Star Beat

Software Architecture

Redatto da:	Erik Frizziero, Pietro Bertoli - Lynx	08/01/2019
Verificato da:	Gabriele De Benetti - Lynx	
Approvato da:	Marco Calvi - Enel Global Digital Solution	

Summary

The aim of this document is to describe the architecture for the Enel Link. For all the terms in brackets [], please refer to Project Glossary.

Distribution List

Document target list

Name	Company

This artifact is available on Confuence:

- Project: IN - Product StarBeat

Document modifications

The following modifications refer to the old document versions.

Changes Description	Reference
First version	1

References

List of the documents

[1] [Krutchen 1995] Philippe Krutchen, Architectural Blueprints - The “4+1” View Model of Software Architecture, IEEE SW 12-1995

[2] 1250_AD5GL_AMMSoftwareArchitecture.docx

Table of Contents

1. Introduction

This Software Architecture document give information about the Product Functionalities available and the architectural solution used in the Star Beat System. It will also provide an overall overview of the systems that are involved in the business processes described.

1.1. Audience

The audience of this document is Enel Global Digital Solution ,and, interested partners and suppliers.

2. Main Functional and Architectural Overview

In following paragraphs will be explained the Star Beat Product features, functions and software architecture.

The Star Beat Product features and functions are explained using the following points of view:

1. The **Features and Functionalities** View. Section that contains a brief description of the available features.
2. The **Use-Case** View, that contains use cases ,and, encompasses scenarios architecturally significant.

The StarBeat Product Technical Architecture documentation is represented by a number of different views, which in their essence are extract illustrating the "architecturally significant" elements. The StarBeat Technical Architecture documentation is composed by:

1. **The Logical View**, which contains the most important design classes and their organization into packages and subsystems, and the organization of these packages and subsystems into layers
2. The **Implementation/ Deployment View**, which contains
   - an overview of the implementation model and its organization in terms of modules into packages and layers. The allocation of packages and classes (from the Logical View) to the packages and modules of the Implementation View is also described.
   - the description of the various physical nodes for the most typical platform configurations, and the allocation of tasks (from the Process View) to the physical nodes. This view need only be used if the system is distributed. It is a subset of the deployment model.

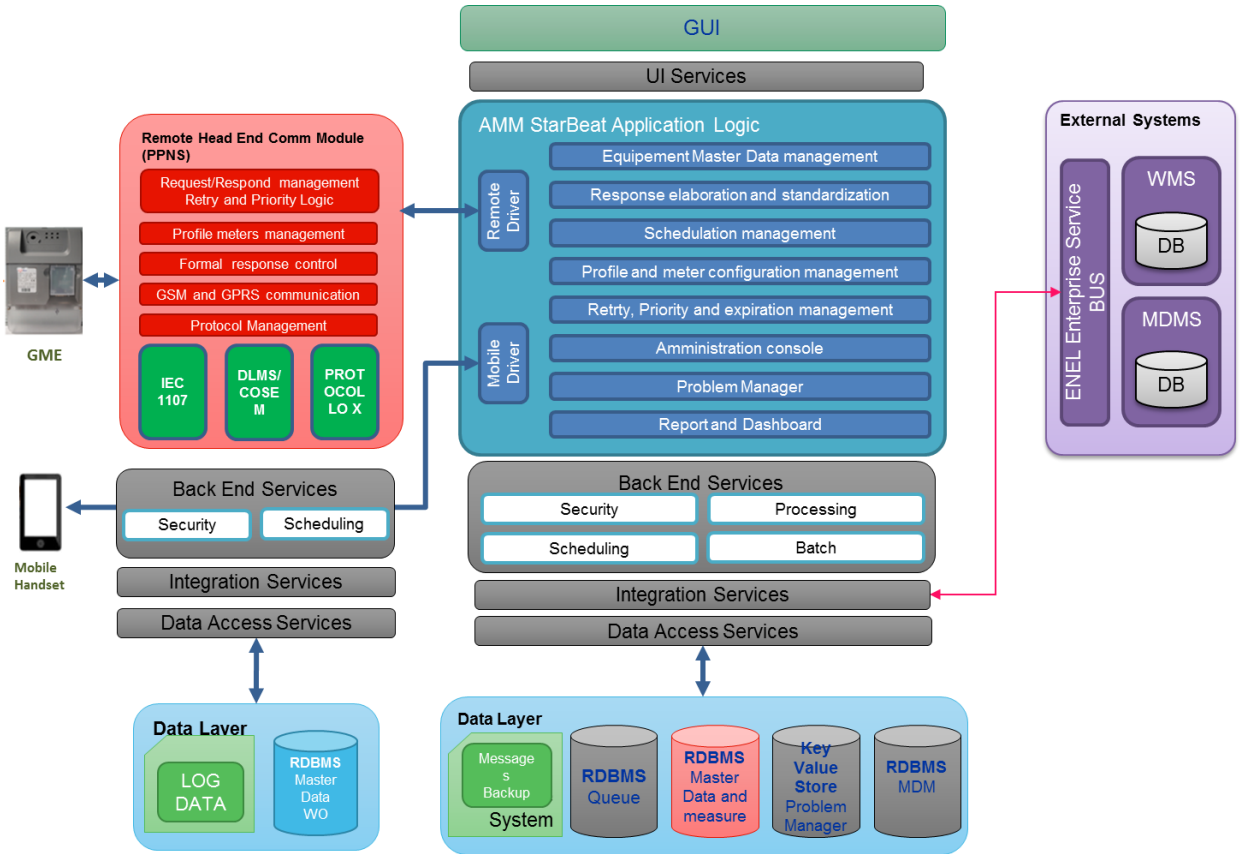

Figure 1: Main Architectural view

2.1. StarBeat Sub-System Model

Star Beat System is composed of the following main three subsystems:

- **StarGroove**
- **StarPulse**
- **StarSync**

The functionalities of these three main subsystems are described below.
The logical view of the interactions is resumed here:

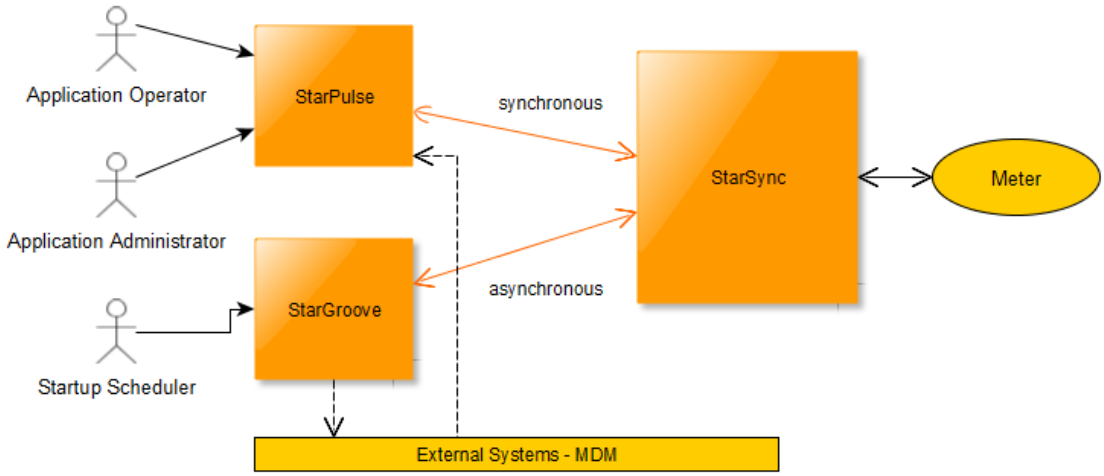

Figure 2: Main Sub-Systems view

These three subsystems uses:

- service modules (like Configurator, Profiler) by which it is possible:
  - to establish authorizations and roles on their functionality
  - to configure the parameters of the systems, appropriately
- persistence modules (see StarPulse, StarGroove, StarSync) to interact with DB

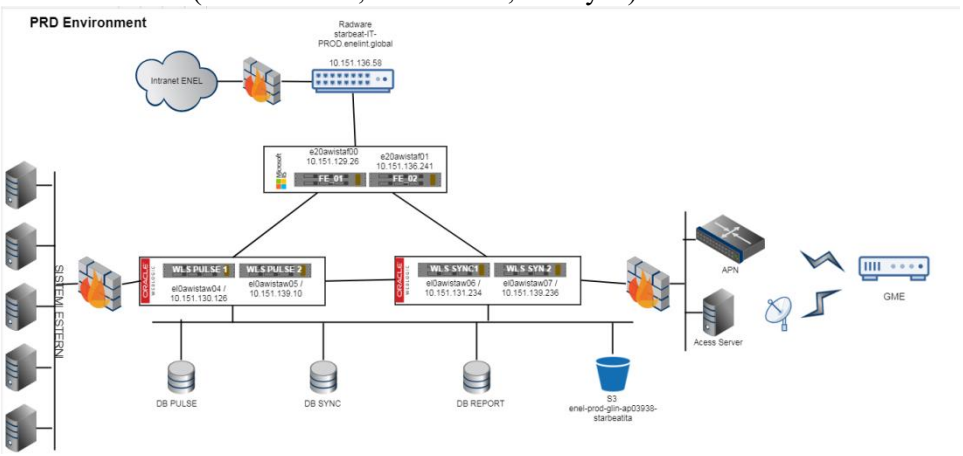

Figure 3: Infrastructure view

In StarBeat there are other modules for example the Reporting Module, allows the user to filter, view and export the Star Beat data (master data, readings execution, etc.) . All these modules are represented and described in the **Technical Architectural View (document/section).**

2.1.1. StarPulse

StarPulse subsystem provides all user interfaces ,and, services that enable a back office operator to handle the inventory of meter, and, create manually scheduling Tasks.
It provides administrative interface for enabling an application administrator to change the system parameters. StarPulse contains services that allow external systems of meter management to upload the inventories. ***Basically it's a subsystem of presentation interfaces and services.***

2.1.2. StarGroove

StarGroove subsystem provides all processes for creating automatically GME meter management tasks, and for dispatching these tasks to StarSync subsystem. It's responsible for processing the responses returned from StarSync and to communicate them to external systems. ***Basically it's a subsystem responsible for automatic creation of scheduled Tasks and for processing the returned response massively.***

2.1.3. StarSync

The StarSync Subsystem takes charge of the activities for meter management created and dispatched by StarGroove or of the spot requests activated by a back office operator in StarPulse (or via Remote system request). It uses appropriate driver to submit these request activities to meters and acquire responses.
Basically it's a subsystem responsible for the communication with the meters for performing the activities submitted by StarGroove and StarPulse.

2.1.4. Report

The Reporting module allows the user to filter and view the acquired readings (eventual errors and warning), the historical master data, and the telereading communication data; this module is described in the **Technical Architectural View (document/section).**

3. Advantages

Advantages

- ✓ Horizontal scalable and High Availability solution
- ✓ Product configuration modularized and customizable
- ✓ Avoid system modules coupling
- ✓ Self-consistent
- ✓ Provide expandability
- ✓ Testable
- ✓ Enel architecture compliant

3.1. Horizontal Scalability and High Availability solution

Star Beat System provides a load balancing and an high available system, in order to satisfy a strict horizontal scalability constraint; in according to the system architecture design.

3.2. Product configuration modularized and customizable

The installation and customized configuration of the product, can be made in according to the system architecture design ,and, the country that uses it (E.g. Protocols, Measurers, and, territorial organization) StarBeat System Is a modularized system in which each component has its own specific responsibilities.

3.3. Avoid tight coupling

The StarBeat System architecture solution avoid tight coupling among its components.

3.4. Self-consistent

StarBeat System is a self-consistent application, that depends only on its own features with no logic implemented inside other systems. When StarBeat System rely on information stored inside an external system, it access it using well-defined APIs compliant to widely adopted open standards.

3.5. Provide expandability

It is possible to add new features to StarBeat System without affecting the current implementation.
The underlying constraint in use is to minimize the use of custom code, to rely on well-known, de-facto standard frameworks, and to exploit middleware services.

3.6. Testable

StarBeat System is a distributed system with strong integration requirements, is designed ,and, developed from start having testability in mind. The "Software testability" means automatic testability first of all, so StarBeat come with a comprehensive suite of unit tests and supports automatic integration testing. Integration tests is implemented so that they can be executed without the need to deploy the application modules to shorten the development cycle. The Documentation (e.g. Catalogue) provided supports to the tests ,and, the interfaces are comply to established open standards supported by common software testing tools.

3.7. Adhere to Enel architecture guidelines

The StarBeat System is compliant with the standards and guidelines issued by Enel Governance.

4. Logical View

This view highlights all the modules and their interactions with the three main subsystems introduced in the previous paragraph. In addition to show all the modules used by the three main subsystems, this view highlights the uses of two Database:

- **StarPulse Database** used by the StarPulse and StarPulse_Batch system.
- **StarSync database** used by the StarSync system.

The separation of two databases allows to tune distinctly the two main massive processes of the application:

- The process in StarGroove that automatically activates the defined schedulation, generating the requests to send asynchronously to StarSync.
- The process in StarSync of sending commands to meters in separate and parallel threads.

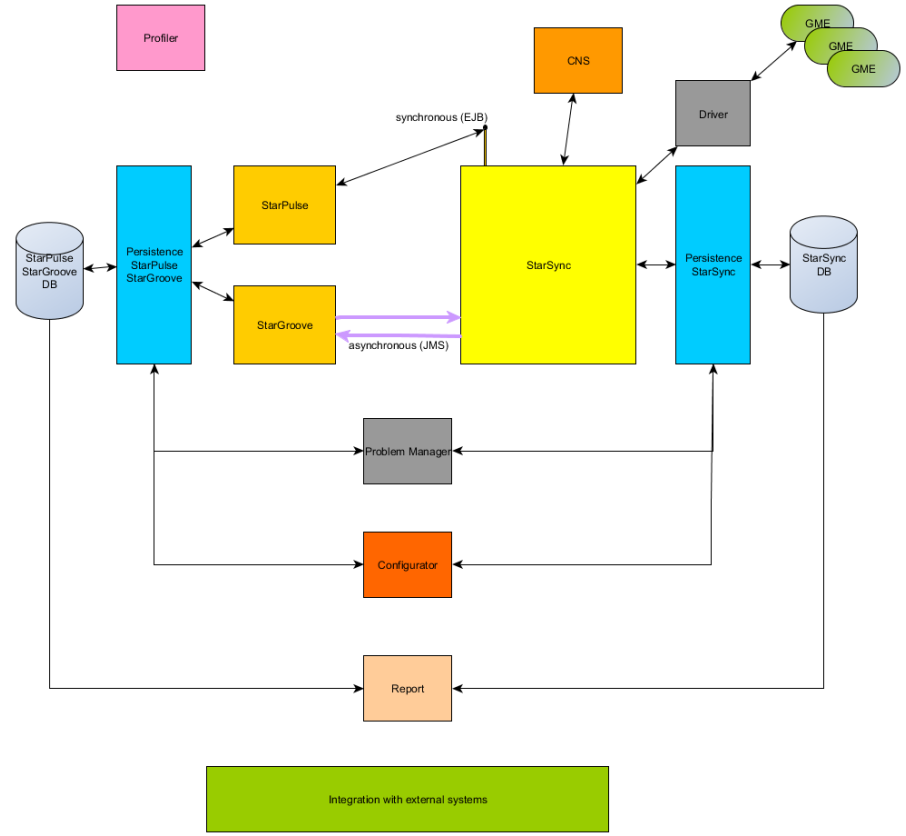

Figure 4: The three main Business Systems and their interaction with other modules

A brief description of all modules and their responsibility is presented below:

4.1.1. The Business Systems

4.1.1.1. The StarGroove system

The StarPulse_Batch system has the responsibility for:

- Scheduling, retry and prioritization logic
- Sending the automatically scheduled activities to GME and Modem
- Elaboration and standardization of acquired data from StarSync

4.1.1.2. The StarPulse system

The StarPulse system has the front end interfaces that can be used by an operator in order:

- to handle the inventory of GME/Modem
- to permit spot readings on devices
- to create scheduled tasks for user-defined sets of GME. These activities should run for a limited time interval, in contrast with automatically scheduled activities.

4.1.1.3. The StarSync system

The StarSync system has the responsibility:

- to remotely send the command to GME, acquire the readings and send to StarPulse/StarPulse-BATCH.
- Meter data acquisition and meter clock synchronization
- GSM and GPRS open communication.

4.1.1.4. The Profiler system

Using the profiler module, it's possible to define profiles and roles that enables to associate the authorization to the various functionalities of the system for every user.
It's used by StarPulse, Report, Problem Manager modules.

4.1.1.5. The Problem Manager system

The Problem Manager is able to manage the problems of the systems, traced using tickets.
It offers automatic processes to correct them.

4.1.1.6. The Configurator system

The Configurator System enables to set/modify the parameters of the application.
There is a configurator module for every main subsystem of the architecture.

4.1.1.7. The Report system

The report module contains all the functionalities that allow to view acquired readings.
It extracts data from StarPulse Database and StarSync database.

4.2. Overview

4.2.1. LOGICAL VIEW OF INTERACTION BETWEEN StarGroove WITH StarSync

The diagram below shows the interaction between StarGroove and StarSync modules.
This solution uses:

- an Executor for dispatching the activities.
- more logical StarSync.
- a Dispatcher for choosing the appropriate StarSync according defined criteria.
- a Response Handler for processing the responses.

In this solutions, more logical StarSync exist and they are customized for Operator/Technology.
The Dispatcher has the responsibility to choose the appropriate StarSync to which the Executor has to route them, according the choice of technology (GPRS over GSM) and the economical convenience of Mobile Operator.

This diagram:

1) Focuses on components in the StarGroove with particular attention to Dispatcher as the component for calculating the routing of the generated activities.
2) Shows that there are more logical StarSync: each of these StarSync is configured ***according to the resources*** reserved for Operator and Technology

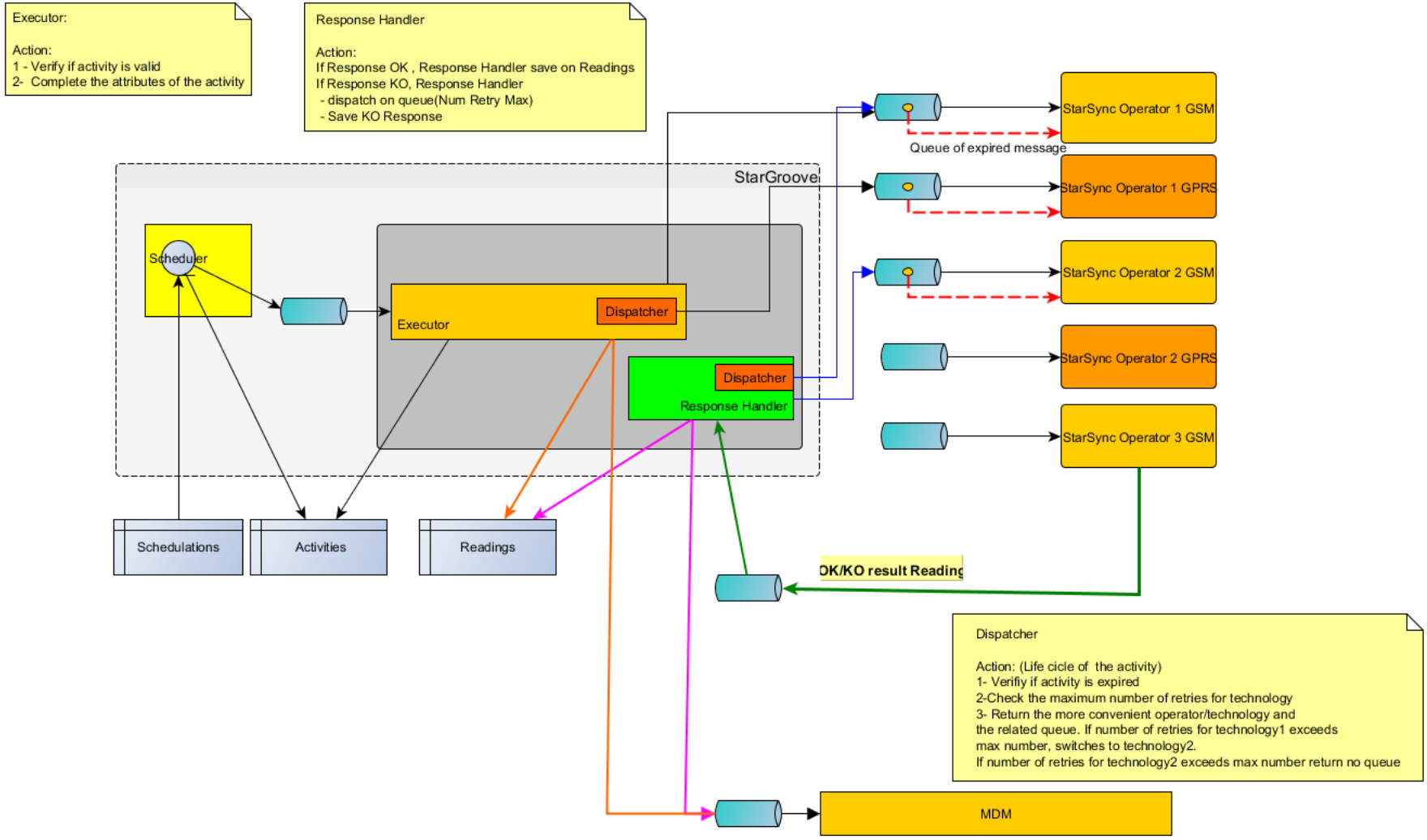

Figure 5: Interaction between StarGroove and StarSync

LOGICAL COMPONENTS OF StarGroove

Scheduler

It's responsible for activating the scheduled activities on meters.
It checks at regular time interval, if there are scheduled tasks to activate for a meter in the Scheduling Persistence Entity.
If they are present, activities are generated for meters, according to the type of commands to send to the meter.
For example if a Registers Reading and a Load Profile Reading are required in the Scheduled Task for a meter, two activities will be generated with the a unique identifier for them.
The scheduler

1) saves the generated activities in the Activities Persistence Entities (Register Activities Entity and Load Profile Activity Entity).
2) sends to Executor a message that represents the activities.

It is also important to highlight **that all the activities that are related to meters belonging to the same chain, will be grouped and sends under the same message**.
In a such way, all the activities related to a multidrop chain, will be taken over by the same message driven bean and treated in the same thread.
The Scheduler is a Singleton component.

Executor

The Executor is responsible for any validation of message and eventually for completing the attributes of activities,
For example if a Load Profile reading of two days is required but the last reading is older than two days, the Number Of days attribute will be increased for reading the missing days. (see the arrow from Executor towards Activities)
If the message is expired, the executor has "to close" the activity on responses and to put a message on a queue towards external systems to inform of the error
(see the arrows from Executor towards Readings Entity and the arrow towards MDM)
If the message is valid, the Executor uses the Dispatcher for choosing the queue and the appropriate StarSync responsible for the processing of the activity message, according technology and operator. (see Asynchronous Dispatcher).

Response Handler

The Response Handler is responsible for the processing of the readings returned by StarSync.
If the reading returned by StarSync is OK, Response Handler has

- to verify the meter identifier with the read serial number.
- to verify if all the expected obis have been read
- to associate the obis of the manufacturer to standard obis.
- saves the readings and sends the results (through a queue) to the external systems.

If the response is KO and the number of retries doesn't exceed the maximum number,
Response Handler has to dispatch the activities again, using the queue returned by the same dispatching algorithm used by Executor.
Response Handler has to save the KO response with the code error and to notify it to the external systems.

Basically Executor and Response Handler have different inputs ma uses the same components.
the first processes the requests generated from scheduler while the second processes the responses returned from StarSync. But both use the same "dispatcher algorithm" and eventually put the activity on a queue (the response handler only in case of error and number of retries not greater than the maximum , and have to "close" the activities on database : (the executor only in case of errors).
So, the same algorithms-components will be injected via Spring to Executor and Response Handler.

Asynchronous Dispatcher

The Dispatcher has the responsibility to choose the queue towards the appropriate StarSync on which to put the messages from Executor and Response Handler according the following rule: ***the Protocol that is associated to meter in inventory and the more convenient operator should be chosen.***

The messages to dispatch can arrive:

1) **from Executor**: these messages are those just generated and validated before the dispatching to the appropriate StarSync
2) **from the Response Handler**:
   In this case, the message are returned from the output code of StarSync
   If the number of retries doesn't exceed the maximum number, the message should be redirect to an appropriate StarSync according the rule described above.

In details:

The dispatcher the first time (in Executor) chooses the more convenient operator and the technology in the inventory (for example GPRS).

The other times (in Response Handler) if the number of retries exceeds the maximum for primary technology (example GPRS), the dispatcher has to choose the secondary technology (for example GSM)

If the number of retries exceeds the maximum for the "first" technology and for the "second" technology, the dispatcher returns no queue.

Queue of expired messages:

If a message in a queue expires will be automatically dispatched to another queue that contains all the expired messages.

These messages are treated in an appropriate manner by StarSync: StarSync creates a KO response with an appropriate code errors and put it on the output queue.

4.2.1.1. LOGICAL COMPONENT OF StarSync

StarSync customized in Threads for number of resources for Operator/Technology.

There are more logical StarSync.

Each logical StarSync is related to a particular technology and operator.

In this way, it's possible to establish the size pool of Message Driven Bean according to the physical resources associated to the operator and to the technology.

The input queues to StarSync represented in figure are distinct JMS Queues.

The output queue of StarSync can be physical distinct queues with the same logical JNDI name.

4.2.2. LOGICAL VIEW OF INTERACTION BETWEEN StarPulse AND StarSync

The interaction between StarPulse and StarSync is synchronous in the sense that the interaction is activated by an operator through a web interface that allows to view the results of the reading at the end of communication.

The diagram below shows the interaction between StarPulse and StarSync modules

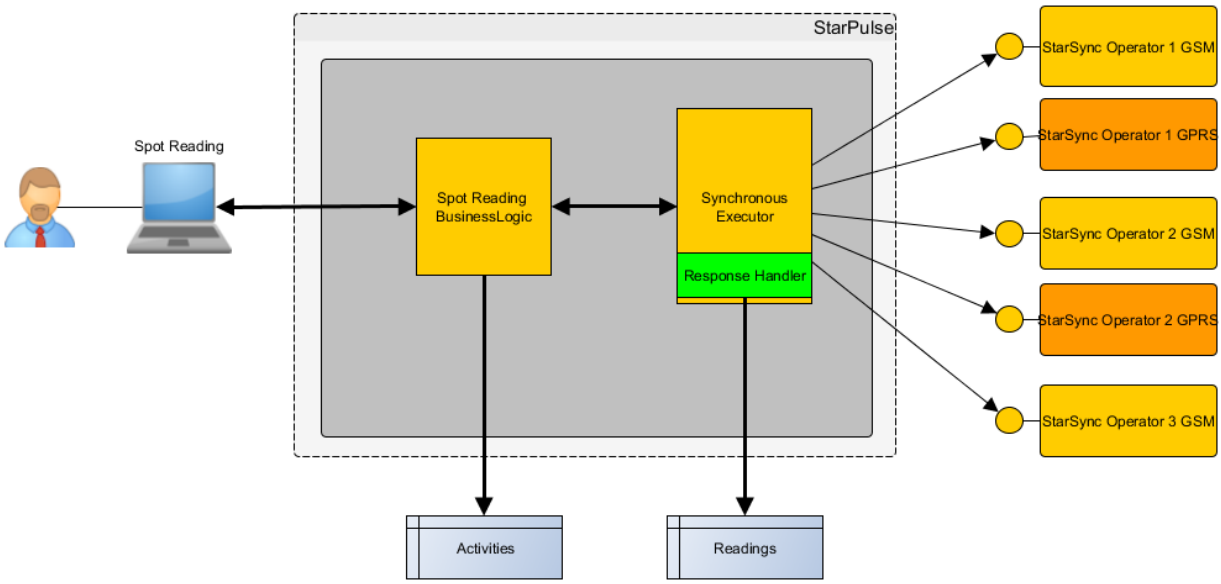

Figure 6: Interaction between StarPulse and StarSync

4.2.2.1. LOGICAL COMPONENT OF StarPulse

Spot Reading Business Logic
It's a Rest Web Service using the security services for checking the authorization of the caller.
It calls the Business Logic for creating the administrative request, saving it in the Activities Persistence Entity and sending it to StarSync through the Synchronous Executor.

Synchronous Executor
In this solution, there is still used the Dispatcher as an algorithm component; in this case the Executor is a synchronous component: it acquires a remote interface from the appropriate StarSync through ejb and calls the remote method of the ejb that allows to send the administrative request to meter.
The Dispatcher uses the same criteria in the Asynchronous Dispatcher to choose the appropriate logical StarSync according to the protocol associated to meter in inventory and the convenience of the Operator.
In contrast with Asynchronous Executor if a message fails for a communication error, the code error is immediately returned to the Executor that can retry or close the activity according to the exceeding the maximum number of retries.

Response Handler
When the results of the readings returns from StarSync, they have to be saved in the Response Persistence Entity.
The Dispatcher saves these results into reading responses using the Response Handler.

4.2.2.2. The Star Beat tiers

The Star Beat is a three tiers architecture system. These tiers are: Presentation, Execution and Information.

1. **Presentation tier**: it's responsible to expose both synchronous and asynchronous interfaces (Web pages, JMS queues) to external actors and take care about security requirements.

2. **Execution tier**: in this tier it will take place the orchestration and execution of business processes and the interaction with the database.
3. **Information tier**: this is the tier where data are persisted, is the logical representation of the database or other persistence solutions.

4.2.2.3. The Star Beat modules

The concrete realization of the logical tiers described in the previous paragraph is made by separate deployments units (Java EAR modules).
Modules are organized across the tiers and are decoupled using Ejb technology for synchronous calls and JMS messages to drive for asynchronous events. These modules are shown in the next diagram

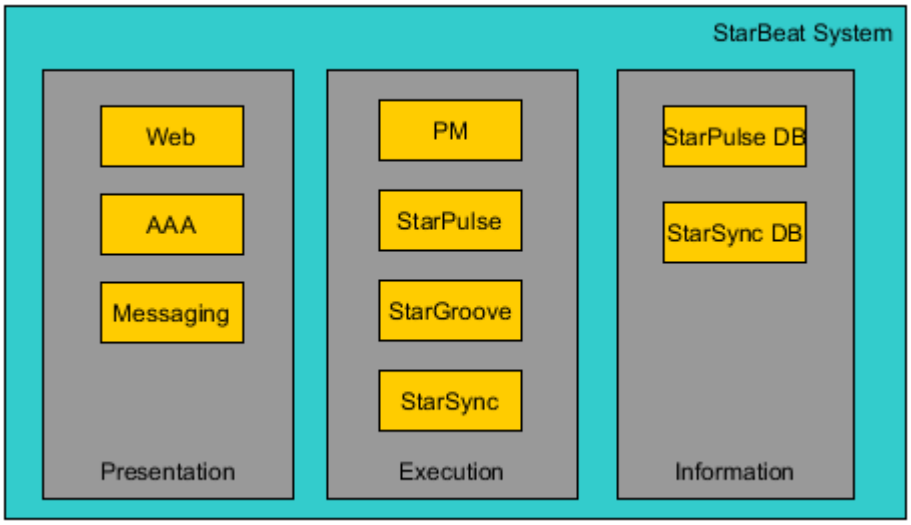

Figure 7: Star Beat modules

The next paragraph will show the logical layers in which the modules are structured.

4.2.2.4. Logical layers of Star Beat modules

Star Beat modules are structured in three layers: Front End, Business, Integration.
The diagram below shows an high level representation of how the three layers solution organize the module structure.

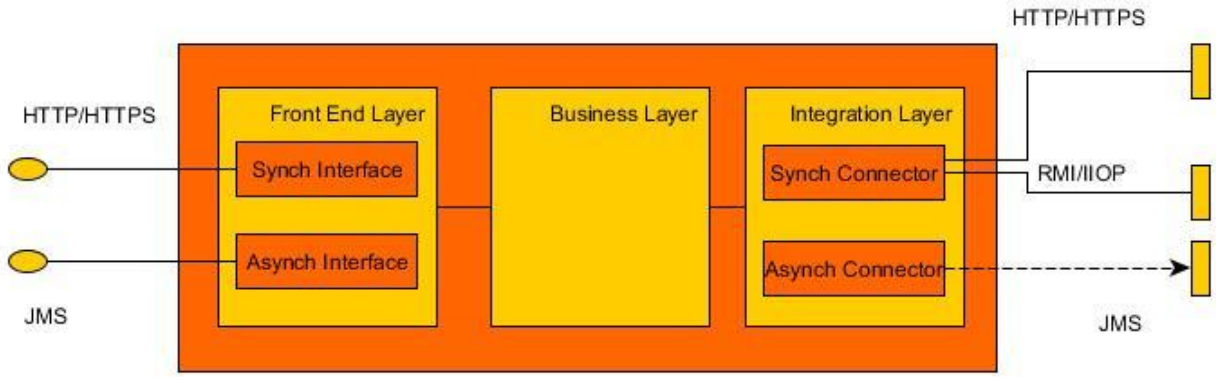

Figure 8: three Layers structure

4.2.2.4.1. The Front End layer

It provides synchronous / asynchronous interfaces.
Synchronous interfaces support the following protocols: HTTP /HTTPS (web pages, web services) and RMI / IIOP (Remote Ejb invocation).
Asynchronous interfaces use JMS protocol.
The choice of the protocol to use depends on the type of interface to implement and the tier where the module resides. For example: Web pages must use HTTP / HTTPS protocol and are packed in a module that resides in the presentation tier.

4.2.2.4.2. The Business Layer

In this layer take place the core business logic of the module. All the business objects adhere to their related business entities descripted in the Project Glossary. The modules located in the Presentation tier will use this layer to implement the business delegate logic to employ the Execution tier. The business logic layer of the modules located in the Execution tier will implement the use cases steps.

4.2.2.4.3. The Integration Layer

This layer provides synchronous and asynchronous connectors to communicate with external systems and other Star Beat modules. These connectors are: remote Ejb accessor, JMS producer, Web services client,.
The MDBs classes are used to consume messages sent to JMS queues.
In the figure the persistence connector and Driver Connector are distinguished from others synchronous connector/Interface in order to highlight the persistence service for the interaction with DB and the Driver Connector as the last connector toward Meters/Modem.

This will be done by configuring in Weblogic a Foreign Server to map the remote JNDI names of the AMM Foreign Connection factory and JMS queues to local names to employ in the weblogic-ejb-jar.xml descriptor.

4.2.2.5. Layer organization in the Star Beat System

The diagram represents the Modules: StarPulse, StarGroove, StarSync highlighting the Synchronous and Asynchronous Interface/Connector when presents.

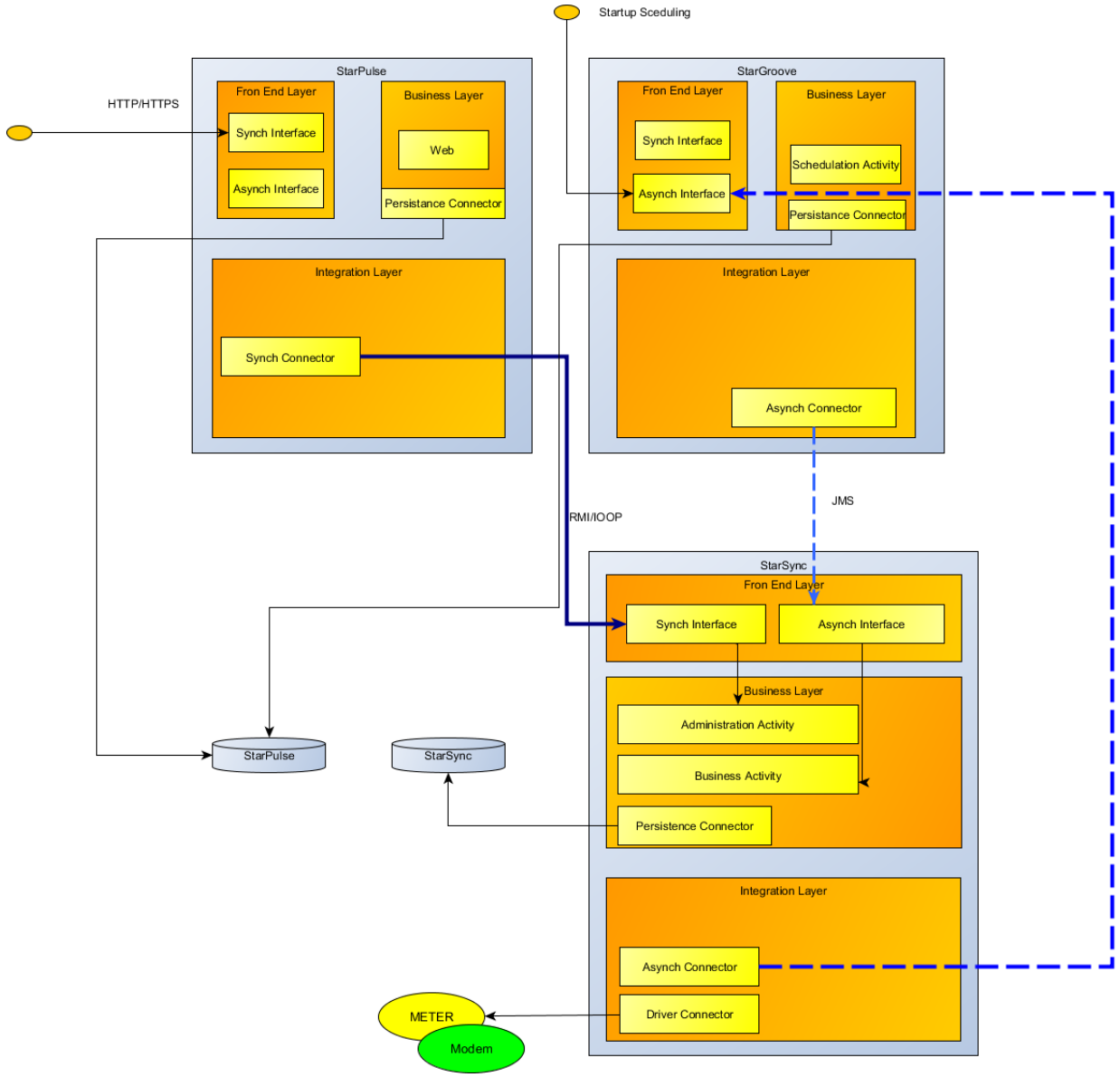

Figure 9: Three Layers structure

In StarPulse Module:

1) In the Front End Layer there is a synchronous interface representing HTTPS/HTTP called from Web Interfaces
2) In the Integration Layer there is a persistence connector in order to connect to ***StarPulse Database***.

In StarGroove Module:

1) In the Front End Layer there is an asynchronous interface representing the scheduler activated by a startup timer scheduling.
   The scheduler activates the defined schedulings as described in the previous paragraph.
2) In the Integration Layer there is a persistence connector in order to connect to ***StarPulse***

Database.

In StarSync Module:

1) In the Front End Layer there is a synchronous interface representing ejb calling from StarPulse for Spot Readings.
   It also contains Asynchronous interface (JMS Queue) in order to receive the scheduled Activities from StarGroove
2) In the Integration Layer there is a persistence connector in order to connect to ***StarSync Database***. There is a Driver Connector toward Meter/Modem and Asynchronous connector that represents the output JMS queue toward StarGroove.

5. Implementation View

HA functionalities and Scalability

The deployed solution is guaranteeing HA functionalities, advanced administration and monitoring, and horizontal scalability.

The horizontal scalability is particularly important since it allows to easily scaling the Star Beat system from the initial number of meters and communications lines to potentially support an increase of meters to handle, and the availability of a greater number of resources assuring the future-proof of the platform.

Core technologies					
Technology	Vendor	Product	License Type	Version	Environment
Java Virtual Machine	Oracle (Sun)	JDK	Open Source (Sun License)	1.6	All
Java Virtual Machine	Oracle (Bea)	JRockit	(Part of Weblogic Server)	1.6	All
Application Server	Oracle (Bea)	Weblogic Server	Commercial	10.3.6	All
IIS	Microsoft	Web server	Commercial	7.5	All
Database Server	Oracle	Database	Commercial	11g	All
Enterprise Reporting	Jaspersoft	JasperReports	LGPL	6.0.0	
Batch Scheduling		Control-M	BMC Control M		

Main Frameworks					
Technology	Vendor	Product	License Type	Version	Environment
Web Services Engine	Apache	Axis2	Open Source (Apache)	1.4.1	N.A.
Web Services Security	Apache	Rampart	Open Source (Apache)	1.4	N.A.
Distributed Cache	Terracotta	EhCache	Open Source (Terracotta)	2.5.1	N.A.
Persistence	RadHat (JBoss)	Hibernate	Open Source (LGPL)	3.5.3	N.A.
XML binding	Java.net	JAXB Reference Implementation	Open Source (Apache)	2.1	N.A.
Presentation MVC	Java.net	JSF Reference Implementation	Open Source (Apache)	1.2	N.A.
Presentation components	RadHat (JBoss)	Richfaces	Open Source (LGPL)	3.3.2	N.A.
Presentation templating	Java.net	Facelets	Open Source (Apache)	1.1	N.A.
Inversion of Control	VMWare (Spring)	Spring Framework	Open Source	2.5	N.A.
Authentication & Authorisation	VMWare (Spring)	Spring Security	Open Source	2.0	N.A.

Client-side Net Protocols	Apache	Commons Net	Apache	3.2	N.A.
HTML	n/a	n/a	n/a	5	N.A.
AngularJS	n/a	n/a	n/a	1.3.16 (1.4)	N.A.
logging library for Java	Apache	Apache log4j	Open Source (Apache)	1.2.14	
Tag library for Display	Display Tag	Display tag	Open Source	1.1.1	
Struts	Apache	Struts Core	Apache	1.3.8	
Java Template	Freemarker	Freemarker	Open Source		
DLMS Driver	Ericsson	Driver	Proprietary		

Table 1 Core technologies and Main Frameworks

- **Java Enterprise**

The technology of choice for implementation of Star Beat System is Enterprise Java**.**
J2EE is the standard for developing Enterprise application and utilities in Enel.
In addition the core framework of Star Beat System will have the same core on which the AMM Suite components are based.
The Application Server on which the application will be deployed will be Weblogic AS.

- **Weblogic**

One of the industry's best Application Servers Across Cloud and Conventional Environments, provides extended administration consoles as well as global system performance monitoring consoles.
Widely used in most Enel projects including Endesa. Limited or no training required for Enel system administrators thanks to Enel Group's consolidated skills with this product.

In front of the Application Server, IIS will be used as Web Server

- **IIS**

IIS works as a proxy for Weblogic Application Server and provides Clustering / Security / Load Balancing features. Moreover IIS is used in this way as a ENEL standard.

UI Layer

About the UI Layer, the following technologies will be used:

- **AngularJS**
- **HTML5**

State-of-the-art technologies for the UI Layer providing a framework for client-side model–view–controller (MVC) and model-view-viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications. Most noteworthy advantages:
Scalability: Most of the logic is performed on the client side reducing the server additional work to serve more clients.

Immediate user response: User interaction is handled by the client, which results in quicker response as the interactions with the server are limited.

Data Layer

- **Oracle 11g (First Phase)**

Oracle DBMS has confirmed itself through the years as highly scalable and reliable platform, confirmed by the market which granted it the leader position in the utility industry.
It has been explored the possibility to shift towards postgres because of its compatibility with other RDBMS (in particular with Oracle) and its ease of use.
While all the documentation retrieved in Internet is agree about this compatibility, there are many discussion about the performance of database in comparison with that of Oracle.
Due to the time-limits, the choice of Oracle has been preferred , having long experience on this DB.
In a second step, it will be possible to explore the using of Postgres in place of Oracle as Open source DBMS.

Furthermore in future it will be possible to explore the possibility to use a NoSQL DB like Cassandra: this solution could be particularly indicated to treat with "punctual" transaction and with little files in respect with a Distributed Filesystem.
A NoSQL Cassandra database has proven to have:
1. the ability to access and deliver data in near real-time. First and most importantly, Cassandra has proven itself capable of delivering near real-time performance to support interactive, Web-based applications at scale.
2. the ability to deploy across data centers. Cassandra can be deployed across multiple, geographically dispersed data centers to provide high-level redundancy, failover, and back-up & recovery capabilities.

In addition Cassandra is a better solution in treating a large number of little files rather than using a distributed file system.

Integration Services

- **REST Web Services**

1. Simple
2. You can make good use of HTTP cache and proxy server to help you handle high load.
3. It helps you organize even a very complex application into simple resources.
4. It makes it easy for new clients to use your application, even if you haven't designed it specifically for them (probably, because they weren't around when you created your app).

Note:

- Weblogic: vers. 10.3.6

The following table lists technologies that are not required by the business functionality for Star Beat System. It can be used as a reference for future requirements.

Other technologies					
Technology	Vendor	Product	License Type	Version	Environment
RADIUS Server	FreeRADIUS Server Project	FreeRADIUS	Open Source (GPLv2)	2.1+	All
DNS Server	PowerDNS.com	PowerDNS	Open Source (GPLv2)	3+	All
Network Access Server	n/a	n/a	n/a	n/a	n/a
Gateway GPRS Support Node	n/a	n/a	n/a	n/a	n/a
OS	IBM	AIX	Commercial	5.2+	All

OS	HP	HP-UX	Commercial	11i+	All
OS	Oracle (Sun)	Solaris	Commercial	8+	All
OS	Novell (SUSE)	SUSE Linux Enterprise	Commercial	9 SP3+	All
OS	Novell (SUSE)	OpenSUSE	Open Source (Misc.)	9 SP3+	Development

Table 2: Other Technologies

5.1.1.1. Clustering

Two core non-functional requirements coming from the customer: **high availability and horizontal scalability.** These requirements are satisfied using Weblogic clustering.
As described in the document 1250_AD5GL_AMMSoftwareArchitecture.docx:

"Clustering allows to provide load balancing and failover both for deployed applications and for JEE services. Clustering is not a single technology but an architectural approach that encompasses several different technologies

- **Clusterwide JNDI tree**: every object available via JNDI (being it an application module like an AMMS session EJB, or a Weblogic services like a JMS queue) is made available to JNDI clients (either running themselves in the cluster or running outside, like standalone client applications) via a centralized, clusterwide cluster. Clients lookup objects on the clustewide JNDI service and are unaware of what actual instance is returned and on which member of the cluster it is running. Members of the cluster can get down and new member may be added (this way providing horizontal scalability) without affecting the clients.

- **T3 remote communication protocol:** this is Weblogic's optimised, proprietary protocol for RMI and, more in general, java-to-java communication. When two objects communicate via T3, a permanent connection is established and the two peers send heartbeats to each other; this heartbeat is used to perform transparent failover in clustered environment when one peer instance fails.

- **Weblogic Proxy Plugin:** the proxy plugin is an extension to the most widely used HTTP servers (Apache, IIS, etc) that enables software load balancing and failover between the HTTP server instance and web applications running on a Weblogic Cluster."

The scalability is a **static** scalability
(Es: It's not possible to handle a new Operator and add ear in a "hot" deploy)

5.1.1.2. Asynchronous mechanisms in Star Beat System

The Star Beat System uses message-oriented architecture to implement the asynchronous mechanisms.
The core components to accomplish them are:

- **The JMS Resources**
- **The Message Driven Bean Components**

5.1.1.2.1. JMS

It's an a totally message-oriented architecture using the following concepts as described in 1250_AD5GL_AMMSoftwareArchitecture.docx:

- **Uniform Distributed Destinations:** these are part of Weblogic JMS clustering set of technologies. UDD are distributed destinations (queues in AMMS' case) in which load balancing and failover are completely managed by the container and completely transparent to the client.

- **XA Balanced Connection Factories:** as part of JMS clustering, Weblogic provides cluster-aware connection factories that provide transparent load balancing and failover to JMS client. Connection factories load balancing behaviour is used by AMMS in its message traffic management policy. See section about message traffic shaping for details.

- **Message Redelivery Policy:** Weblogic provides a message redelivery policy that can be defined for all queues or at queue level. The redelivery is triggered when the consumer does not complete the processing of the message and an exception is thrown back at the container. This feature is used by Star Beat System to perform retries in case of transaction rollback, see section about Transactions for more information.

- **Message Time to Deliver:** this is a Weblogic extension to the standard JMS message producer class, that allows the sender to specify for each message a time to deliver. When the message is committed to the target queue, it is not relayed to the consumer until the time to deliver delay has passed. This feature is used by Star Beat System for the following purposes:
    - Perform a programmatic reschedule of a task execution in case of application-controlled retry, according to the use case
    - Schedule a task for execution in the future
    - Work around the intrinsic limitations of two-phase commit protocol, that does not specify the order with which resource managers must commit.
      Once the transaction manager gives the order to commit to the resource managers, the will execute the order with a speed that depends on their performances. It is possible that at a certain point in time the message on target queue is committed and the row on the DB is not committed; if the message consumer looks for the updated row in the DB, it will not find it causing an exception.
      To overcome this limit, for transactions where the message recipient needs a record update by the message sender, we set a time to deliver in order to delay the message consumption in order to give time to the DB for committing the record update.

- **Message Paging:** when there are too many messages on queues, the paging feature offloads them to disk and keeps in memory only the message headers. This allows to preserve precious heap space avoiding out of memory errors, given the huge amount of messages coming in batches form external systems.

- **JDBC Persistent Store**: Star Beat System uses persistent point-to-point messages. The JMS specification ensures that each message will be delivered to one and only one consumer. The JDBC store is used to provide message persistence. Persistence of the message in the JDBC store is a hidden part of the distributed transaction involving the queue to which the message is sent so if the message cannot be persisted the entire distributed transaction is rolled back.

- **Foreign Connection Factories and Destinations**: foreign JMS resources are mappings of concrete JMS resources belonging to an external JMS server to virtual JMS resources belonging to the local server. Foreign resources are seen by applications running on a server as local resources, and the remote resources are completely hidden from the application.

5.1.1.2.2. Message Driven Bean

A message Driven Bean is an enterprise bean that allows Java Enterprise applications to process message asynchronously.

In particular, the following features of message Driven Bean are used:

- They are invoked asynchronously.
- They are relatively short-lived.
- They do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.

When a message arrives, the container calls a well defined message-driven bean's method (onMessage) to process it. This message handles it according with the application's business logic.
The onMessage method can use the architectural components of the AMM-Switch Framework. (WorkFlowManager , Executor) for implementing the logic to accomplish its goals.

The messaging system is used in particular in the implementation of:

- **Scheduler** : A persistent JMS Queue with only a one Message Driven Bean
- **Asynchronous Executor:** the number of Message Driven Bean will be tuning according to the workload in the scheduled activities
- **Asynchronous components in StarSync modules:** The size of the maximum pool will be tuned according the free resources in communication lines/resources.

5.1.1.3. Managing Transactions in Star Beat System

In Star Beat System the transaction are handled in two ways:

5.1.1.3.1. Container-Managed Transactions

In the case of Container Managed Transactions, it's not necessary to include statements that begin and end the transactions. In this type of managing, the Transaction boundaries are always marked by the start and end of EJB business methods.
The CMT will be used when possible in particular in StarPulse and StarGroove only when all the resources involved are all transactional.

5.1.1.3.2. Bean-Managed Transactions

In the case of Message Driven Beans of StarSync, responsible for checking the free resources and establishing the communication with Meters, the Bean-Managed Transactions will be used: in fact the communication with the Meter is not transactional by its nature (it's not possible to rollback a command, sent to the meter).
Furthermore, we need to implement a lock mechanism on the resources (in particular on the communications resources) that needs a partial transaction inside the context of the main method.

For this reason in Message Driven Bean of StarSync, there will be a main Transaction and other autonomous transactions that will be handled independently.

5.1.1.4. Synchronous Communications in Star Beat System

The Synchronous Communications between two remote subsystems in Star Beat is implemented using a stateless enterprise java bean.
The remote business interface exposes business logic to remote clients - clients running in a separate application from the EJB.
In particular, this mechanism will be used:

- in the functionality of Spot Reading, when from the StarPulse System, we need to call the StarSync in order to read a particular Meter.

5.1.1.5. Persistence Components in Star Beat System

The Persistence components in Star Beat System are implemented using Data Access Objects making use of the **Hibernate** implementation for the Java Persistence API (JPA) specification.
We call its interface in the Service Layer to do the CRUD operations on the database Tables
In this way we'll separate the low level data access operations from high level business services.

One of the cases that deserves to be highlighted is the implementation of Spot Reading:
This is a case of a long running HTTP request that causes a browser Timeout.

5.1.1.6. Implementation Components of Spot Reading as case of Long Running Request from Web Interfaces.

When we need to solve the problem of a long running HTTP request and we cannot reduce the time of the long running task, the only way to proceed is the following:

1) It's necessary to avoid that the Web Interface stay waiting for the result of the long running task. It's necessary to write something to the response: the Web Interface has to do at regular interval "a polling call" to the server: this can be a just a simple visualization of the progress.

2) The long-running task has to be implemented in a background thread instead of the HTTP request thread.

This is the case of Spot Reading of a Meter started by an operator in StarPulse.
The reading can take several minutes that could cause a client Timeout.

For implementing the Spot Reading, *three Rest Web Services will be created:*

1) **Web Service for starting an asynchronous task and immediately returning a unique identifier of the request.**

   When it receives the reading request, it has to start a Thread in which the calling to StarSync is encapsulated.

Because of instances of own thread could interfere with the container's ability to control its components' lifecycle, one possibility is to create the thread using the same asynchronous mechanism described in the previous paragraphs: a dedicated JMS Queue and the related Message Driven Bean.
In this case the maximum size of pool will be set according the number of spot requests we want to manages simultaneously.

Furthermore, the service has to generate a unique identifier of this request in order to return it "immediately" to the client. The client will use this identifier as a key to check the progression in the asynchronous task.

2) Web Service for verifying the progression of the task

In order to avoid the client is blocked waiting the end of the long running task,
the client has to call this web service at regular Time.
Using the key generated in the previous step, it verifies the progression in reading meter,
(querying, for example, an events table)
Furthermore it verifies if the asynchronous task has ended, checking the presence of a particular ending event in the same event table.

3) Web Service for returning the results

The Web client calls this Service for having the readings results associated to the request.

NB: It will be possible to implement three methods in the same Web Service.

5.2. Transition solution

In order to accomplish the main project dead line an architectural transition solution will be put in place. The transition architecture expects all the solutions listed in this document except has two peculiarities:

- Use of Oracle advanced queuing in order to communicate message from StarGroove module and StarSync module
- Use of scheduled stored procedure in the StarSync module in order to run and execute activities.

This transition solution expects the reuse of specific modules already implemented in the SMILE solution, this approach will be used just for the first version of the application and will be replaced in the second version with the final architecture design.

6. Implementation components and Deployment View

6.1. StarSync WL Cluster

StarSync is packaged as an ear containing the Message Driven Beans responsible of the processing of the activities messages forwarded by the Dispatcher.

As described in the previous paragraphs, these activities messages contain the scheduled readings that we need to do on meter.

A message driven bean is a type of enterprise bean which is invoked by EJB container when it receives a message from queue.
Message driven bean is a stateless bean and is used to do task asynchronously.

The StarSync ear is deployed inside a Clustered Domain on which the following queues are configured:

- The input queues in front of which the Message Driven Beans are.
  These input queues are physically distinct and each one has its own logical name.

- The output queues on which the message to return to StarGroove is put.
  They can be configured more physical output queues with the same logical jndi name.
  (in the figure: *out.queue*)

These queues are jms persistent queue of weblogic: these are targeted on cluster.
Using the jms persistent queue we have a single point of failure with respect to the message persistence. Only we lose the disk or database, we have lost messages.

It will be built an ear for each StarSync Operator present in the figure: **¡Error! No se encuentra el origen de la referencia.**.

The ear that is associated to ***the i-th Operator***, contains the three main types of Message Driven Bean for

- handling the GSM requests coming from the Dispatcher for the i-th Operator
- handling the GPRS requests coming from the Dispatcher for the i-th Operator.
- a third type of Message Driven Bean will be used to take charge of the expired messages.
  In fact it's possible to configure the jms gsm/gprs queues in order to routes the expired messages in an another garbage queue.

The parameters in the weblogic-ejb-jar.xml will be set according to the free resources for that Operator.
In particular the <max-beans-in-free-pool> of the two Message Driven Bean will be set according the maximum number of ISDN lines and GPRS resources for that Operator.

The <destination-jndi-name> are set respectively to the values:

- jndi name of the physical GSM queue name in input to StarSync:
  in.queue.GSM.Operator i-th
- jndi name of the physical GPRS queue name in input to StarSync:
  in.queue.GPRS.Operator i-th

NB:

All the StarSync ear have the same configuration and uses the same classes. In this way, we haven't the need to maintain different StarSync code.

In the GSM, GPRS Message Driven Bean, the delegate classes for handling the specified technology will be injected.
However, if a message of OperatorX is put on the queue of OperatorY the MDB/StarSync of OperatorY will be able to process the message.
The only difference in the various StarSync is the ***fine tuning*** *in the values of size pool according to the queues of which the Message Driven Beans are subscribers.*

In the below figure, there are three StarSync for handling three operators.
If we need to handle another operator, it is sufficient to deploy another ear and to tune the size pool of GSM and GPRS Message Driven Beans according to the resources of the four Operator.

In the StarSync ear , there are the Service Rest for implementing "the Configurator module" that is used to deploy the application console.
This application console allows to change and save the settings of applications like the couples <ip, port> for Access Server associated to Operator, etc.

6.2. StarGroove WL Cluster

StarPulse Bach is packaged as an ear targeted on StarGroove Cluster.
If there will be the need to increase the workload of the system, the size pool of the asynchronous components (Message Driven Bean of Executor and Dispatcher) will be increased together with the related Work Managers of the servers.
If the number of Threads, that the server has to handle, is too high, another node will be added and the resources will be balanced.

The the creation of scheduled tasks is driven by a scheduler
The Scheduler must be the unique point on which the scheduled tasks are activated.
The Scheduler is implemented using a JMS persistence queue with a ONE Message Driven Bean.
A startup process puts a message in the queue. The Message Driven Bean executes the logic to do at regular time interval and puts again the Message in the queue with the appropriate redelivery Time.

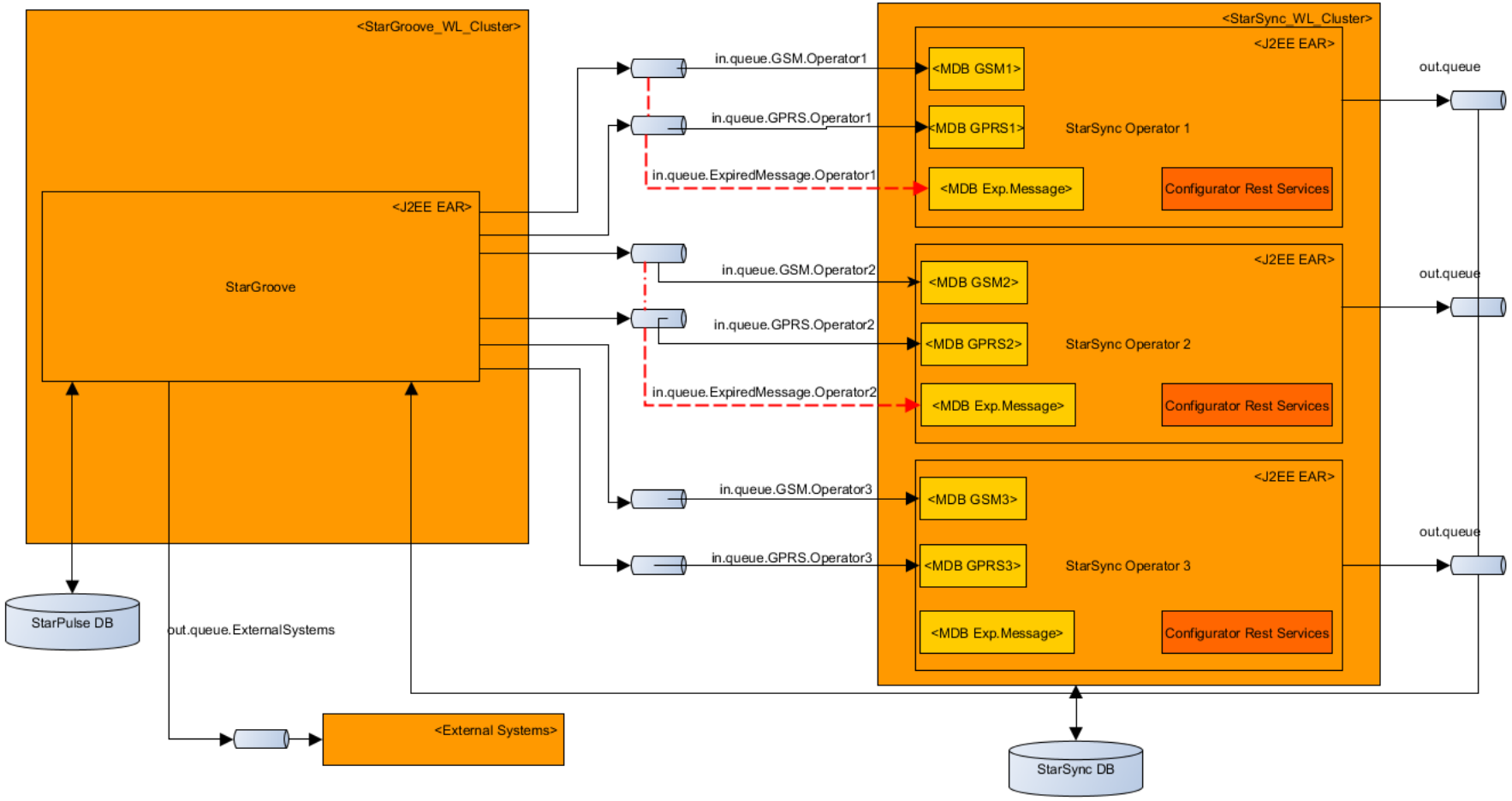

Figure 10: StarSync and StarGroove Cluster

6.3. StarPulse WL Cluster

The StarPulse system has to interact with the StarSync system in the case of Spot Reading.
In order to interact with StarSync, it has to expose a synchronous interfaces. The synchronous communication between different Systems in Star Beat System is implemented using a Stateless Bean.
Using this stateless Bean we are able to:

- Create a remote interface exposing the business methods.
- This interface will be used by the ejb client application.

The StarPulse and StarSync Cluster is described here:

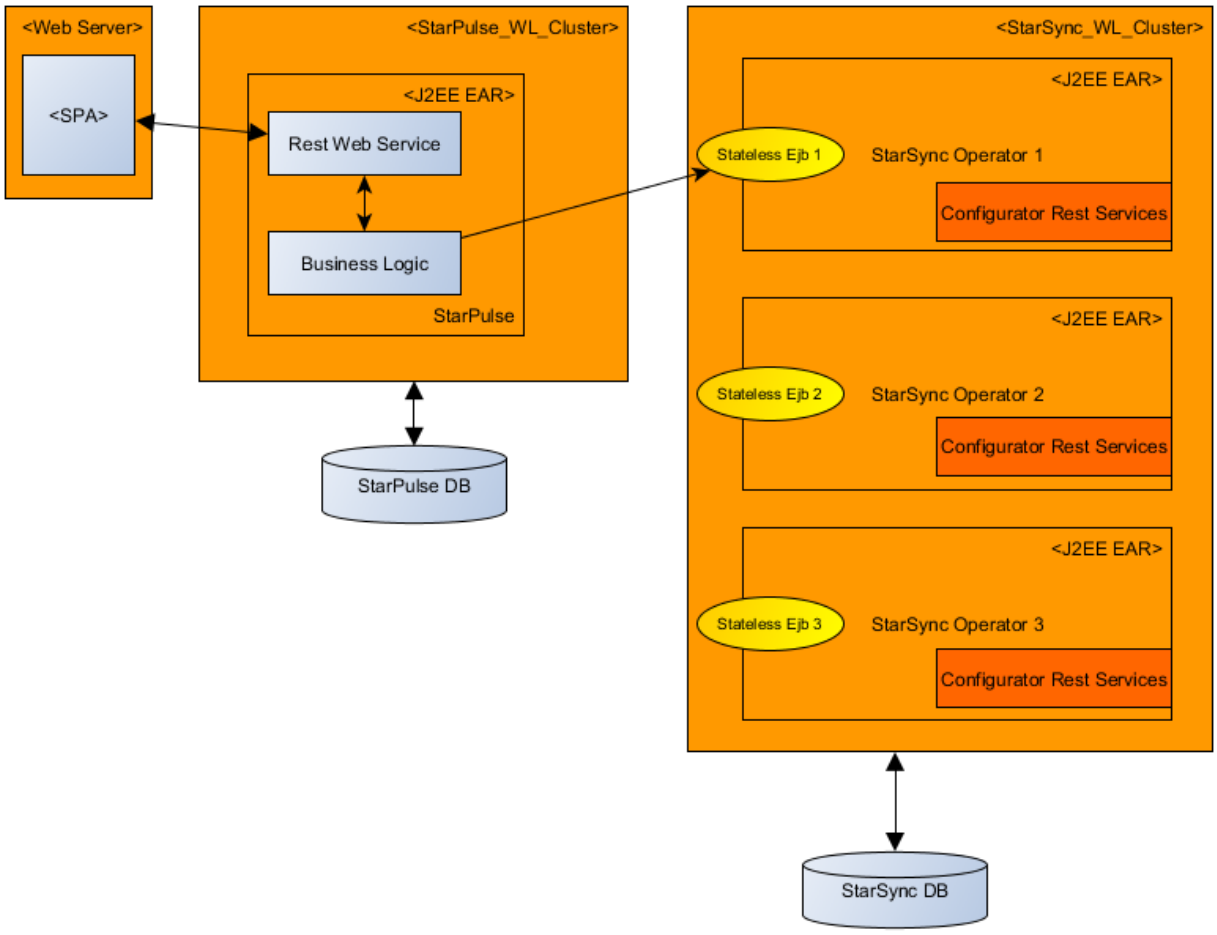

Figure 11: StarSync and StarPulse Cluster with interaction

The figure highlights that the pages implementing user interfaces can be installed in the Web Server.
In fact the Web interfaces are implemented using a Javascript Framework (AngularJS)
according to the pattern of a single-page application (SPA),
"A SPA is a web application or web site that fits on a single web page with the goal of providing a more fluid user experience similar to a desktop application. In a SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load and the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user action"

The logic of the framework is described in the following figure:

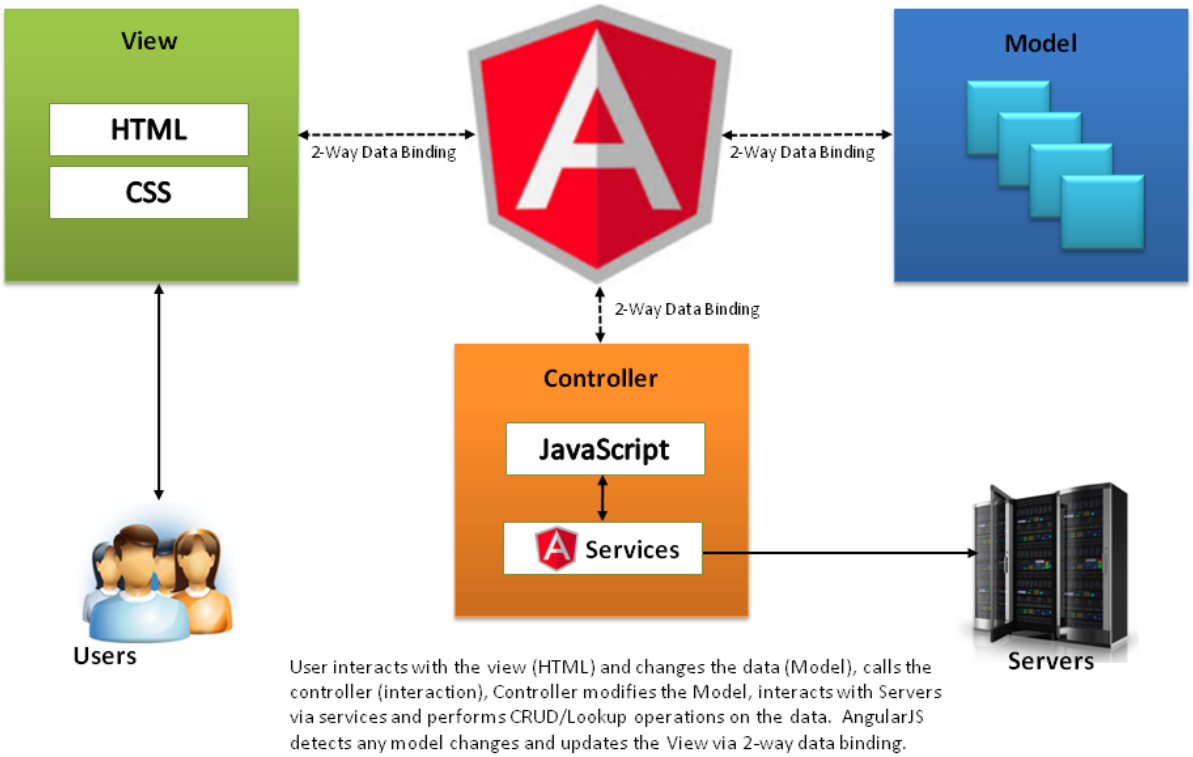

Figure 12: AngularJS Framework

In the ear deployed in StarPulse System, there are all the REST Web Services that serve the application requests.

REST Web service follows four basic design principles:

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation (JSON), or both.

They are light in the sense that there aren't SOAP XML Messages but use of HTTP methods establishing

a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods.

The Rest Web Service returns the result to display to the pages in the JSON format.

6.4. Star Beat Web Application Structure

In defining the web application structure with new technologies and in particular with AngularJs (MVC framework - client), the following aspects must be taken into account:

- Single Page Application
- Modularity
- Maintainability

6.4.1. Installation on server

Below the structure of Web application Star Beat with two modules is described

```
iis
└──static
  └──sb
    ├──application
    │  │  index.html
    │  │
    │  ├──conf
    │  │    conf.json
    │  │
    │  ├──css
    │  │    external-min.css
    │  │    sb-min.css
    │  │
    │  ├──fonts
    │  ├──img
    │  ├──js
    │  │    sb-min.js
```

```
│   │
│   └──libs
│       external-min.js
│
├──moduleName1
│   ├──css
│   │   moduleName1-min.css
│   │
│   ├──img
│   ├──js
│   │   moduleName1-min.js
│   │
│   ├──SubModule1Name
│   │   └──views
│   │       subModule1Name-view1.html
│   │       subModule1Name-view2.html
│   │
│   └──SubModule2Name
│       └──views
│           subModule2Name-view1.html
│
└──moduleName2
    ├──css
    │   moduleName2-min.css
    │
    ├──img
    ├──js
    │   moduleName2-min.js
    │
    ├──SubModule1Name
```

```
│ └──views
│      subModule1Name-view1.html
│
├──SubModule2Name
│ └──views
│      subModule2Name-view1.html
│
└──SubModule3Name
  └──views
      subModule3Name-view1.html
```

In the definition of the structure above, each module is self-consistent and divided into submodules.
In more detail:

- "**application**": It contains files of the whole Star Beat application and in particular: the configuration , style sheets , fonts , images , routing among application modules , third party javascript libraries and the index.html file that is the single page of the whole application , according the SPA frameworks.

- "**ModuleName**": It contains files of the specific module. In particular :
  - images eventually used by the module,
  - the file " moduleName - min.js " containing the javascript code of module that has been " minified " and contains the definition of the module and the definition of the routing among the views of sub-modules

    PS: **Minification** in JavaScript, is the process of removing all unnecessary characters from source code without changing its functionality. These unnecessary characters usually include white space characters, new line characters, comments, and sometimes block delimiters, which are used to add readability to the code but are not required for it to execute.

  - The file "moduleName-min.css" containing the CSS of the module concatenated and minified. The module is divided into sub modules, where the directory "**SubModuleName**" contains the views (templates) of the specific sub module that will be injected into the single page application.

Before the execution of each REST Service, the system needs to verify that the user is authenticated and authorized to perform the operation.
They will be implemented some classes that implement the interface HandlerInterceptor of Spring and act as a filter to direct calls to the servlet Spring (DispatcherServlet) .

6.4.2. Authentication

A class ***AuthenticationInterceptor*** (for example: ***com.enel.sb.fe.srv.skel.AuthenticationInterceptor*** verifies that one of the following fields has a not null value in the request header:

Proxy-Remote-User
LOGON_USER
REMOTE_USER
REM_USER

In this case, the process verifies the authentication.
Otherwise the execution is interrupted and the following code error is returned:

```
{
 "errorMessages": [
   "Not authenticated user"
 ],
 "output": null
}
```

The class AuthenticationInterceptor is mapped in order to intercept all the requests towards DispatcherServlet. So, in the implementation of REST Services, it's not needed to make any implementations about authentication

6.4.3. Authorization

A class **<*Function*> *AuthorizationInterceptor*** (for example: ***com.enel.wom.fe.srv.skel.sb.<Function>AuthorizationInterceptor***) where <Function> is the name of the functionality to verify the authorization, for example ***MeterInventoryManager.***
The interceptor verifies that the user is authorized to execute the functionality for the DTR input parameter (parameter in the url, for example : territory='DD')
In more details:

The code of the activity is a property initialized using Spring Configuration File
The interceptor verifies if the "territory " parameter is present . If no, the following error code is returned:

```
{
 "errorMessages": [
   "missing parameter: territory"
 ],
 "output": null
}
```

If the territory parameter is present, it's must be called ***a functionality of Profile Module*** in order to verify if the user is authorized to execute the activity in the specified territory.

If the user is authorized, the Rest Service is executed.
Otherwise, the following error is returned

```
{
 "errorMessages": [
       "User: "user" not authorized to execute the activity: in the territory: DD"
 ],
 "output": null
}
```

6.5. The deployment unit

6.6.

7. Definitions, Acronyms, and Abbreviations

All terms, acronyms and abbreviation are contained in the Project Glossary and here below.

Abbreviations - Acronyms	Definition
AMM	Automated Meter Management
AMM Suite	Global AMM Solution including multiple systems
AMMS	Automated Meter Management System
ASDU	Data Unit Application Services
ATR	Third Party Network Access
BO	Back Office
CPD	Data center
CUPS	Universal Supply Point Code
DB	Data Base - Base Dati
DH	Hourly discrimination
EI	External Interface - Interfacce Esterne al Sistema
EMD	Equipment Master Data - Dati Anagrafici Attrezzature
EXABEAT	Measure system
FE	Front End
GME	Electronical Measuramente Device - Gruppo di Misura Elettronico
GUI	Graphical User Interface - Interfaccia Grafica Utente
Heart Beat	Master Data System
Job	Group of tasks related to the same communication session (multidrop chain). A job can also contain tasks related to a single meter.
Meter	Measure Point
Multidrop	The multidrop feature, in when more than one meter that aim to the same node and are bounds in a chain to be read and in the same communication session.
N.A.	Does not apply
OLA	Online Activity/Activities
POD	Point of delivery
PS	System Parameter - Parametri di Sistema
RPT	Report
SB	Star Beat
SBS	Star Beat System
SBS	Star Beat System - Smart Metering System for the acquisition of measurement data from devices connected to the high & medium voltage grid
SCH	Scheduler - Modulo Star Beat che si occuppa di pianificare le attività da eseguire sugli Equipment
SI	System Interface - Interfaccia di Sistema
Task	Single operation (e.g. Load Profile Reading) to execute.
TOUs	Territorial Organizational Units
UI	User Interface - Interfaccia Utente
UOT	Territorial Organizational Unit
WO	Work Order - Ordine di lavoro (Richiesta di esecuzione attività su GME)
WORK BEAT	Work Order System
XSD	Scheme XML definition

Figure and Table Indexes

Tables

Non è stata trovata alcuna voce dell'indice delle figure.