D5.21

Front End Stargate Implemented – First version

Document Owners	Cinzia Rubattino [PIKS]
Contributors	Gianluigi Di Vito [PIKS], Klaus Fischer [DFKI], Liudmila Dobriakova [RPLY], Artur Feilic[CAS]
Dissemination	Public
Date	30/09/2014
Version	Final

Table of Contents

DOCUMENT HISTORY			
Version	Version date	Responsible	Description

Executive Summary

THIS IS A DOCUMENT ACCOMPANYING THE TECHNICAL PROTOTYPE

As stated in the Description of Work (DOW), this deliverable is a prototype deliverable.

As such, this document's only purpose is to briefly describe the prototype functionality as well as to provide instructions on how to use it. This document is an accompanying document to the corresponding software.

1 Introduction

1.1 Objective of the Deliverable

This document accompanies the technical prototype of the Front-End Stargate released at M12. It provides a report about activities held in Task 5.2 of the OSMOSE project; the objective is to describe the status of the development of the prototypical implementation of the OSMOSE Stargate.

1.2 Structure of the Deliverable

In its details the document provides an overview of the Stargate components about the following items:

- General information about the architectural component
- The components internal architecture and major functionalities
- Technical information about software used in the component (libraries, databases, etc.)
- Licensing information regarding mainly third party software incorporated or used in the software
- Technical manual in order to allow technicians to download, install, configure and run the system
- User manual supporting humans in accessing and using GUI based prototypes
- Conclusions and future plan.

This document is intended to be read by both technicians and end-users.

1.3 Applicable Documents

- OSMOSE Description of Work(DOW) providing the basis for the entire project and this deliverable content
- Deliverable D2.11 “First User Scenarios & Business Expectations” and D2.21 “First User Requirement & PoC Specification” providing test cases description, end user requirements and proof of concepts specification
- Deliverable D3.11 “First OSMOSE Models and Architecture” describing the high-level architecture and first work on technology scouting
- Deliverable D5.11 “Front End Stargate Specification and Design – First version” (in parallel to this deliverable) specifying the functionalities and design of the Front End Stargate.

2 Stargate Architecture and Components

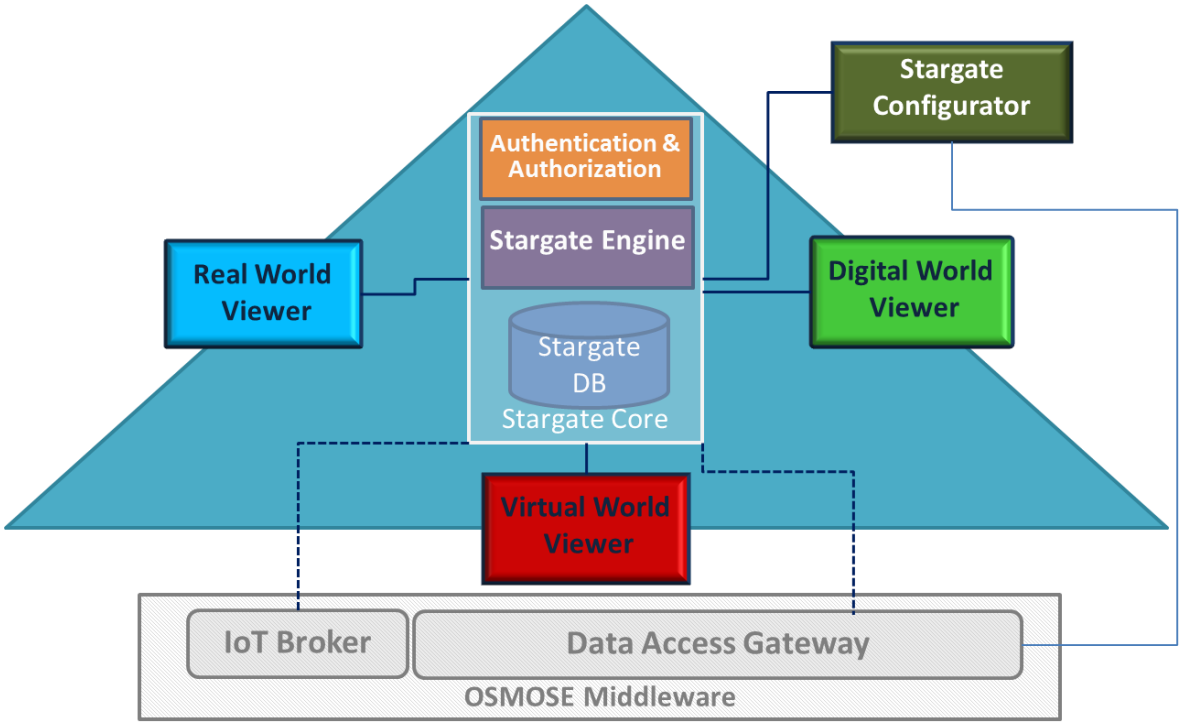

Figure 1 Stargate overall architecture

The architecture of the Front End Stargate (as defined in D5.11 simply Stargate) is depicted in Figure 1. It consists of three viewers: Real World Viewer, Digital World Viewer and Virtual World Viewer. The consistency between the three viewers is kept by the Stargate Core component in the centre of the triangle. The three worlds' viewers provide different views of the real-digital and virtual assets for different contexts on different devices. The following sections describe the main components of the Stargate.

2.1 Real World Viewer

2.1.1 Overall Data

Component Name	**Real World Viewer**
Reference WP – Task	WP5 – Task 5.2
Responsible	PIKS
Version	1.0
Source Control	n/a
Contact Person	Cinzia Rubattino, Gianluigi Di Vito
Short Description	A viewer supporting the human user when working in the real environment complementing the real world with digital and virtual contents

2.1.2 Architecture and Functionalities

2.1.2.1 Introduction

The Real World Viewer (briefly **RW-Viewer**) is specifically designed to support the human user when working in the real environment complementing the real world with digital and virtual contents. Taking advantages of Augmented Reality technologies the system recognises the objects in the Real World and superimposes digital contents upon them.

2.1.2.2 Summary of Functionalities

The RW-Viewer enhances the user's perception of the real world by displaying digital contents that the user cannot directly detect on his own. The information conveyed by the digital contents helps the user perform real-world tasks.

The RW-Viewer is an Augmented Reality system which allows annotating objects and environments with data coming from the Osmose Middleware.

2.1.2.3 Component Architecture

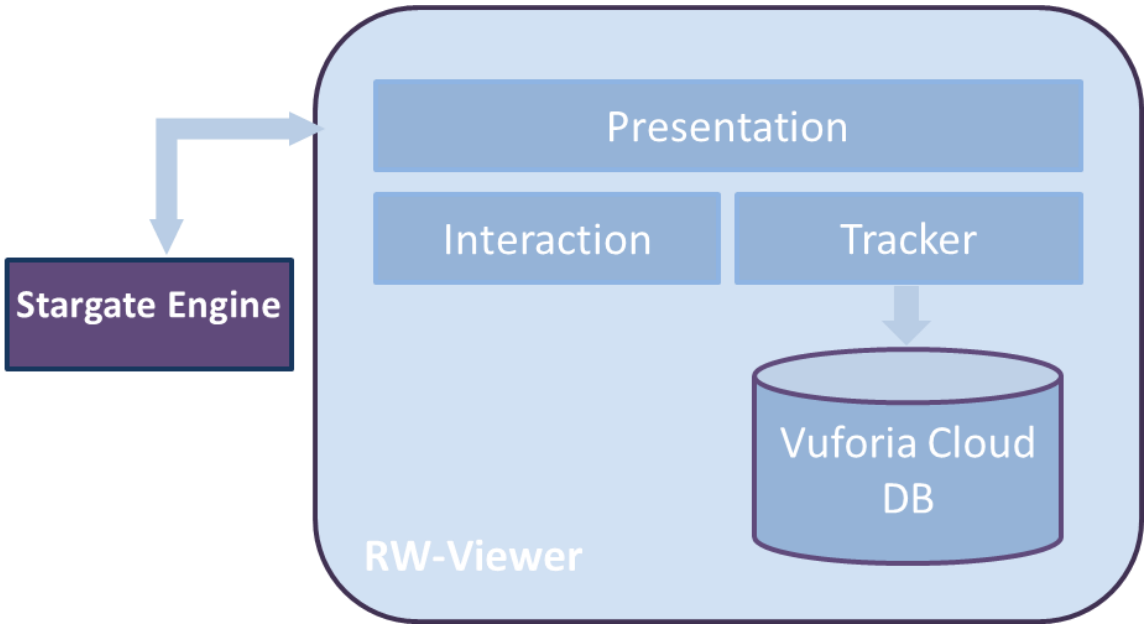

Figure 2 RW-Viewer architecture

The architecture of the RW-Viewer is depicted in Figure 2.

The component is developed using Vuforia AR Extension for Unity. To recognise the objects of the real environment the system uses markers in particular QR code and Data Matrix, and this is done by integrating ZXing library. ZXing ("zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java.

The tracker sub-component detects and tracks real-world objects. The markers that have to be recognised are stored in the Vuforia Cloud DB.

The Interaction sub.component gathers and processes any input that the user makes deliberately.

The Presentation sub-component displays system output to the user. The core of this component is the part that draws the objects that augment the user's perception in the correct distance, position and orientation based on the user's pose.

The Augmentation library retrieves the data to display from the Stargate Engine.

2.1.2.4 Technical Information

Name	AROsmose.apk
Nature	Vision-based augmented reality application
Programming Language	C#, Javascript
Development Tools	Vuforia SDK v3.0,Unity SDK, Android SDK
Additional libraries	SimpleJSON, ZXing ("Zebra Crossing") project
Application Server	//
Databases	//

2.1.2.5 Licensing

Vuforia SDK: https://developer.vuforia.com/legal/vuforia-30-license-agreement

Unity SDK: http://unity3d.com/legal/eula

Android SDK – Open Source (Apache Software License, Version 2.0)

ZXing - Open Source (Apache Software License, Version 2.0)

SimpleJSON: http://wiki.unity3d.com/index.php/SimpleJSON

2.1.2.6 Technical Manual

Step 1 - Download App

Download the AROsmose.apk to your Android device.

Step 2 - Allow Installation

Navigate to your device's Settings, select Security or Applications (depending on device) and check the Unknown Sources box.

Step 3 - Install

Launch AROsmose.apk from Notifications or Downloads, tap Install when prompted.

2.1.2.7 User Manual

- Launch **AROsmose** app from your device. Your device's camera is activated.
- Line up the camera on your device with the QR code or Data Matrix you want to

scan, and hold the device steady until the app can read the code in front of it.

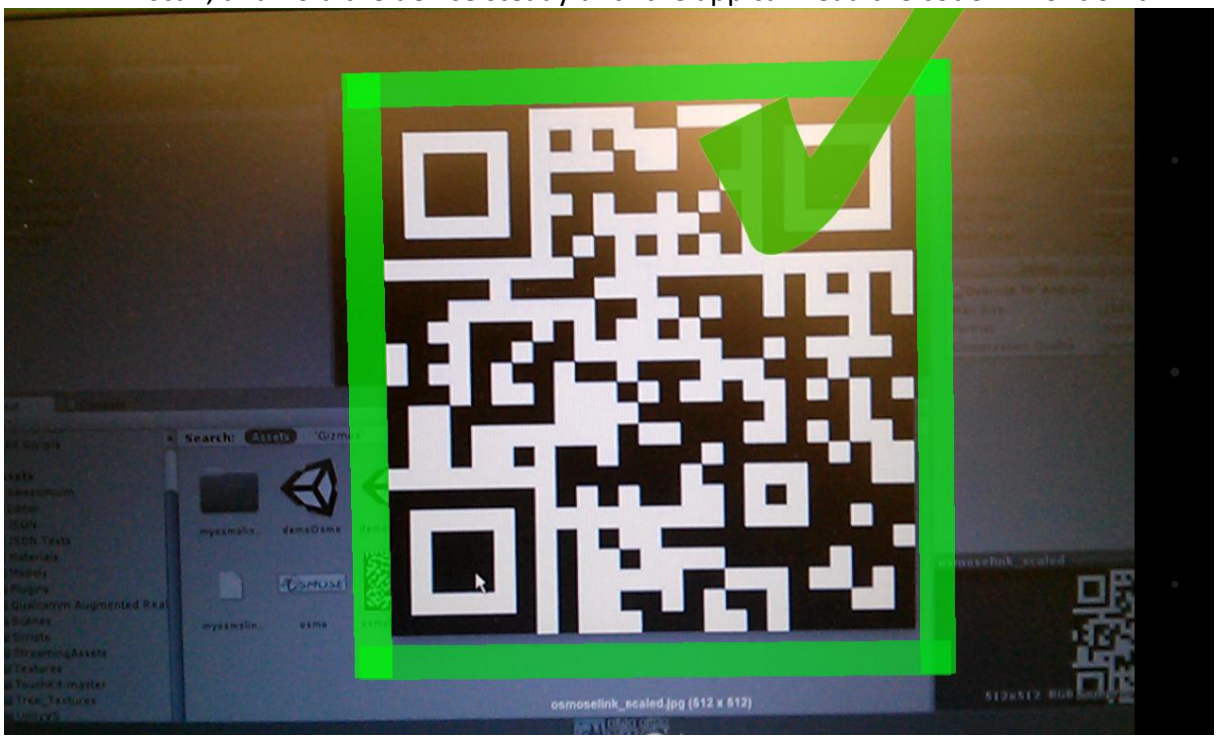

Figure 3 QR code recognition

- Once the object is recognised it is edged (see Figure 3). Tip inside the square to visualize information about the selected object.
- A summary about the asset selected is displayed on top-right on the object (Figure 4).
- Use the button below to view more information about the asset and to jump to DW-Viewer.
- Use the “Close” button to select another QR code or Data Matrix.

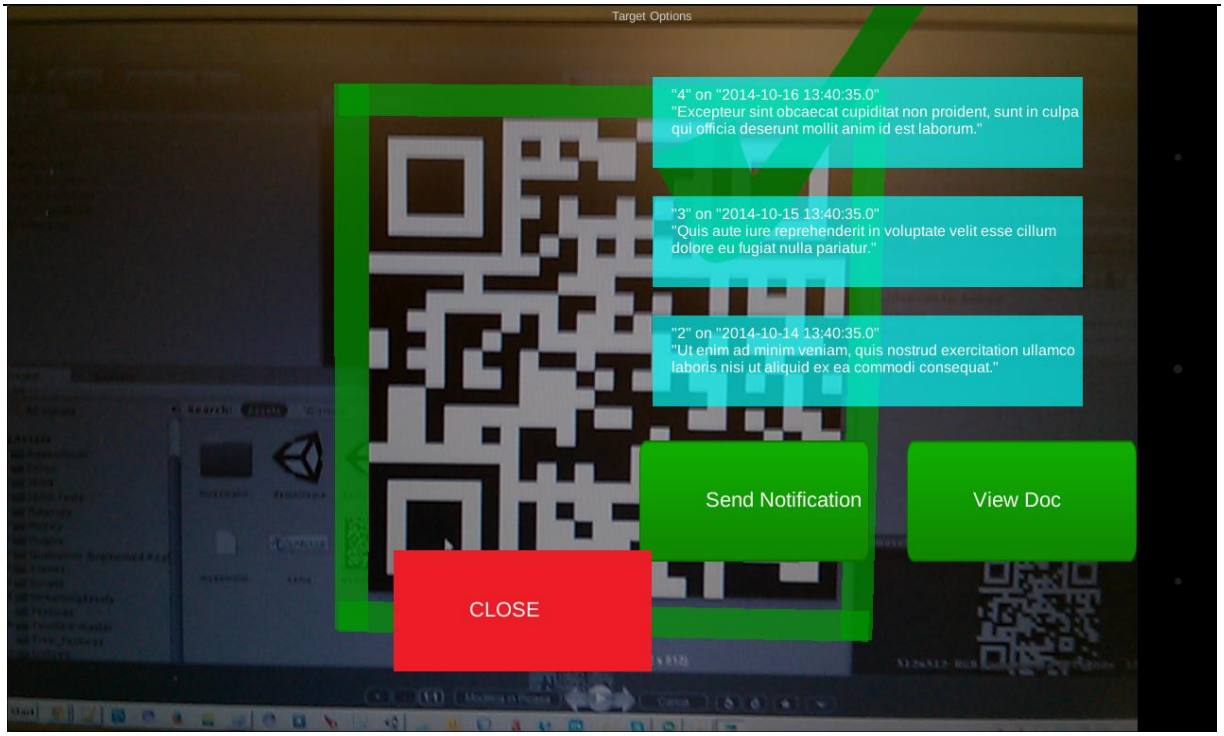

Figure 4 Digital information displayed on top of a QR code

2.1.2.8 Conclusions and Future Plans

In the deliverable 5.22 a more detailed user manual will be inserted.

The current version of RW-Viewer works on Android (4.x) device (smartphone and tablet) and on Windows PC. For the next development a support for other smart devices, such as smart glasses, is under evaluation.

2.2 Digital World Viewer

2.2.1 Overall Data

Component Name	**Digital World Viewer**
Reference WP – Task	WP5 – Task 5.2
Responsible	PIKS
Version	1.0
Source Control	n/a
Contact Person	Cinzia Rubattino, Gianluigi Di Vito
Short Description	A portal environment to assess the data in an intuitive and comprehensive way

2.2.2 Architecture and Functionalities

2.2.2.1 Introduction

The Digital World Viewer (briefly **DW-Viewer**) is a portal like environment which allows the users to browse and retrieve the relevant digital-real and virtual asset in a simple and intuitive way.

2.2.2.2 Summary of Functionalities

The major functionalities of the DW-Viewer are:

- Navigate and search the OSMOSE entities
- View content associated to an entity (document, media file, DB entry…)
- Navigate the history of an entity
- Change the view

2.2.2.3 Component Architecture

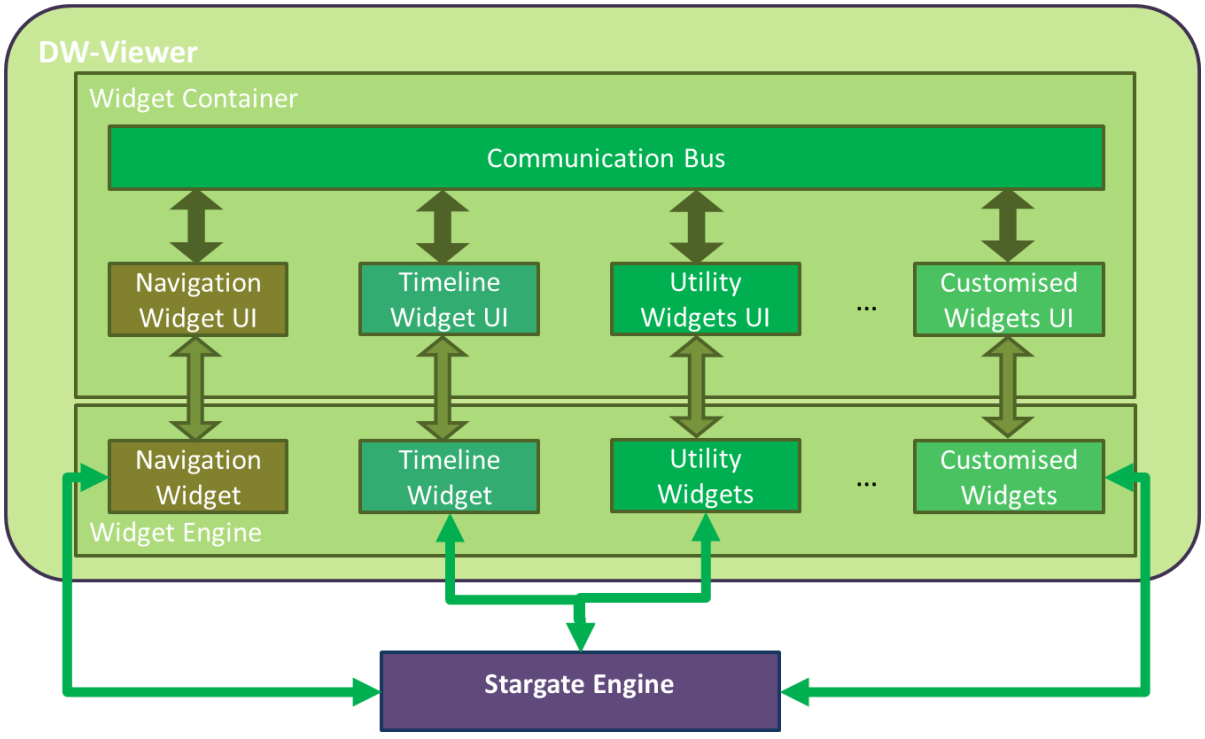

Figure 5 DW-Viewer architecture

The DW-Viewer is a portal environment which results from the mashing up of different widgets. Every user will be able to access his/her dashboard and set the preferences.

The Widget Container is based on Apache Rave. Apache Rave has been selected since it is built on open standard, it supports both Open Social and W3C widgets and it implements IWC thanks to the integration with Open Ajax Hub.

The Widget Engine exposes services useful to widgets for accessing remote services. Osmose Widgets take advantages of this feature to access additional server logic running in the Stargate Engine. The Stargate Engine supplies also the services to retrieve the data from the Stargate Database.

2.2.2.4 Technical Information

Name	Apache Rave
Nature	Web and social mashup engine
Programming Language	Java
Development Tools	Eclipse
Additional libraries	//
Application Server	Tomcat
Databases	//

Name	OpenSocial Widgets
Nature	Web based applications
Programming Language	HTML, Javascript
Development Tools	Notepad++
Additional libraries	CytoscapeJS, TimelineJS, AmaranJS, Animate.css, Font Awesome, Open Ajax Hub
Application Server	Tomcat, Apache Shinding
Databases	//

2.2.2.5 Licensing

Eclipse - Eclipse Public License (EPL) https://www.eclipse.org/legal/epl-v10.html .

Tomcat - Open Source (Apache License, Version 2.0)

Apache Rave - Open Source (Apache License, Version 2.0)

Apache Shinding - Open Source (Apache License, Version 2.0)

CytoscapeJS – LGPL: http://www.gnu.org/licenses/lgpl.html

TimelineJS - the Mozilla Public License, v. 2.0.

AmaranJS: https://github.com/hakanersu/AmaranJS/blob/master/LICENCE.txt

Animate.css - MIT license

2.2.2.6 Technical Manual

Install Apache Rave (https://rave.apache.org/documentation/installing.html)

Widget installation:

Open the admin interface [http://localhost:8080/portal/app/admin/]in a browser.

Click on Widget Store on the top bar

Click on Add new widget

Select OpenSocial tab

Insert in Location Url form the URL of the specific widget (ex. http://demos.polymedia.it/iks/semanticplayer/notifier/timeline.xml)

Click on Get widget metadata

Click on add widget

Repeat procedure for each Osmose widget

2.2.2.7 User Manual

Open Apache Rave in the browser [http://localhost:8080/portal/]

Select an asset from the assets list

Navigate through the available widgets to see information about the asset in the proper format.

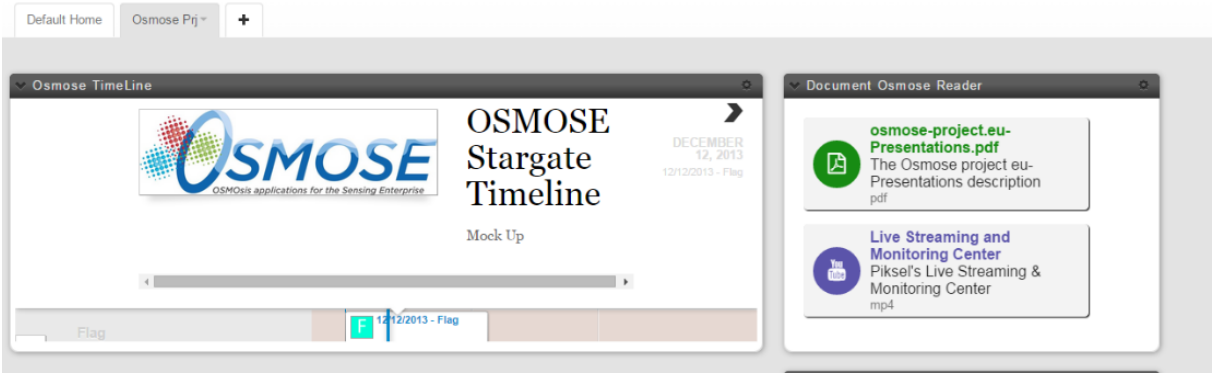

Figure 6 Home page of the DW-Viewer

2.2.2.8 Conclusions and Future Plans

In the deliverable 5.22 a more detailed user manual will be inserted.

2.3 Virtual World Viewer

2.3.1 Overall Data

Component Name	**Virtual World Viewer**
Reference WP – Task	WP 5, Task 5.2
Responsible	DFKI
Version	0.1
Source Control	
Contact Person	Klaus Fischer

Short Description	Synchronized visualization of 3D content across multiple Web browser clients.

2.3.2 Architecture and Functionalities

2.3.2.1 Introduction

The Virtual World Viewer is based on the DFKI's XML3D technology and FiVES synchronization server. In the context of the Stargate the 3D Widget can be used for synchronized visualization of 3D content, i.e. multiple users can concurrently view 3D scenes or models where dynamic changes in the scene or model are synchronized among all participating clients. The visualisation infrastructure combines several generic enablers from FI-PPP and FI-WARE: FiVES as multi-user synchronisation framework, including KIARA as transport layer and XML3D as web-based 3D visualisation approach for the Web clients. Nevertheless, other client frameworks can be supported by implementing the required client plugins.

2.3.2.2 Summary of Functionalities

XML3D is a representation format for 3D content which allows direct visualization of the content in a standard Web browser with WebGL support. The design of XML3D allows seamless integration of 3D content into HTML pages. The interactive 3D content can be embedded directly into the HTML content. This way, 3D content can be displayed in arbitrary websites without any plug-in being installed.

FiVES provides a generic synchronization infrastructure with an extension to synchronize virtual worlds. It includes a server-side architecture which supports world representation based on the ECA model and client-side applications for rendering web-based 3D graphics with interactions. The server side architecture is extendable to allow application specific customization based on a plugin concept.

FiVES allows fast synchronization of virtual world scenes both for editing time and simulation time. It supports real-time interaction between clients on different devices (workstation, mobile, etc.). FiVES can be easily extended and customized to include application and use case specific functionality.

2.3.2.3 Component Architecture

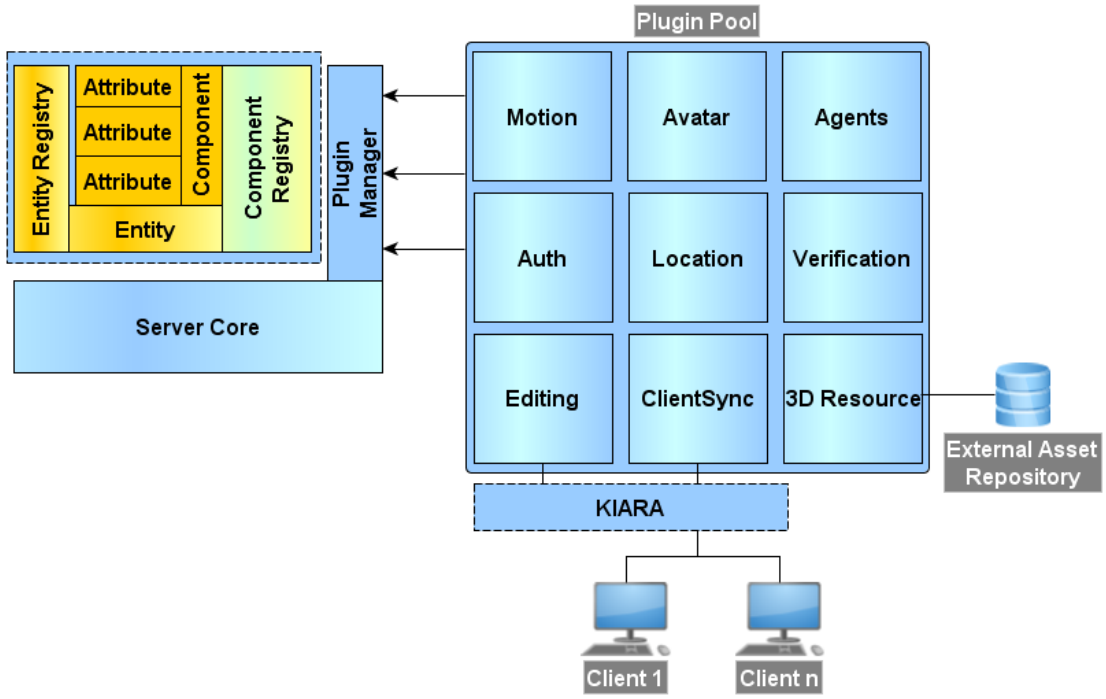

Figure 7: Architecture of FIVES

Figure 7 depicts the overall architecture of the 3D Widget. The core is formed by the synchronization server. The visualization of the 3D content which is assumed to be provided in XML3D format is visualized by standard Web browsers like for example Google Chrome or Firefox. For now the external asset repository is provided by files in a standard file system.

2.3.3 Technical Information

Name	Virtual World Viewer
Nature	Prototype
Programming Language	C#, Java Script
Development Tools	n./a.
Additional libraries	n./a.
Application Server	n./a.
Databases	n./a.

2.3.4 Technical Manual

2.3.4.1 FiVES Core

Basically, the server core consists of three building blocks:

- The World that contains all Entities

- The Plugin Manager that loads Plugins at server startup time. Please refer to the chapter on the Plugin System for further explanation
- The Component Registry that maintains Component Definitions, describing which sets of attributes of what type can be accessed on each Entity. Please refer to the chapter World Representation for a detailed description of the role of components.

To maintain flexibility to adapt easily adapt FiVES to the use case of your choice, FiVES core does not implement any domain specific logic, such as providing data for 3D rendered resources or movement - imagine your virtual environment should run completely in 2D sprites or even text only, then you would have to circumvent the 3D mechanics of the core to make your world run.

2.3.4.2 FiVES Plugins

Plugins are the means to add domain-specific logic to your FiVES application. Plugins are provided as .dll and loaded automatically when the server starts up, once they are copied to FiVES' plugin folder.

To simplify the re-use of plugins and avoid writing redundant code, FiVES provides a concept of Plugin Dependencies and Component Dependencies. If a plugin depends on another plugin or a specific component, the depending plugin is not loaded to the application, if the dependency cannot be resolved.

In the following, we describe in more detail the plugins in the plugin pool (see **Error! Reference source not found.**).

Auth <Prototype>

Auth plug-in provides mechanisms for client authentication. This plug-in was originally planned to be used for both authentication and authorization, hence the "Auth" name. However, it turns out that authorization requires the knowledge of the subject in question to define permissions and the like, therefore this plug-in performs only authentication for now and should be renamed into "Authentication".

Moreover, the plug-in does not check the validity of credentials yet. Users may log-in with arbitrary names and passwords. Having done so, the provided user name is linked to the connection established by the particular user.

Plug-In Dependencies:

- KIARA: Connection class is used as parameter to identify the authenticating client

Avatar

Adds 3D avatars to the world that represents connected users. Whenever a user successfully opens a new connection to FiVES, a new entity is created that is linked to the specific user. The 3D representation is provided via attributes from the Renderable plugin.

Plugin Dependencies

- Auth: Used to relate an avatar to an authenticated user account

- ClientManager: Used to create avatars for connected clients and synchronize their states to other clients
- KIARA: Used to access connection states and parameters of connected users to update avatar states accordingly

Component Dependencies

- meshURI: For reference to 3D view used to render the avatar
- scale: To change the visual appearance of the avatar concerning size
- velocity: To express motion of the avatar in the world
- rotVelocity: To express spin of the avatar

Client Manager

Client Manager implements the KIARA-based service interface that realizes client connections and interactions. This plug-in is so far the only measure to establish server-client connections, and intentionally designed as plug-in to allow exchanging the communication mechanics of the current implementation with custom implementations.

Client Manager is implemented in such a way that all components and attributes of loaded plugins are synchronized automatically, i.e. it is not necessary to implement a new synchronization routine for every new plugin.

Client Manager moreover provides a C#-Interface for other plugins to introduce new KIARA services. This should be replaced by an IDL-based approach in the near future.

Plugin Dependencies

- KIARA : Used to communicate with the clients.
- Auth: Used to authentication clients.

Component Dependencies

- location
- scale
- meshURI

Note: From the conceptual point of view, Client Manager should not depend on any kind of components, as it should handle all components in the same way. Component dependencies may disappear completely in future updates.

Editing

Editing plug-in implements a KIARA interface that allows clients to create new entities, delete them, or change position and orientation of existing entities.

Plugin Dependencies

- Renderable : Uses MeshResource struct type.

Component Dependencies

- location
- scale
- meshURI

EventLoop

Event loop provides an event that is fired in fixed intervals. Plugins can register to this event to invoke recurrent, fixed interval computations, like for example simulation updates. Using this plug-in for recurring computations is highly recommended.

Keyframe Animation

KeyframeAnimation adds interfaces and functionality to synchronize key frame animations between clients. Updating the key frames can either be done by each client individually, invoked by a single message to them, or on the server which in turn then sends the updated key frame values to clients. The plugin supports multiple animations per entity, referenced by name, and invoking animation playback with different parameters for speed, cycles and key ranges.

Plugin Dependencies

- EventLoop : Used to continuously update key frames on server side
- ClientManager: Used to register KIARA service interface for client synchronization

Component Dependencies

- None

KIARA

Implements the KIARA Middleware. Please refer to the documentation on the KIARA Service Interface for further information. In a future release, this plugin is to be changed to only include the official KIARA middleware C# library.

Location

Introduces components to describe a 3D location of an entity, consisting of a position in 3D Vector format, and orientation as Quaternion. Introduces KIARA service functions to update position and orientation of an entity.

Motion

Introduces capabilities to express motion of entities in the world. Based on the velocity attributes introduced by this plugin, server side computations update position and orientation values of moving entities periodically.

Plugin Dependencies

- EventLoop : Registers to EventLoop's TickFired event for the periodical position and orientation updates

Component Dependencies

- location

Renderable

Introduced components to include externally stored assets as renderable objects. Attributes contain an URI to the externally stored resource which needs to be processed by the client, and a visible attribute to toggle the mesh visible or invisible.

Static Scenery

Adds one single entity to the world as soon as server starts. MeshResource and Offset of the entity to the world's origin can be specified in the plugin's config file.

Plugin Dependencies

- none

Component Dependencies

- position
- meshURI

Terminal

Implements a command line terminal for server monitoring. The plugin provides an API to register new commands, which allows to introduce also monitoring for new plugins. A list of all commands currently registered in Terminal is printed when typing “?” or “help”.

The FiVES repository already comes with a set of plugins to choose from. Please refer to the individual Plugin Documentation documents that are linked from the FiVES Wiki.

2.3.4.3 FiVES World Representation

FiVES stores the world internally represented in an ECA (Entity-Component-Attribute) model. In this model, every object in the world is represented as an *Entity*. Entities do not carry any data on their own. They can be considered as 'empty objects'. Instead, information is attached to entities by specifying *Components*, which in turn contain a set of typed *Attributes*. That means, that the information that describes an entity is entirely contained in its components.

FiVES does not make any assumptions about what components are needed to describe an *Entity* in a meaningful way, i.e. there are no components that are added to entities by default. It is completely up to the user to decide which components to use by selecting the respective set of FiVES plugins.

2.3.4.4 Network synchronization

FiVES provides its capabilities to the clients via the ClientManager plugin which exposes them as a set of KIARA functions. Clients can use these functions to send data to the server and/or to registers callbacks for updates from the server. The interface may be extended by other plugins who may register their extensions via ClientManager’s API (see ClientManagerPlugin.ClientManager.RegisterClientService). For a list of services and functions that are currently implemented in FiVES, please have a look at the documentation of the KIARA-Service-Interface.

2.3.4.5 FiVES client

FiVES comes with a sample implementation of a browser-based Web-client. The client is implemented entirely in JavaScript and uses XML3D as implementation of the FI-WARE 3D-UI GE. However, FiVES is designed to synchronize between any client that implements the KIARA Service Interface, independent of the choice of programming language or platform. You may also write a plugin to provide synchronization logic other than FiVES' KIARA synchronization.

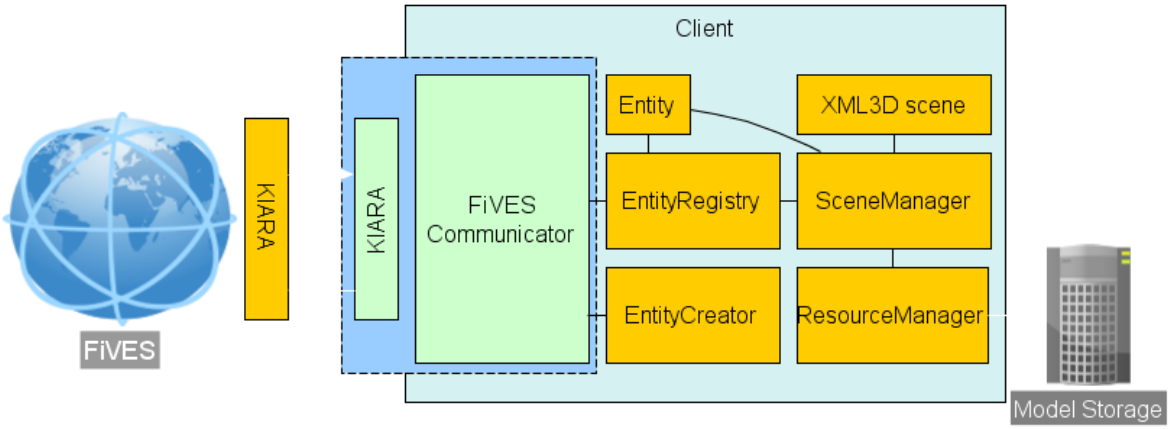

Figure 8: FiVES client architecture

The Web-client mirrors the world representation that is introduced by the core on server side. When the client connects to the server, it receives the current world state in terms of *Entity* objects that are then maintained in the *Entity Registry*. Having established the connection once, the client continuously receives updates for specific entities from the server, and updates the changed attributes of updated entities accordingly.

2.3.5 User Manual

2.3.5.1 Installation

2.3.5.1.1 Server:

From Source: Checkout the latest release or stable development build from https://github.com/rryk/FiVES. The solution uses NuGet-Package manager for third party libraries. Let NuGet restore the packages and build the solution. FiVES.exe and libraries for all Plugins will then be found in the Binaries/Release (or Binaries/Debug for Debug build) subfolder of the Repository's root directory.

2.3.5.1.2 Client:

The client is a HTML5 / JavaScript based browser-application using XML3D. It needs to be deployed somewhere into an HTTP Web-server. To do so, copy the folder "WebClient" contained in the FiVES Repository (https://github.com/rryk/FiVES) and place it somewhere within the htdocs folder of the server.

2.3.5.2 Configuration

2.3.5.2.1 Server:

Configuring Plugins:

In the folder of the FiVES server executable, you will find a configuration file *FiVES.exe.config.* In the section *appsettings,* you may specify the following values:

- PluginDir: The directory in which FiVES will search for precompiled Plugins (*.dll*)
- PluginBlackList / PluginWhiteList: Names of Plugins that should not be loaded (or name of Plugins that should only be loaded) when the server is startet, given as comma separated names, eg. *value="Location,Motion"*
- *ProtocolDir*: The directory in which FiVES will search for precompiled Protocols used by KIARA (*.dll)*
- ProtocolBlackList / ProtocolWhiteList: Names of Protocols that should not (or should only) be loaded when the server starts.

There will always only be a key for either Black- or Whitelist, i.e. no entries for both PluginBlackList *and* PluginWhiteList. If you would like to use Whitelists instead of Blacklists, make sure to replace the key *PluginBlackList* with *PluginWhiteList*. The same holds for *Protocols*.

Some of the plugins provide configuration files themselves. They will be called *<pluginname>.dll.config*, e.g. *StaticScenery.dll.config,* located in the same folder as the plugin .dll itself. The key values in these config files should be self-explanatory.

Configuring Client Communication:

So far, Server-Client-Communication is implemented in the *ClientManager*-Plugin, using KIARA as communication Middleware. Communication parameters can be specified in the file *clientManagerServer.json.* FiVES does currently not implement all features of KIARA, but only to start one server with given protocol on a given port. As a default, FiVES runs a websocket-json protocol on port 34837. These values usually don't need to be changed, but you may choose another port here.

2.3.5.2.2 Client

The KIARA configuration file fives.json for the WebClient is located in the /kiara/ subfolder of the Web-client root directory. Make sure to enter the IP address (for "host") and port ("port") on which the FiVES server is running. This must be a public IP address under which the server can be accessed from the outside (i.e. not localhost, if the client should be accessible publicly, even if the HTTP server that is running the client is running on the same machine as FiVES).

2.3.5.3 Usage

2.3.5.3.1 Server

Start the Server by double-clicking *FiVES.exe* in Windows, or typing *mono FiVES.exe* in the console in Linux. The server should start up. If *Terminal* plugin is loaded (highly recommended), the console should prompt

The Server is up and running use "quit" command to stop it …

>>

Type "help" or "?" to get a list of all available commands in Terminal.

2.3.5.3.2 Client

Navigate to *client.xhtml* located in the root directory of the client deployment on your Web-server. You will be prompted with a login dialogue. Enter arbitrary credentials (no user accounts and thus no credentials check implemented so far!). Control your avatar using W/A/S/D - keys on the keyboard.

2.3.5.4 Conclusions and Future Plans

A deeper integration with the other viewers is envisaged for the next prototype.

2.4 Stargate Configurator

The Stargate Configurator is formed by three components: the Stargate Viewers Configurator which will provide the functionalities to configure the Stargate itself and the viewers (it will be developed in the second phase of the project), the Ontology Configurator which allows to access and manipulate the OSMOSE ontologies and the Knowledge Link Configurator which consists of a graphical interface to manage the Knowledge Links.

2.4.1 Knowledge Link Configurator

2.4.1.1 Overall Data

Component Name	**Knowledge Link Configurator**
Reference WP – Task	WP5 – T5.2
Responsible	Artur Felic (CAS)
Version	1.0
Source Control	-
Contact Person	Artur Felic (CAS)
Short Description	Graphical Knowledge Link Editor for managing Knowledge Links between the OSMOSE Worlds or the Stargate and the Worlds.

2.4.1.2 Architecture and Functionalities

2.4.1.2.1 Introduction

The OSMOSE Worlds connected to each other via the OSMOSE Middleware. Each of the World import the ontologies of the Context Manager in order to extend the knowledge concepts with own specific concepts that belong only to the OSMOSE World itself. Thus, the knowledge bases of the OSMOSE Worlds can be evolved separately without changing the common structure which could lead to interoperability problems. Following this concept, the OSMOSE Worlds define different knowledge domains that doesn't provide a shared vocabulary anymore. Specific concepts stay specific while concepts defined under existing

concepts from the Context Manager can be classified. Although this is demanded, the shadow images of an entity in the three worlds share data via the three pairs of osmiotic processes and thus keeping background consistency. For this purpose, fomalized links between the shadow images i.e. their knowledge representations are necessary that are defined once by authorized users and running in the background.

In order to formalize complex knowledge connections between different knowledge domains, the Knowledge Link Configurator enables authorized users to model Knowledge Links graphically. Modified or newly created Knowledge Links can be stored and uploaded to the Context Manager.

2.4.1.2.2 Summary of Functionalities

Major functionalities of the Knowledge Link Configurator are:

1. Import ontologies and available Knowledge Links:
   Ontologies of the OSMOSE Middleware and the OSMOSE Worlds can be loaded from their location and imported into the graphical editor.
2. Create/Modify/Delete Knowledge Links:
   Use the graphical rule sheet of the graphical editor to (re-)define or delete Knowledge Links. Use the imported ontology concepts by dragging and dropping them on the rule sheet in order to reference ontology concepts.
3. Store/Upload Knowledge Links:
   Store the Knowledge Links locally or upload them to the Context Manager to allow background execution during OSMOSE Platform operation.

2.4.1.2.3 Component Architecture

The Knolwedge Link Configurator is based on the CAS M.Model, a graphical modelling editor of the CAS Configurator Merlin[1]. It is based on the Eclipse Rich Client Platform (RCP)[2]. For an architectural overview, please refer to the whitepaper: https://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html .

2.4.1.2.4 Technical Information

Name	CAS M.Model
Nature	Proprietary
Programming Language	Java
Development Tools	Eclipse
Additional libraries	Eclipse RCP
Application Server	-
Databases	-

[1] http://www.cas.de/crm-software/konfigurator/produktkonfigurator-merlin.html
[2] http://wiki.eclipse.org/Rich_Client_Platform

2.4.1.2.5 Licensing

Non-Commercial academic license.

2.4.1.2.6 Technical Manual

Run the 'Setup.exe' and follow the instructions on the screen to install and configure the Knowledge Link Configurator.

2.4.1.2.7 User Manual

A User Manual for the CAS M.Model can be requested via E-Mail. Please contact artur.felic@cas.de. A manual for the Knowledge Link Configurator will be provided with D5.22.

2.4.1.2.8 Conclusions and Future Plans

Our next steps involve integration and testing with the OSMOSE Platform and models. Additionally, we are going to provide a full user manual of the functionalities.

2.4.2 Ontology Configurator

2.4.2.1 Overall Data

Component Name	**Ontology Configurator**
Reference WP – Task	WP5 – T5.2
Responsible	Artur Felic (CAS)
Version	1.0
Source Control	-
Contact Person	Artur Felic (CAS)
Short Description	Ontology management tool for the OSMOSE Middleware and the OSMOSE Worlds

2.4.2.2 Architecture and Functionalities

2.4.2.2.1 Introduction

The knowledge of the OSMOSE Platform is structured with ontologies. Generic knowledge about events, services, entities and processes as well as middleware and Stargate specific knowledge is located in the Context Manager of the OSMOSE Middleware. The OSMOSE Worlds import the knowledge concepts defined inside the Context Manager and extend the concepts according to each worlds specific knowledge concepts.

The purpose of the Ontology Configurator is to manage the ontologies of the OSMOSE Platform. It allows Stargate users to access and load the ontologies of the OSMOSE Platform and the Worlds and edit, add or delete knowledge concepts.

2.4.2.2.2 Summary of Functionalities

Major functionalities of the Ontology Configurator are:

1. Create/Open/Save ontologies:

New ontologies can be created in order to allow modular ontologies for the OSMOSE Worlds. Ontologies of the Context Manager can be loaded and saved after modification.

2. Edit ontologies:
   The opened/created ontology can be edited. Entities, Classes, Object or data properties can be viewed and modified. New elements can be added or existing elements can be deleted.
3. Ontology visualization:
   The active ontology can be visualized in order to provide a convenience way to browse its elements.
4. Querying:
   SPARQL queries can be used for information retrieval.
5. Reasoning:
   Infer elements of the active ontology.

2.4.2.2.3 Component Architecture

The Ontology Configurator is based on Protégé and can be directly called from out the Stargate Configurator. Further information about Protégé can be found at: http://protege.stanford.edu/

2.4.2.3 Technical Information

Name	Protégé
Nature	Open Source
Programming Language	Java
Development Tools	Eclipse
Additional libraries	-
Application Server	-
Databases	-

2.4.2.4 Licensing

Open Source

2.4.2.5 Technical Manual

Please refer to the Protégé Developer Manual: http://protegewiki.stanford.edu/wiki/Protege5DevDocs

2.4.2.6 User Manual

Please refer to the Protégé User Manual: http://protegewiki.stanford.edu/wiki/Protege4UserDocs

2.4.2.7 Conclusions and Future Plans

Our next steps involve integration and testing with the OSMOSE Platform and models.

2.5 Stargate Core

This section describes the components of the Stargate Core: the Authentication and Authorization, the Stargate Engine and the Stargate DB.

2.5.1 Authentication and Authorization

2.5.1.1 Overall Data

Reference WP – Task	WP5.2
Responsible	Reply
Version	0.1
Source Control	n/a
Contact Person	Andrea Martelli, Fabrizio De Ceglia
Short Description	Security Authentication Layer

2.5.1.2 Architecture and Functionalities

2.5.1.2.1 Introduction

The goal of the security authentication layer is to create a trusted access to the Osmose platform and a correct granting of the resources based on the client that requests the access.

This layer will manage authentication and authorizazion to the Osmose platform and, eventually, among some of its sub-systems.

2.5.1.2.2 Summary of Functionalities

Different tool combinations have been tested to find suitable solutions.

As described in [D5.11, chapter 3.3] the tools that have been chosen to implement the Security Authentication Layer are:

- OpenAM;
- Shibboleth.

Here are three combinations of different tools to have the solutions that have been tested:

- Solution #1: OpenAM configured to work as *Identity Provider* and *Service Provider,* using the *Policy Agent* solution provided by the OpenAM platform. Access control

rules could be managed by the *Policy Agent*;

- Solution #2: OpenAM configured to work as *Identity Provider* and Shibboleth configured as *Service Provider; Service Provider* could also manage the access control rules;
- Solution #3: Same as Solution #2, but the entity where Shibboleth installed act as a reverse proxy to another web resource (an ipotetic Osmose instance); *Service provider* could also manage the access control rules; this solution has been tested to increase the performance, the reliability and the security of the platform.

2.5.1.2.3 Component Architecture

2.5.1.2.3.1 *Solution #1: OpenAM as* Identity Provider *and* Service Provider

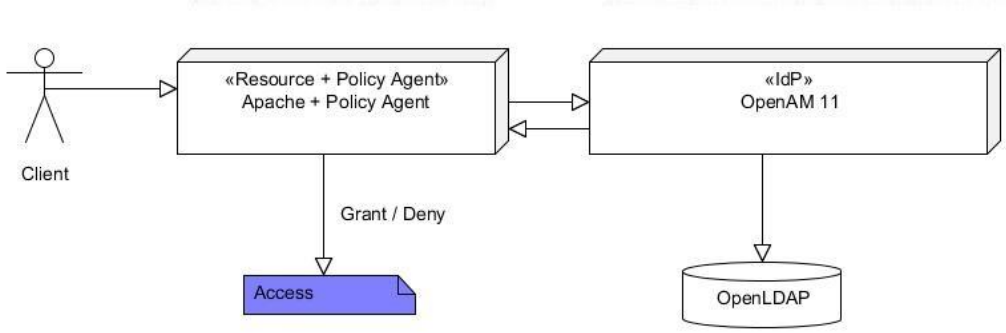

Figure 9: Solution #1 architecture/flow schema

In the solution #1, the client (e.g. an end user, client application) tries to access the protected resource; the *Web Policy Agent* installed on the same machine of the resource, as shown in **Error! Reference source not found.**, intercept the request and redirect the user to the login page of the *Identity Provider*; after the authentication, OpenAM set a session cookie and redirect the client to the requested source.

The *Web Policy Agent* use the token set by OpenAM to make the policy decision (access control) and grant or not the access to the requested resource.

As shown in **Error! Reference source not found.** OpenAM use an external LDAP directory as user store.

2.5.1.2.3.2 *Solution #2: OpenAM as* Identity Provider *and Shibboleth as* Service Provider

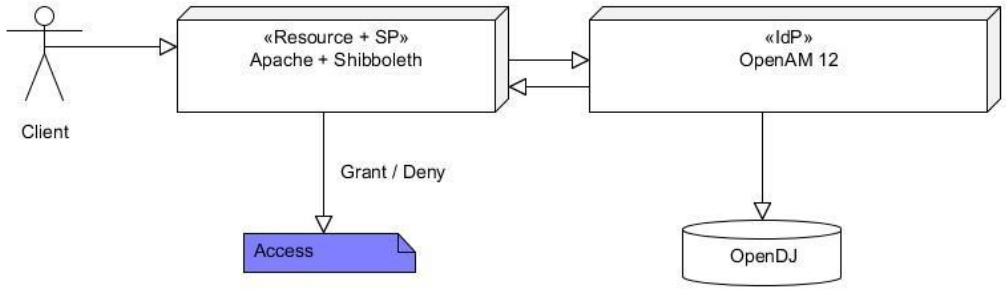

Figure 10: Solution #2 architecture/flow schema

Same as solution #1, the client (e.g. end user, client application) tries to access the protected resource; on the same machine, the Shibboleth 2 *Service Provider* module is installed.

Shibboleth generates a SAML assertion and redirects the user to OpenAM, the *Identity Provider* for the solution #2.

OpenAM authenticates the user and sends a SAML assertion to the *Service Provider* that analyses the response and decides whether to grant or not access to the protected resource.

As show in **Error! Reference source not found.**, OpenAM uses an internal OpenDJ (a particular implementation of LDAP for OpenAM by Forgerock) directory as user store.

In this solution, the access control is managed directly by rules defined in the web container of the *Service Provider.*

2.5.1.2.3.3 *Solution #3: OpenAM as* Identity Provider, *Shibboleth as* Service Provider *and* Reverse Proxy

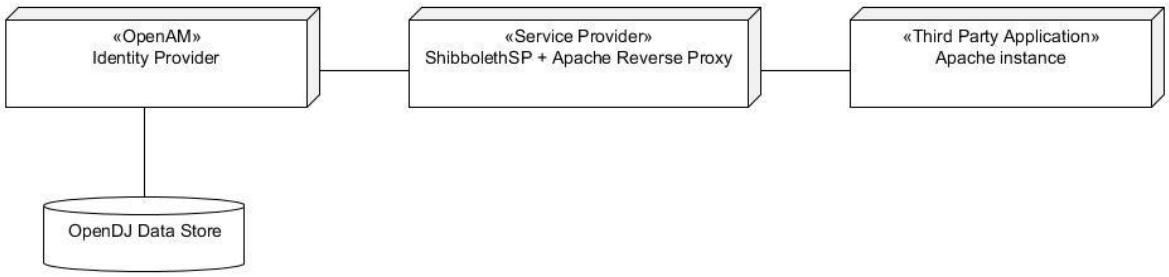

Figure 11: Solution #2 modified with Reverse Proxy

The solution #3 is a modified version of the solution #2.

In this case, the protected resource is a reverse proxy that will hide the requested resource of the Osmose platform, as shown in **Error! Reference source not found.**.

In this solution, like Solution #2, the access control is managed directly by rules defined in the web container of the *Service Provider.*

More specifically, through appropriate directives in the Apache or Shibboleth configuration files, it is possible to implement relatively simple but yet powerful authorization rules. It is in fact possible to restrict access to a location (e.g.: an HTTP URI directory) to a specific user or group of users, to use wildcards and regular expression and to use any LDAP attribute as a filter to take the final authorization decision. The only pre-requisite to use this kind of policies is to have LDAP attributes correctly mapped into SAML identity providers' configuration.

2.5.1.2.4 Technical Information

Name	OpenAM
Nature	Idenity Provider / Service Provider
Programming Language	Java
Development Tools	//
Additional libraries	//
Application Server	Jetty, Tomcat, GlassFish,IBM Web Sphere, JBOSS, Oracle Web Logic Server
Databases	LDAP directory, OpenDj, relational DB,OTP

Name	Shibboleth Sp
Nature	Service Provider
Programming Language	C++, Java, Java Script
Development Tools	//
Additional libraries	//
Application Server	Apache
Databases	//

Name	OpenDJ
Nature	Data Store
Programming Language	Java
Development Tools	OpenDJ SDK
Additional libraries	//
Application Server	Tomcat, Jetty
Databases	//

Name	Web Policy Agent
Nature	Policy Agent for access control and policy decision
Programming Language	Java

Development Tools	//
Additional libraries	//
Application Server	Apache, IIS, Oracle iPlanet, Sub Web Server
Databases	//

Name	Apache
Nature	HTTP Server
Programming Language	C
Development Tools	//
Additional libraries	//
Application Server	//
Databases	//

Name	Tomcat
Nature	Application Server
Programming Language	Java
Development Tools	//
Additional libraries	//
Application Server	//
Databases	//

2.5.1.2.5 Licensing

All the software used in the three solutions explained in the **Error! Reference source not found.** is open-source and free to download.

OpenAM project is provided under multiple licenses (http://openam.forgerock.org/license.html):

- CDDL-1.0
- CC BY-NC-ND 3.0

Shibboleth project is released under the Apache Software License.

All the components of the Forgerock family (OpenAM, the Policy Agent and OpenDJ) are also available in an enhanced version, which involves a paid subscription with extended support and mid-version updates.

2.5.1.2.6 Technical Manual

2.5.1.2.6.1 Solution #1: install, configure

To verify this solution virtual machine inside the Reply cloud infrastructure has been created and configured with the following parameters:

- Ubuntu 12.02 LTS
- Jetty 9
- Apache 2.4
- OpenAM 11
- Web Policy Agent 3.3.0

Download the OpenAM 11 war file from the download section of the OpenAM web site:

- http://forgerock.org/downloads/openam-builds/

Follow this tutorial to install and configure OpenaAM:

- http://docs.forgerock.org/en/openam/11.0.0/install-guide/

Browse to *http://servername:8080/openam/console/* to access to console.

Follow this tutorial to install the *Web Policy Agent:*

- http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/web-install-guide/

Follow this tutorial to configure the web policy agent and configure the policies (access control):

- http://docs.forgerock.org/en/openam/10.1.0/admin-guide/index/chap-authz-policy.html#configure-authz-policy

Browse to the protected resource set in the Policy Agent setup and the Policy Agent will make sure that the user is authenticated at the *Identity Provider (login screen should be presented)*.

After submitting the valid credentials user will be redirected back to the *Service Provider* application to the originally requested URL.

2.5.1.2.6.2 Solution #2: install, configure

In the Solution#2 the goal is to achieve SSO between an OpenAM *IdentityProvider* and a Shibboleth *Service Provider*.

- **Identity Provider** (**IdP)** - holds all information about the user (for example in LDAP), and also it is the IdP's job to authenticate the users, and decide on what kind of information it shares about the users with other providers.
- **Service Provider** (**SP**) - is as an extra layer in front of the webapplication. The SP's job is to authorize page requests, and if there is no authenticated session at the SP, initiate an authentication request to the IdP.

To verify this solution Reply cloud virtual machine has been used with the following parameters:

- Ubuntu 14.04 LTS
- Tomcat7
- Apache 2.4
- Shibboleth Sp 2.5.2

- OpenAM 12 (Snapshot 29.08.2014)
- OpenDJ 2.7.0 (Snapshot 29.08.2014)

OpenDJ should be downloaded from the link:

- http://download.forgerock.org/downloads/opendj/20140829020001/OpenDJ-2.7.0-20140829.zip

OpenDJ should be installed in the desired folder following the description from the following link:

- https://wikis.forgerock.org/confluence/display/OPENDJ/OpenDJ+Installation+Guide

OpenAM 12 should be installed and configured based on the following description:

- http://docs.forgerock.org/en/openam/11.0.0/install-guide/

Configure OpenAM as a hosted identity provider following these steps:

- Log in to the admin console and on the Common Tasks pane click on the Create Hosted Identity Provider link
- Select No for Do you have metadata for this provider
- Use http://servername:8080/openam as Name
- Select the default test Signing Key
- Use cot as the name of the New Circle of Trust
- Leave the Attribute mapping table empty
- Press the Configure then the Finish button

Install and configure Shibboleth SP following the description from the following link:

- http://blogs.forgerock.org/petermajor/2011/10/federation-with-shibboleth-sp-apache-module/

Change Apache2 config to support Shibboleth authentication. To do so, create an *.htaccess* file in the *docroot (/var/www/html)* containing:

```
AuthType shibboleth
ShibRequireSession On
require shibboleth
```

Now enable the Apache Shibboleth module by running this command:

a2enmod shib2

Browse to *https://servername/Shibboleth.sso/Metadata* to obtain the Shibboleth SP metadata. Save the metadata file then browse to OpenAm console and login as admin.

1. Browse to **Register Remote Service Provider** on the common task pane
2. Upload the metadata file just downloaded
3. Click on the configure button
4. Go to the **Federation** page and open the SP's page located in the **Entity Providers** table

5. Go to the **Assertion Processing** tab
6. In the **Attribute Map** list add this value:

   urn:oasis:names:tc:SAML:2.0:attrname-format:uri|urn:oid:0.9.2342.19200300.100.1.1=uid
7. Save the configuration and logout

Browse to the Shibboleth machine and the configuration created in the .htaccess file will make sure that the user is authenticated at the *Identity Provider (login screen should be presented)*.

After submitting the valid credentials user will be redirected back to the *Service Provider* application to the originally requested URL.

2.5.1.2.6.3 Solution #3: install and configure

To verify this solution, the third party application Apache instance running on another Reply cloud machine that simulate an ipotetic Osmose instance)will be used.

Configure the Apache 2.4 instance on the *server* machine to work as a reverse proxy.

Delete the *.htaccess* file in the */var/www/html* directory created in the **Error! Reference source not found.**.

Activate the proxy module on apache running this command:

a2enmod proxy_http

Update the virtual host configuration of the machine to configure the reverse proxy, following this tutorial:

- http://www.apachetutor.org/admin/reverseproxies

Browse to the Shibboleth machine and the configuration created in the virtual host file will make sure that the user is authenticated at the *Identity Provider (login screen should be presented)*.

After submitting the valid credentials user will be redirected back to the *Service Provider* application to the originally requested URL and the proxy will reverse to the third party application.

2.5.1.2.6.4 Solution #2 & Solution #3: Configure the access control

As said in **Error! Reference source not found.**, the access control in the Solution #1 is managed by the *Policy Agent*, a component of OpenAM; it is configurable through OpenAM console and system console as described in **Error! Reference source not found.**.

For Solution #2 and Solution #3, the access control can be managed directly from the Apache instance where the *Service Provider* is running, taking advantage of some users attributes stored in the data store.

To implement access control in Solution #2 and Solution #3 the attribute used is "*isMemberOf*".

This attribute indicates the membership of a user with a certain group in the data store.

To do this, follow these steps:

- update the Shibboleth Service Provider configuration adding the "isMembeOf" attribute in the attribute map of the Shibboleth, following this tutorial: https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPAddAttribute
- update the OpenAM federation and user store configuration adding the "isMemberOf" attribute in the attribute configuration of the OpenDJ data store, by following this tutorial: http://docs.forgerock.org/en/openam/11.0.0/admin-guide/index.html
- update the Apache reverse proxy configuration adding access control rules, by following this tutorial: http://httpd.apache.org/docs/current/howto/access.html

2.5.1.2.7 User Manual

At the time no software components have been developed.

Refer to the official guides of the used tools to retrieve the specific user manual:

- OpenAM: http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/admin-guide/index.html
- Shibboleth: https://wiki.shibboleth.net/confluence/display/SHIB2/Home

2.5.1.2.8 Conclusions and Future Plans

As described in [D5.11, chapter "Security Authentication"], the big advantage of using OpenAM is that we don't have to modify, redeploy or redesign the application that need a security layer; also OpenAM support the majority of the authentication/authorization protocols (Oauth2, OpenID, etc) and expose lots of useful REST API. These features make it possible to integrate and federate different services in a way that is language and platform agnostic, and to support a number of Service and Identity Providers when needed.

The policy agent solution described in the **Error! Reference source not found.** is very simple to configure and very powerful for defining policy rules: for example we can define specific rules for specific users, groups of users and specific directory. A con is that the Policy Agent is not a standard and that is not completely open source; in fact some of the bugs are solved only in the subscription version of the product.

Shibboleth *Service Provider*, used in the Solution #2 and Solution #3, is a completely open source product, with a more complete documentation and based on the SAML 2.0 standard.

OpenDJ data store, used in the Solution #2 and Solution #3, is a great solution combined with OpenAM; in fact it is fully supported and integrated in the OpenAM solution; OpenDJ also provide a complete set of REST API but is not a standard and his data schema in not standard.

The reverse proxy in Solution #3 is a good start for building a platform with a higher flexibility; with this solution we can attach different services behind the proxy with simple configuration (e.g. Apache proxy configuration) and a firewall configuration for each service. The disadvantage of this configuration is that there is one more component in the architecture to be added, the reverse proxy.

2.5.2 Stargate Engine

2.5.2.1 Overall Data

Component Name	**Stargate Engine**
Reference WP – Task	WP5 – Task 5.2
Responsible	PIKS
Version	1.0
Source Control	n/a
Contact Person	Cinzia Rubattino, Gianluigi Di Vito
Short Description	Back end services to guarantee consistency between the viewers and to provide access to the Stargate DB

2.5.2.2 Architecture and Functionalities

2.5.2.2.1 Introduction

The Stargate Engine represents the point of contact of the world's viewers. It provides also the methods to access the Stargate DB. All the data stored in the Stargate DB are retrieved from the Stargate Engine.

2.5.2.2.2 Summary of Functionalities

The Stargate consists of a set of REST services. The main services provided are:

1. **POST – addARNotification**

   @FormParam("userid")/@FormParam("objid")/@FormParam("functionId"): insert a new notification for the user "userid" and the object "objid" in the function "functionId"

   http://localhost:8080/OsmoseServices/addARNotification

2. **GET – retriveUserNotications:**
   returns the list of the notification for the user specified in the request

   http://localhost:8080/OsmoseServices/retriveUserNotications/user1

3. **POST – consumeNotification @FormParam("idNotification"):**
   Mark as red the notification specified

   http://localhost:8080/OsmoseServices/consumeNotification

4. **GET – retriveObjHistory:**
   Returns the history of the object specified

   http://localhost:8080/OsmoseServices/retriveObjHistory/1020e101

5. **GET – retriveObjTimeline**
   Returns the timeline of the selected object

http://localhost:8080/OsmoseServices/retriveObjTimeline/1020e101

6. **GET – retriveObjMediaAttach**
   Returns the ids of the content attached to the selected object

   http://localhost:8080/OsmoseServices/retriveObjMediaAttach/1020e101

2.5.2.2.3 Component Architecture

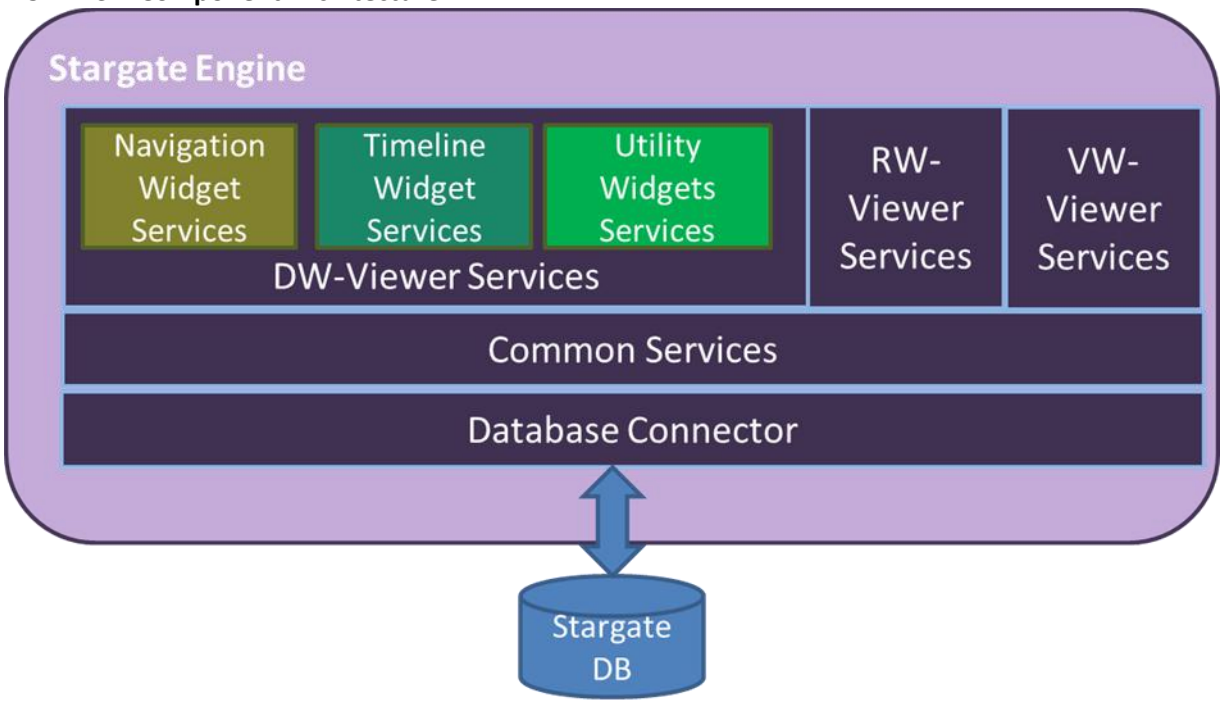

Figure 12 Stargate Engine architecture

The Database Connector groups the services which provide the access to the Stargate DB. The level just above the Database Connector, called Common Services, includes a set of basic services shared by all the components accessing the Stargate Engine. The Common Services guarantee also the consistency between the worlds' viewers; these services implement the mechanism which allows the users to jump from one perspective to another preserving the navigation history and letting the user to have the better perspective of the assets every time and on almost every device(PC, tablets, smartphones…).

2.5.2.2.4 Technical Information

Name	Stargate Engine
Nature	RESTful web services
Programming Language	Java
Development Tools	Eclipse
Additional libraries	Jersey 2.x, mySqlConnector Jettison
Application Server	Tomcat
Databases	//

2.5.2.2.5 Licensing

Eclipse - Eclipse Public License (EPL) https://www.eclipse.org/legal/epl-v10.html .

mySqlConnector – Open Source (GPL License)

Tomcat - Open Source (Apache License, Version 2.0)

Jettison – Open Source (Apache License, Version 2.0)

2.5.2.2.6 Technical Manual

Install Apache Tomcat: http://tomcat.apache.org/tomcat-8.0-doc/setup.html

Deploy OsmoseService.war file

In a browser, enter the following URL to open the Tomcat Manager application: http://localhost:8080/manager/html

Enter the user name and password.

In the Deploy > WAR file to deploy section, click Choose file.

Select the OsmoseService.war file.

Click Deploy. The OsmoseService.war file is deployed, started, and included in the Applications section. The OsmoseService.war file is also copied to the CATALINA_HOME/webapps directory.

2.5.2.2.7 User Manual

The user manual is not applicable for this component.

2.5.3 Stargate DB

2.5.3.1 Overall Data

Component Name	**Stargate DB**
Reference WP – Task	WP5 – Task 5.2
Responsible	PIKS
Version	1.0
Source Control	n/a
Contact Person	Cinzia Rubattino, Gianluigi Di Vito
Short Description	The Database of the Stargate

2.5.3.2 Architecture and Functionalities

2.5.3.2.1 Introduction

The Stargate DB contains the information that is shared by the different viewers. All the viewers are able to access the Stargate DB through the Stargate Engine which provides methods to retrieve and write data in the Database.

2.5.3.2.2 Summary of Functionalities

The Stargate DB is a simple rational database developed using MySQL.

In the current prototype the Stargate DB will be used also to store the data needed to simulate the integration with the Data Access Gateway of the Middleware.

2.5.3.2.3 Component Architecture

The Stargate DB is formed by a number of tables which allows storing the information about the notifications; the Figure 13 depicts such model.

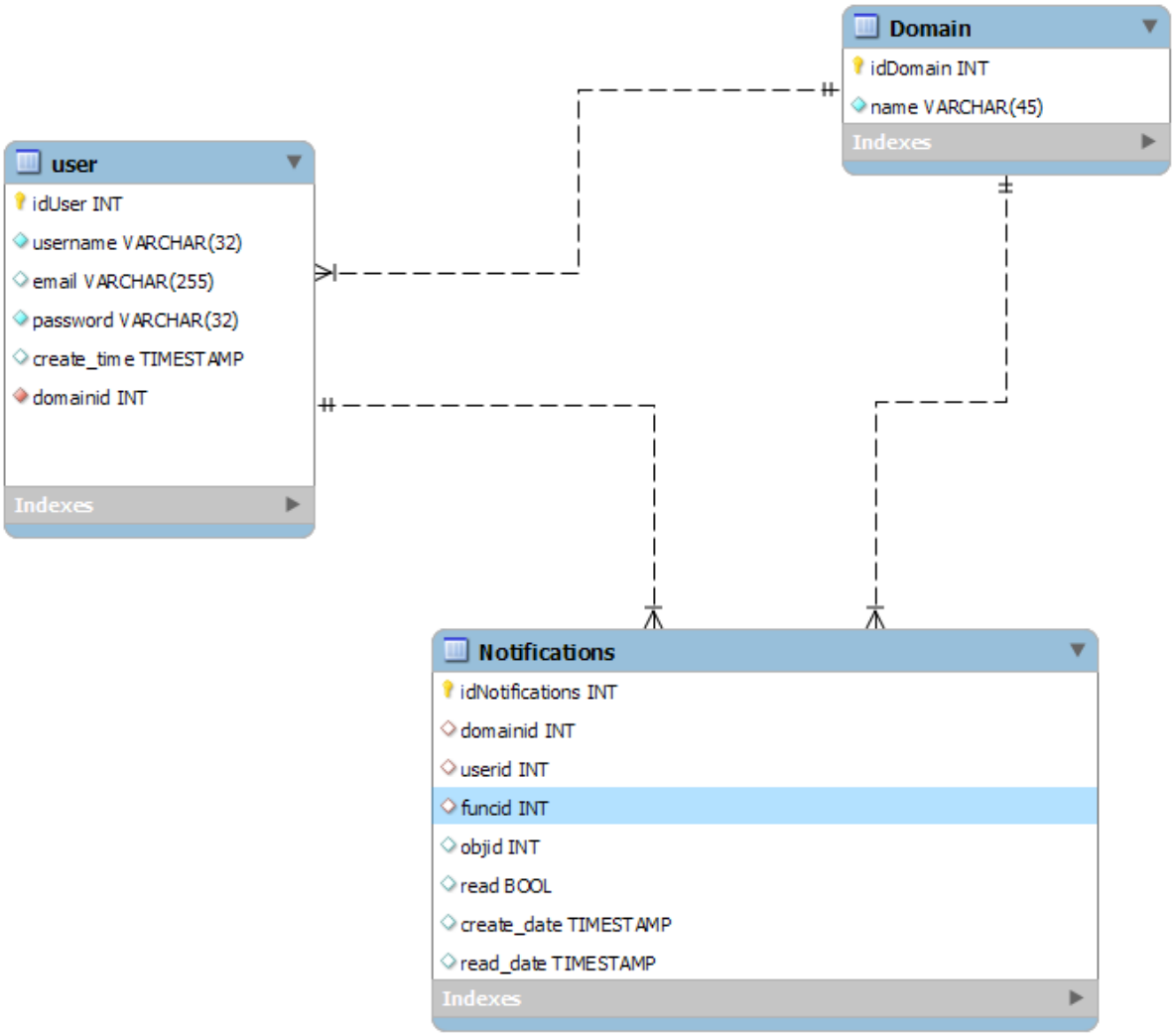

Figure 13 Stargate Notification Model

Given that the integration with the OSMOSE middleware and the legacy system of the pilots is not yet available, the Stargate DB provides also a model for storing the information simulating the data of the pilots. This structure guarantees the correct operation of the Stargate and provides the first input for the specification of the integration process. The Figure 14 represents the structure simulating the EPC data.

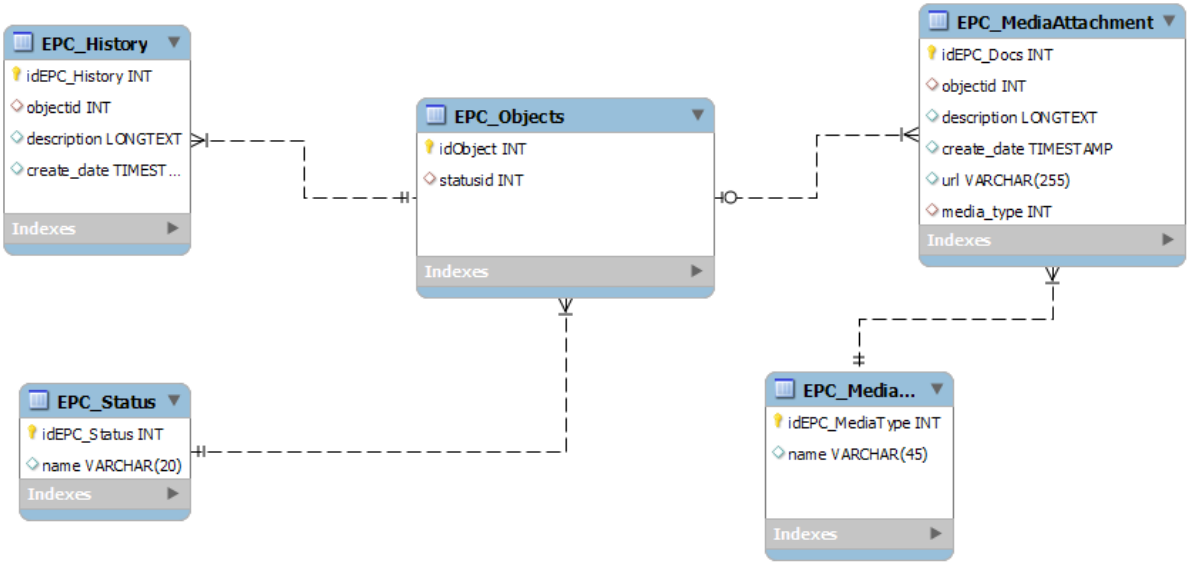

Figure 14 EPC data structure

2.5.3.2.4 Technical Information

Name	OpenSocial Database
Nature	Data Base
Programming Language	SQL
Development Tools	MySql WorkBench
Additional libraries	//
Application Server	//
Databases	MySQL

2.5.3.2.5 Licensing

MySQL DB Community Edition — Open Source (GPL License)

MySQL Workbench Community Edition — Open Source (GPL License)

2.5.3.2.6 Technical Manual

Install MySQL: http://dev.mysql.com/doc/refman/5.6/en/installing.html

Install MySql WorkBench: http://dev.mysql.com/downloads/workbench/

Open MySql WorkBench and connect to MySQL server

Click on Data import/Restore

Click on Import from Self-Contained File and select the **OsmoseDB.sql** file.

Click on Start import

2.5.3.2.7 User Manual

The user manual is not applicable for this component.

2.5.3.2.8 Conclusions and Future Plans

Once the integration with the Middleware is performed the Stargate DB structure will be revised deleting the structure simulating that integration and a merging with the OSMOSE database presented in D4.11 will be evaluated. Moreover a more strict integration with the Apache Rave Database is foreseen.

3 Conclusions and Future Steps

In this document the architectural components specifications of the Stargate have been provided in order to accompany the technical prototype.

The main components of the Stargate have been implemented by M12 and will be integrated in the whole architecture by M18. The integration will take place in WP3. The integrated system will be the major result of the MS4 project milestone at M18.