	log4j.xml	Log4j configuration
src/main/resources /ehcache	ehcache.xml	EhCache configuration
src/main/resources /messages	*.properties	Internationalizated Messages
src/main/resources /properties	*.properties	Properties
/	pom.xml	Multimodule POM
/	pom.xml	Backend POM
/	pom.xml	GWT Frontend POM
/	pom.xml	GWT Shared POM

Table 4 Configuration Files Overview

5.3.2. Sample Configuration Files

5.3.2.1. Sample Web Application Deployment Descriptior

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" version="2.5">

  <display-name>Project Name</display-name>

  <!-- ========================================================================= -->
  <!-- Log4j Spring Integtration Loader Listers (Requieres a expanded WAR)     -->
  <!-- ========================================================================= -->
  <listener>
    <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
  </listener>

  <context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>/WEB-INF/classes/log4j/log4j.properties</param-value>
  </context-param>

  <!-- ========================================================================= -->
  <!-- Spring Application Context Loader Listener                                -->
  <!-- ========================================================================= -->
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring/application-config.xml</param-value>
  </context-param>

  <!-- ========================================================================= -->
  <!-- Spring Security Filter                                                    -->
  <!-- ========================================================================= -->
  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!-- ========================================================================= -->
  <!-- Spring MVC Dispatcher Servlet                                             -->
  <!-- ========================================================================= -->
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/*.rpc</url-pattern>
  </servlet-mapping>
</web-app>
```

5.3.2.2. Sample Spring MVC Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-
mvc-3.0.xsd" default-autowire="byName">

  <!-- Spring MVC Annotation Activation                                    -->
  <mvc:annotation-driven/>

   <!-- Autodiscovery for annotated controllers                            -->
  <context:component-scan  base-
pakage="org.appverse.web.framework.backend.frontfacade.gxt.controllers"/>
</beans>
```

5.3.2.3. Sample Spring Aplication Context Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:ehcache="http://ehcache-spring-annotations.googlecode.com/svn/schema/ehcache-spring"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-
aop-3.0.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-
3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://ehcache-spring-annotations.googlecode.com/svn/schema/ehcache-spring http://ehcache-
spring-annotations.googlecode.com/svn/schema/ehcache-spring/ehcache-spring-1.1.xsd"
default-autowire="byName">

  <!-- ========================================================================= -->
  <!-- Property Files Loader                                                     -->
  <!-- ========================================================================= -->
  <context:property-placeholder location="classpath:properties/*.properties" />

  <!-- ========================================================================= -->
  <!-- JPA ORM                                                                   -->
  <!-- ========================================================================= -->
  <bean id="entityManagerFactory"
        class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"/>

  <!-- ========================================================================= -->
  <!-  myBatis Data Mapper                                                       -->
  <!-- ========================================================================= -->
  <bean id="sqlSessionFactory"

        class="org.mybatis.spring.SqlSessionFactoryBean"

        p:mapperLocations="classpath:mybatis/*DataMapper.xml" p:dataSource-ref="dataSource"
/>

  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"
        p:basePackage="org.organization.project.backend.services.integration"

p:annotationClass="org.appverse.web.framework.backend.api.services.integration.helpers.Mapper
"        p:sqlSessionFactory-ref="sqlSessionFactory" />

  <!-- ========================================================================= -->
  <!--  Transaction Manager                                                      -->
```

```
  <!-- ========================================================================= -->
  <bean id="transactionManager"
        class="org.springframework.orm.jpa.JpaTransactionManager" />

  <!-- ========================================================================= -->
  <!-- Dozer converter                                                           -->
  <!-- ========================================================================= -->
  <bean id="conversionService"  class="org.dozer.spring.DozerBeanMapperFactoryBean">
    <property name="mappingFiles" value="classpath*:dozer/*bean-mappings.xml"/>
  </bean>

  <!-- ========================================================================= -->
  <!-- EHCache caching                                                           -->
  <!-- ========================================================================= -->
  <ehcache:annotation-driven />

  <ehcache:config>
    <ehcache:evict-expired-elements interval="60" />
  </ehcache:config>

  <bean id="cacheManager"
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
    <property name="configLocation"  value="classpath:ehcache/ehcache.xml"/>
  </bean>

  <!-- ========================================================================= -->
  <!--  JSR-303 Validation                                                       -->
  <!-- ========================================================================= -->
  <bean id="validator"
        class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" />

  <!-- ========================================================================= -->
  <!-- Autowired Logger BeanFactoryPostProcessor                                 -->
  <!-- ========================================================================= -->
  <bean

class="org.appverse.web.framework.backend.api.helpers.log.AutowiredLoggerBeanPostProcessor"
/>

  <!-- ========================================================================= -->
  <!-- AOP                                                                       -->
  <!-- ========================================================================= -->
  <tx:advice id="txAdvice">
    <tx:attributes>
      <tx:method name="retrieve*" read-only="true" propagation="SUPPORTS"/>
      <tx:method name="search*" read-only="true" propagation="SUPPORTS"/>
      <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
  </tx:advice>

  <bean id="profilingAdvice"
        class="org.appverse.web.framework.backend.api.aop.advices.ProfilingAdvice" />
  <bean id="exceptionAdvice"
        class="org.appverse.web.framework.backend.api.aop.advices.ExceptionAdvice" />
  <bean id="validationAdvice"
        class="org.appverse.web.framework.backend.api.aop.advices.ValidationAdvice"/>
  <bean id="profileManager"

class="org.appverse.web.framework.backend.api.aop.managers.impl.live.ProfileManagerImpl" />
  <bean id="exceptionManager"

class="org.appverse.web.framework.backend.api.aop.managers.impl.live.ExceptionManagerImpl" />
  <bean id="validationManager"

class="org.appverse.web.framework.backend.api.aop.managers.impl.live.ValidationManagerImpl"
/>

  <aop:config>
    <aop:pointcut id="allPresentationServicesMethodsCalls" expression="execution(*
org.organization.project.backend.services.presentation..*.*(..))"/>
    <aop:pointcut id="allBusinessServicesMethodsCalls" expression="execution(*
org.organization.project.backend.business.integration..*.*(..))"/>
    <aop:pointcut id="allIntegrationServicesMethodsCalls" expression="execution(*
org.organization.project.backend.services.integration..*.*(..))"/>
    <aop:pointcut id="allServicesMethodsCalls" expression="execution(*
org.organization.project.backend.services..*.*(..))"/>
    <aop:pointcut id="allGWTPresentationServicesMethodsCalls" expression="execution(*
org.organization.web.framework.backend.frontfacade.gxt.services.presentation..*.*(..))"/>
```

```
    <aop:pointcut id="allPresentationSaveMethodsCalls" expression="execution(*
org.organization.project.backend.services.presentation..save*(..))"/>

    <aop:advisor advice-ref="txAdvice" pointcut-ref="allPresentationServicesMethodsCalls"/>
    <aop:advisor advice-ref="profilingAdvice" pointcut-ref="allServicesMethodsCalls"/>
    <aop:advisor advice-ref="exceptionAdvice" pointcut-ref="allServicesMethodsCalls"/>
    <aop:advisor advice-ref="validationAdvice" pointcut-
ref="allPresentationSaveMethodsCalls"/>
  </aop:config>

  <!-- ======================================================================== -->
  <!-- Imports and Component Scans                                              -->
  <!-- ======================================================================== -->
    <import resource="database-config.xml" />
    <import resource="security-config.xml" />

    <context:component-scan  base-
package="org.organization.project.backend.services.presentation.impl.live"/>
    <context:component-scan  base-
package="org.organization.project.backend.services.business.impl.live"/>
    <context:component-scan  base-
package="org.organization.project.backend.services.integration.impl.live"/>
    <context:component-scan  base-package="org.organization.project.backend.converters.*"/>

    <context:component-scan  base-
package="org.appverse.web.framework.backend.api.converters.*"/>
    <context:component-scan  base-
package="org.appverse.web.framework.backend.api.services.presentation.impl.live"/>
    <context:component-scan  base-
package="org.appverse.web.framework.backend.frontfacade.gxt.converters.*"/>
    <context:component-scan  base-
package="org.appverse.web.framework.backend.frontfacade.gxt.services.*.impl.live"/>
</beans>
```

5.3.2.4. Sample Spring Security Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans  xmlns="http://www.springframework.org/schema/security"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="http://www.springframework.org/schema/beans"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.0.xsd">

  <!-- ======================================================================== -->
  <!-- Auto Configuration and URL Interceptors                                  -->
  <!-- ======================================================================== -->
  <http auto-config='true'>
    <intercept-url pattern="/login.jsp*" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
    <intercept-url pattern="/favicon.ico" access="IS_AUTHENTICATED_ANONYMOUSLY" />
    <intercept-url pattern="/**" access="IS_AUTHENTICATED_FULLY" />
    <form-login login-page='/login.jsp'/>
    <logout logout-success-url="/login.jsp"/>
  </http>

  <!-- ======================================================================== -->
  <!-- Method Pointcuts                                                         -->
  <!-- ======================================================================== -->
  <global-method-security>
    <protect-pointcut
        expression="execution(*
org.organization.project.backend.services.business..*.*(..)))"
       access="ROLE_USER"/>
  </global-method-security>

  <!-- ======================================================================== -->
  <!-- Authentication Managers                                                  -->
  <!-- ======================================================================== -->
  <authentication-manager alias="authenticationManager">
    <authentication-provider user-service-ref="userService"/>
    <authentication-provider user-service-ref="jdbcUserService"/>
  </authentication-manager>
```

```
  <!-- ========================================================== -->
  <!-- In-Memory Authentication Manager                           -->
  <!-- ========================================================== -->
  <user-service id="userService">
    <user name="admin" password="admin" authorities="ROLE_USER, ROLE_ADMIN"/>
    <user name="user" password="user" authorities="ROLE_USER"/>
  </user-service>

  <!-- ========================================================== -->
  <!-- JDBC Authentication Manager                                -->
  <!-- ========================================================== -->
  <jdbc-user-service id="jdbcUserService" data-source-ref="dataSource"
        authorities-by-username-query="select users.username, authorities.id from us-
ers,authorities,userauthorities where username = ? and users.pk=userauthorities.userid and
authorities.pk=userauthorities.authority"/>
</beans:beans>
```

5.3.2.5. Sample Spring Database Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:p="http://www.springframework.org/schema/p"
xmlns:jdbc=http://www.springframework.org/schema/jdbc
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-
jdbc-3.0.xsd " default-autowire="byName">

  <!-- ========================================================== -->
  <!-- JPA ORM                                                    -->
  <!-- ========================================================== -->
  <bean id="jpaVendorAdapter"
    class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter"
    p:databasePlatform="org.eclipse.persistence.platform.database.MySQLPlatform"
    p:showSql="true" />

  <!-- ========================================================== -->
  <!-- Data Source                                                -->
  <!-- ========================================================== -->
   <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
       p:driverClassName="${db.driverClassName}" p:url="${db.url}"
p:username="${db.username}"
       p:password="${db.password}" p:initialSize="${db.poolInitialSize}"
       p:maxActive="${db.poolMaxActive}" p:minIdle="${db.poolMinIdle}"
       p:maxIdle="${db.poolMaxIdle}" p:testOnBorrow="${db.testOnBorrow}"
       p:testWhileIdle="${db.testWhileIdle}"
       p:timeBetweenEvictionRunsMillis="${db.timeBetweenEvictionRunsMillis}"
       p:validationQuery="${db.validationQuery}"
       p:connectionProperties="${db.connectionProperties}"/>
</beans>
```

5.3.2.6. Sample Spring Inmemory Database Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:p="http://www.springframework.org/schema/p"
xmlns:jdbc=http://www.springframework.org/schema/jdbc
xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-
jdbc-3.0.xsd" default-autowire="byName">

  <!-- ========================================================== -->
  <!-- JPA ORM                                                    -->
  <!-- ========================================================== -->
      <bean id="jpaVendorAdapte”
            class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter"
            p:databasePlatform="org.eclipse.persistence.platform.database.HSQLPlatform"
            p:showSql="true" />
```

```
  <!-- ========================================================================== -->
  <!-- Data Source                                                                -->
  <!-- ========================================================================== -->
          <jdbc:embedded-database id="dataSource">
              <jdbc:script location="classpath:/sql/HSQLPlatform-schema-create.sql"/>
      </jdbc:embedded-database>
</beans>
```

5.3.2.7. JPA Persistence Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">
  <persistence-unit name="persistence-unit" transaction-type="RESOURCE_LOCAL">
    <shared-cache-mode>NONE</shared-cache-mode>
    <validation-mode>NONE</validation-mode>
      <properties>
        <property name="eclipselink.weaving" value="static"/>
        <property name="eclipselink.weaving.changetracking" value="false"/>
        <property name="eclipselink.logging.logger" val-
ue="org.appverse.web.framework.backend.persistence.helpers.log.EclipselinkLogger"/>
        <property name="eclipselink.session.customizer" value="
com.gft.riaframework.backend.services.integration.ELCacheSessionCustomizer" />
        <property name="eclipselink.jdbc.batch-writing.size" value="1000" />
        <property name="eclipselink.jdbc.fetch-size" value="5000"/>
        <property name="eclipselink.jdbc.batch-writing.size" value="1000" />
        <property name="eclipselink.jdbc.fetch-size" value="5000"/>
    </properties>
  </persistence-unit>
</persistence>
```

5.3.2.8. Sample SQL Generator JPA Persistence Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">
  <persistence-unit name="persistence-unit">
    <class>org.organization.example.backend.model.integration.UserDTO</class>
  </persistence-unit>
</persistence>
```

5.3.2.9. Sample JPA ORM Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_2_0.xsd" version="2.0">
  <persistence-unit-metadata>
    <persistence-unit-defaults>
      <entity-listeners>
        <entity-listener
class="org.appverse.web.framework.backend.persistence.services.integration.helpers.EntityList
ener">
          <pre-persist method-name="prePersist"/>
          <pre-update method-name="preUpdate"/>
        </entity-listener>
      </entity-listeners>
    </persistence-unit-defaults>
  </persistence-unit-metadata>
</entity-mappings>
```

5.3.2.10. Sample myBatis Data Mappers

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="org.organization.project.backend.services.integration.UserDataMapper">
        <select id="retrieveUser"
resultType="org.organization.project.backend.model.integration.UserDTO"  fetchSize="200"
parameterType="org.organization.project.backend.model.integration.SearchDTO">
                select company.id as id ,company.company_id as
companyId,company.commercial_name as companyName, corporation.summit_corporation_code  as
corporationName, DECODE ( (SELECT COUNT  (*) FROM company_setup_cp csc, company_setup cs
WHERE  cs.ID = CSC.COMPANY_SETUP_ID AND COMPANY_ID= company.id   AND approval_status =
'SAVED'),  0, 'N',  'Y') as blocked
                from company,corporation
                <if test=managers != null">
                  ,company_user,users
                </if>
                where corporation.id=company.corporation_id
                <if test="companyId != null">
                  and upper(company.company_id) LIKE upper('%' || #{companyId} || '%')
                </if>
                <if test="companyName!= null">
                  and upper(company.commercial_name) LIKE upper('%' || #{companyName} ||
'%')
                </if>
                <if test="corporation != null">
                  and upper(corporation.summit_corporation_code) LIKE upper('%' || #{cor-
poration} || '%')
                </if>
                <if test="managers != null">
                  and company.id = company_user.company_id
                  and company_user.id=users.id
                  and users.type = 'm'
                    <foreach item="manager" index="index" collection="managers" open="and
(" separator="or" close=")">
                          upper(users.email) LIKE upper('%' || #{manager} || '%')
                    </foreach>
                </if>
        </select>
</mapper>
```

5.3.2.11. Sample Dozer Mappers

```
<?xml version="1.0" encoding="UTF-8"?>

<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://dozer.sourceforge.net
http://dozer.sourceforge.net/schema/beanmapping.xsd">

<!-- Business to Integration Mappings (Sort in alphabetical order)-->

  <mapping map-id="account-p2i">
    <class-a>org.organization.project.backend.model.presentation.User</class-a>
    <class-b>org.organization.project.backend.model.integration.UserDTO</class-b>
    <field>
      <a>user</a>
      <b>username</b>
    </field>
    <field>
      <a>pass</a>
      <b>password</b>
    </field>
  </mapping>
</mappings>
```

5.3.2.12. Sample EhCache Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">
```

```
    <defaultCache maxElementsInMemory="10000" eternal="true" overflowToDisk="false"/>
</ehcache>
```

5.3.2.13. Sample Log4j Configuration

```
log4j.rootLogger = error, stdout, RE, RI, R
log4j.logger.org.organization.project = debug
log4j.logger.org.appverse.web.framework = info
log4j.logger.org.appverse.web.framework.backend.api.aop.managers.impl.live.ProfileManagerIm
pl = debug

#-------------------------------------------------------------------------------
#
#  The following properties configure the console (stdout) appender.
#  See http://logging.apache.org/log4j/docs/api/index.html for details.
#
##-------------------------------------------------------------------------------
log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
log4j.appender.stdout.threshold = debug
log4j.appender.stdout.layout.ConversionPattern = [project] [%p] %d{yyyy-MM-dd HH:mm:ss,SSS}
[%c{1}.%M(%L)]  %m%n

#-------------------------------------------------------------------------------
#
#  The following properties configure the Daily Rolling File appender.
#  See http://logging.apache.org/log4j/docs/api/index.html for details.
#
#-------------------------------------------------------------------------------
log4j.appender.RE = org.apache.log4j.DailyRollingFileAppender
log4j.appender.RE.Threshold=error
log4j.appender.RE.File = ${webapp.root}/WEB-INF/logs/project.error.log
log4j.appender.RE.Append = true
log4j.appender.RE.DatePattern = '.'yyy-MM-dd
log4j.appender.RE.layout = org.apache.log4j.PatternLayout
log4j.appender.RE.layout.ConversionPattern = [Project] [%p] %d{yyyy-MM-dd HH:mm:ss,SSS}
[%c{1}.%M(%L)]  %m%n

log4j.appender.RI = org.apache.log4j.DailyRollingFileAppender
log4j.appender.RI.Threshold=info
log4j.appender.RI.File = ${webapp.root}/WEB-INF/logs/project.info.log
log4j.appender.RI.Append = true
log4j.appender.RI.DatePattern = '.'yyy-MM-dd
log4j.appender.RI.layout = org.apache.log4j.PatternLayout
log4j.appender.RI.layout.ConversionPattern = [Project] [%p] %d{yyyy-MM-dd HH:mm:ss,SSS}
[%c{1}.%M(%L)]  %m%n

log4j.appender.R = org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File = ${webapp.root}/WEB-INF/logs/project.all.log
log4j.appender.R.Append = true
log4j.appender.R.DatePattern = '.'yyy-MM-dd
log4j.appender.R.layout = org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern = [Project] [%p] %d{yyyy-MM-dd HH:mm:ss,SSS}
[%c{1}.%M(%L)]  %m%n
```

5.3.2.14. Sample Internationalizated Messages

```
0=Correcto
1=Error de comunicacion
2=Error de proceso
3=Error de datos
```

5.3.2.15. Sample Properties

```
property=value
```

5.3.2.16. Sample Multimodule POM

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.appverse.web.framework.poms</groupId>
    <artifactId>appverse-web-superpom</artifactId>
    <version>1.0.1</version>
    <relativePath/>
  </parent>

  <groupId>org.organizatiop.project</groupId>
  <artifactId>project</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>Project</name>

  <build>
    <defaultGoal>clean package tomcat:deploy-only</defaultGoal>
  </build>

  <modules>
    <module>project-backend</module>
    <module>project-gwtfrontend</module>
    <module>project-gwtshared</module>
  </modules>
</project>
```

5.3.2.17. Sample Backend POM

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.appverse.web.framework.poms.modules.backend.core.parent</groupId>
    <artifactId>appverse-web-modules-backend-core-parent-pom</artifactId>
    <version>1.0.1</version>
    <relativePath></relativePath>
  </parent>

  <groupId>org.anakala.project.backend</groupId>
  <artifactId>project-backend</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>Project GWT Backend Module</name>

  <properties>
    <main.name>project</main.name>
    <main.group>org.organization.project</main.group>
  </properties>

  <build>
    <filters>
      <filter>src/main/filters/${env}.properties</filter>
    </filters>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
         <filtering>true</filtering>
      </resource>
    </resources>

    <plugins>
      <plugin>
         <groupId>org.appverse.web.framework.poms.tools.jpaddlgenerator.plugin</groupId>
          <artifactId>appverse-web-tools-jpa-ddl-generator-plugin</artifactId>
        </plugin>
    </plugins>
  </build>
```

```
  <dependencies>
    <dependency>
      <groupId>org.appverse.web.framework.modules.backend.core.api</groupId>
      <artifactId>appverse-web-modules-backend-core-api</artifactId>
      <version>${project.parent.version}</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>org.appverse.web.framework.modules.backend.frontfacade.gxt</groupId>
      <artifactId>appverse-web-modules-backend-frontfacade-gxt</artifactId>
      <version>${project.parent.version}</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>org.appverse.web.framework.modules.backend.core.persistence</groupId>
      <artifactId>appverse-web-modules-backend-core-persistence</artifactId>
      <version>${project.parent.version}</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>org.organization.project.shared</groupId>
      <artifactId>project-gwtshared</artifactId>
      <version>1.0-SNAPSHOT</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
     <groupId>org.organization.project.gwtfrontend</groupId>
      <artifactId>project-gwtfrontend</artifactId>
      <version>1.0-SNAPSHOT</version>
      <classifier>GWT-Frontend-Resources</classifier>
      <type>zip</type>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

5.3.2.18. Sample GWT Frontend POM

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.appverse.web.framework.poms.modules.frontend.gwt.parent</groupId>
    <artifactId>appverse-web-modules-frontend-gwt-parent-pom</artifactId>
    <version>1.0.1</version>
    <relativePath></relativePath>
  </parent>

  <groupId>org.organization.project.gwtfrontend</groupId>
  <artifactId>project-gwtfrontend</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Project GWT Frontend Module</name>

  <dependencies>
    <dependency>
      <groupId>org.appverse.web.framework.modules.frontend.gwt.api</groupId>
      <artifactId>appverse-web-modules-frontend-gwt-api</artifactId>
      <version>${project.parent.version}</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.anakala.project.shared</groupId>
      <artifactId>project-gwtshared</artifactId>
      <version>1.0-SNAPSHOT</version>
      <scope>provided</scope>
    </dependency>
```

```
  </dependencies>
</project>
```

5.3.2.19. Sample GWT Shared POM

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.appverse.web.framework.poms.modules.frontend.gwt</groupId>
    <artifactId>appverse-web-modules-frontend-gwt-pom</artifactId>
    <version>1.0.1</version>
    <relativePath></relativePath>
  </parent>

  <groupId>org.organization.project.shared</groupId>
  <artifactId>project-gwtshared</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Project GWT Shared Module</name>

  <dependencies>
    <dependency>
      <groupId>org.appverse.web.framework.modules.frontend.gwt.api</groupId>
      <artifactId>appverse-web-modules-frontend-gwt-api</artifactId>
      <version>${project.parent.version}</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

6.1. Technologies

Technologies	Libraries
Java	Sun Java JDK 7 (1.7.x) (http://www.oracle.com/es/technologies/java)
Container	Spring Framework 3.x (http://www.springsource.org/spring-framework)
Security	Spring Security 3.x (http://www.springsource.org/spring-security)
Request Handling	Spring MVC 3.x (http://www.springsource.org/spring-framework)
Logging	Simple Logging Facade for Java (SLF4J) (http://www.slf4j.org)
	log4j (http://logging.apache.org/log4)
Caching	Spring Cache Abstraction 3.x (http://www.springsource.org/spring-framework)
	Ehcache (http://ehcache.org/)
Bean Mapper	Dozer 5.x (http://dozer.sourceforge.net)
XML to Bean Mapper	Jaxb (http://jaxb.java.net/)
Web Services	Spring WS 2.1 (http://www.springsource.org/spring-web-services)
	Apache Axis 1.4 (http://axis.apache.org/axis/)
Json Procesor	Jackson (http://jackson.codehaus.org/)

Table 5 Tecnologies

6.2. Tools

Module	Tools
Java	Sun Java JDK 7 (1.7.x) (http://www.oracle.com/es/technologies/java)
IDE	Eclipse IDE with JEE extensions (http://www.eclipse.org/)
Project Management Tool	Maven 3.x Project Managment Tool (http://maven.apache.org)
Continous integration Server	Jenkins Continous Integration Server (http://jenkins-ci.org/)
Maven Artifacts Repository	Nexus Maven Artifacts Repository (http://nexus.sonatype.org/)
Software Quality	Sonar Source Code Quality Managment Platform (http://www.sonarsource.org/)
Design & Modelling	Sparx Enterprise Architect 9.2 Corporate (http://www.sparxsystems.com/) (Editor & Viewer)
HTTP Server	Apache HTTP Server (http://httpd.apache.org/)
Development Server	Apache Tomcat 7.x Web Application Server (http://tomcat.apache.org)
Other environtment Server	Oracle WebLogic J2EE Server (http://www.oracle.com/technetwork/middleware/weblogic/overview)

Table 6 Tools