MOEEBIUS

Modelling Optimization of Energy Efficiency in Buildings for Urban Sustainability

D4.4 MOEEBIUS Building-Level Middleware

Version number: 1.0
Dissemination Level: PU
Lead Partner: HON
Interim Due date: 30/04/2017
Type of deliverable: Other
STATUS: Delivered

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680517

Published in the framework of:

MOEEBIUS - Modelling Optimization of Energy Efficiency in Buildings for Urban Sustainability

MOEEBIUS website: www.moeebius.eu

Authors:

Jakub Malanik, Jiri Rojicek, Zdenek Schindler – HON

Borja Tellado Laraudogoitia, Pablo de Agustin Camacho – TECNALIA

Georgios Kontes – THN

With contributions from the project consortium as regards system deployment requirements to be met.

Revision and history chart:

VERSION	DATE	EDITORS	COMMENT
1.0	29/04/2017	HON	Submitted to the EC

Disclaimer:

Table of content

List of tables

List of figures

Glossary

Acronym	Full name
Amazon AWS	Amazon Web Services
Amazon EC2	Amazon Elastic Compute Cloud
AMI	Amazon Machine Image
API	Application Programming Interface
BMS	Building Management System
CQL	Cassandra Query Language
CSV	Comma-separated values
DB	Database
EIP	Enterprise Integration Patterns
HTTP(S)	Hypertext Transfer Protocol (over TLP/over SSL/Secure)
IoT	Internet of Things
JMS	Java Message Service
JSON	JavaScript Object Notation
MOEEBIUS	Modelling Optimization of Energy Efficiency in Buildings for Urban Sustainability
MQTT	Message Queue Telemetry Transport
REST	Representation State Transfer
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
URL	Uniform Resource Locator
WSO2	Open source technology company developing service-oriented architecture middleware
WSO2 DAS	WSO2 Data Analytics Server
WSO2 DSS	WSO2 Data Services Server
WSO2 ESB	WSO2 Enterprise Service Bus
XML	eXtensible Markup Language

1 Executive summary

The aim of this deliverable is to describe the process of selecting and building MOEEBIUS building level middleware layer.

The document starts with the middleware requirements and expectations resulting in possible options and choices to be made. Once the platform is selected, the document elaborates on hardware choices, which are important to provide expected performance at reasonable cost.

To explain the middleware expected use, it then describes key middleware building blocks and their relationships. The most crucial part describes which functionalities developers can expect from the middleware and how they are expected to utilize it, however, the contents of this document should not be mixed with software development documentation. Finally, this part is accompanied by selected use cases.

Please note, that the key deliverable of the tasks 4.3 and 4.4 is not this document, but the middleware itself — this document describes the journey how it was achieved.

This deliverable significantly overlaps with D4.5 district level middleware.

2 Introduction

The concept of middleware is to introduce a layer providing services to software operations beyond those available from the operating system. It simplifies software development especially by unifying communication and data access in general.

Within MOEEBIUS framework the middleware serves two primary functions

- Data integration: unify data collection, i.e. offering pilot sites as well as future site owners support to easily send data to a common data store without developing any site-specific drivers;
- Application integration: allow analytics developers to easily hook their tools and algorithms to live and archive data without any extra effort on developing data management layer.

These core and fundamental requirements are also accompanied by others, also quite essential like security, authentication, scalability, messages managements, system monitoring, etc. — and such functionality should be transparent to end users.

This concept is valid regardless the abstraction level applied to the functionalities that want to be provided as a service.

2.1 Scope of this documentation

This document summarizes the requirements of the MOEEBIUS building level middleware, underlying the specific platform choice and the overall implementation concept. However, it does not include a detailed description on how MOEEBIUS functionality is to be implemented on this middleware and should not be mixed with software development documentation.

The reader should have an understanding of general MOEEBIUS goals.

2.2 Document structure

This chapter focuses on motivation and general concept of middleware. Chapter 3 is the key part of this deliverable describing MOEEBIUS middleware requirements, infrastructure platform choice, hardware selection and individual software components. It also covers expected data workflow and some communication details. Chapter 4 outlines selected building level MOEEBIUS-related use cases and describes how they can be supported with by the middleware developed.

2.3 Middleware concept

2.3.1 Definition

Middleware is software that functions as a conversion or translation layer. Middleware is also a consolidator and integrator. Custom-programmed middleware solutions have been developed for decades to enable one application to interface with another, which either runs on a different platform or comes from a different vendor. In particular, within European Projects several middleware solutions for buildings and smart grids have been reported. Some examples here include e.g. the IFC4-based middleware developed within BaaS fp7 project, the LinkSmart middleware developed within FP6 European Project HYDRA, the middleware developed within ENCOURAGE project of the ARTEMIS Industry Association call and many others. The main downsides of these solutions are the following:

- they are (heavily) tailored towards the requirements of each umbrella project;
- their level of maturity, reliability, stability and security is below available commercial solutions;
- there is a huge lack of community support and public documentation in all these solutions.

Today, this effort can be reduced as there is a diverse group of commercially available middleware products which can be configured and tailored for specific needs.

Middleware supports and simplifies complex distributed applications. It may include web servers, application servers, messaging and similar tools that support application development and delivery. Middleware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture.

Middleware often enables interoperability between applications that run on different operating systems, by supplying services so the application can exchange data in a standards-based way. Middleware sits "in the middle" between application software that may be working on different operating systems. It is similar to the middle layer of a three-tier single system architecture, except that it is stretched across multiple systems or applications.

At this point, regarding application in buildings and building level middleware, we have to emphasize the difference in MOEEBIUS between the Building Management System (BMS) and the middleware. According to the MOEEBIUS project view, the BMS of each building is responsible for operating all the building systems and collecting the related sensor and control commands data (and of course is also responsible for the low-level control of all the devices). Consequently, each BMS system uses a gateway to "push" the required data to the middleware defined here. These data in turn are complemented with data uploaded from other sources

(e.g. NOD devices, weather predictions, signals from the grid, etc.) and are made available to the analytics services.

2.3.2 Service oriented architecture and microservices

In order to allow independent development of individual analytics, MOEEBIUS design is based on service oriented architecture, where services are provided to other components. Services then communicate through standard communication protocols over a network.

Services or microservices can be seen as components loosely linked together to create an application. In MOEEBIUS context, data validation or generally data management, various analytics, etc. are planned to be implemented as microservices.

Generally, each microservice provides an API endpoint, often but not always a stateless REST API which can be accessed over HTTP(S) just like a standard webpage. To use microservices effectively, some kind of orchestration is required, which is provided by Enterprise Service Bus.

2.3.2.1 Enterprise Service Bus (ESB)

An enterprise service bus (ESB) is a software architecture construct that enables communication among various applications. Instead of having to make each of the applications communicate directly with each other in all their various formats, each application simply communicates with the ESB, which handles transforming and routing the messages to their appropriate destinations.

An ESB provides its fundamental services through an event-driven and standards-based messaging engine (the bus). Thanks to ESB, integration architects can exploit the value of messaging without writing code. Developers typically implement an ESB using technologies found in a category of middleware infrastructure products, usually based on recognized standards. As with a Service-Oriented Architecture (SOA), an ESB is essentially a collection of enterprise architecture design patterns that is now implemented directly by many enterprise software products.

2.4 Building- vs. district-level middleware

The above described concept of middleware applies for any level (building or district); it basically only differs in scale. Once scalability is ensured, it makes no longer sense to build separate middleware layers for the different levels, since the same implementation/solution can be utilized for all cases. Should network traffic be the issue, the very same middleware can be deployed locally and let the ESBs communicate as any other service. Following chapter lists all other requirements of the MOEEBIUS middleware and describes the process of selecting the right platform capable to serve as both building- and district-level middleware. This is also a reason why D4.4 and D4.5 significantly overlap.

3 MOEEBIUS building level middleware requirements

The key question for MOEEBIUS was either to develop custom middleware or to customize an existing one. To address it properly, the following aspects were inspected:

3.1 Security and privacy

Name	Description	Note	Priority
Secure internet communication	All internet communication must be carried over a secure channel	Certificate for supporting the secure communication	Must
Private data protected	The privacy related data (e.g. emails), must be stored in the cloud separately and must not be shared with not entitled parties or components		Must
License agreement	Data owners must agree with storing their data in the platform		Must
Support for Standard Based Authentication Protocols	Usage of standard protocols like OAuth 2.0 will pave the way for smoother security integration in the future.		High

3.2 Usability

Name	Description	Note	Priority
Tools maturity	The tools for interacting with the solution must be standard and well accepted and supported by community	Tools for external partners or end users must be simple enough and easy to use, while administration tools used by the consortium members may be relatively complex.	High

3.3 Interoperability / Compatibility

Name	Description	Note	Priority
External access to cloud platform	It should be possible to allow the consortium members securely accessing the cloud platform remotely from their premises, i.e. without need to install their services in the cloud.	The access should be monitored to evaluate the impact and possibly detect security threats.	Low
Seamless Data Access	In order to protect the platform from future potential changes to data sources, the data access layer should be capable of accessing different data sources		Medium

	without involving higher architecture layers.		
Big Data Database Support	In order to cover potential future big data needs supporting big data databases like Cassandra or MongoDB would be beneficial.		Medium
Distributed Messaging Support	This feature would be beneficial to be able to handle large number of request with short response time and without any downtime.		Medium
Machine Learning Capabilities	Having both capabilities of distributed machine learning and performing analytics on a single machine in order to support different use cases.		Medium
Real-time Data Streaming Capabilities	Because this part deals with real time events, in order to not miss any events in the system, supporting high availability features is crucial. Also this section should be capable of supporting different message contents such as XML, json and text and should be	Used protocols will be specified based on pilot needs.	Must

	compatible with different protocols such as JMS, SOAP, REST and MQTT.		

3.4 Performance

Name	Description	Note	Priority
Linear Scalability	Every part of the platform should be equipped with high availability features which provide a predictable linear performance growth rate.		High
No Single Point of Failure	Through high availability features we should have the possibility of having a system with no single point of failure.		Medium
Support for Transactional Behavior	The system should be capable of calling a group of services in a transactional manner with rollback capabilities in the cases witch this feature is needed.		Low
Guaranteed Message Delivery	In some situations, like sending sensitive messages to third-party systems while the recipient is offline, this feature would be required.		Medium

3.5 Management

Name	Description	Note	Priority
Service Publication	A centralized storage and management solution that plays the role of a proxy for seamless access to services provided by different parties.		Medium
Service Access Authorization	Having a central access authorization system preferably based on single sign on to facilitate ease of use by different parties.		High
Service Dependency Management	This system will provide the relationship/dependencies between services published by different parties.		Medium
Service Traffic Management and Monitoring	This feature will provide the ability for real-time monitoring of services in order to detect any faults or stress points and staying compliant with the Service Level Agreements.		Medium
Custom Application Repository	Defining a central storage platform for defining third party applications, access rights and usage metrics.		Medium
Centralized Visualization Dashboards with access control level policies	This capability allows creating all required dashboards for monitoring different aspects of the platform's functionality and allows access to different parties based on their needs and access level.		High

SOA Governance	This capability allows registering different shared assets like API services in a centralized manner. It also enables managing dependencies between services, managing their lifecycle, checking constant compliance with predefined standards and informing consumers about any changes in service definitions.		Low

Taking into account the software development complexity, we decided to adopt and configure an existing middleware solution. Among those available (e.g. Microsoft Azure, Amazon AWS, etc.), we found WSO2 as the only one offered as the open source software meeting our requirements (a detailed analysis on the requirement matching can be found in ANNEX I of MOEEBIUS D3.1).

The selection of WSO2 for MOEEBIUS middleware does not imply that the middleware is finalized with no further work required. On contrary, WSO2 components have to be set up properly and configured for MOEEBIUS specific needs. Selecting WSO2 is rather like selecting an environment for building the middleware.

3.6 Infrastructure as a Service

After selecting WSO2 as the middleware platform, the next question concerns the deployment of the middleware solution itself. Here, the following three options have been considered:

- Custom-built platform or on-premise (possible for building level only)
- Microsoft Azure cloud;
- Amazon AWS.

Following, a short analysis on the advantages and disadvantages of each option are presented.

3.6.1 Custom-built platform

This option was soon eliminated as it requires a good IT support and infrastructure (internet access) without any restrictions and even though IT support was possible utilizing some partners' infrastructure (Honeywell, THN, Tecnalia), strict IT policies in all companies ruled out any deployment on their premises.

3.6.2 Microsoft Azure

This is a viable option with no significant limitation; Azure however provides better support for Microsoft technologies.

3.6.3 Amazon AWS

Amazon was finally selected as the best option. It provides reliability, accessibility and pricing comparable to other commercially available platforms, but it also comes with good history of hosting WSO2 services — the latter implies rich community support.

3.7 MOEEBIUS Middleware Beans

All instances running on the AWS are based on the bitnami-cassandra-3.9-1-r28-linux-ubuntu-14.04.3-x86_64-hvm-ebs Amazon Machine Image (AMI). This AMI contains Ubuntu 14.04.3 Linux operating system and preinstalled Cassandra node readily available with working default configuration.

Following, selected components of the WSO2 architecture (specific parts of the overall architecture presented in Figure 1 that are relevant/essential to the MOEEBIUS solution are described.

3.7.1 Middleware platform

After exploring available possibilities, we have selected the WSO2 integration platform as the framework for the middleware. The WSO2 integration platform is a mature technology used by a large number of companies. The WSO2 integration solutions are based on WSO2 Carbon which hosts components for integration, security, clustering, governance, statistics, and other features in the middleware space.

The usage of Attune platform built on top of the Tridium Niagara Framework as a middleware framework mentioned in the DoW was not evaluated as the most viable solution when compared to the solution finally accepted. The reasons for this are following:

- Niagara for pilot instrumentation would rely on local Honeywell braches to install Niagara (JACEs), some pilot owners planned to make a move in this direction, but it never happened (it was not and cannot be driven by HON team in MOEEBIUS).
- Niagara is a commercial product which may restrict the possibilities due to inability to change the underlying platform and may also incur deployment complications because of licensing issues.
- Attune was intended as a platform for analytics, however since the compilation of DOW, Honeywell strategy has moved a bit here and Attune is no longer a good choice. Moreover – we'd have to deploy the whole Attune

production environment (Honeywell proprietary) and make any analytics Attune-compatible. Having an open platform (WSO2), we can avoid obvious problems and take any algorithm from any environment (including Attune) and connect it to our middleware.
- Selecting Attune platform could lead to us experiencing difficulties from currently limited Honeywell support of Attune platform.

Selection of WSO2 integration platform provides us with numerous advantages, some of which are:

- It is 100% open source project which means it is completely free to use. Anybody can view the source code which is beneficial in understanding how the platform works. Additionally it offers flexibility in configuring and extending the open source code to meet our requirements.
- The componentized architecture spanning the entire breadth of Service Oriented Architecture (SOA) enables us to deploy only what we need when we need it, so we can easily expand the already deployed middleware with new components depending on our needs.
- The WSO2 platform is fully cloud ready.

One of the greatest strengths of the WSO2 platform will be apparent when all the development activities are started and we will be able to easily create and fine-tune the routing sequences, interfaces and APIs required by devices and services developed by various partners. The routing sequences can perform conditional routing based on the message content or type along with message transformation, which allows the middleware to orchestrate operations among multiple services. These changes are made online while the platform is running and can be done either in Management Console in web browser or offline and then uploaded to the platform as a .capp (Carbon Application Package) package. There is ESB Tooling plugin to Eclipse which add intuitive graphical user interface for developing of ESB sequences and APIs into Eclipse.

In the current state the middleware exposes number of general APIs which are used to access the Cassandra database. In further project phases greater number of APIs and routing sequences will be added to conduct orchestration between the diverse MOEEBIUS framework components such as DAE, DSS, BEPS, etc. Once all the configuration and integration of the components into the middleware is completed, the documentation will be updated.

3.7.2 WSO2 Carbon

WSO2 Carbon redefines middleware by providing an integrated and componentized middleware platform that adapts to the specific needs of any enterprise IT project - on premise or in the cloud.

Being 100% open source and standards-based, WSO2 Carbon enables developers to rapidly orchestrate business processes, compose applications and develop

services using WSO2 Developer Studio and a broad range of business and technical services that integrate with legacy, packaged and SaaS applications.

The lean, complete, OSGi-based platform includes more than 175 components – OSGi bundles or Carbon features. The WSO2 Carbon core framework functions as "Eclipse for servers" and includes common capabilities shared by all WSO2 products, such as built-in registry, user management, transports, security, logging, clustering, caching and throttling services, co-ordination, and a GUI framework.

3.7.3 MOEEBIUS Enterprise Service Bus Implementation

WSO2 ESB is a fast, light-weight, and versatile enterprise service bus. It is 100% open source and is released under Apache Software License Version 2.0, one of the most business-friendly licenses available today. Using WSO2 ESB allows performing a variety of Enterprise Integration Patterns (EIPs) [13] using mediators, including filtering, transforming, and routing SOAP, binary, plain XML, and text messages that pass through your business systems by HTTP, HTTPS, JMS, mail, etc. An overview of the ESB architecture is shown in Figure 1.

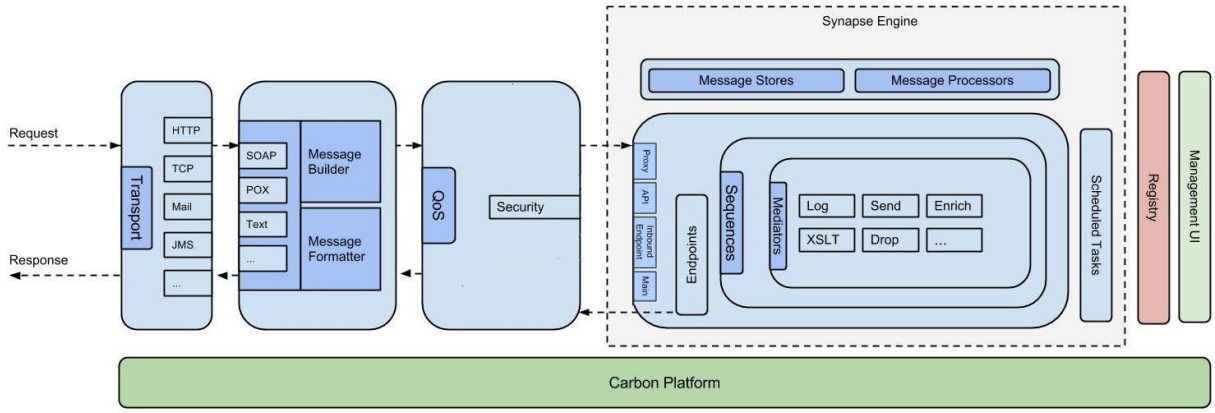

Figure 1: Component based architecture of ESB

The following diagram (Figure 2) illustrates how two applications can exchange messages using the WSO2 ESB (the components of the pipes are not in a specific order):

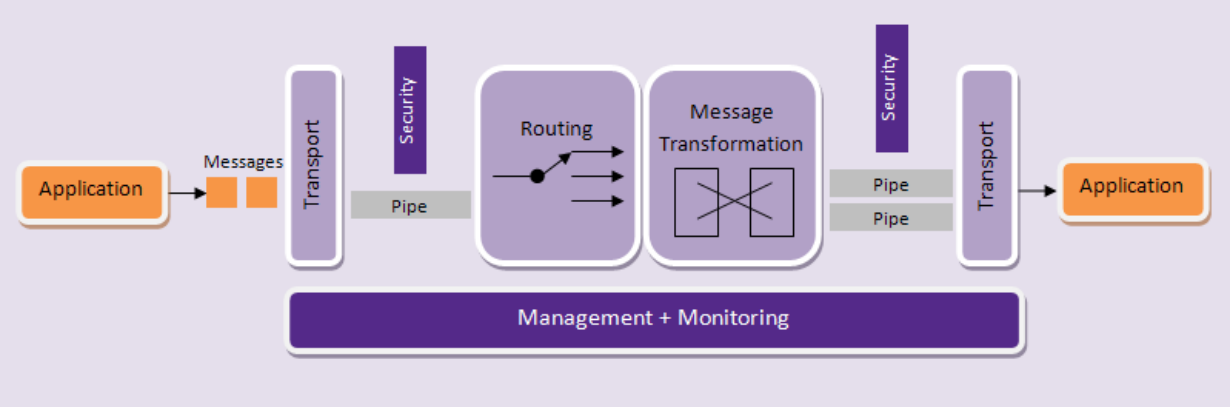

Figure 2: Messaging based architecture of ESB

1. An application sends a message to the WSO2 ESB.
2. The message is picked up by a transport.
3. The transport sends the message through a message pipe, which handles quality of service aspects such as security. Internally, this pipe is the in-flow and out-flow of Apache Axis2. The WSO2 ESB can operate in two modes:
   - Mediating Messages - A single pipe is used.
   - Proxy Services - Separate pipes connecting the transport to different proxy services are used.
4. Both message transformation and routing can be considered as a single unit. As the diagram specifies, there is no clear separation between message transformation components and routing components. In WSO2 ESB, this is known as the mediation framework. Some transformations take place before the routing decision has been made while others take place after the routing decision. This is part of the Synapse implementation.
5. The message is injected to the separate pipes depending on the destinations. Here again, quality of service aspects of the messages is determined.
6. The transport layer takes care of the transport protocol transformations required by the WSO2 ESB.

The above diagram shows how a request propagates to its actual endpoint through the WSO2 ESB using its architecture. Response handling is the reverse of this operation. There are other areas like Working with Scheduled Tasks and Events that are not shown in the diagram. All these components can be analysed and monitored through WSO2 ESB Analytics.

Messages can be sent directly into the WSO2 ESB using rest, by utilizing the API features of WSO2. An API in WSO2 ESB is similar to a web application deployed in the WSO2 ESB runtime: each API is anchored at a user-defined URL context,

much like how a web application deployed in a servlet container is anchored at a fixed URL context. An API will only process requests that fall under its URL context. The API defines one or more resources, which is a logical component of an API that can be accessed by making a particular type of HTTP call.

As stated before, once a message is received, it can be further processed (e.g. transformed, filtered or routed) using mediators. WSO2 ESB includes a comprehensive mediator library that provides functionality for implementing widely used Enterprise Integration Patterns [13]. In addition, custom mediators that provide additional functionality can easily be developed, using various technologies such as Java, scripting, and Spring.

Moving one step higher in the messaging architecture hierarchy, a sequence is a set of mediators organized into a logical flow, that allow implementing message pipes and filter patterns. Sequences can be added to proxy services and APIs.

WSO2 ESB version 5.0.0 is deployed in the middleware. WSO2 released new package containing WSO2 ESB, DSS and other similar products called WSO2 Enterprise Integrator, however by the time of the release, the middleware was already deployed and configured.

Figure 3 illustrates the Deployed APIs section in WSO2 ESB Graphical User Interface accessible via web browser. Names of the APIs are self-describing – GetDataAPI is used for routing requests for data from Cassandra database, InsertDataAPI routes incoming requests containing new data to be saved into Cassandra database and EchoTestAPI was used for testing purposes during development.

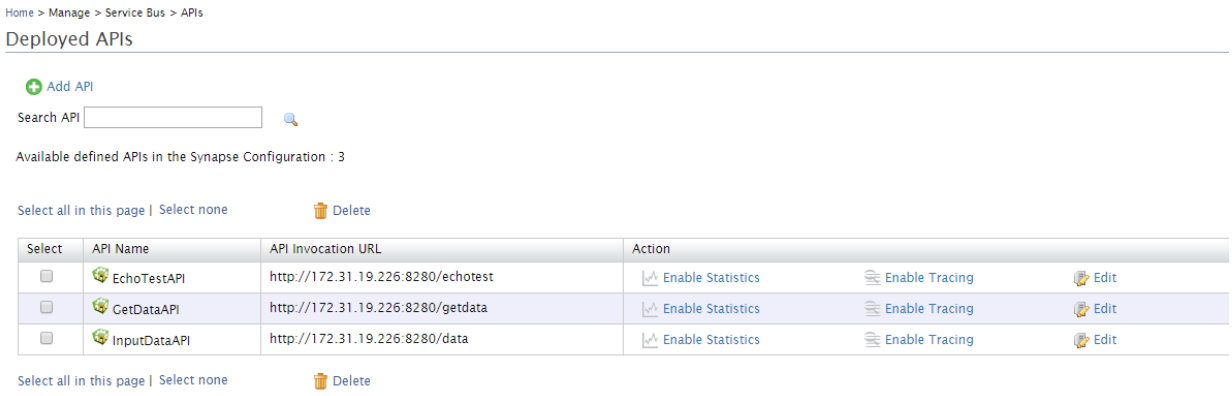

Figure 3: Deployed APIs tab in WSO2 ESB GUI

The following figure illustrates one of the REST API sequences (InputDataAPI) defined within MOEEBIUS. First, the message is logged. Depending on the content-type of the message it is processed either as a JSON or an XML. This processing transforms the message into a format which is accepted by the instance running the WSO2 Data Service Server (described in the next Section) where it is sent. Incompatible content-type results in returning the message

immediately. After being processed by the WSO2 Data Service Server (e.g. data is saved to a Cassandra database or retrieved from it) the result is send back to the requesting application.

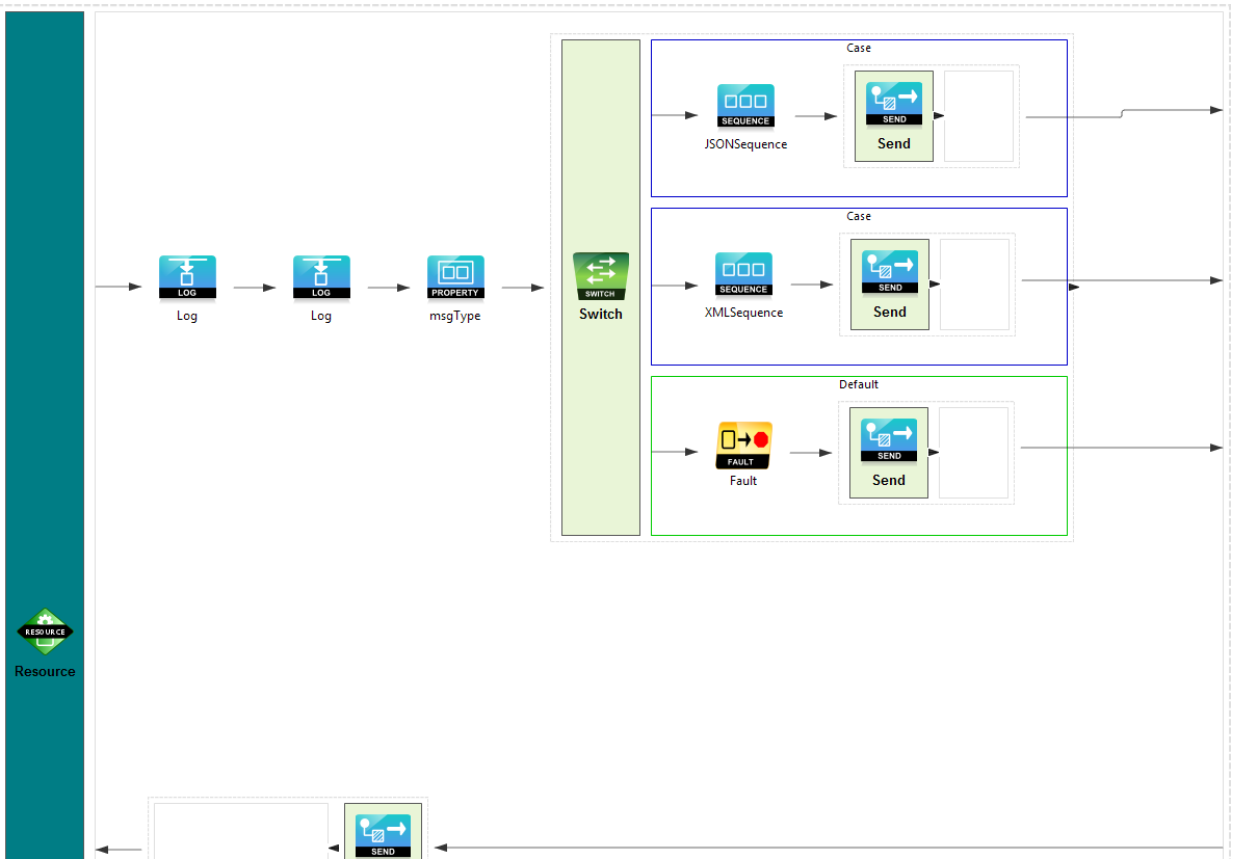

Figure 4: MOEEBUS API resource sequence in Eclipse Developer studio

Figure 5 drills down the JSONSequnce sequence which is part of one branch of the switch mediator in Figure 4. In this sequence the message is mapped from JSON to XML format accepted by the WSO2 DSS and then sent to the WSO2 DSS endpoint.

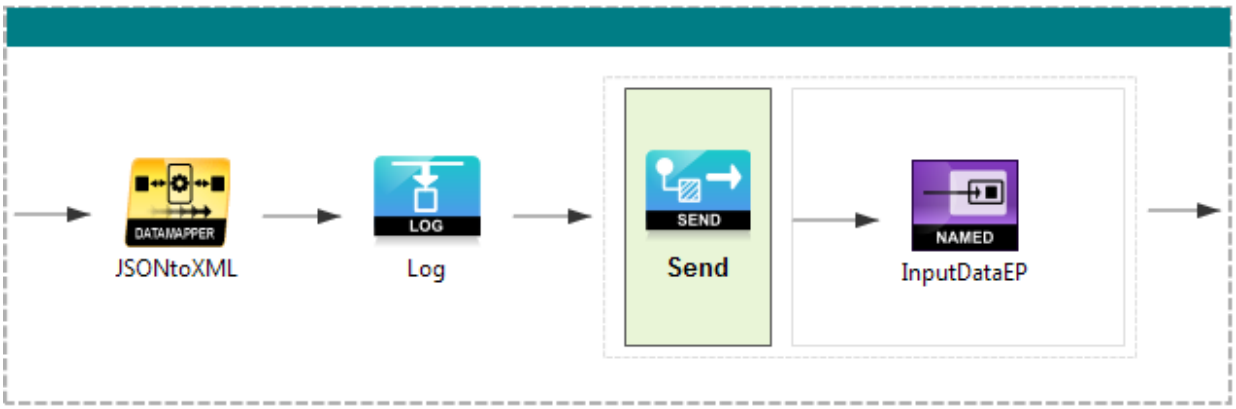

Figure 5: Detailed view of the JSONSequence used in InputDataAPI

The following figure shows detailed view at the JSONtoXML datamapper, which allows easy mapping from one format of the message to another. There is possibility of using arithmetic operations or parsing from string to number or vice versa.

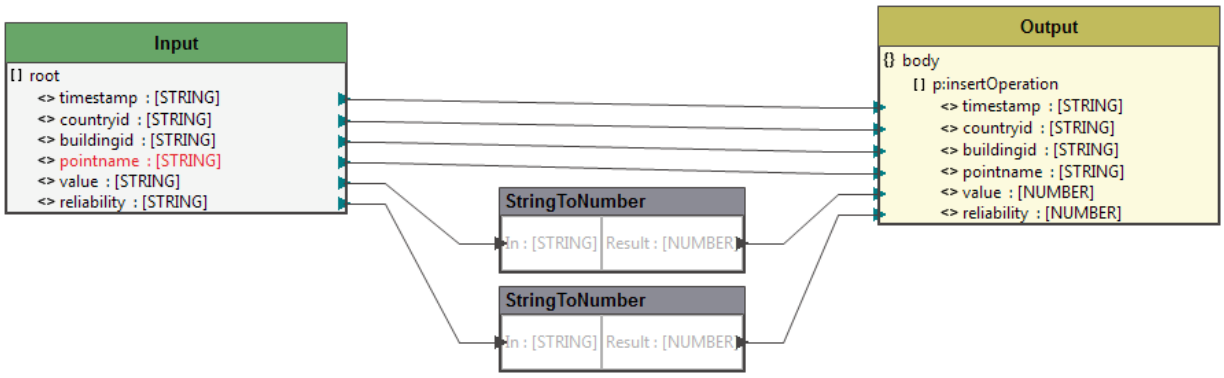

Figure 6: Detailed view of JSONtoXML datamapper used in JSONSequence

There are multiple publicly accessible similar REST APIs for use with data in the MOEEBIUS middleware. These APIs are consumed by sending an HTTP request (GET, POST) to the http address of the EC2 instance hosting the WSO2 ESB with the URL of the API. It is possible to send new data to store it in the Apache Cassandra DB (described in the following Section) or to query for data already stored in the same DB. Some constraints for querying can be specified in the request depending on the API. The following table lists the defined APIs.

Request type	Endpoint	Description
POST	http://35.156.215.81:8280/data	Stores data in the Cassandra DB.
GET	http://35.156.215.81:8280/getdata	Returns all data points.

Table 1: List of available ESB APIs.

Endpoint for inserting new data points located on http://35.156.215.81:8280/data accepts POST requests with body in JSON or XML format. Example of the JSON containing 3 new data points follows.

```
[{
        "pointname" : "pointname1",
        "countryid" : "countryid1",
        "buildingid" : "buildingid1",
        "timestamp" : "2017-03-09T13:20:00",
        "reliability" : "1",
        "value" : "1.77"
},
{
        "pointname" : "pointname2",
        "countryid" : "countryid1",
        "buildingid" : " buildingid1",
        "timestamp" : "2017-03-09T13:20:00",
```

```
        "reliability" : "1",
        "value" : "2.88"
},
{
        "pointname" : "pointname3",
        "countryid" : "countryid1",
        "buildingid" : "buildingid1",
        "timestamp" : "2017-03-09T13:21:00",
        "reliability" : "1",
        "value" : "3.99"
}]
```


The figure below describes the ESB's role in the MOEEBIUS data marshalling process. The ESB component is in charge of routing the requests (and responses) done by each of the MOEEBIUS modules. This mechanism makes the access to MOEEBIUS modules independent of their location. The data flow illustrates the simplified data flow for occupancy profiling process and the conceptual items that are part of the involved MOEEBIUS modules.

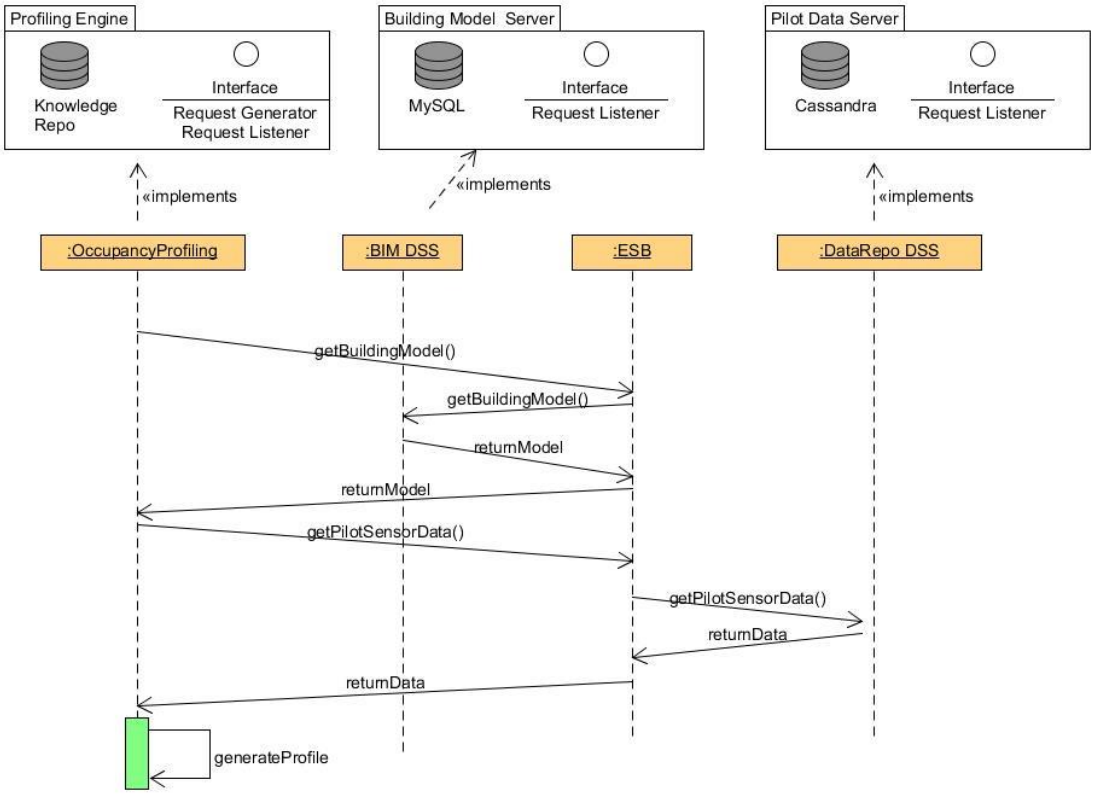

Figure 7: Role of WSO2 ESB in MOEEBIUS data marshalling process

3.7.4 MOEEBIUS Data Services Server Implementation

WSO2 Data Services Server (WSO2 DSS) augments service-oriented architecture development efforts by providing an easy-to-use platform for integrating data stores, creating composite data views, and hosting data services. It supports secure and managed data access across federated data stores, data service transactions, and data transformation and validation using a lightweight, developer friendly, agile development approach. It provides federation support, combining data from multiple sources in single response or resource and also supports nested queries across data sources. The internal architecture of the server is illustrated in Figure 8.

Data Services provide a convenient mechanism to configure a Web service interface for data in various data sources such as relational databases, CSV files, Microsoft Excel sheets, Google spreadsheets etc. These data services provide unprecedented data access and straightforward integration with business processes, mashups, gadgets, business intelligence and mobile applications.

We use the WSO2 DSS version 3.5.1 as a part of the middleware in MOEEBIUS to expose the Cassandra database as a REST API.

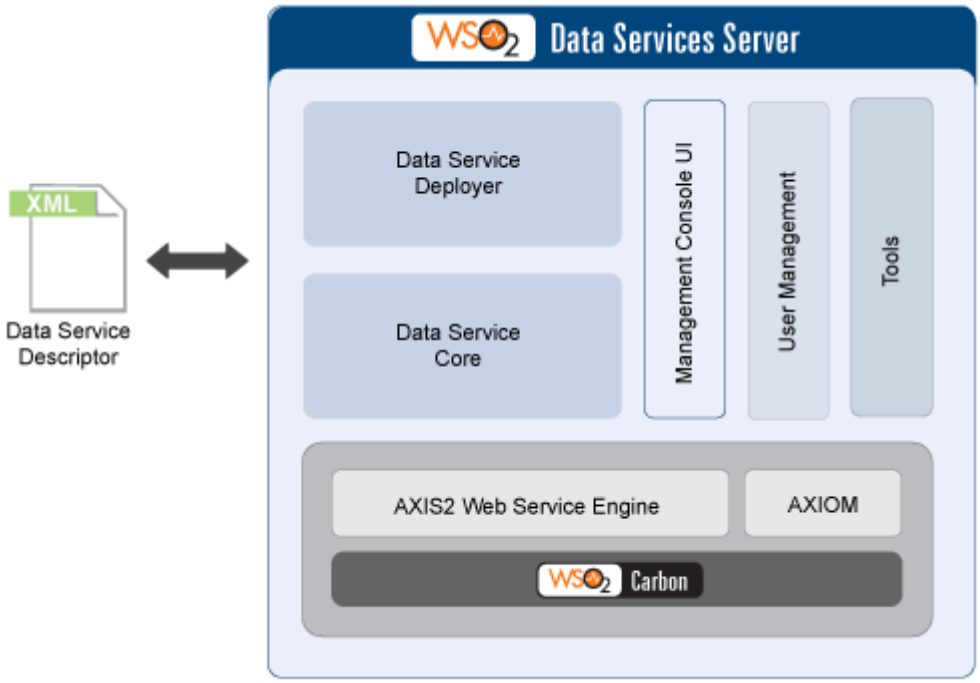

Figure 8: Overview of the WSO2 Data Services Server architecture

We created a Data Service named CassandraService which connects to the Cassandra DB and serves as a gateway to access the database. Datasource was set up and configured with the details of the Cassandra DB like the address, user name, password, cluster name and keyspace. There is a number of queries which correspond to the ESB APIs. At the moment, there is one query for inserting data

and multiple queries for data retrieval with different number and types of input parameters. Each query is linked to its resource. The resource serves as an API, where resource paths denote paths of individual APIs. Resource also passes query parameters found in the URL to the queries.

In the following table are listed endpoints currently available on the DSS instance. These endpoints are called mostly internally by ESB to access the Cassandra DB.

Request type	Endpoint	Description
POST	http://35.156.22.237:9763/services/CassandraService/data	Stores data in the Cassandra DB.
GET	http://35.156.22.237:9763/services/CassandraService/data	Returns all data points.
GET	http://35.156.22.237:9763/services/CassandraService/pointndata?pointname='pointname'&limit=20	Returns up to a specified number (limit) of data points with specified pointname.
GET	http://35.156.22.237:9763/services/CassandraService/pointdaterange?pointname='pointname'&startdate='2017-02-14T08:00:00+00:00'&enddate='2017-02-16T20:00:00+00:00'	Returns all data points with specified pointname between start date and end date.

Table 2: List of available DSS APIs.

It is recommended to use APIs located on WSO2 ESB instance described above to insert data. The WSO2 ESB API is capable of parsing JSON and XML formatted data and then marshalling the request to WSO2 DSS instance in correct format. However it is possible to send POST request on WSO2 DSS instance directly in following XML format:

```
<body xmlns:p="http://ws.wso2.org/dataservice">
  <p:insertOperation>
    <timestamp>YYYY-MM-DDTHH:MM:SS</timestamp>
    <countryid>countyrid</countryid>
    <buildingid>buildingid</buildingid>
    <pointname>pointname</pointname>
    <value>value</value>
    <reliability>reliability</reliability>
  </p:insertOperation>
</body>
```

GET requests return list of entries satisfying constraints given by the query parameters. Returned timestamps are in unix time format. Example response to a request follows.

Request:

http://35.156.22.237:9763/services/CassandraService/pointdaterange?pointname='SI01.BL00.CL01.SM02.EN00'&startdate='2017-04-10T12:00:00'&enddate='2017-04-11T00:00:00'

Response:

```
<entries xmlns="http://ws.wso2.org/dataservice/PointnameTimestampRangeQuery">
   <entry>
      <pointname>SI01.BL00.CL01.SM02.EN00</pointname>
      <reliability>1.0</reliability>
      <value>328257.0</value>
      <countryid>ES</countryid>
      <timestamp>1491825808000</timestamp>
      <buildingid>BL00</buildingid>
   </entry>
   … 142 records …
   <entry>
      <pointname>SI01.BL00.CL01.SM02.EN00</pointname>
      <reliability>1.0</reliability>
      <value>328258.0</value>
      <countryid>ES</countryid>
      <timestamp>1491868726000</timestamp>
      <buildingid>BL00</buildingid>
   </entry>
</entries>
```

3.7.5 MOEEBIUS Data Analytics Server

WSO2 Data Analytics Server (WSO2 DAS) listens to a constant stream of events that represents the transactions and activities of an enterprise from different sources, processes them in real time and communicates the results in a variety of interfaces. This allows organisations to quickly respond to their environments, thus gaining an advantage over their competitors. In addition, WSO2 DAS combines real-time analytics with batch analytics (equipped with incremental processing), interactive and predictive (via machine learning) analysis of data into one integrated platform to support the multiple demands of Internet of Things (IoT) solutions, as well as mobile and Web apps.

Used version of WSO2 DAS is 3.1.0. We have created a simple flow consisting of an Event Receiver, an Event Stream and an Event Publisher. This flow allows us to persist data coming from some specific sensors to a specialised DAS table. Custom scripts then can be set up to run against this table at specific time intervals given by a cron expression[1]. We also created a simple script which implements a moving averaging of the data for illustration purposes.

[1] https://en.wikipedia.org/wiki/Cron

3.7.6 MOEEBIUS Cassandra Implementation

The central MOEEBIUS data-store is an Apache Cassandra Database version 3.9.1, which is a highly-scalable partitioned row store, where rows are organized into tables with a required primary key. Partitioning means that Cassandra can distribute the data across multiple machines in an application-transparent matter. Cassandra will automatically repartition as machines are added and removed from the cluster. Row store means that like relational databases, Cassandra organizes data by rows and columns. The available Cassandra Query Language (CQL) for querying the DB is a close relative of SQL.

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. Cassandra aims to run on top of an infrastructure of hundreds of nodes (possibly spread across different data centres). At this scale, small and large components fail continuously. The way Cassandra manages the persistent state in the face of these failures drives the reliability and scalability of the software systems relying on this service. While in many ways Cassandra resembles a database and shares many design and implementation strategies therewith, Cassandra does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format. Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency. Figure 9 shows a simple Cassandra cluster.

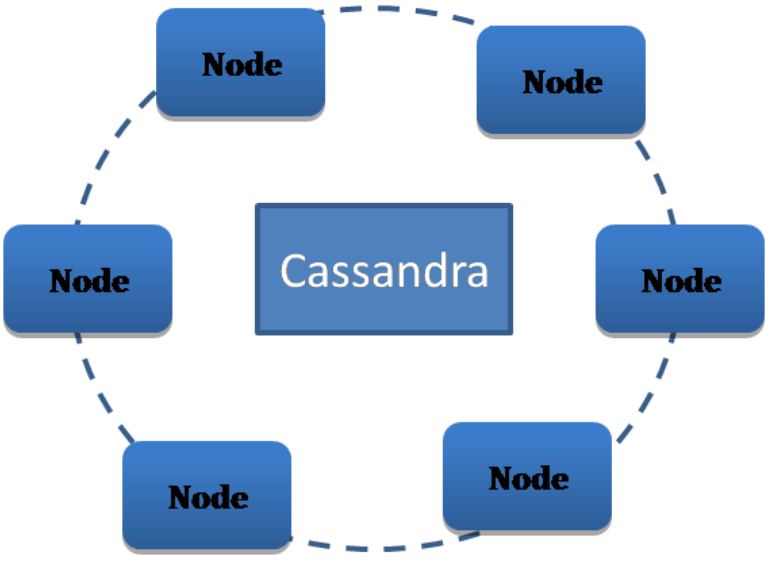

Figure 9: Simple illustration of a Cassandra cluster.

For the purposes of the MOEEBIUS project we created a Cassandra cluster running on the AWS. We started with Amazon Machine Image (AMI) by Bitnami with preinstalled basic Cassandra setup. It was necessary to edit the "Cassandra.yaml"

configuration files to prepare the individual Cassandra instances to form a cluster. This step required the configuration of addresses for the individual instances and specifying the cluster name. The Cassandra cluster which stores data recorded by sensors from various partners consists of 3 nodes. Each node is installed on a different EC2 instance and all of the nodes are connected. The resulting cluster maintains some redundancy between individual nodes, which means that when one node fails, the cluster will still contain a copy of the data on another node and to the outside user the operation of the cluster will not change.

Additional information about the Cassandra cluster can be found in the following table.

Node addresses	35.156.22.237, 35.156.215.81, 35.156.149.159
Used port	9042
Cluster name	MOEEBIUS Cluster
Used keyspace	Moeebiusdata
Tables	Bmsrawdata, Lastprocessedtimestamp
Replication factor	3

Table 3: Detailed information about the Cassandra cluster.

Table "bmsrawdata" is designed to be universally used for storing data from various sources. Its structure and data types are detailed in the following table.

Column	Type	Key
pointname	Text	Partition key
countryid	Text	Clustering key, ASC ordering
buildingid	Text	Clustering key, ASC ordering
timestamp	Timestamp	Clustering key, ASC ordering
value	Double	
reliability	Double	

Table 4: Structure of "bmsrawdata" table.

Table "lastprocessedtimestamp" is a helper table which stores latest timestamps of data points which were processed by some kind of analytics. The analytics can then later resume calculation where it ended after accumulating more data since the latest run. The analytics has to keep track of the records in this table itself, it is not automated in any way. The structure of "lastprocessedtimestamp" is described in the following table.

Column	Type	Key
pointname	Text	Partition key
countryid	Text	Clustering key, ASC ordering
buildingid	Text	Clustering key, ASC ordering
timestamp	Timestamp	

Table 5: Structure of "lastprocessedtimestamp" table.

3.8 MOEEBIUS Deployment

We have deployed all the WSO2 components in the Amazon AWS cloud. The following figure depicts architecture and relationships of these individual components with the rest of MOEEBIUS components, and relevant role that ESB plays

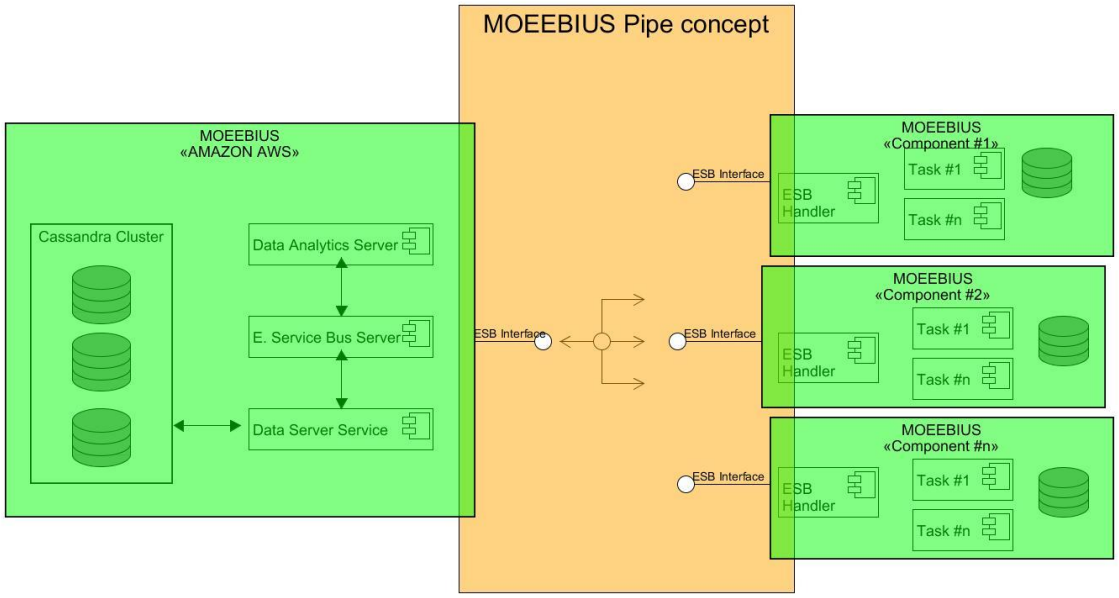

Figure 10: MOEEBIUS Architecture of the middleware with WSO2 ESB bridge to DSS and DAS services implemented in MOEEBIUS

3.9 Hardware

The middleware is physically located on the cloud computing platform Amazon Web Services (AWS) provided by Amazon.com. Part of the services offered in AWS is the Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Honeywell is currently administering three EC2 instances for the MOEEBIUS project.

The two instance types (virtual machines) used for the deployment of the middleware are the following:

- t2.micro:
  - 1 vCPU, 1GiB RAM;
  - 2.3 GHz Intel Xeon® E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors;
- m3.medium:
  - 1 vCPU, 3.75GiB RAM;
  - High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge) Processors or Intel Xeon E5-2670 (Sandy Bridge) Processor running at 2.6 GHz;
  - SSD-based instance storage for fast I/O performance.

We started with all instances as t2.micro and later scaled up to m3.medium in the cases of WSO2 ESB and WSO2 DSS. The scaling up was needed because the instances weren't stable enough under the given computational load. However, the process of scaling up on AWS proved to be simple and fast and during the upgrade there were no performance issues. In the case of the WSO2 DAS instance, we stayed on the t2.micro because there is no load on it now, but as stated it can be easily upgraded in the future.

4 Selected MOEEBIUS-related use cases

This section aims to illustrate relevant MOEEBIUS use case, especially with respect to the middleware layer. It does not aim to describe all considered use cases.

4.1 Occupancy Profiling

The occupancy profiling module target is to forecast the building spaces occupation ratio. For that purpose, the Occupancy Profiling engine requires information about the deployed building sensors topology as well as their values. This means the intervention of 3 different modules

- BIM server: Deployed over WSO2 DSS stores building topology repository and enables read/write operations.
- Pilot data server: Deployed over Cassandra framework stores the values gathered from pilot sites. The values stored are directly linked to the ones defined in the BIM server
- Occupancy Profiling Engine: Supported by the previously mentioned modules, calculates, stores and serves the occupation ratio forecasts.

The picture below describes how the WSO2 ESB enables the data exchange among three different MOEEBIUS modules wrapping their respective WS interfaces in one single data exchange layer.

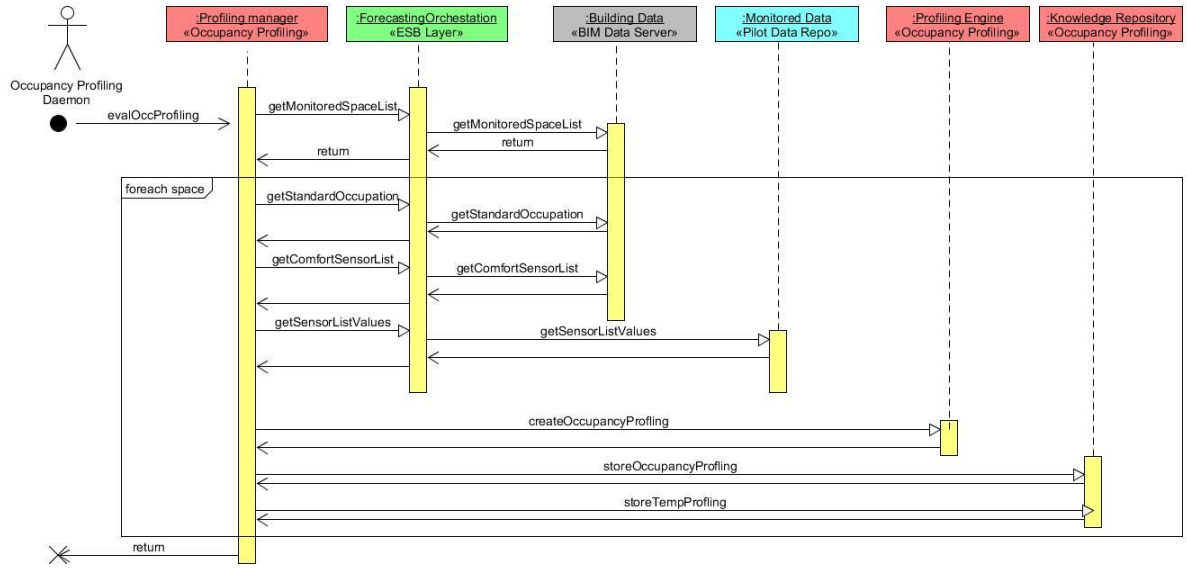

Figure 11: Example of WSO2 ESB usage with MOEEBIUS modules

The role of the WS02 framework in the current use case is mainly to ensure the interoperability among all the MOEEBIUS components involved in it. It could be possible to split this interoperability in two layers.

- Transport layer: The WSO2 framework, and in this case more precisely the ESB, routes the HTTP calls fired by the "Occupancy Profiling" to the appropriate target, in other words, it makes the services location independent.
- Application layer: The WSO2 will ensure that the data structures exchanged among the different modules in the use case are compatible in terms of data format. Additionally, the application layer will handle the data exchange error management.

Note: Even if it does not fit strictly to the definition of Transport and Application layer used in IT domain, we adopted this definition for descriptive purposes.

4.2 Building Dynamic Assessment

The building dynamic assessment (herein after BDA) is probably the most complex use case applicable at building level. The BDA involves MOEEBIUS modules involved in the building daily operation as well as modules related to the MOEEBIUS decision support tool implementation.

The MOEEBIUS modules involved in the current use case are:

- Building Performance Simulation Engine: Taking into consideration the BIM calculates the building level energy demand/consumption forecast.
- Dynamic Assessment Engine: DAE's aim is to fine tune the energy demand/consumption forecasts done by the Building Performance Simulation Engine. DAE's activity can be split in two sub tasks.
    - Calibration: Executed offline, the calibration tunes building model parameters (isolation level, air infiltration ratio, etc.) in order to calculate correction factors related to them.
    - Assessment: Once the model is calibrated, then during the forecasting phase the "Assessment" task applies the appropriate correction factor.
- BIM server: Deployed over WSO2 DSS stores building topology repository and enables read/write operations.
- Pilot data server: Deployed over Cassandra framework stores the values gathered from pilot sites. The values stored are directly linked to the ones defined in the BIM server
- Occupancy Profiling Engine: Supported by the previously mentioned modules, calculates, stores and serves the occupation ratio forecasts.
- Predictive Maintenance Engine: The predictive maintenance aims to predict scenarios in which the building systems will underperform and trigger preventive operations. Talking in terms of optimization, the predictive maintenance engine will have to minimize the HVAC systems non activity periods related to failures and reparation activities.

The picture below describes how the WSO2 ESB enables the data exchange among different MOEEBIUS modules wrapping their respective WS interfaces in one single data exchange layer. Additionally, the ESB layer triggers WSO2 framework functionality, the Data Analytics feature. As mentioned the DAS (Data Analytics Server) component may execute different regression or heuristic methods to predict future scenarios.

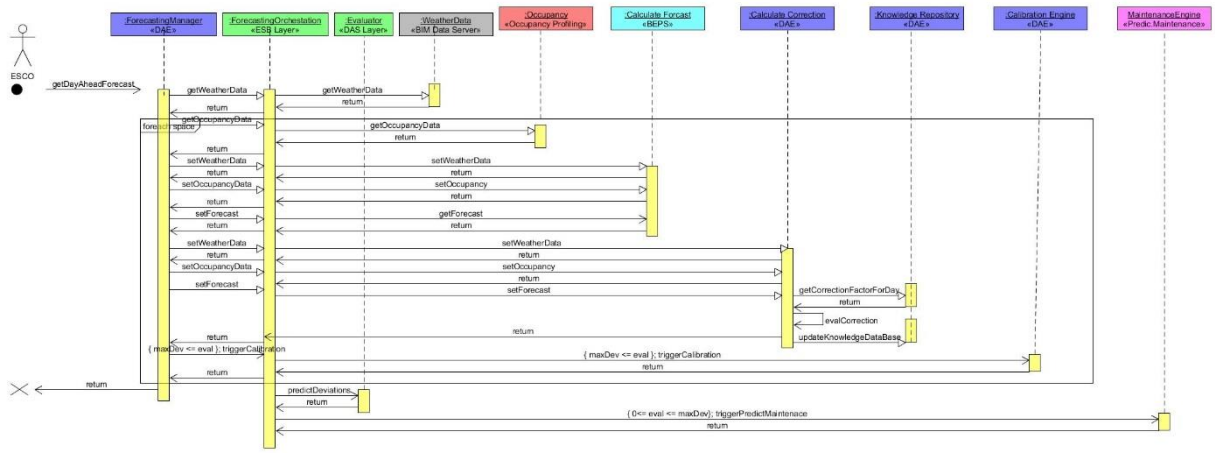

Figure 12: Example of WSO2 ESB usage with MOEEBIUS modules, including WSO2 DAS

The role of the WS02 framework in the current use case is not only used to ensure the interoperability among all the MOEEBIUS components involved in it and labeled as transport and application layers. Additionally, the ESB layer triggers a process in the data analytics layer to evaluate if a predictive action is required. This behavior can be labeled as "third party triggering".

The use case figure describes the process in which a ESCO or building planner request the "day ahead forecast" and in parallel a "business intelligence" process, that is impendent to the main request, evaluates the operation scenario and triggers parallel actions if necessary.

4.3 Retrofitting Advisor Service

The RAS server main goal is to predict buildings' future energy consumption after some retrofitting actions.

The retrofitting advisor use case is similar to the demand aggregation and flexibility calculation use case but in the data analytic methodologies to apply. The DAaF module targets a combinational problem, this is searching among a set of options the best alternative, on the other hand the Retrofitting Advisor Service (RAS) target a forecasting problem. In others words, given a certain initial conditions predict the future behaviour.

The figure below summarizes the retrofitting advisor activity diagram.

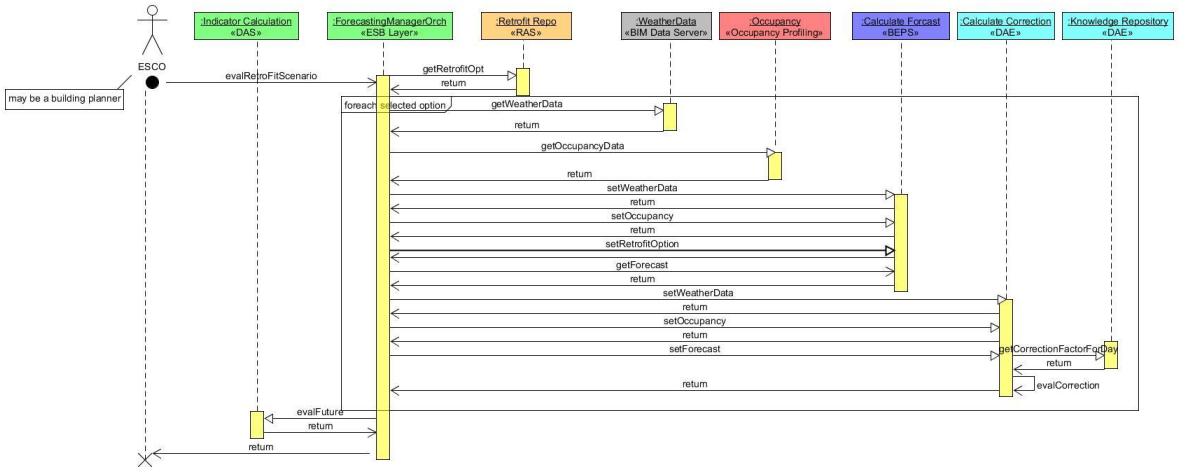

Figure 13: Retrofitting Advisor activity diagram

The MOEEBIUS modules involved in the current use case have been already described, in this context, we will focus on describing additional WSO2 features taking as reference the RAS use case:

- Forecasting Manager Orchestration: Already described in the D4.4 WS02 ESB module can provide abstract services from their location, this is, the ESB may act as service router. Additionally, the ESB layer may implement a orchestration that fetches all the input data and finally calls the required functionality.
  In the picture above, both steps are clearly identified. The data fetching process is grouped in a "foreach" loop. The evaluation function is finally invoked by the "evalFuture" call.

- DAS predictive features: Represented by the "Indicator Calculator" module in the figure above, the DAS will implement predictive functions.

The predictive methods in the data analytics domain are covered by a wide spectrum of mathematical approaches. Predictive applications may be implemented using simple regression methods or more complex state of the art neural networks.

The predictive methods, whatever analytic approach they follow, require a set of reference values. Usually the reference values come from historical data, the MOEEBIUS project will consider the scenario in which there is not available any reference value (historical data).

The approach for the scenarios in which there is not available any historical data will rely in EnergyPlus simulation for periods of years. The background for this assumption is that, once the building models' parameters have been fine-tuned by

the DAE, the EnergyPlus simulation will be accurate enough to support multi-year timeframe predictions.

5 Conclusion

After evaluation of the alternatives, the WSO2 platform was selected as a framework for the middleware, Amazon Web Services was chosen for hosting the middleware platform and Cassandra was selected as the data storage. We have registered MOEEBIUS Middleware AWS account and instantiated three EC2 instances for running required WSO2 components and Cassandra database cluster. Cassandra cluster was set up by interconnecting Cassandra databases on all three instances and tables for sensor data were created with parameters based on discussion with partners. Individual WSO2 components were deployed on the EC2 instances. Inbound and outbound ports were configured to allow communication with the middleware from outside of the cloud.

WSO2 Data Services Server (DSS) was configured to be able to access the Cassandra data store. For this purpose, a data service CassandraService was created and configured with Cassandra cluster details. Endpoints for reception of requests for data storage and retrieval were created. Currently there are endpoints which return either all data points or endpoints which expect query parameters constraining the scope of the data to be returned.

Several general APIs were created in WSO2 Enterprise Service Bus (ESB) to accommodate incoming requests for data input or data querying. These APIs contain sequences with optional conditional logic which can route or transform received messages based on their content. This allows us to truly separate the concerns of data acquisition, processing and analytics since we have the ability to dynamically change the shape and the destination of the messages according to our needs. Addition of new APIs, sequences and proxies is a live process. We can seamlessly implement new interface elements tailored to the needs of the new MOEEBIUS components.

WSO2 Data Analytics Server (DAS) was prepared for use by creating basic event receiver, event stream and event publisher. This event flow is needed to enable real-time streaming analytics. Data for streaming analytics can be routed to WSO2 DAS by WSO2 ESB according to some conditional logic. This also allows us to perform batch analytics, e.g. aggregation, on data stored in temporary tables. Results of this aggregation can be used to read total energy consumption for given building, whole district, etc.

Some partners already connected their services to the middleware and are sending data which are stored in the Cassandra data store. By the time of writing there is over 500,000 records in the database.

Future steps for the middleware development are linked with the development of other MOEEBIUS components. We will be cooperating with other partners

developing the MOEEBIUS components to create tailored interfaces for these components to connect to the middleware and orchestrate the subsequent calls with minimal effort. This documentation will evolve as new MOEEBIUS components will be connected to the middleware and new APIs will be deployed to the WSO2 ESB.

References

[1]. MOEEBIUS Description of Actions, MOEEBIUS Project (680517)
[2]. MOEEBIUS D2.1 End-user & business requirements, MOEEBIUS Project
[3]. MOEEBIUS D2.2 New Business Models and Associated Energy Management Strategies, MOEEBIUS Project
[4]. MOEEBIUS D2.3 MOEEBIUS Energy Performance Assessment Methodology, MOEEBIUS Project
[5]. MOEEBIUS D2.4 Functional and Non-functional requirements of the MOEEBIUS framework and individual components
[6]. MOEEBIUS D3.1 Framework Architecture including functional, technical and communication specifications
[7]. OPENSOURCE, MICROSERVICES [Online]. Available: https://opensource.com/resources/what-are-microservices
[8]. Fowler, A., MICROSERVICE [Online]. Available: https://martinfowler.com/articles/microservices.html
[9]. WSO2 [Online]. Available: http://wso2.com/
[10]. Martin, S., Hernandez, J., & Valmaseda, C. (2015, March). A novel middleware for smart grid data exchange towards the energy efficiency in buildings. In Networked Systems (NetSys), 2015 International Conference and Workshops on (pp. 1-8). IEEE.
[11]. Jahn, M., Eisenhauer, M., Serban, R., Salden, A., & Stam, A. (2012, July). Towards a context control model for simulation and optimization of energy performance in buildings. In 3rd Workshop on eeBuildings Data Models, Proc. of ECPPM conference, Reykjavik, Iceland.
[12]. Le Guilly, T., Skou, A., Olsen, P., Madsen, P. P., Albano, M., Ferreira, L. L.,... & Gangolells, M. (2016, September). ENCOURAGEing results on ICT for energy efficient buildings. In Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on (pp. 1-8). IEEE.
[13]. Hohpe, G., & Woolf, B. (2004). Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional.Hohpe, G., & Woolf, B. (2004). Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional.