



COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement N° 609043

D7.6.2 Integration Plan (Updated)

WP7: Use cases Adaptation, Integration and Experimentation

Version: 1.0

Due Date: 30/04/2015

Delivery Date: 30/04/2015

Nature: Report

Dissemination Level: Public

Lead partner: ICCS/NTUA

Authors: G. Kousiouris, A. Marinakis, P. Bourellos, O. Voutyras (NTUA), , P. Ta-Shma (IBM), A. Akbar, F. Carrez (UniS), B. Târnaucă, L. Plitu, (Siemens), S. Ballaguer, A. Martin (EMT), Juan Sancho (ATOS)

Internal reviewers: Juan Rico (ATOS), Paula Ta-Shma (IBM)

www.iot-cosmos.eu



The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 609043

Version Control:

Version	Date	Author	Author's Organization	Changes
v0.1	16/2/2015	George Kousiouris	NTUA	Circulating the ToC, adding new subchapters
V0.2	23/3/2015	George Kousiouris	NTUA	Adaptation of previous content (update of charts etc.), nodered
V0.3	3/4/2015	George Kousiouris	NTUA	Inputs on Madrid Scenario concretization
V0.4	6/4/2015	G. Kousiouris, A. Martin, S. Balaguer	NTUA/EMT	Inputs of EMT Bus API and adaptation
V0.41	10/4/2015	Paula Ta-shma	IBM	Inputs on Data Model
V0.42	15/4/2015	Leo Pitu	SIEMENS	Inputs on VE side integration
V0.43	19/4/2015	J. Sancho, A. Akbar	ATOS	Updates on 5.2
V0.44	22/4/2015	Panagiotis Mpourelos	NTUA	Updates on London scenario concretization
V0.45	23/4/2015	George Kousiouris	NTUA	Inputs on Application Definition Framework, archetypes, finalized chapters
V0.5	25/4/2015	George Kousiouris	NTUA	Version ready for QA
V0.6	27/4/2015	Paula Ta-Shma, Juan Rico	IBM, ATOS	Internal review performed
V1.0	28/4/2015	George Kousiouris	NTUA	Comments addressed, document ready

Table of Contents

Executive Summary	11
1. Introduction	13
1.1. Objectives.....	14
1.2. Differences to previous version	14
2. Integration Strategy	15
2.1. COSMOS Overall Goal/Vision	15
2.2. Integration Goals.....	17
2.2.1. COSMOS Platform and cross-component integration	17
2.2.2. Data Model/Template	17
2.2.3. VE description and linking in the COSMOS Platform	17
2.2.4. VE-side COSMOS components integration	17
2.2.5. Application definition, creation and deployment	18
2.3. Integration Stages Definition	18
2.4. Integration Timeline in a nutshell	18
2.5. Integration Stages Template Fields.....	20
2.5.1. Involved Integration Goals	20
2.5.1.1 Subgoals	20
2.5.2. Subsystems and integration points involved	20
2.5.2.1 Components involved.....	20
2.5.3. Use Case specific aspects involved and concretization.....	21
2.5.4. Testing environment	21
2.5.4.1 Testing infrastructure.....	21
2.5.4.2 Involved Tester roles	21
2.5.5. Deviations from Plan	22
2.6. Generic Considerations	22
2.6.1. Software Packaging	22
2.6.2. Helper Tools	22
3. Integration Stage 1 (M10-M16).....	23
3.1. Involved Integration Goals and Refinement to Subgoals.....	23
3.2. Subsystems involved	24
3.2.1. Data Feed, Annotation and Storage	24
3.2.2. Metadata Search and Storlets.....	25

3.2.3.	Modelling and Storage Analytics.....	25
3.2.4.	Security, Privacy and Storage	27
3.2.5.	Autonomous Behavior of VEs with minimum integration with the Platform	27
3.2.6.	Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform	28
3.2.7.	Components involved.....	29
3.3.	Overview of Use Case specific aspects involved and concretization	29
3.3.1.	Camden UC Data Feed.....	30
3.3.2.	Madrid UC Data Feed	31
3.4.	Testing environment	31
3.4.1.	Testing infrastructure - The COSMOS Platform Setup	31
3.4.2.	Involved Tester roles	32
4.	Integration Stage 2 (M17-M22).....	33
4.1.	Involved Integration Goals and Refinement	33
4.2.	Subsystems/Subgoals involved	34
4.2.1.	VE Instances Descriptions and Registry population	34
4.2.1.1	Prerequisites: COSMOS Ontology and Domain Specific Ontologies Linking ...	34
4.2.1.2	VE Registration Subsystem.....	34
4.2.2.	Application Definition Framework through NodeRed Flows	35
4.2.3.	Data Fields Definition	38
4.2.4.	VE side COSMOS Components integration	39
4.2.4.1	VE Resources specification.....	39
4.2.5.	Application Archetypes Definition	40
4.2.5.1	Application VE2VE archetype	40
4.2.5.2	Application with centralized logic archetype	41
4.3.	Use Case specific aspects involved.....	42
4.4.	Testing environment and roles	42
5.	Integration Stage 3 (M23-M25).....	44
5.1.	Involved Integration Goals and Refinement	44
5.2.	Subsystems/Subgoals involved	45
5.2.1.	Application Definition Framework through NodeRed Flows (continuation)	45
5.2.2.	VE Instances Descriptions, Registry population and DSOs (continuation).....	46
5.2.3.	VE side COSMOS Components integration	46
5.2.3.1	VE endpoints definition.....	46

5.2.3.2	Components installation to VE instances.....	46
5.2.4.	JSON Schema retrieval subsystem	47
5.2.5.	COSMOS Platform and cross components integration (continuation)	47
5.2.5.1	CEP, Situational Awareness and Machine Learning Cooperation	47
5.2.5.2	Planning, Experience sharing and Situational Awareness.....	48
5.2.5.3	COSMOS Process Web UI integration	50
5.2.6.	Data feeds integration.....	52
5.2.6.1	Madrid Bus API UC data	52
5.2.6.2	Taipei UC data	58
5.3.	Use Case specific aspects involved and concretization.....	58
5.3.1.	Application scenarios concretization (Madrid Case).....	59
5.3.2.	Application scenarios concretization (Camden Case)	61
5.4.	Testing environment and roles	62
6.	Integration Stage 4 (M26-M34).....	63
6.1.	Involved Integration Goals and Refinement	63
6.2.	Subsystems/Subgoals involved	64
6.2.1.1	Components installation to VE instances.....	64
6.2.1.2	VE Instances Descriptions and Registry population	64
6.2.1.3	Complex Applications creation and deployment- Nodered template flows ..	64
6.2.1.4	Application archetypes potential extension	64
6.2.1.5	Linking external platforms/services (Optional).....	65
6.2.1.6	COSMOS Process Web UI	65
6.3.	Use Case specific aspects involved and concretization.....	66
6.4.	Testing environment and roles	66
7.	Traceability Matrix of Capabilities to Use Cases	67
8.	Conclusions	69
Annex A	Components Involved	70
•	COSMOS platform	70
▪	Nodered Environment.....	70
▪	Data Mapping.....	70
▪	Message Bus.....	70
▪	Cloud Storage-Metadata Search	70
▪	Cloud Storage- Storlets.....	70
▪	Event Detection and Situational Awareness	71

▪ Prediction	71
▪ Semantic Description and Retrieval	71
• VE Level	71
▪ Privelets	71
▪ Planner	71
▪ Event Detection and Situational Awareness	72
▪ Experience Sharing	72
Annex B Project Computing Testbed Details	73
• Basic Requirements	73
▪ Accessibility	73
▪ Security	73
• Components to Virtual Resources Mapping	73
• Testbed Description	74
Annex C Software Packaging and Delivery	75
• Installation - Execution	75
• Standard Naming Convention	76
• Standard Readme File	76
• Standard License File	76
• Manuals	76
• Acceptance Procedure	77
• Integration Tools	78
▪ Revision Control System	78
▪ Alfresco	78
▪ Wiki	78
▪ Code Quality checks	78
▪ Template Adaptation and Validation	78
References	79

Table of Figures

Figure 1: COSMOS Vision. Functionalities and Roles	15
Figure 2: COSMOS Integration Strategy Overview.....	16
Figure 3: COSMOS Integration Timeline Gantt Chart.....	19
Figure 4: Overall COSMOS Architecture.....	24
Figure 5: Data Feed, Annotation and Storage Subsystem	25
Figure 6: Metadata Search and Storlets Subsystem	26
Figure 7: Modelling and Storage Analytics Subsystem	26
Figure 8: Security, Privacy and Storage subsystem.....	27
Figure 9: Autonomous Behavior of VEs with minimum platform integration subsystem	28
Figure 10: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform	28
Figure 11: Camden UC data.....	30
Figure 12: Madrid UC data	31
Figure 13: COSMOS Platform Setup Process.....	32
Figure 14: VE Descriptions and Registry population	35
Figure 15: Indicative Nodered flow for CEP Rules update	37
Figure 16: Testing a given flow.....	38
Figure 17: Roles interaction with Flows repository	39
Figure 18: VE2VE application archetype based on defined system cases of D2.3.2.....	41
Figure 19: Application with centralized logic archetype based on defined system cases	42
Figure 20: Generic application design process.....	46
Figure 21: JSON schema storage and association	47
Figure 22: Cooperation of Inference/Prediction with Event Detection (from D2.3.2) and involved integration points.....	48
Figure 23: Experience Sharing for propagation of Situational Awareness.....	49
Figure 24: The Situational Awareness Nodered process	50
Figure 25: Situational Awareness, Application Logic/Planner and Experience Sharing.....	50
Figure 26: Example GUI Mock-up for COSMOS Front-End.....	51
Figure 27: Example of Bus SDK reply for bus info	56
Figure 28: Example of Taipei data	58
Figure 29: Madrid Assisted Mobility application runtime.....	60
Figure 30: The Flat VE and its connections and characteristics.	62
Figure 31: Web UI expected functionalities.....	65

List of Tables

Table 1: Test Case Template	20
Table 2: Web Services from the Bus SDK	53
Table 3: Attributes and possible values for DatosCoche XML response.....	54
Table 4: Stops fields description for estimated arrival	55
Table 5: Traceability Matrix of capabilities to UC scenarios	67
Table 6: Software requirements for the Nodered Environment.....	70
Table 7: Software requirements for the Data Mapping.....	70
Table 8: Software requirements for the Message Bus.....	70
Table 9: Software requirements for the Metadata Search	70
Table 10: Software requirements for the Storlets	70
Table 11: Software requirements for the Event Detection at the COSMOS platform.....	71
Table 12: Software requirements for the Event Detection at the COSMOS platform.....	71
Table 13: Software requirements for the Semantic Description and Retrieval	71
Table 14: Software requirements for the Privelets.....	71
Table 15: Software requirements for the Planner	71
Table 16: Software requirements for the Event Detection at the VE level.....	72
Table 17: Software requirements for the Experience Sharing.....	72
Table 18: Components to VMs mapping and VM configuration requirements.....	73

Acronyms

Acronym	Meaning
ARM	Advanced RISC Machines
CBR	Case-Based Reasoning
CEP	Complex Event Processing
CG	Care Giver
CPU	Central Processing Unit
D	Deliverable
DoW	Description of Work
DSO	Domain Specific Ontology
ETA	Estimated Time of Arrival
FC	Functional Component
GPS	Global Positioning System
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IoT	Internet of Things
IP	Integration Point
JSON	Java-Script Object Notation
MB	Message Bus
MS	Milestone
OS	Operating System
PE	Physical Entity
REST	Representational State Transfer
SDK	Software Development Kit

SOAP	Simple Object Access Protocol
SP	Special Person
SSH	Secure Shell
SVN	Subversion
TCP	Transmission Control Protocol
UC	Use Case
UI	User Interface
URL	Uniform Resource Locator
UTM	Universal Transverse Mercator
VE	Virtual Entity
VM	Virtual Machine
WP	Work Package
XML	Extensible Mark-up Language

Executive Summary

The main focus of this document is to highlight the integration plan to be followed in order to enable the advanced and combined capabilities of the COSMOS platform. To this end, a relevant strategy has been defined, that separates the final goal into 5 partial functional goals. The latter are further refined to 16 subgoals that involve more concrete aspects and specific involved subsystems of the COSMOS ecosystem. For each part of the ecosystem, relevant integrating roles have been identified and their integration points with the platform have been defined. For these integration points specific details and instructions will be offered in the context of the “Integration of Results” document series (D7.7.x).

A division is performed also with regard to the timeline of the project, coordinated with its milestones, and prioritized based on the stepwise gradual increase of the offered functionalities and incorporated elements. To this end, 5 relevant integration periods have been defined and concrete time lines for the goals and subgoals have been drawn. In order to properly prepare these periods, a relevant description template has been defined, incorporating the necessary details that need to be in place such as description of the main subgoals of this period, involved subsystems, components and roles, specific tests or test setups and the aspects of the UCs that can or must be concretized in order to enable the application scenarios. This description drives also the reporting of the results in the relevant D7.7.X document series, in terms of necessary content and type of information. The initial envisioned integration period details have been included in the previous version of the document for period 1 (PM10-16), but have been maintained here for consistency and global view of the integration process. In this version of the document, the main focus is on the current and upcoming integration periods 2 (PM17-22) and 3 (PM23-25), while highlighting the aspects for period 4 (M26-34). This is considered a live document and thus it will be periodically updated with new details as the periods progress. The main focus of these periods is to continue integration for new platform functionalities, concretize UC scenarios and investigate how to implement them through the exploitation of the available COSMOS services and VE components. What is more, a relevant baseline tool has been identified (namely Nodered), and its usage in the context of COSMOS is investigated. This is a very important aspect since it will enable the ability to combine, test, store and reuse workflows that are created or needed in the context of an application scenario. Thus it will be necessary for component developers to provide these types of flows for external roles to utilize or combine their component functionalities. The investigation also of a front end portal to the COSMOS platform is performed, that may aid to further abstract configuration aspects and prove to be a guideline for COSMOS roles (e.g. application developers, VE developers etc.) during their interaction with the platform.

Furthermore, a traceability matrix has been created in order to map the offered COSMOS capabilities to the project UCs. The main goal is that at the end of the project each capability must have been applied to at least one of the scenarios. Thus the inclusion status can be monitored via this structure and provide valuable feedback or identify gaps in the process on which we will need to focus. 2 applications from this matrix have been identified at this stage as the main goal for Y2 integration, namely the Camden Heating Schedule and the Madrid Assisted Mobility scenarios.

Finally, a set of necessary processes has been defined, in terms of practical aspects of integration, such as documenting component dependencies, individual testing needs, relevant tools and software packaging (in terms of recommendations about the internal structure of

the software components and the creation of the software packages and related documentation), as part of an acceptance procedure at the component level, and included in the relevant Annexes. The contents of the previous version have been retained in this version in order to ensure a complete picture, a single point of information access and overall view.

1. Introduction

The aim of this document is to describe the strategy and plan of integration of the different parts developed within COSMOS, the included testbed and the process put in place to reach this goal. The integration and demonstration purposes of the project are both formally defined in WP7. The different project partners are expected to showcase the outcomes of the research and development WPs of COSMOS, verifying their applicability through the representative smart city Use Case scenarios and providing useful feedback about the COSMOS concepts and technologies.

After providing detailed definitions and design for the Use Cases, partners are expected to utilize the methodologies, frameworks and tools offered by COSMOS in order to realize the corresponding application scenarios. According to the scenarios analysis and definition, the use-cases will be implemented, the experiments will be prepared and the evaluation of the experimentation will follow in the context of this WP. In this process, it is of major importance to identify the intermediate steps that are necessary in order to progress development and integration between the major building blocks of COSMOS, in order to provide added value and functionality that can be used in the context of the defined Use Cases. Finally, the capabilities of the COSMOS technologies must be tested under real-life smart city conditions and be applied in a realistic context in order to verify the applicability of the implemented technology in different application domains.

To keep the consistency of all the developments and to ensure that all the components follow the same direction towards the overall solution, an integration plan has been defined and put in place, covering the coherence of the development activities inside the technical WPs and the integration and testing of the software components. Relevant subsystems and their according functionality have been defined, in order to gradually progress towards the final COSMOS platform prototype. This plan may also be used in order to prioritize work in the respective technical WPs but also identify the testing needs of the functionalities, based also on the roles that are envisioned to be included in their usage. Furthermore, applicability of the various functionalities against the defined UC scenarios is highlighted and ways of adapting the functionalities to the scenario needs are described. Frameworks for integration and for component functionality combination and abstraction are also described.

The document proceeds by describing the overall COSMOS vision and how this can be separated into smaller fragments for better manageability, but also identifying their relationship to the COSMOS Architecture and the involved components. Specific time periods are highlighted, along with their primary goals, and relevant points have been defined that are of specific interest for entities that wish to cooperate with the COSMOS platform itself. Furthermore, based on the requirements imposed by the applications and by the platform itself, an initial description of practical aspects is included (e.g. the testbed setup, component functional requirements, etc.).

Finally some recommendations in terms of software packaging, internal structure, installation, and execution of components, are given in Annex C. These guidelines give the set of rules that developers should follow to have a group of components with a similar structure and similar management commands.

1.1. Objectives

The overall objectives of this deliverable, as defined in the DoW, are the following:

- Describe the methodology and time plan followed to perform the integration activities
- Describe the integration process and planned activities
- Describe the tools that are used
- Describe the components that are integrated
- Describe the infrastructure that is used

1.2. Differences to previous version

With relation to the previous Y1 version, new information is mainly included in Chapter 4, extending the details of Integration Stage 2, including advancements in used frameworks such as Nodered, the refinement of the subgoals and their purpose and the creation of the application archetypes.

Chapter 5, regarding the details of Integration Stage 3, has been introduced in this version, including the details on the specific subsystems (many of which are continuations from previous periods including the points that are distributed in each period), along with an initial concretization of the selected application scenarios. Chapter 6 is also introduced for Integration Stage 4, including the investigation of a COSMOS Portal introduction and coarse grained investigation of the period goals. This is a living document, so details on this period will be further enhanced in the following months.

Section 7 has also been revised, as well as Annex A in order to include modifications in component requirements. The overall Gantt chart of the timeline in Section 2 is also updated with inclusion or modification of subgoals in terms of namings (to make them more clear) or duration, including the addition of two subgoals.

The contents of the previous version have been retained in this version in order to ensure a complete picture, a single point of information access and overall view.

2. Integration Strategy

2.1. COSMOS Overall Goal/Vision

COSMOS final goal is to give the ability for creating applications that can combine information coming from Smart City platforms with the COSMOS platform and VE side added value services in order to achieve a desired outcome (in terms of the actual application scenarios) for developers, authorities and citizens. During this process it engages a number of system capabilities that need to be integrated and combined in order to provide added value and then exposed to the various roles, whose interaction with the system should also be examined. The main difference with relation to the roles envisioned in the COSMOS Architecture is the specialization of the Application Developer role to 4 main subcategories, one generic for the COSMOS App Developer and three more specialized roles that can integrate with the platform only partially and with relation to a specific aspect. In that sense, an Analytics expert could be involved only in implementing a relevant data analytics component (in the form of Storlets), or a specific Domain's expert (e.g. mechanical engineer) could create a specific set of complex events rules in order to optimize a concrete function of the platform (e.g. bus service management and early malfunction identification).

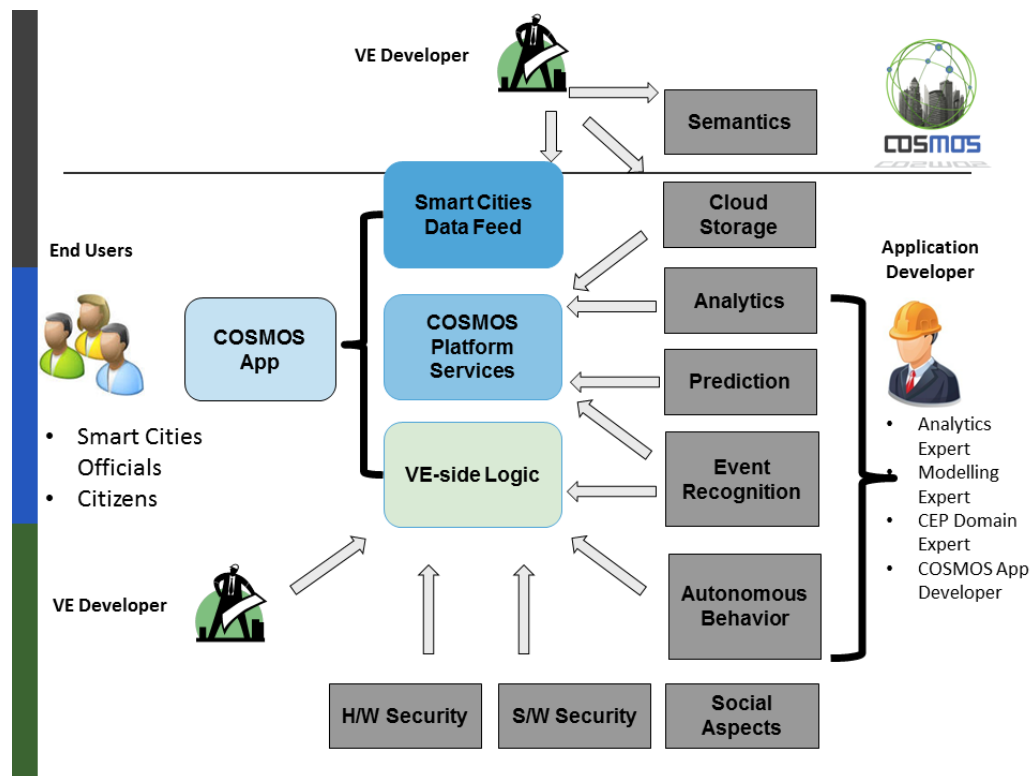


Figure 1: COSMOS Vision. Functionalities and Roles

In order to drive this effort, a suitable integration strategy needs to be in place. The integration strategy of COSMOS revolves around a number of high level **integration goals**, that are progressive steps towards the aforementioned final goal, bringing the main parts of the overall system closer at each step. These high level goals may span across different **integration**

periods within the project, that have been defined based on components delivery and key milestones of the project. In each period, the high level goals become more fine-grained on **subgoals**, regarding a specific aspect. These subgoals include the integration of one or more **subsystems** in order to be tested and provide the envisioned capability. These subsystems are intended to be completed within the specific period and are the main unit of integration and testing from the system capabilities point of view. For these cases also, relevant **integration points** must be identified. These are specifically the points of involvement of “external” entities/roles, for which specific testing processes should be described (In D7.7.1[13] Integration of results) for their incorporation, in order to cover the second aforementioned integration target. The integration points (IPs) have been separated into 4 major categories:

- **IP1:** It implies end user involvement, which should be accompanied by respective Graphical User Interfaces
- **IP2:** It implies Application Developer involvement, which should be accompanied with the definition of a relevant process and format
- **IP3:** it implies VE Developer involvement, in terms of endpoints definition and format
- **IP4:** it implies VE Developer involvement, in terms of definition of a description template and instance creation

The overall process appears in Figure 2.

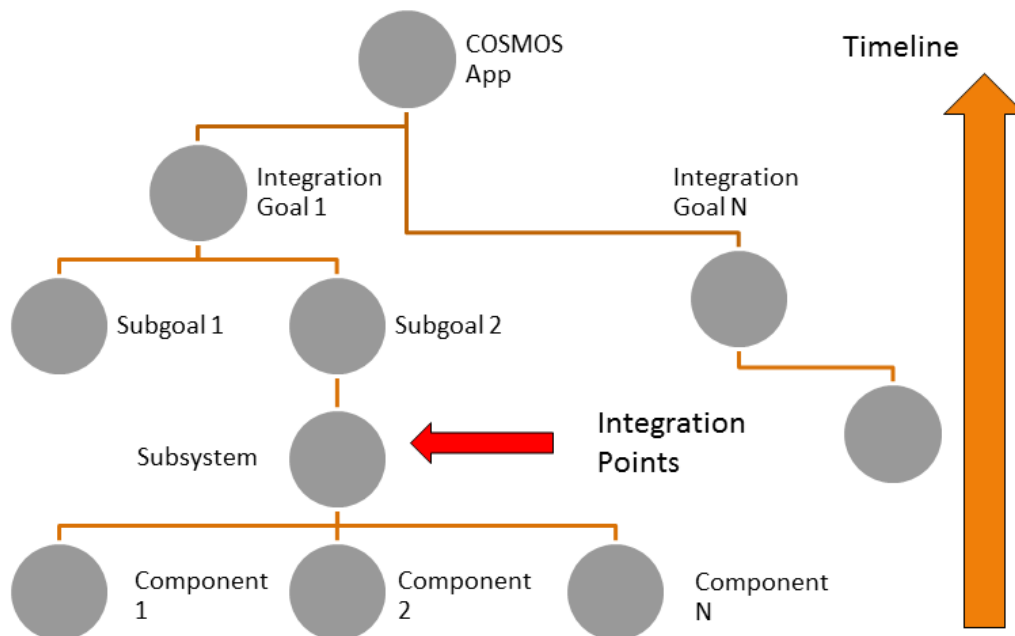


Figure 2: COSMOS Integration Strategy Overview

In the following paragraphs, more details are provided for the identified goals along with their mapping on integration periods.

2.2. Integration Goals

2.2.1. COSMOS Platform and cross-component integration

The first integration goal consists of the COSMOS platform being the central point of integration of the various components and how the latter can be combined in order to produce increased functionality and added value (e.g. data ingestion,). This includes the way other existing platforms (like VEPROT and Camden EnergyHive) could interconnect with the main COSMOS platform in order for the latter to have access to the specific data. It includes also the manner through which a COSMOS component developer would integrate their respective components (e.g. new prediction model, new analytics Storlet, new data annotation etc.).

Especially for the Uce Case platform data feeds, these can be separated in two major points:

- The endpoints and interfaces through which this information may be obtained by the COSMOS platform
- The data format and semantics (in terms of fields annotations and meaning)

2.2.2. Data Model/Template

The second integration goal refers to the definition of a common Data Model and Template as an agreement between the COSMOS UCs and the COSMOS Platform. This template will contain the amount and type of information provided by the UCs and adapted to a format structure that is understandable and usable by the various components and applications. This does not refer to the information representation standard to be used, since this has been agreed from an early stage of the project that it will be based on JSON.

2.2.3. VE description and linking in the COSMOS Platform

This goal is responsible for integrating Smart City elements in the COSMOS platform, in terms of their semantics, their description of capabilities and functionalities. To this end, relevant tools available by the platform may be used by the role of VE Developer, in order to create description templates of the various physical entities (e.g. buses, flats, traffic lights) and to instantiate them based on the actual available physical objects. This includes also the necessary interfaces that need to be implemented in order for the data streams from these objects to reach the COSMOS components.

2.2.4. VE-side COSMOS components integration

According to the COSMOS architecture, a number of components are expected to be deployed on the VE side. For these components a suitable integration must be in place, taking under consideration the UC platform side, in order for a VE-side COSMOS component to be incorporated and cooperate in the context of the defined scenario. Furthermore, issues of communication between these components or with the COSMOS platform need to be investigated and adapted.

2.2.5. Application definition, creation and deployment

Application creation may include one or more features from the described capabilities of the platform in Chapter 2.1, depending on the stage of the project. The role of Application Developer is needed in this case, an entity that will be responsible for combining information, services and logic from multiple sources in order to provide added value in the form of an application. The definition framework of such an application is a goal of integration, along with the creation of the necessary support structures by the platform (e.g. creating a data channel that combines the application-defined sources of information). These applications are initially defined by the UCs. Initially limited functionality is foreseen, not including all aspects of involvement. This inclusion is expected to be completed during the following periods of the project. However, the ability to have the framework for at least defining preliminary applications should exist at the end of Y2.

2.3. Integration Stages Definition

The integration periods that can be defined at this stage may be separated to the following intervals:

- M10-M16: Following the release of the initial software prototypes, platform component integration may kick in, in order to enable advanced capabilities and initial platform features. This corresponds to Milestone MS6 of the project. Initial discussions are needed also in terms of necessary metadata annotations.
- M17-M22: For this period, background work in data model agreement and necessary adaptations that will drive Y2 component development is foreseen, along with extensions of the platform design and initial VE integration (VE descriptions etc.) from the UCs. This does not include data format aspects, since these have been defined early in the project to be JSON-compatible. M22 corresponds to Milestone MS8 of the project.
- M23-M25: For this period, and following the release of Y2 components, the goal is to have the platform available, along with a set of elementary applications and VE integration (in terms of described VEs, getting info etc.). This will be completed one month prior to Milestone MS9, in order to receive feedback on the results.
- M26-M34: For this period, the final point of VE-side components integration is expected to be completed, thus leading to the ability to create full-blown applications covering the entire range of COSMOS functionalities (combining VE data, VE side actions, COSMOS services and generic smart city data).
- M35-M36: Final version of the applications creation, exploiting the results of the previous period.

2.4. Integration Timeline in a nutshell

The integration timeline incorporating the goals and subgoals identified so far is included in Figure 3. This includes also tasks that have been identified for the upcoming periods and for which we will extend their description in the next iterations of this document. As mentioned previously, the main target is for:

- End of Y1 (M16) to have advanced platform capabilities and their definition
- End of Y2 (M25) to have extended platform capabilities and elementary applications that utilize COSMOS services and Smart City data
- End of Y3 (M36) to have completely incorporated the VE side components and enable full blown COSMOS applications

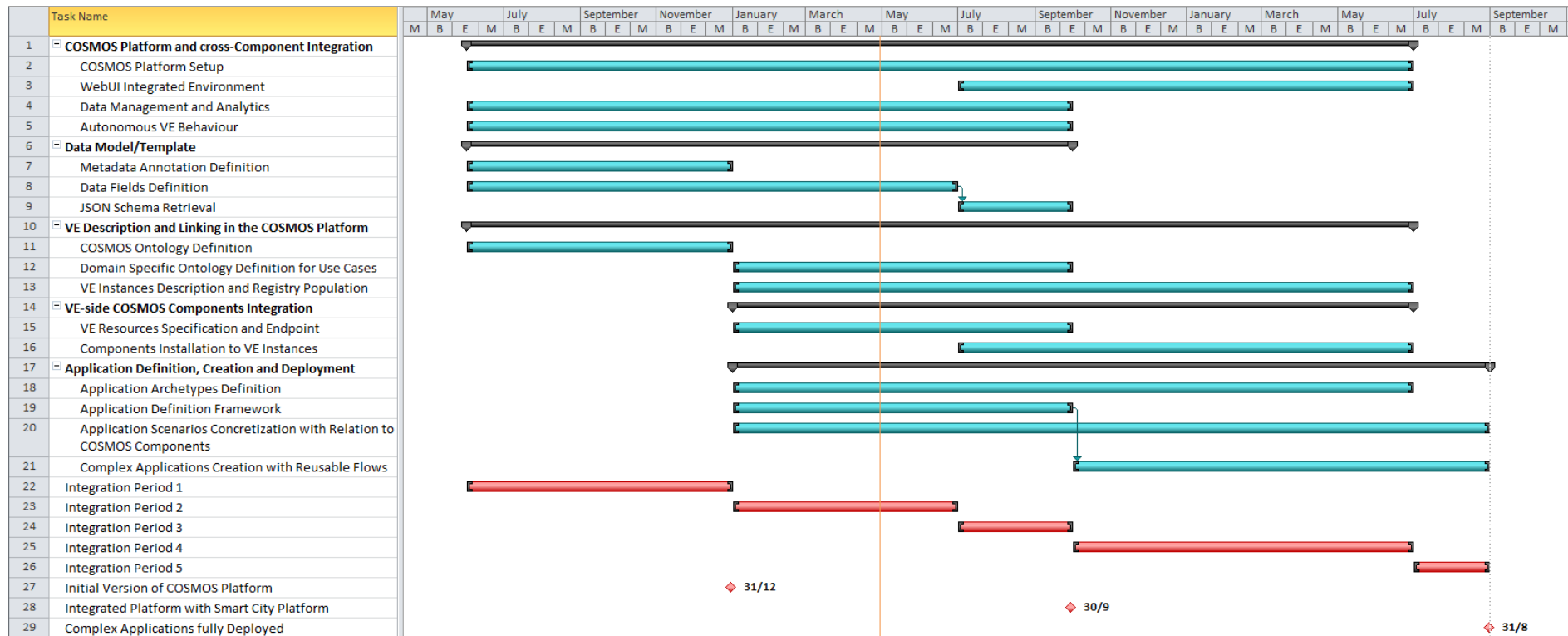


Figure 3: COSMOS Integration Timeline Gantt Chart

2.5. Integration Stages Template Fields

Each integration stage description needs to incorporate a set of fields that will drive integration actions and will guide/prioritize the overall project (including the technical WPs) towards the fulfilment of the necessary steps. In the following subsections, an analysis of these template fields and their necessary information is portrayed.

2.5.1. Involved Integration Goals

This section includes the high level integration goals that are partially or entirely completed within this stage. These goals may span across multiple periods, and can be further refined in concrete aspects (subgoals), that usually refer to a system capability.

2.5.1.1 Subgoals

The subgoals, which refer to a concrete capability of the system, in relation also to the architecturally defined ones in the relevant documentation, are based on one or more subsystems (collections of components).

2.5.2. Subsystems and integration points involved

In order for a subgoal to be completed, a number of components must cooperate in order to provide the overall functionality or variations of it, thus forming a relevant subsystem. The overall functionality of the subsystem needs to be tested through the defined scenarios that also involve the expected inputs from the Use Cases of the project. Integration points, as mentioned in the strategy, must be identified at this part.

2.5.2.1 Components involved

In this section of each period, the components to be included in the subsystems examined in the specific period should be identified. Before the components are included in the platform, they must be tested in order to verify their behaviour. After passing this verification process, that checks components behavior, they may be included in the platform. Furthermore their functional requirements must be documented (in terms of OS etc.). Given that some components may be included in more than one subsystems or periods and in order not to repeat information, these requirements have been included in a relevant Annex (Annex A) and cross-referenced from that location. The testing of these components (based on a template Test Case table that appears in Table 1) will be included in the results deliverable (D7.7.X) in a similar manner (cross-referenced from a respective Annex).

Table 1: Test Case Template

Test Case Number Version	Ordinal of the test case, no special numbering policy is enforced, component name should precede
Test Case Title	Short name that describes the test case.
Module tested	Name of the module to be tested.

Requirements addressed	To which requirements the module functionality can be mapped.
Initial conditions	Initial conditions that we need to set before starting the test.
Expected results	List of the results that are expected when the test is run.
Owner	Person responsible for the test case.
Steps	List of the exact steps that the owner must follow to perform the test case.
Passed	“Yes” if the test has passed, “No” if it has failed.
Bug ID	Bug ID, if the test has failed and a bug report was opened. Naming convention should include Test Case Number Version and Bug Number.
Problems	Any problems encountered while running the tests.
Required changes	Any suggested changes to the test or the module tested.

2.5.3. Use Case specific aspects involved and concretization

The purpose of this section is to highlight at this stage what is the expected concrete usage of a COSMOS service/component in the context of a specific UC. Examples of this may include concrete CEP rules definition, concrete models and predictions and cases for the CBR approach, specific data feed adaptations and annotations.

2.5.4. Testing environment

2.5.4.1 Testing infrastructure

In this section of the periods, the infrastructure that is needed in order to proceed with the deployment and usage of the components may be included. This may include internal project testbed that will emulate the COSMOS platform, usage of external services or elements of the UC infrastructure, client side components, h/w devices etc. Again, given that this information may be repeated, a relevant Annex (Annex B) has been defined.

2.5.4.2 Involved Tester roles

COSMOS ecosystem is comprised of a set of functionalities, features or capabilities that are expected to be used by different roles/actors involved. A mapping may be performed between the various COSMOS features and these roles that are intended to be engaged. The latter should be also the ones that test the offered services and report back with their feedback. Thus in each period the suitable roles need to be identified and suitable feedback mechanisms need to be in place. Potentially not all roles will be completely enabled by the end of the project since in many cases this would imply an increased level of abstraction (e.g. GUIs) that may not be feasible to create in the project lifetime or resources. For these cases, the project’s technical partners are expected to act as mediators for the Use Cases to adapt to their specific aspects and needs.

Examples of fully flexible roles may include Domain Experts (e.g. a mechanical engineer, either external or from EMT) to provide their experience and expertise to optimize e.g. bus maintenance, potentially through the definition of proper CEP rules (e.g. if ABS is activated more than 3 times in a non-humid day -> check tire conditions). Another example would be of a Domain Expert on modelling, creating prediction models from the available information and using the COSMOS services (e.g. storage and Storlets) to create an algorithm for training etc.

Roles are identified also with relation to specific subsystems and the defined integration points. Thus this information may also be included in the Subsystems relevant sections.

2.5.5. Deviations from Plan

For each period, and based on the anticipated inputs and results, a number of deviations may be identified. This information is expected to be included in the D7.7.1 “Integration of Results” in a respective section.

2.6. Generic Considerations

2.6.1. Software Packaging

Software packaging needs to follow a number of conventions in order to have a uniform nature. More information on this is provided in Annex C and is relevant to all the software artefacts produced.

2.6.2. Helper Tools

A set of tools may aid in the integration of the components. Details on these tools are included in Annex C.

Following, a list of the upcoming integration Stages, as identified at this moment in time, is included. For the initial stages more concrete information can be defined. For the future periods their details will be more fine grained in the next iterations of this document, following its application in the context of the project and feedback.

3. Integration Stage 1 (M10-M16)

3.1. Involved Integration Goals and Refinement to Subgoals

In this first Integration Stage, the main focus is on the first Integration Goal, the COSMOS Platform and cross-component integration, that spans across Y1 and Y2 of the project. For Integration Period 1, this can be refined to the subgoals of “Data Management and Analytics”, “COSMOS Platform Setup” and “Autonomous VE Behaviour”.

Data management refers to the receipt and ingestion of data feeds from the UC platforms, their subsequent annotation with adaptable tags and their grouping as storage objects. Analytics refers to the creation of relevant Storlets, which are computational components that are executed near the actual data, and which are used for any specific need of data manipulation.

Autonomous VE Behaviour refers to the initial prototype of the VE-side component functionality, resulting in a suitable definition of a Case Based Reasoning approach for problem solving, that is enhanced with social aspects in order to share experiences and solutions. This does not include the actual deployment on the VE side, since this is included in future goals (VE-side COSMOS components integration).

These subgoals are using the following subsystems:

- Data Management and Analytics (Subgoal)
 - Data Feed, Annotation and Storage (Subsystem)
 - Storage and Analytics which can be divided into
 - Metadata Search Storlet (Subsystem)
 - Modelling and Storage Analytics (Subsystem)
 - Security, Privacy and Storage with Analytics (Subsystem)
- Autonomous VE Behaviour (Subgoal)
 - Autonomous Behaviour with minimal integration to the platform (Subsystem)
 - Autonomous Behaviour with Platform involvement and automated event detection (Subsystem)

Information with relation to the subsystems is included in the following paragraphs.

COSMOS Platform Setup refers to the way the current version of the components needs to be installed and configured, so that a running instance of a COSMOS Platform provider is enabled, and it is more of a practical nature. Thus details on this task are included in Section 3.4.1.

Furthermore, based on the presented Gantt Chart in Figure 3, other subgoals that are activated are:

- Metadata Annotation and Definition (included in the Data Model Goal)
- Data Fields definition (included in the Data Model Goal)
- COSMOS Ontology Definition (included in the VE Description and Linking Goal)

Given that the Annotations and Fields topics are highly related to the Data Management aspects, they have been incorporated in the respective subgoal in terms of how they were applied in Y1. For the COSMOS Ontology definition, given that it is the first step towards an overall description of the VEs, the outcomes will be reported in the following integration periods of the project, along with the future steps that will further concretize the approach.

3.2. Subsystems involved

With relation to the main architectural diagram, depicted in Figure 4, the subsystems that have been identified above can be highlighted. For each case, relevant integration points (of the respective category defined in Section 2.1) are highlighted, for which a specific process should be realized in D7.7.1 (Integration of Results) that would enable the testing with regard to this feature.

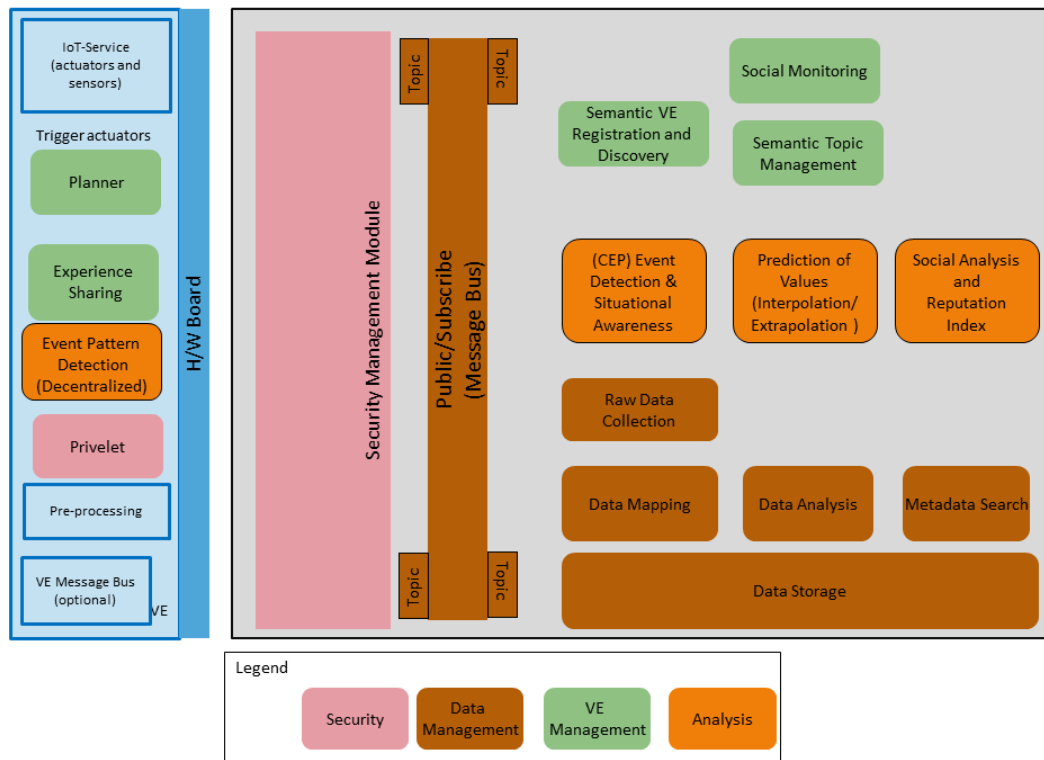


Figure 4: Overall COSMOS Architecture

3.2.1. Data Feed, Annotation and Storage

The components involved in this subsystem appear in Figure 5. The main flow includes the data source (real-time data coming from the UCs, and more specifically the Camden platform) that is adapted to the format and protocol of the COSMOS platform (i.e. the format accepted by the Message Bus component, IP3). The format to be used in this case is JSON. The VE developer must also specify (in terms of a relevant configuration file), what are the annotations (metadata tags) to be associated with the individual fields of information in the data feed (IP3). This information is collected by the Raw Data Collector, grouped into objects (along with the actual data) by the Data Mapper and stored in the backend storage system. From there it can be retrieved based on the metadata tags using relevant constraints.

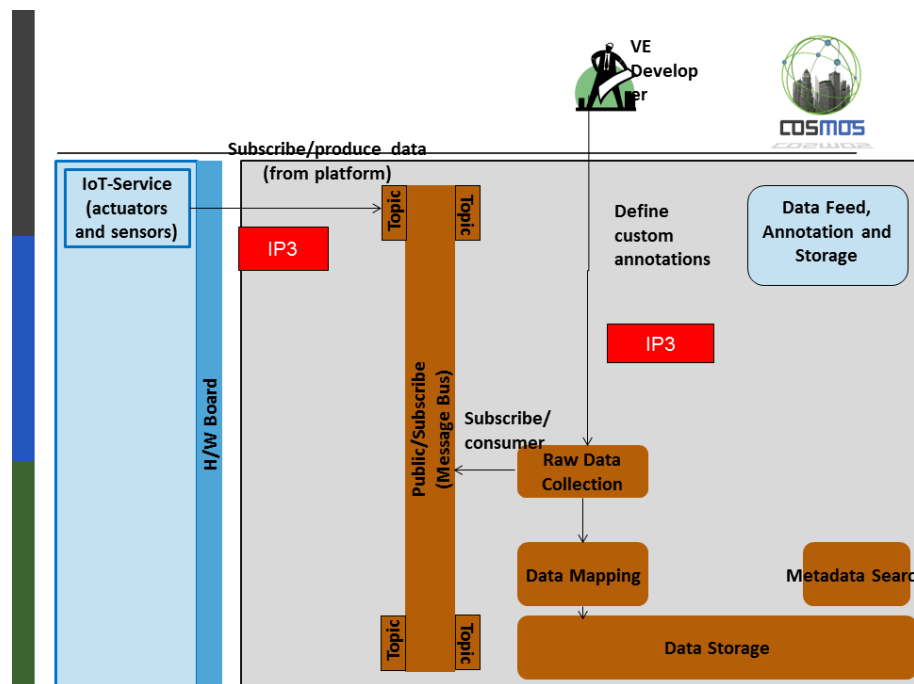


Figure 5: Data Feed, Annotation and Storage Subsystem

3.2.2. Metadata Search and Storlets

The components involved in this subsystem appear in Figure 6. The main flow includes the data source (historical or static data coming from the UCs, and more specifically the Madrid static Bus Routes definitions) that is adapted to the format of the COSMOS platform. The format to be used in this case is JSON. The VE developer must again specify (in terms of a relevant configuration file- IP3), what are the annotations (metadata tags) to be associated with the individual fields of information in the data feed (this action is omitted since it is included also in the previous subsystem). Furthermore, an Application Developer, with the specialization of Analytics Expert, is responsible for creating and integrating a Storlet script (IP2) in order to perform a set of specific computation actions (such as geolocation conversion and metadata enrichment) on the ingested data. This functionality enables the End Users to search for bus routes that include stops in a given geographic box (IP1).

3.2.3. Modelling and Storage Analytics

The subsystem and roles for this case appears in Figure 7. The main flow includes the data source (historical data coming from a smart building in Surrey). The specific source was selected since it included the features that were needed by the respective model of occupancy detection, since the main purpose of this subsystem is to integrate the process and flow of model creation. In the upcoming periods, models more adapted to the UC needs will be pursued. This subsystem includes the incorporation of an Application Developer (specialized as an Analytics Expert) in order to create a relevant Storlet for data preprocessing (IP2), based on the needs of the other Application Developer (specialized as a Modelling Expert- IP2), e.g. if the latter needs average values from the raw data or other forms of preprocessing (outlier detection etc.). The modelling Expert may also include a relevant model creation and training algorithm, through the usage of Apache Spark. Domain experts will need to be in close communication and perform collaborating coding. In the future there may be certain libraries COSMOS offers with a number of ready-made Storlets/modelling options which can sometimes avoid the need for coding.

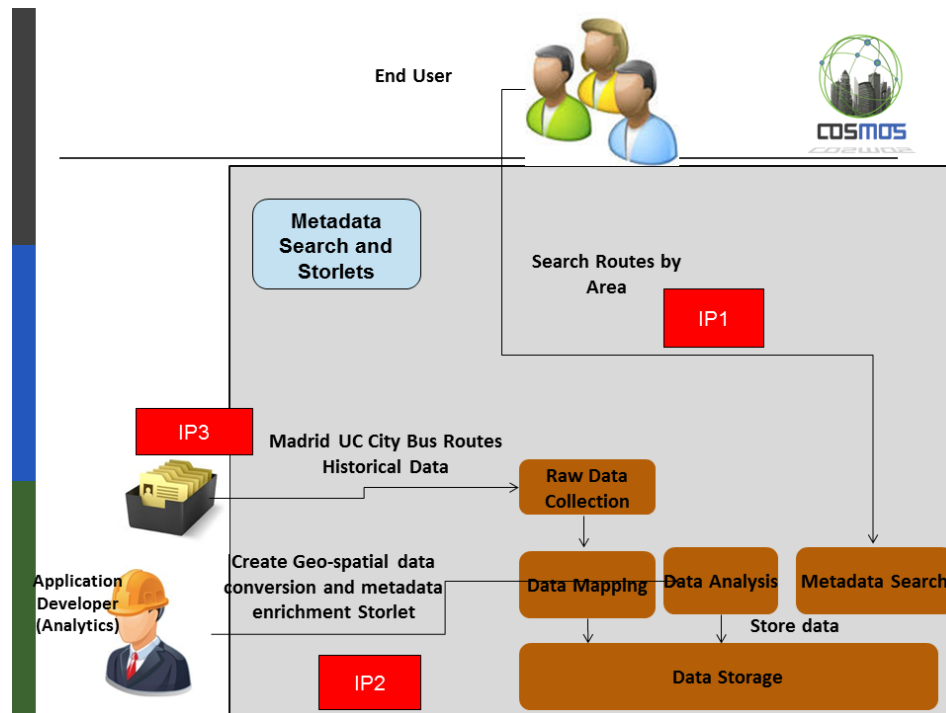


Figure 6: Metadata Search and Storlets Subsystem

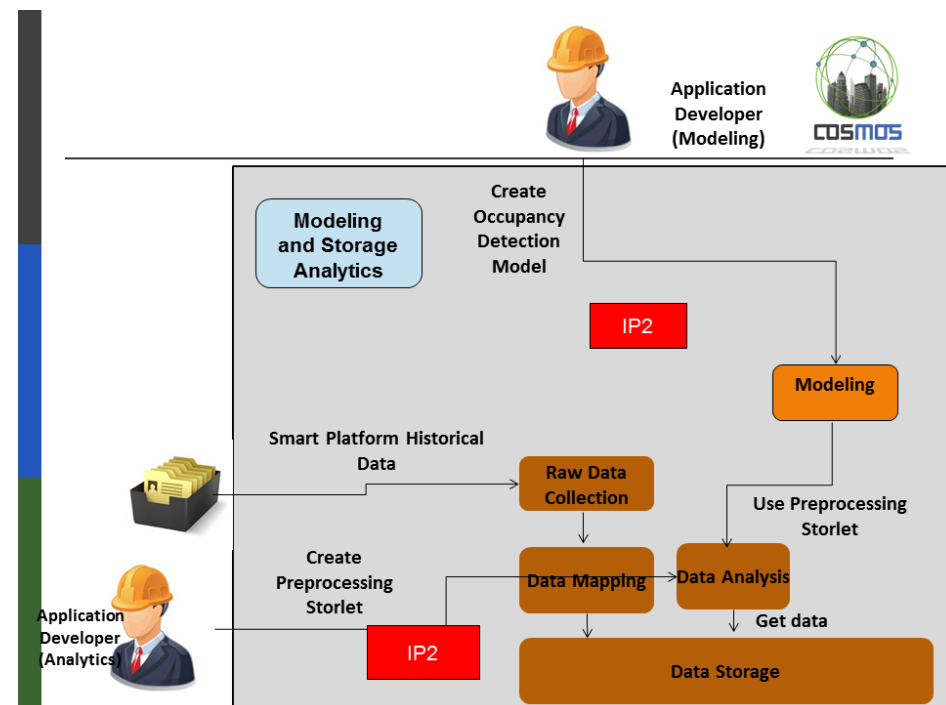


Figure 7: Modelling and Storage Analytics Subsystem

3.2.4. Security, Privacy and Storage

The subsystem and roles for this case appears in Figure 8. The main flow includes the data feed (real-time data coming from a camera) that is ingested securely in the COSMOS storage, following a hardware-enabled preprocessing step at the source (IP3), that is responsible for compressing and encrypting the image before its transmission. This subsystem includes the incorporation of an Application Developer (specialized as an Analytics Expert) in order to create a relevant Storlet for data privacy processing (IP2), which is in charge of enabling external users (End Users role included) retrieval on the images (IP1), but based on their access rights. Thus the Storlet may allow or deny access to the image, or retrieve it and blur the faces of the frame, based on the authorization level of the End User. The incorporation of real data from the UCs (especially the Madrid UC that includes on board bus cameras) is expected to be performed in the future integration periods.

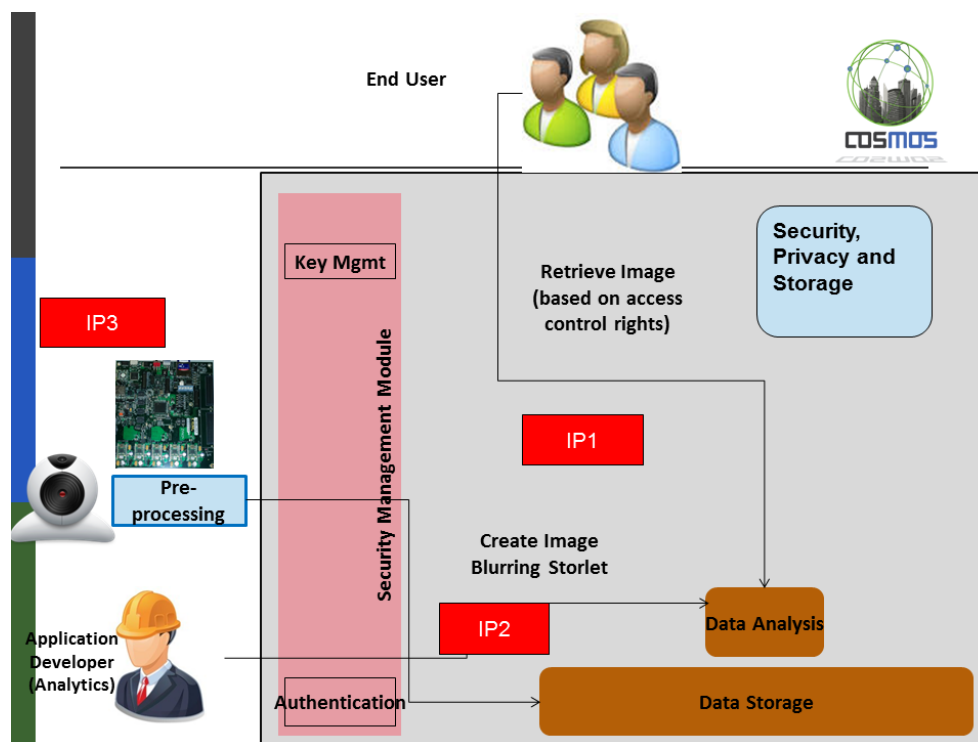


Figure 8: Security, Privacy and Storage subsystem

3.2.5. Autonomous Behavior of VEs with minimum integration with the Platform

The subsystem and roles for this case appears in Figure 9. The main flow in this case includes the instantiation of the Planner component by an Application Developer role (IP2), through the definition of the kind of problem the component is intended to solve (case-problem structure definition). Thus it gives the capabilities to the End Users of the application to automate management aspects, through setting the problem parameters (e.g. arrival at one hour and desired flat temperature of 25°C- IP1). The solutions may come either from the planner's local Case Base or through interaction with other similar VEs. In that case the Experience Sharing kicks in, in order to find friend VEs that have dealt with similar situations in the past, thus instructing it on the necessary solution to follow. The solution is rated in the end in terms of its effectiveness, information that is kept in the social network.

3.2.6. Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

The subsystem and roles for this case appears in Figure 10. The main flow in this case includes the same flow as previously, however the platform capabilities (in terms of CEP or storage) are also enabled. Thus new functionalities can be achieved (e.g. logging of historical data, event recognition and alert etc.). The data now pass through the COSMOS platform, thus including the VE developer to adapt the relevant flows (IP3).

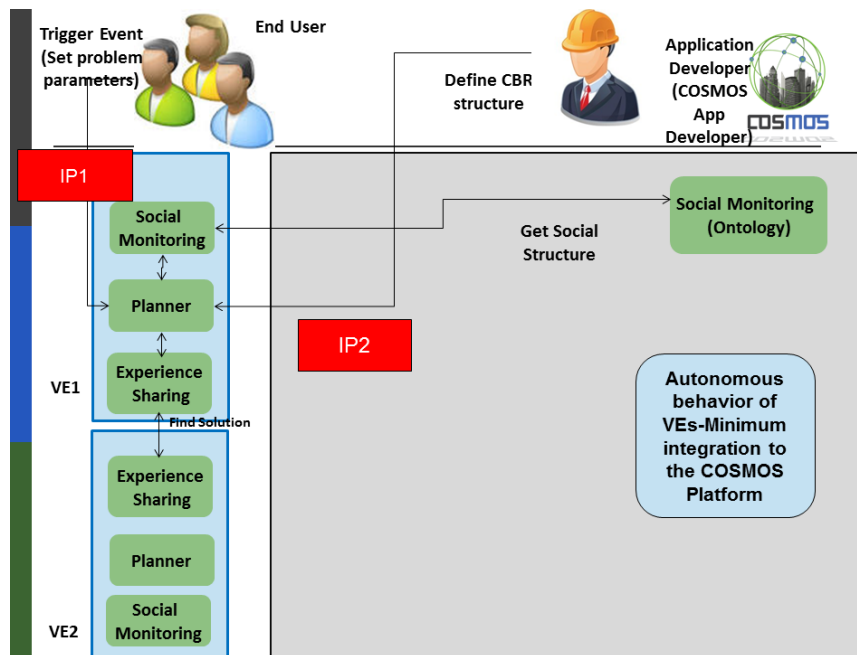


Figure 9: Autonomous Behavior of VEs with minimum platform integration subsystem

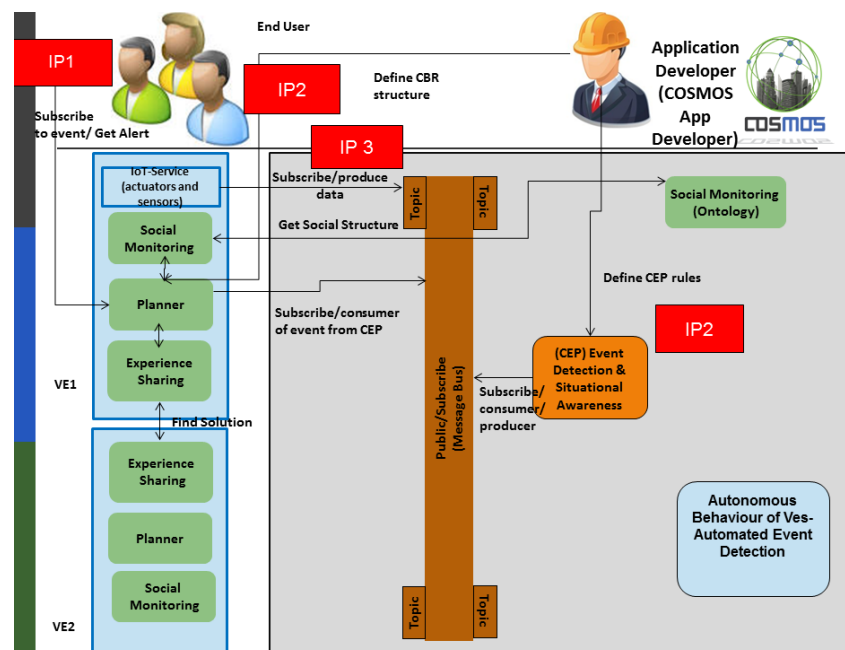


Figure 10: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

3.2.7. Components involved

The components involved based on the previous subsystems and their functional requirements, as these can be identified by the current development process, have been included in Annex A. We do not include them here in detail, since these components may appear in more integration periods and in order not to repeat information we cross-reference them directly from the respective Annex.

3.3. Overview of Use Case specific aspects involved and concretization

For this integration period, UCs are expected to be incorporated in the following manner:

- Data coming from the platforms and adapted to the COSMOS needs for ingestion (Data Feed, Annotation and Storage subsystem). Data may be ingested either through real-time feeds (Camden UC) or through historical file-based formats (Madrid UC), annotated accordingly and stored to COSMOS storage structures. More information on this is provided in Sections 3.3.1 and 3.3.2.
- Analytics capabilities based on the annotated data, mainly in the form of metadata search and filtering (Madrid UC), in the form of searching for bus routes based on area highlighting. This is mainly tested through the Metadata, Search and Storlets subsystem.
- The image blurring and parametric image retrieval application is expected to be used in the case of Madrid, in the context of a wider application scenario and is included in the Security, Privacy and Storage subsystem.
- Data pre-processing Storlets are also expected to be used in the context of UC-tailored prediction models, included in the Modelling and Storage subsystem.
- For the Camden UC, the applied aspects include a concretization of the Autonomous Behaviour of VEs subsystems, in the following manner:

In smart home environment, total energy consumption is measured in real time with the help of smart meters.

1. Every flat is modelled as a VE. Every flat contains a thermometer (sensor) and a boiler whose temperature can be measured (sensor), as well as set (actuator).
2. During the creation of an application, COSMOS offers the developer the opportunity to define how cases relevant to his application are going to be stored, created and maintained.
3. A user downloads an application for a VE. This application defines how the several cases (both complete and incomplete) are going to be created. For example, the flat VE continuously records the actions of a human user regarding the heating of the flat. In our case, it may record that at a room with a temperature of 6 °C, the user turned the heating on for a total of 10 minutes (600 seconds) and stopped at 20 °C, using hot water of 70 °C.
4. That way, the VE builds a case of {6, 600, 20, 70} and, in a similar, fashion its case base.
5. COSMOS manages the CBR cycle.
6. Each time a new incomplete case is created, the VE searches its Case Base for a solution. For example, the user may inform the VE that he/she will return after a specific time interval and request a certain target temperature in that certain time limit.
7. If no solution is detected, the VE initiates the Experience Sharing service and asks its friends for help.

8. A number of cases is returned by the friends.
9. Based on the dependability index of the friends and the similarity of the cases (the weights of these criteria can be defined), the cases are sorted and the best case is chosen.

The case is evaluated based on its results and the Trust of the friend that shared its case is recalculated. The extensive details on the available data sets and interfaces for the UCs is given in D7.1.1 [10] However we include here a short overview with relation to the concrete aspects used during this integration period.

3.3.1. Camden UC Data Feed

Endpoint

Camden flats send their data through Mosquitto (MQTT) which is a lightweight publish/subscribe messaging protocol. In order to inject them in the COSMOS platform through the Message Bus, Camden UC provided us with an endpoint, credentials and a relevant topic.

Data Format

The data are structured in JSON format, which is the one adopted in the COSMOS platform.

The Figure 11 shows an example of these data:

```
{
  "estate": "Dalehead",
  "hid": "cPGKKhiI4q6Y",
  "ts": "1405578854",
  "instant": "226",
  "returnTemp": "72.3",
  "flowTemp": "72.4",
  "flowRate": "742",
  "cumulative": "15703"
}
```

Figure 11: Camden UC data

“estate” refers to one of three 21-storey tower blocks (Dalehead, Gillfoot and Oxenholme), all located within the boundaries of the London Borough of Camden. “hid” corresponds to one single flat and is unique within the heating system. “ts” indicates the timestamp of the specific observation, written in Epoch (Unix) format. “returnTemp” and “flowTemp” keys refer to the temperature of the heating water, whereas “cumulative” represents the cumulative energy consumption of the flat and is used for charging.

3.3.2. Madrid UC Data Feed

EMT data were provided through files containing historical data for bus trips. Each excel file contained data for a particular bus line during a particular time period. The data were injected to the COSMOS platform through the Data Mapper component.

An example of a bus trip data is depicted in Figure 12:

LINE	STOPID	NAME	ROUTE	METERS	POSX	POSY
3	1885	PUERTA DE TOLEDO	1	0	439735	4473313
3	1884	GRAN VIA SAN FRANCISCO-PTA.TOLEDO	1	172	439686	4473457
3	1882	GRAN VIA SAN FRANCISCO-AGUILA	1	348	439597,3	4473606
3	1880	PZA.SAN FRANCISCO-BAILEN	1	571	439554	4473816
3	1878	BAILEN-YESEROS	1	713	439553	4473945
3	1876	BAILEN-MAYOR	1	1009	439583	4474239
3	1886	MAYOR-PZA.DE LA VILLA	1	1348	439866	4474332
3	1887	MAYOR Nº 21	1	1653	440159,8	4474406
3	1888	PTA.DEL SOL-CARRETAS	1	1991	440493,1	4474453
3	2004	CEDACEROS-ZORRILLA	1	2400	440863,8	4474489
3	4108	GRAN VIA-HORTALEZA	1	2817	440735,2	4474810
3	5376	HORTALEZA-INFANTAS	1	3087	440697	4474949
3	278	HORTALEZA-GRAVINA	1	3361	440839	4475208
3	5639	PZA.DE SANTA BARBARA	1	3737	441009,8	4475514

Figure 12: Madrid UC data

“STOPID” and “NAME” keys both refer to a specific bus stop, while “METERS” indicates the whole distance covered by the bus from the beginning of its trip. “POSX” and “POSY” correspond to the UTM coordinates of the relevant bus stop.

3.4. Testing environment

3.4.1. Testing infrastructure - The COSMOS Platform Setup

The testing infrastructure to be used comprises of an internal testbed, for the components to be deployed. This emulates the role of a COSMOS Platform Provider. In order to create this facility, a number of steps must be performed (Figure 13):

- Creation of a set of virtual appliances (Virtual Machines) that will be the environment for the platform components to run. Given that components may have different dependencies (e.g. operating systems, java versions etc.), in order to have functional separation we used this approach for the isolation of deployed components. These dependencies led to the mapping between components and VMs that needs to be taken under consideration for the determination of the VM types to be created. The virtualization approach is also helpful in case the COSMOS Platform provider is based on a Cloud (private or public) implementation. Details on the COSMOS testbed are included in Annex B, again following the logic that it may be used in multiple integration periods and thus it should not be included in the individual period description. Furthermore it will be easier to update this information based on new component deployments or added requirements. The description includes component grouping in these VMs, needed open ports and virtualization dependencies.
- Deployment of the created VMs on a respective hardware platform.

- VM configuration. Based on the mapping of components to VMs, the necessary dependencies (e.g. libraries etc.) need to be installed, along with the configuration of the network ports. Component requirements and software dependencies are described in Annexes A and B
- Component installation. Finally, the respective components need to be installed and configured in the VMs. This information is provided in great detail in deliverables DX.2.1 (where $X\{3,4,5,6\}$), the project's technical WPs prototype documentation.

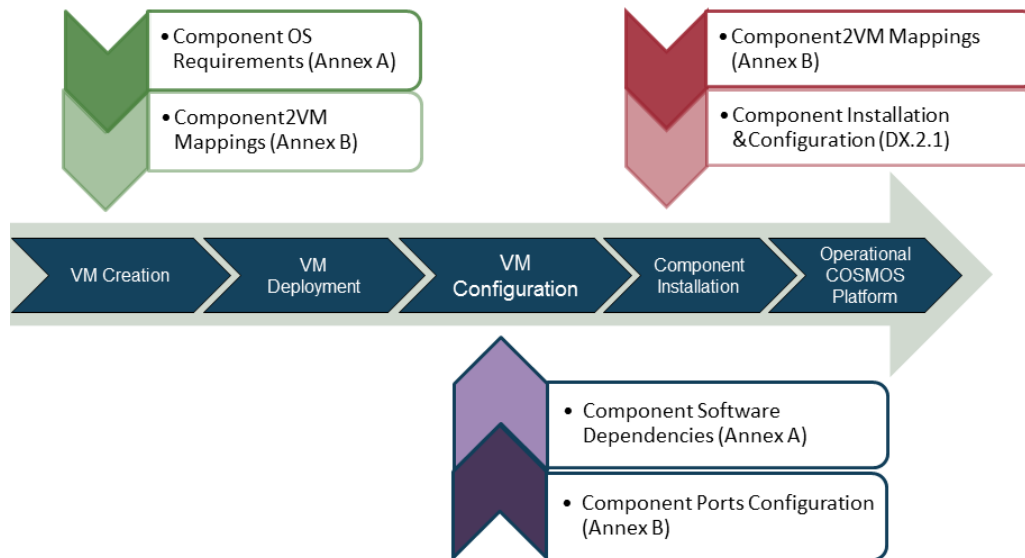


Figure 13: COSMOS Platform Setup Process

It is expected that in the end of the project, the virtual appliances created by the COSMOS project will be made available as an alternative and more flexible means of platform installation.

3.4.2. Involved Tester roles

During this period, the main roles involved are the COSMOS Component Developers, that aim to integrate the various components in cooperating entities. These testers will also emulate the role of the other entities (e.g. VE developer, App Developer, End user, Domain Expert) that have been highlighted in the defined subsystems.

4. Integration Stage 2 (M17-M22)

4.1. Involved Integration Goals and Refinement

Following the results of the first integration period and the initial deployment and definition of the COSMOS platform, the second stage of integration aims at further enhancing the linking in the platform and better incorporation of the UC elements. To this end, and based on the Gantt Chart in Figure 3, one of the tasks of this period include the definition of the data model to be used across the platform and in cooperation with the component needs and the UC data feeds. This implies mainly the identification of component combinations (not necessarily in the form of subsystems), from which the information flow will pass.

The VE Description and Linking to the platform is one of the main goals of this period, which aims at resulting in having semantically enriched VEs in the COSMOS platform, together with their annotations and endpoints description. To this end, it will include the following subgoals:

- COSMOS Ontology Definition
 - This includes mainly the different ways through which UC data can be directed to the COSMOS platform. Thus the ontology definition should be generic enough to cover the different means of data acquisition (e.g. through REST interfaces, through appropriate topics in the Message Bus etc.) and have the capabilities at the same time to be instantiated based on the available interfaces per case.
- Domain Specific Ontology Definitions for Use Cases
 - Given that the COSMOS ontology should be abstract enough to describe endpoints, the semantics of these endpoints are linked with Domain Specific ontologies, in order to indicate the type of information contained in a specific endpoint. Thus relevant definitions must exist for each UC, adapting to its individual needs and enabling the mapping of the endpoint to the concept.
- VE Instances Descriptions and Registry population (subsystem)
 - Based on the two previous steps, concrete VE descriptions should be made available in the COSMOS Registry, in order to describe the UC elements and be used in the following goal.

Furthermore, another critical aspect that integration will center around is the initial version of the application scenarios, which implies work towards the following subgoals:

- Application Definition Framework
 - This relates mainly to the sources of data and how these can be combined in the context of a specific application, how they can be discovered by an Application Developer and included (the endpoints), in the application logic (and based on what the latter needs to achieve).
- Application Archetypes Definition (Subsystem combination)
 - This relates mainly to high level ways of combining subsystems, as these have been defined in D2.3.2[14], in order to create conceptual template applications that may be reused in similar scenarios. In order to have a more integrated approach and include the COSMOS UCs in the loop, a number of the scenarios identified by the UC partners in D7.1.1 and D7.1.2 have been taken under consideration and have been abstracted in this process.

Finally, the third point of attention will be the initialization of the discussion on the overall integration process at the VE side, which includes the following:

- VE resources specification and endpoint
 - This refers to the VE side components of COSMOS and how these can be deployed on the actual UC testbeds. Component deployment should take under consideration specification of the available resources, and this information should be also propagated to the development WPs.

Also the initial integration goal of “COSMOS Platform and Cross Component integration” is expected to carry on, but mainly in order to incorporate any needed changes and new advancements. However testing of these capabilities will be performed in the following period, according to the availability of the prototypes delivered in M22.

4.2. Subsystems/Subgoals involved

This section will be further populated in the next iterations of the document, following the update on the COSMOS architecture. For the moment we can identify two subsystems that can be the target of integration in this period.

4.2.1. VE Instances Descriptions and Registry population

The subsystem and roles for this case appear in Figure 14. The main involved entity is the VE developer, who needs to provide the concrete VE descriptions, based on the provided ontologies. This corresponds to integration point IP4, and includes the definition of the templates and then the population of the registry with existing entities (with their endpoints and semantics).

4.2.1.1 Prerequisites: COSMOS Ontology and Domain Specific Ontologies Linking

As identified in D2.3.2, in order to decouple domain specific knowledge from the abstracted and generic features, DSOs need to be created that contain the specific features of a domain, and linked with the concepts in the generic COSMOS ontology. In this period, a DSO for the Madrid UC is expected to be created, to act as a guideline for the process and linking to the COSMOS ontology. This action is expected to continue in the following periods.

4.2.1.2 VE Registration Subsystem

In order to prepare for registration, the VE developer needs to create the VE and IoT-services, according to the format defined in the COSMOS project (IP4). Then based on the ontologies defined in the previous section, they need to provide the semantic description of the PE and the services (VE and IoT-services) and register the VE to the COSMOS platform (IP3). Finally they need to download the appropriate COSMOS components to the device where the VE resides. Following the definition of the combined semantic structure, the VE developer needs to insert information on specific VEs at the Registry component. This process is handled through the subsystem of VE Registration, that appears in the following figure (Figure 14). Except for the generic information of the COSMOS ontology this combined information (core ontology fields values and URIs to concepts defined in the DSO -agnostic to the Registry-) will

be stored combined in one "line" in the triple store functionality of the registry. So actually the integration is made indirectly (and abstractly) in the triples, decoupling the DSO concepts from the COSMOS ontology concepts (related to interfaces and protocols).

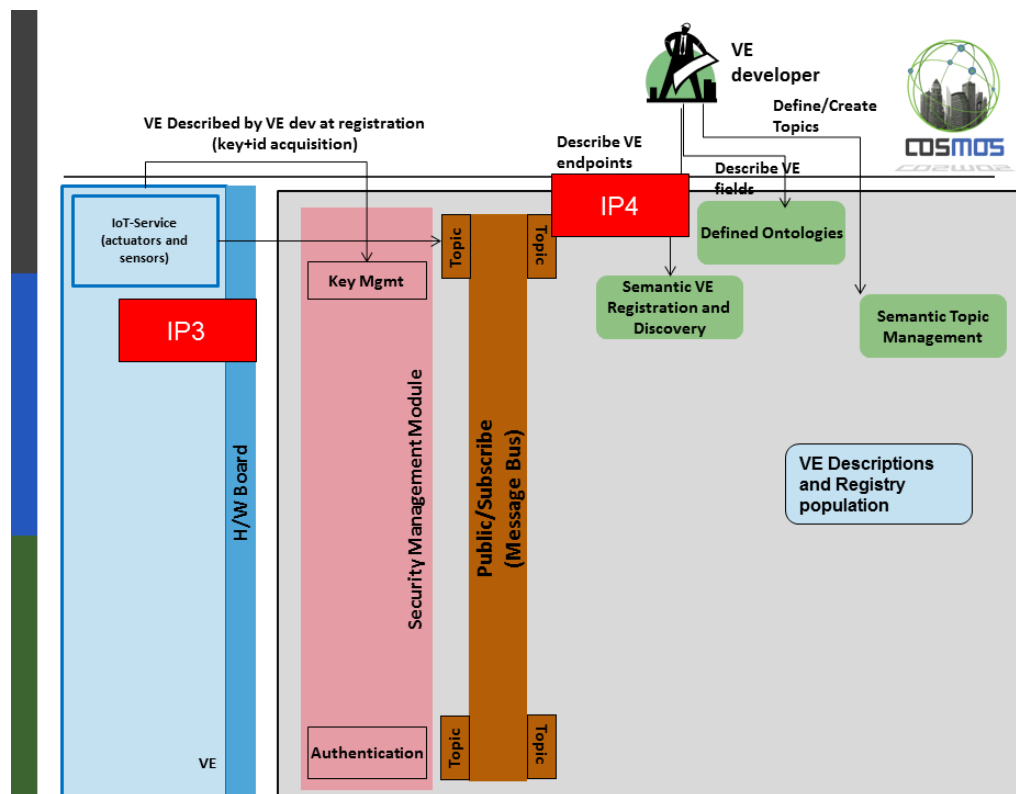


Figure 14: VE Descriptions and Registry population

4.2.2. Application Definition Framework through NodeRed Flows

The COSMOS ecosystem is consisted of a number of services and devices, all exposing interfaces through which interested entities can acquire information. These interfaces, according to D2.3.2, may be of two types, either offered through the Message Bus functionality, in the form of topics, or directly through REST-based services. These available interfaces can then be used, either in the context of the specific platform functionalities or in generic combinations that may be designed dynamically and create new types of services or enable a specific application design and scope. In order to enhance this process, a very interesting approach is Nodered[11], graphical tool based on Node.js, for orchestrating flows between elementary components. Nodered covers a set of requirements that are necessary for the intended use:

- It is abstracted, thus aiding developers with limited experience in the vast IoT domain to combine existing services in the context of a wider application
- It is reusable, in the sense that template flows can be created and shared.
- It is adaptable and configurable, meaning that the template flows may be initialized easily for the specific service instances to be utilized and new nodes with specific purpose can be created and used in the same context

- It can be used both at the platform and VE side for deploying services and application flows

A very interesting feature is that it also comes with a built-in library of useful nodes, that cover the basic requirements of COSMOS (e.g. including MQTT and REST adapters for data flows). Each type of roles in the COSMOS ecosystem (COSMOS Platform Provider, VE Developer, Application Developer) may utilize this tool for their specific purpose. JSON is the main format that is used, which is also in agreement with the COSMOS approach. Furthermore, to ease the integration process, sets of actions and sequences can be defined through Nodered, in order to be reusable in the future. This does not mean that Nodered handles everything, since for example service implementations for specific aspects (e.g. CEP rule update) should be present independently, but be linked through the Nodered tool and relevant flow. Following, indicative purposes for which Nodered may be used are detailed.

Definition of a data feed by an Application Developer

The vision of COSMOS is that Application developers are able to combine and interlink data coming from different sources, according to the envisioned application needs. However, in order to do so, a suitable mash-up kind of tool needs to be in place, including the various COSMOS data sources that may be combined. For the purposes of the COSMOS UCs, we will provide such archetype combinations (that can also serve as examples for anyone interested in building similar application structures), that are mainly derived from the scenarios themselves, and are mainly related to the communication model, specific requirements or the necessity to include a kind of application logic. At this stage we have identified two main basic application archetypes, one with a centralized mechanism at the COSMOS platform, including server side logic, that relates more to the Madrid Case based on the scenarios definition, and one with a decentralized mechanism that relates more to the Smart homes envisioned applications. This analysis is conducted in Section 5.3. These archetypes will be extended when the remaining application scenarios are investigated. Through Nodered we can define the basic template flows (including communication models such as centralized MB topics or decentralized VE2VE service invocations) through which the data will be passed. Thus Nodered will act as our main front end towards Application Developers in order to create new applications from the offered services.

Definition of Services by Application/VE developers

Application/VE Developers may also easily create services directly from REST templates in order to complement the given functionalities of the COSMOS services, either at the VE, application or platform side. This is especially useful (for the Application Developer) when part of the application logic may also reside in a server-side component, as is the case of Madrid mobility scenario. At the VE level, relevant adaptation service interfaces may be created by the VE developer in order to expose IoT services through the interfaces defined by COSMOS or adapt the data format.

Definition of Installation and Deployment Flows

This process is expected to aid significantly VE developers and COSMOS Platform providers mainly. COSMOS VE side components can be installed through specific processes designed with Nodered. To this end, relevant installers (in the form of e.g. shell scripts) may be created

and triggered through a flow, on the target devices. It is a prerequisite that the device is supported by Nodered. Currently support exists for Raspberry type devices. It must be investigated whether the envisioned hardware board to be used as a VE side integration point can be included in this list. A similar process can also be defined for specific aspects of the COSMOS platform, that will aid the according provider role to install and configure platform-side components more efficiently.

Definition of automated runtime adaptation and reconfiguration aspects of the platform

COSMOS developers can also utilize adapted flows for the runtime adaptation and reconfiguration functionalities that may be necessary from their side. Concrete flows in this category may include update of the service endpoint in the Registry, in case of dynamic conditions (e.g. change of IP address). This may be achieved by event-based functions in the Nodered flow, that indicate a change in the IP address and use the REST API of the Registry component (together with the VE's id) in order to update the new endpoint. This flow may be automated and only the specific part of which element changes dynamically (in our example the IP address identification logic) should be altered per case.

Definition of automated runtime adaptation and reconfiguration aspects of the application features derived from COSMOS services

Other aspects may include CEP rules dynamic update, which can be automated from the point where a VE or Application Developer (both roles apply here since a specific rule detection may be either an inherent part of the VE or an extended part of an application) upload a new rules file. This indicative flow appears in Figure 15. The CEP rules definition in Dolce is located in a file at /home/cosmos directory. Whenever a change is identified in this file (which can be made through e.g. an upload of a new rules file, an update of the boundaries of the rules by the cooperation of machine learning), an http request is launched towards the CEP engine REST interface, in order to update the rules in that specific engine. Similar flows may be prepared by the COSMOS technical WPs in order to aid in the integration and configuration process. In the specific example, the need for configuration would be located in the directory where the CEP rules file is located and the IP of the CEP engine and could be driven even by automated processes, such as the ones identified in Section 5.2.5.1.

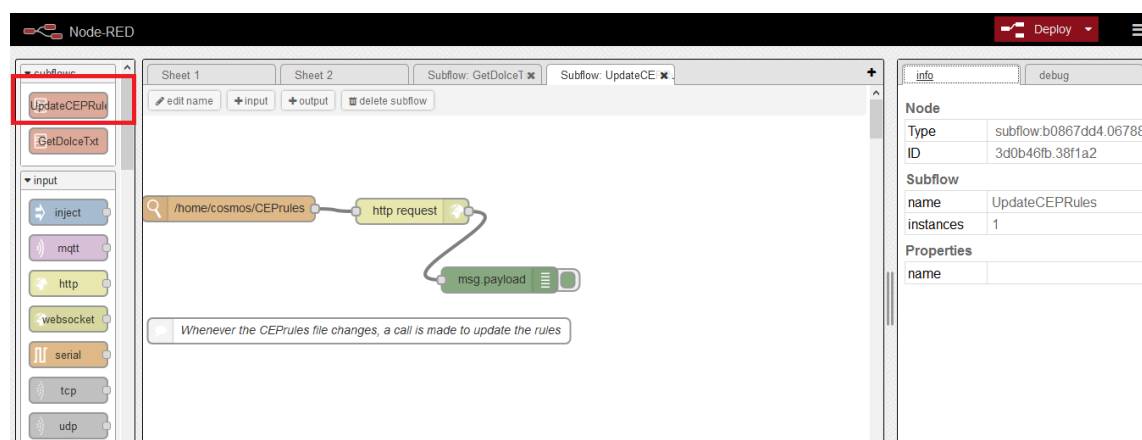


Figure 15: Indicative Nodered flow for CEP Rules update

Testing the flows

For all cases, an extended usage of the tool involves also the ability to test the designed flows, with specific triggering components and debugging abilities. An example of this appears in Figure 16. In this case, we have defined and created a simple REST service (black box) that when triggered by an external GET request returns the working directory. In order to test the service, we use the part highlighted by the red box, which triggers a GET request towards this service, either manually or whenever a specific directory in the filesystem changes. The output is then produced in the Debug tab in the right side of the image (green box). Thus through this functionality we will be able to test the created flows in the context of COSMOS.

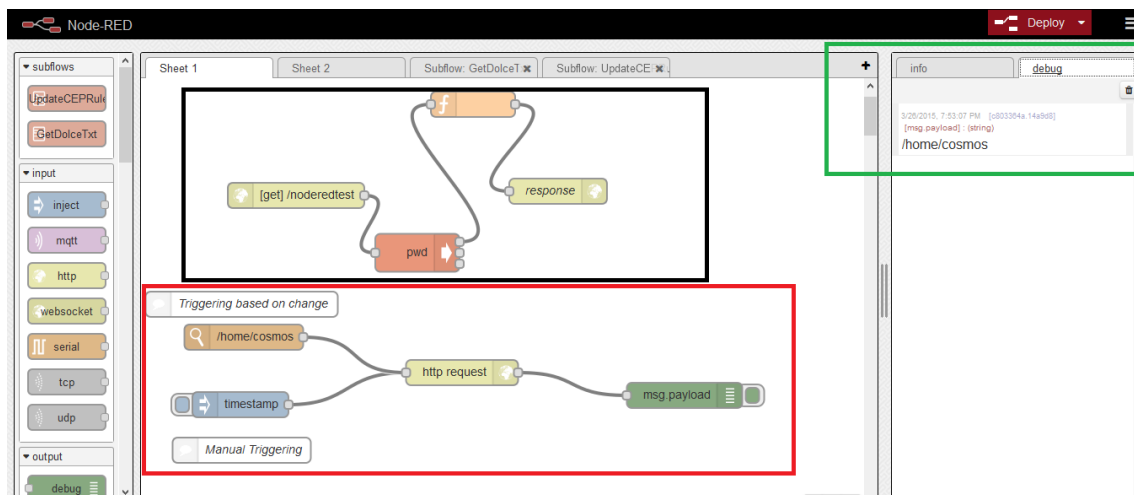


Figure 16: Testing a given flow

Saving, packaging, sharing and reusing flows

Saving and reusing flows is one of the key features of Nodered, and one of the main arguments for using it as an automation and integration tool. In the aforementioned example in Figure 15, the UpdateCEPRules action stream has been defined as a subflow, which enables it to be considered practically as a new node (indicated by the red box in the main node pane in the left upper side of the image). This node can then be used directly and in an abstracted manner in more complicated diagrams. Stored flows can be shared and distributed via repository structures (Figure 17). We will create the necessary repositories for the COSMOS defined flows. Furthermore, we may use existing libraries that have been created by the very active Nodered community[12] and that extend the basic functionalities of the original library. It is within our intentions to contribute also towards this community flows that will be created in the context of the project.

4.2.3. Data Fields Definition

According to the Gantt chart, in this period the data fields definition should be made available. The Message Bus data structure is defined in section 9.2.1 of Deliverable D2.3.2. The important points are that messages should be in JSON format, and should always have fields for VE id and timestamp. Geospatial location is an optional field. Finally there is an additional field for the data payload itself which should be a nested JSON object. This payload may be abiding to per case defined templates. Thus decoupling and adaptation per case may be achieved, with the

only requirement being the usage of JSON. The integration of the relevant subsystem is foreseen for Integration stage 3 (Section 5.2.4).

4.2.4. VE side COSMOS Components integration

The VE side integration will start during this period, mainly with the specification of the H/W board capabilities, followed by the initial component installations in the following period. This process is expected to continue through Y3 of COSMOS, until all VE side components are packaged and running on the device.

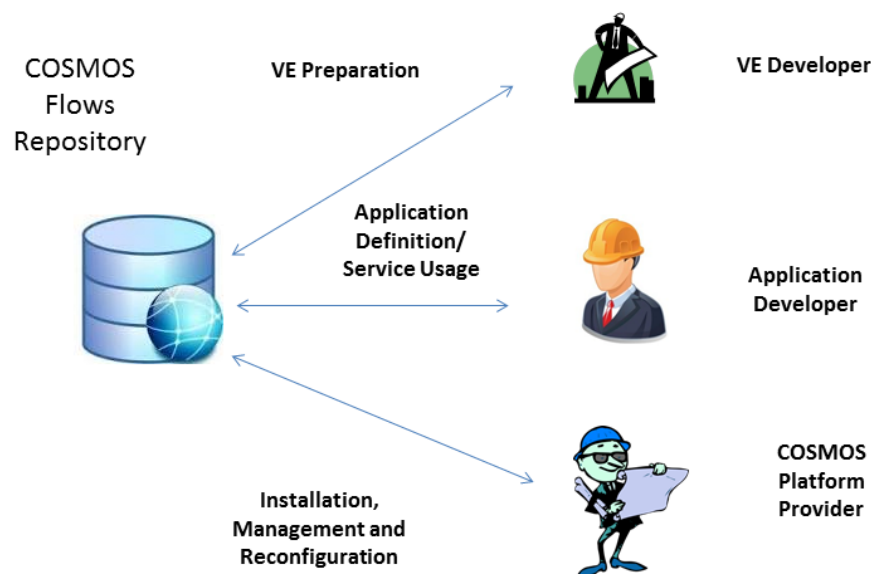


Figure 17: Roles interaction with Flows repository

4.2.4.1 VE Resources specification

The H/W Board provides means for VE integration. The H/W Board consists, as described in D3.1.1[15] and D3.1.2[16], of an embedded ARM CPU which runs a Linux based custom operating system. The H/W Board provides basic functional primitives such as computing, storage and communication infrastructure but also security components. Once the H/W Board has been enrolled into COSMOS, as presented in D2.3.2, components which run on it can make use of all available resources exposed as a service (e.g. encryption/decryption service or checksum computation). The H/W board therefore provides the basic embedded platform for VE to run on.

The main traits of the H/W Board are: Zynq-7000 XC7Z020-1CLG484C AP SoC (containing 2 x ARM Cortex A9 Application Processors)

- 1 GB DDR3 component memory (four 256 Mb x 8 devices)
- 128 Mb Quad SPI flash memory
- USB 2.0 ULPI (UTMI+ low pin interface) transceiver
- Secure Digital (SD) connector (4-8GB cards can be used as main storage)

- Ethernet
- USB-to-UART bridge
- I²C bus
- 3 clock sources (for the FPGA fabric)
- Dual 12-bit 1 MSPS analog-to-digital front end.

4.2.5. Application Archetypes Definition

As mentioned in the introduction of this section, application archetypes are generic combinations of subsystems that have been highlighted from the analysis of the UC scenarios. At this stage, two such kinds of archetypes have been defined, one including the usage of a distributed application, based on COSMOS offered VE capabilities, and one having a more centralized nature, including application server side logic, that involves significant interaction with the COSMOS platform. Other combinations are also feasible (e.g. VE side application logic deployment), that will be investigated in the future and in accordance with the defined UC scenarios.

4.2.5.1 Application VE2VE archetype

The application archetype combination of subsystems appears in Figure 18. In this case the formalization has been made based on the available and defined system use cases in Chapter 9 of D2.3.2, and the relevant sections are included in the name of each connector arrow. For example “CBR Case Creation (9.3.3.4)” refers to the relevant subsystem whose complete description is in section 9.3.3.4 of D2.3.2 Only the initial step of these subsystems has been included for better visibility, that is why we cross-reference the relevant sections of D2.3.2. Internal components that are included in these subsystems are also not included, if they are not involved in the first triggering step. Thus the observed components in the figure are a subset of the overall platform or VE level components used. In this case, service orchestration is used however only in order to define the application structure and perform the deployment on the VE level. This archetype represents the minimal interaction with the COSMOS platform, that was initially highlighted in the Y1 version of this document (section 3.2.5) and contains two integration points, one for the Application developer to create the design (IP2) and one for the End User to handle the created application (IP1).

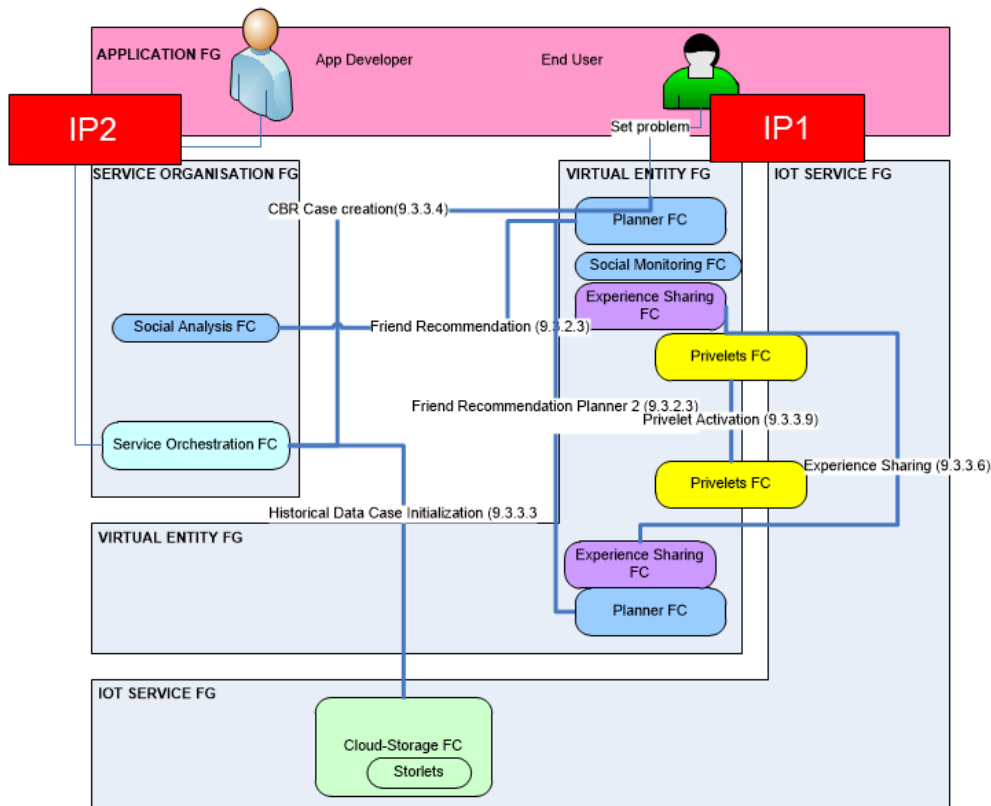


Figure 18: VE2VE application archetype based on defined system cases of D2.3.2

4.2.5.2 Application with centralized logic archetype

The subsystem and roles for this case appear in Figure 19. The main involved entity is the Application Developer, who needs to define the application workflow for the centralized part (including any interactions/configurations/selections, data channel creations etc.) based on the envisioned application logic needs (IP2) and create also client applications that can themselves adapt to the service calls that need to be made towards the COSMOS components (IP2). Again the same notation as in the previous section is used, by cross-referencing the relevant subsystems from the according sections of D2.3.2. Internal components that are included in these subsystems are also not included, if they are not involved in the first triggering step, in order to enhance readability of the image. Thus the observed components in the figure are a subset of the overall platform or VE level components used.

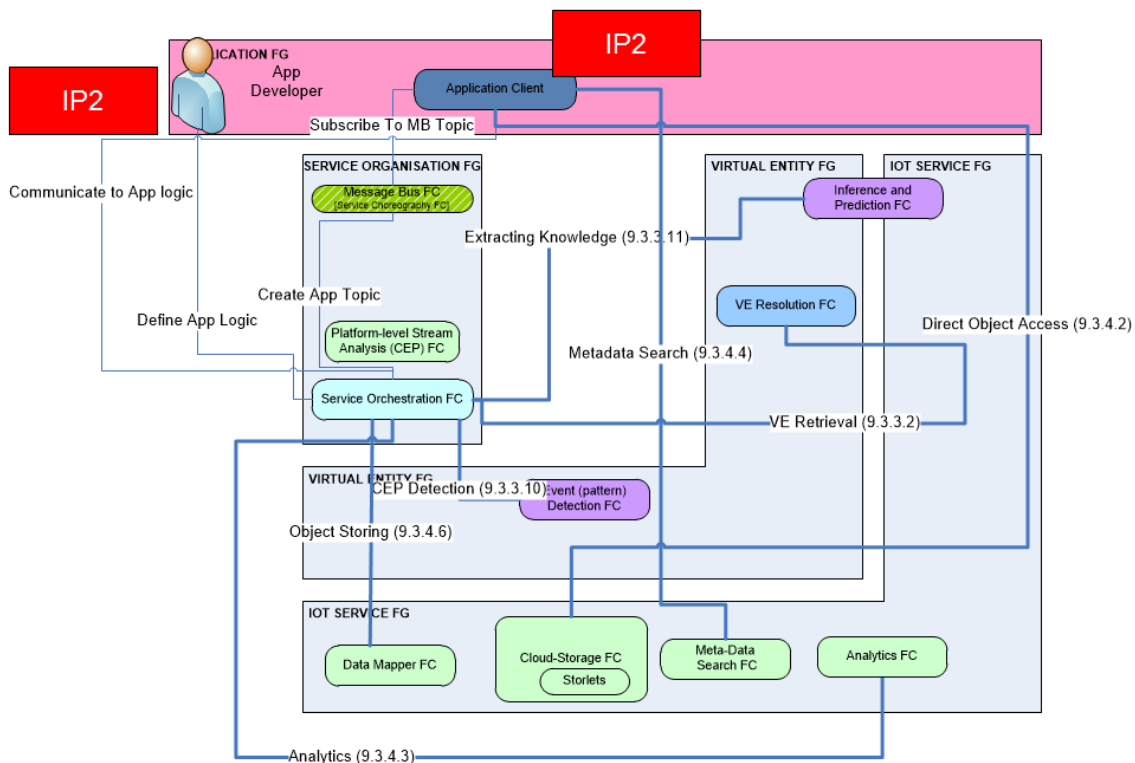


Figure 19: Application with centralized logic archetype based on defined system cases

4.3. Use Case specific aspects involved

The application archetypes defined in the previous sections have been abstracted directly from the defined UC scenarios, specifically the ones that are anticipated to be ready for the end of Y2 (M25- September 2015). For these cases the components that are going to participate in the respective COSMOS app and from which perspective (how are they going to be used in the context of the scenario) are highlighted in the application archetypes. More information on the concrete scenarios is included also in the following chapter, since they are mainly part of Integration Stage 3.

What is more, derived from the UCs directly are the DSOs mentioned in Section 4.2.1.1, that capture the concrete characteristics of the UC VEs and of course the UC specific VEs are the main target of the overall “VE Instances and Registry Population” subgoal.

4.4. Testing environment and roles

The testing infrastructure at this stage is expected to be comprised by the COSMOS platform, described in Chapter 3, enhanced by the newly introduced components in this cycle, and by the UC platforms. A significant advancement is the expectation to have available the VEPROT platform from the Madrid UC, in order to obtain real-time feeds from the respective VEs. Furthermore, we will extend the testing of the VE side components to lower capabilities platforms such as the Raspberry Pi, in order to benchmark the implementation of specific components like the Planner.

The involved tester roles at this stage include mainly the VE developers, that are expected to interact with the COSMOS platform developers for creating the DSOs of the UCs. Furthermore, the Application Developer role should be included in order to take under consideration how the different available services can be combined to create an elementary COSMOS App, obtaining information from the city platforms and using this as input to the COSMOS services, through the Nodered environment. The later involves also the interaction with COSMOS component developers, based on its anticipated usage. However this role mainly starts the interaction in the following period.

In a nutshell the testing environment is expected to comprise of:

- The necessary COSMOS services running at the COSMOS platform side testbed
- Domain Specific Ontologies created
- Raspberry Pi for initial investigation of lower level devices for VE components in an effort to benchmark capabilities
- Nodered environment deployment for investigation and initial integration purposes

5. Integration Stage 3 (M23-M25)

5.1. Involved Integration Goals and Refinement

Following the results of the previous integration period, we expect at this stage, with the advent of new functionalities as expressed in D2.3.2, and the new prototypes in M22, that the need for integration will be enlarged. Initially the goal of COSMOS Platform Setup and cross-component integration is expected to continue, for the new functionalities provided during Y2. New advancements on subsystems are anticipated, for the following subgoals:

- **Data Management and Analytics:** in this case, we will investigate the integration between the Experience Sharing and Situational Awareness components, for regarding as experiences not only the currently supported cases of the Planner component, but also extended types (e.g. events or situations). Another aspect of extension is the achievement of CEP rules boundary definitions through machine learning analysis. The analysis should involve both real time and historical data.
- **Autonomous behaviour:** in this case we will examine direct CEP rules editing and update, as well as cooperation between the planner and the storage analytics for initial Cases creation and the planner and privelets for direct VE2VE communication.
- **Web UI Integrated environment:** in this case the specifications will be pursued, so that implementation can start in the following Integration Stage 4.

Furthermore, we expect to progress on the goal of Data Model/template, in terms of the following subgoals:

- **JSON Schema retrieval subsystem:** in this case we will examine the ability to store separate JSON schemas per topic in the Cloud Storage and associate them in the Registry to each topic.

Another goal of this period is the VE Description and Linking to the Platform goal, from which the following subgoals will be investigated:

- **DSO definition for Use Cases:** in this case we expect to carry on work in case delays are encountered. However at least one DSO should be ready up to this stage in order to enable the testing of the following subgoal.
- **VE Instances Description and Registry population:** in this case we expect to be able to define and describe a VE, its capabilities and its interfaces and thus include the semantic descriptions and capabilities in the demonstration lifecycle.

What is more, progress with relation to the VE side components integration is expected, mainly in the following subgoals:

- **VE endpoint definition:** in this case, how the VE services are going to be exposed
- **Components installation:** in this case, the installation of all COSMOS components in one shared VE device will be initialized, as baseline for the following period

Finally in this period advancements are expected in the case of the Application Definition, Creation and Deployment goal, in terms of the following subgoals:

- Application Scenarios concretization: in this case we expect to define how the application scenarios will be instantiated and implemented with the usage of the appropriate COSMOS services and components This relates mainly to the initial design and development of one or more of the concrete Application Scenarios that have been defined, in relation to how the available COSMOS enabled functionalities can be used or defined in order to serve the application needs. This includes aspects such as data flow from multiple sources of information, event definition and identification, specific Storlet creation and integrated usage, adapted predictive models for a specific metric (as indicated by the application) etc.
- Application Definition framework: in this case, initial Nodered flows may be investigated, to aid in the previous subgoal

5.2. Subsystems/Subgoals involved

5.2.1. Application Definition Framework through NodeRed Flows (continuation)

In this period, following the analysis made in Section 4.2.2, the various roles that are expected to interact with the Nodered environment should produce the relevant flows that are realizable per case, and based on the anticipated provided functionality. Especially technical components are expected to engage in this process and produce flows that abstract the functionality and interfaces, where applicable (IP without numbering, since it involves COSMOS developers). Furthermore, the applications that are defined in Section 5.3 may begin to be encoded (the aspects that are at least applicable e.g. in the case of the centralized application archetype mentioned in the previous chapter), by an application developer (IP2).

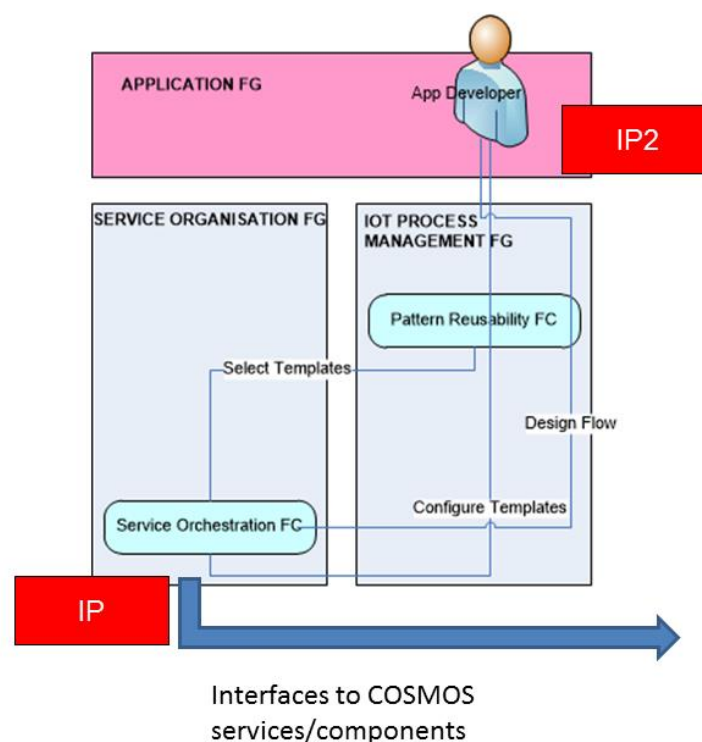


Figure 20: Generic application design process

5.2.2. VE Instances Descriptions, Registry population and DSOs (continuation)

Following the preparatory work in the previous period with regard to this subgoal, in this period the instantiation of concrete VEs for Madrid is expected, for testing the VE registration subsystem. In this period the action is expected to continue in order to finalize the concrete Domain Specific Ontologies for Camden and Taipei and potentially create instances for the Camden case. The concrete subsystem along with the integration points has already been defined in the previous chapter (Figure 14 of Section 4.2.1.2).

5.2.3. VE side COSMOS Components integration

Given the fact that a VE consists of various software components, the VE side integration views each VE as a set of applications which are performing a given number of tasks. One or more VE's can run on a H/W Board the only limitation being the given performance of the system versus the complexity of the VE's.

The first step in integrating VE's on the H/W Board will be the simple instantiation of the software components which comprise the embedded platform. A further step would be to provide means for seamless extension/instantiation of new VE's using existing ones as a starting point.

VE's can make use of a Code Ubuntu distribution (version 12.10 currently) which offers the advantages of the H/W Board components combined with the power of a fully-fledged Linux distribution. Web services can be exposed using RESTfull interfaces using the build-in web-server while security uses the hardware primitives described in D3.1.2.

5.2.3.1 VE endpoints definition

In this case it will be investigated whether an intermediate layer needs to exist between the VE services endpoints used in the project and the external world (e.g. existing services interfaces), in order to adapt them to a defined template from COSMOS. This layer would be needed if we want to homogenize endpoints in the context of the project, but it would also produce an overhead for the VE developer to adapt their VEs in this format. Such wrappers could also be created through Nodered flows. The tradeoffs of this process will be investigated in this period.

5.2.3.2 Components installation to VE instances

Once a VE is instantiated new components can be installed to it, like for example Situational Awareness or Planner modules or CEP engine. These are plug-in applications which are instantiated by the VE. On the back-end side, each VE is a user living inside the Linux operating system which powers the H/W Board. The VE acts like the administrator within its own user and has control over the other sub-components instantiated. Component installation needs human intervention which means that each component installed needs to be loaded and configured manually so that the VE knows of its existence and can make use of it.

Currently the H/W Board supports one VE which instantiates a CEP engine optimized for low resource embedded system. Privelets and security components (e.g. authentication, authorization, etc.) are by-default included as they form the basis for each VE. The goal for this period is to have a functional VE which can push private aware data to COSMOS using secure communication channels.

5.2.4. JSON Schema retrieval subsystem

A relevant subsystem should be defined for retrieving the JSON schema of a specific topic of the MB. According to D2.3.2, each topic may use its own schema, as defined by the VE developer. The Cloud Storage can store these available schemata, and the Registry may include a necessary field to hold the reference to that storage location. This subsystem should be defined in this period.

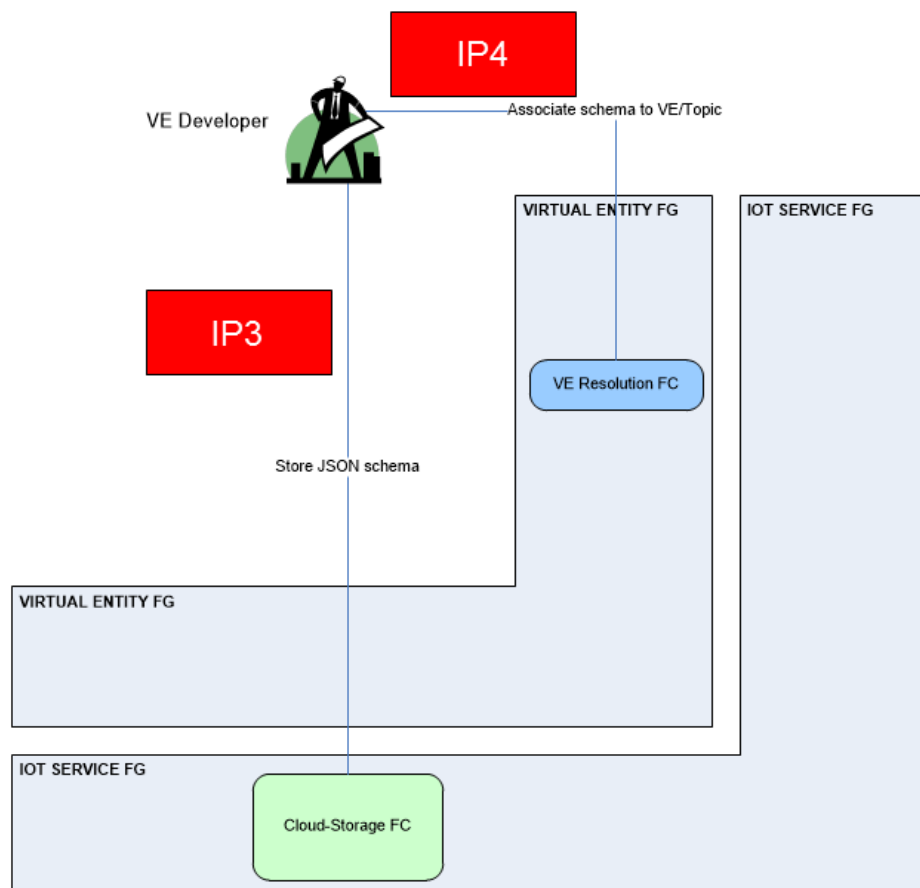


Figure 21: JSON schema storage and association

5.2.5. COSMOS Platform and cross components integration (continuation)

5.2.5.1 CEP, Situational Awareness and Machine Learning Cooperation

The CEP engine being used in COSMOS, formerly known as μ CEP, facilitates the achievement of two levels of Situational Awareness, Level 1 *Perception* and Level 2 *Understanding* –as

explained in the work done in WP6—. In order to make available this component to the rest of COSMOS components, specific *Event Collectors* are designed to connect to the Message Bus in its 2 implementations: RabbitMQ (year 1)[18] and Apache Kafka (year 2)[19]. In addition, a REST Admin API has been defined so to allow system developers to add new or modify existing DOLCE Rules files. Finally, a WebGUI assisted wizard is being created in order to facilitate those non-technical application developers (e.g. sustainability officer role in Figure 22) in the task of defining rules and populating them into the μ CEP engine. All in all, the SA mechanism requires the selection of relevant data sources, the application of appropriate complex rules, and furthermore, the projection of the evolution of the situation of the monitored situation, what is enabled by the connection of the generated value-added information (in the form of complex events) with dedicated prediction components.

Further integration is expected to interconnect Inference/Prediction FC with the Event Detector FC. The former will access historical data and run machine learning algorithms to estimate optimized parameters for CEP rules which will be updated automatically through the aforementioned REST interface of Event Detection FC (Figure 22).

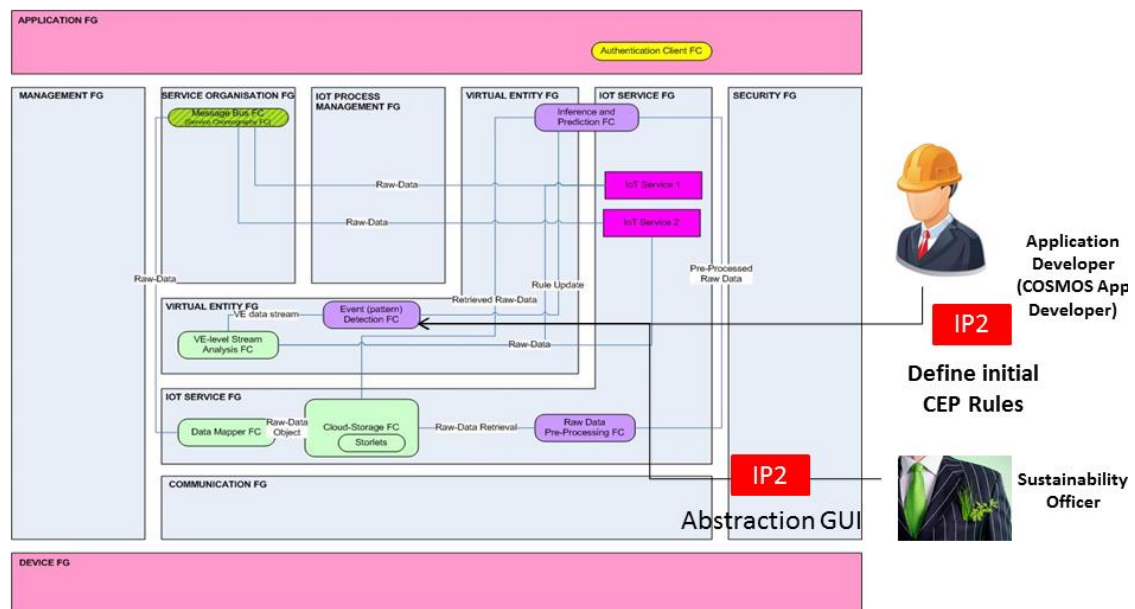


Figure 22: Cooperation of Inference/Prediction with Event Detection (from D2.3.2) and involved integration points

5.2.5.2 Planning, Experience sharing and Situational Awareness

Following the previous integration attempts, the final step is to integrate what is identified as a Level 2 SA, with the Experience Sharing mechanisms. Experience sharing is designed as a mechanism to disseminate relative information to relevant participants through the use of social networks. Thus it would be interesting to work towards sharing any kind of experience, including situational awareness aspects. In order to do so, we could also add (at the VE side) Nodered capabilities and thus include an “application logic” part, similar to the centralized applications of Madrid explained in Section 5.3.1, but this could also be directly embedded in the planner component (as the solution to the given problem, the problem being the specified event from SA2, the solution being getting the similar VEs and using VE2VE communication to propagate the event). The combined system cases for the latter case appear in Figure 23, where the IP2 refers to the application developer making the necessary combinations.

However in order for this combination to be achieved, adaptation of the current Experience sharing mechanism may be required, since at the moment it is used to send actuation plans from the callee to the caller, while in this case the callee gets an actuation plan from the caller. In this case it is critical to exploit COSMOS social monitoring components, in order for the callee to investigate the ranking and reputation of the caller that propagates the event. Thus it would not continue propagating the event to its own contact list, if the original sender cannot be trusted (avoidance of panic creation etc.). This may be used in UCs in the form of propagating emergency signals (e.g. fire in the apartments, amber alert for missing SP in the Madrid case etc.).

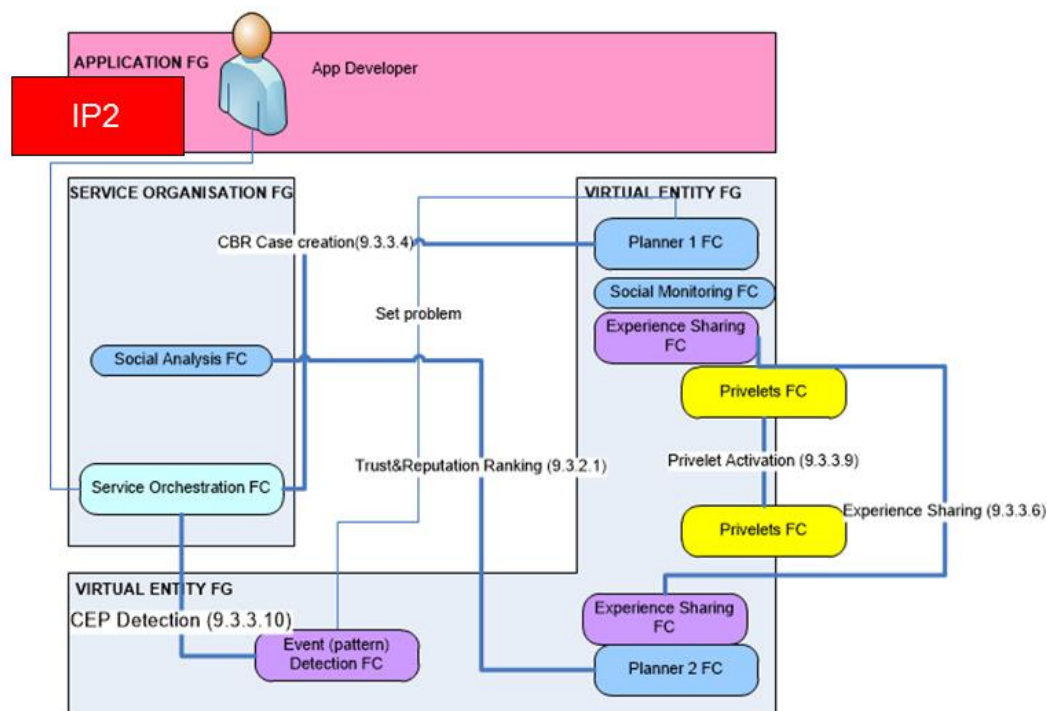


Figure 23: Experience Sharing for propagation of Situational Awareness

The cooperation of all these components can be identified as a Nodered sequence, as identified in Figure 24 (for the core SA and machine learning cooperation) and Figure 25 (for the SA and Planner/Experience sharing cooperation).

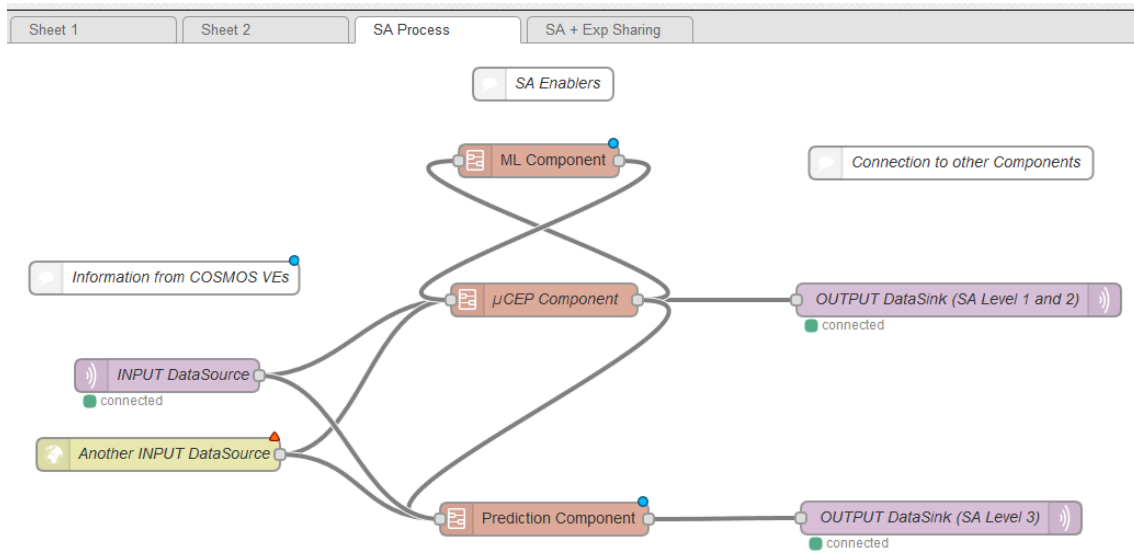


Figure 24: The Situational Awareness Nodered process

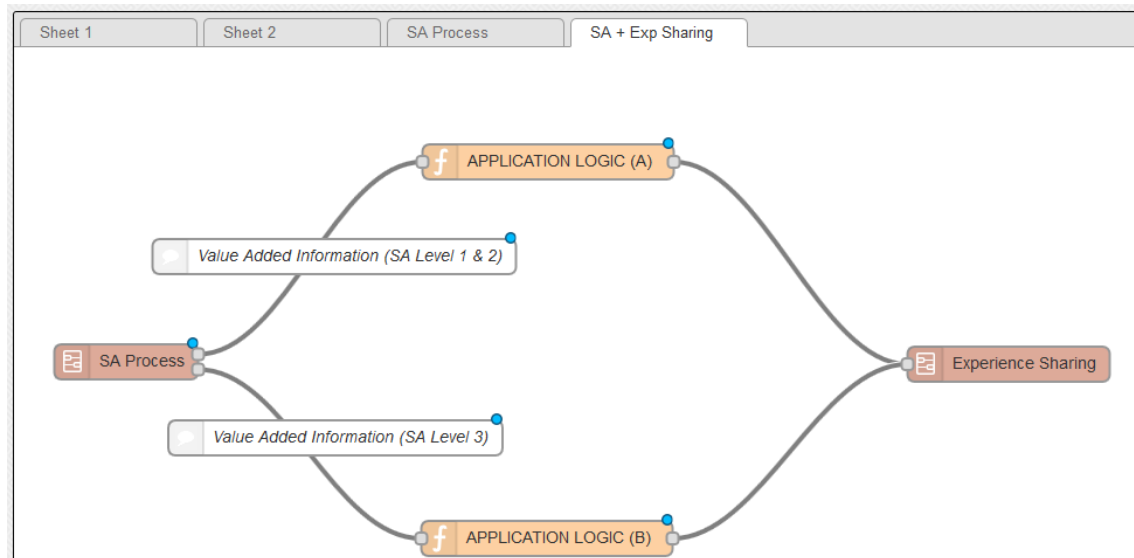


Figure 25: Situational Awareness, Application Logic/Planner and Experience Sharing

5.2.5.3 COSMOS Process Web UI integration

In this period it will be investigated whether a specific COSMOS portal front end is required in order to abstract and present the actions needed by each role. Thus it is applicable to the components identified in this document that have integration points with the according roles. Such a front end should include a set of specifications that will be clarified during this period. At this stage some anticipated actions include

- A user management system for different roles
- Different tab activation for these roles, based on the expected actions that they are going to perform with relation to the VEs, the platform services etc.
- Linking to existing or to be created GUIs of the subsystems of COSMOS, some of them already included in D7.7.1, for example:

- Uploading of specific storlets to the Cloud storage
- Uploading of CEP rules files definition
- Redirecting or embedding the Nodered GUI
- GUI for the repository of available Nodered flows that can be imported
- Etc.

Through this functionality relevant instructions may be provided for each step, directing each role to the necessary action steps. For this reason a separate GUI should be created by component owners where appropriate. This is compatible to the existing architecture in which most of the components are based on REST interfaces. Furthermore it may enhance aspects that are included in D7.4.1 as criteria for assessing the COSMOS components. Specifically it will enhance the following aspects identified in the aforementioned document:

- **Functionality**
 - Satisfaction
 - Ease of Use
 - Reusability
- **Construction**
 - Structuring

An indicative screenshot of such a front end appears in Figure 26. In this period the main focus will be on specifying the needed operations of such a tool.

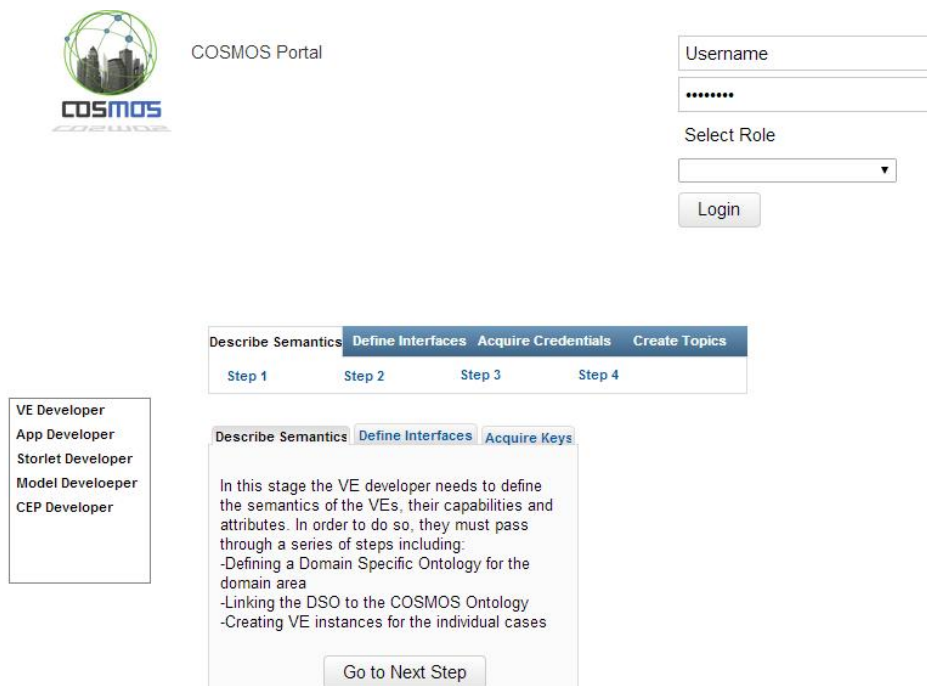


Figure 26: Example GUI Mock-up for COSMOS Front-End

5.2.6. Data feeds integration

5.2.6.1 Madrid Bus API UC data

In this period we are mainly focusing on integrating with the Madrid Bus API, since as will be described in Section 5.3.1, it is one of the points needed for the application scenarios.

The Central Processing Unit on the Bus (UCP) has all the necessary information in real time to make the correct operating system inside the vehicle and SAE offer the driver and passengers about the route of the line through the peripherals (driver console, interior and exterior panels, synthesizer audio). The wireless network in the bus allows different users to connect to the EMT network to obtain information in their mobile devices. The aim is to provide functions so that the users can have access through their mobile devices to information about the operation of the line and the bus in real time. This information will be available through web services that are easily usable from mobile applications using standard protocols.

The information provided by the CPU to external systems via Web server is follows:

1. Data bus and its position on the line
2. Estimated time of arrival at the following stops
3. Data path of the line

This information is available upon request at any time of a web client. To ensure proper use of these services a secure connection using the SSL protocol will be used as well as user authentication for a service. Restricted access to services based on user and access type will be also used. The URL on the bus will be emulated through an online service of the form:

[http://some_IP/rests/?srv=\[service name\]&\[parameters\]](http://some_IP/rests/?srv=[service name]&[parameters])

A "virtual" bus running continuously on line 1 will be simulated for testing. Communication between the client device and the Web server CPU use the client server model. Web services are implemented with the REST protocol that enables to design services less lightweight than SOAP based ones. Consulting a Web service will be a TCP connection to server port (port 443 for a secure connection using SSL), sending a GET operation. Depending on the Web service, parameters may be omitted or may be more than one. Response is obtained by the same TCP connection as an HTTP document with the message body encoded in an XML document format, containing the returned data. User authentication shall be in default in the http connections, including the username and password in the http header of the document sent to the Web service request. SAE is the system in the EMT premises responsible for returning the information to the application request.

Error Handling

In the aforementioned process there are two types of errors:

1. Errors detected while trying to build the question and send it to the simulated system at EMT central.
2. Errors detected by the SAE in a given application service request

In the first case the XML document returned in response to REST service contains a single node with <error> name. This node will contain the "E" attribute with a value number indicating the error code and data as a text description of the error. An example error response might be:

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

```
<error E="1">Service undefined</error>
```

The errors that can occur are:

- 1) Has not been defined with the data service "srv"
- 2) Error in connection with the SAE
- 3) Time-out waiting for response from SAE
- 4) Operation not permitted for the user

In turn, the application of SAE can detect any problems in drawing up the requested response. In this case the answer will be a node with the name of the requested service and an attribute "E" containing a numeric value with the error code. Optionally you can contain a node named "error" which will have a text data describing error.

An example of this type of error could be:

GET /infoemt/?srv=NoDefinido

```
<?xml version="1.0" encoding="UTF-8"?>
<NoDefinido E="1">                                (not defined)
<error>Orden no valida</error>  (Order not valid)
< NoDefinido>
```

These errors may be generic, common to any service or specific to each service. Common errors to all services may include:

- 1) Order not valid
- 2) Wrong parameters
- 3) Error when creating the service. Required Resources
- 4) duplicate identifier question
- 5) Internal Error sending message
- 6) Time-out waiting for the response internal
- 7) Incorrect Data
- 8) Data from the incorrect response
- 9) Error while processing the command
- 10) Maximum size of a parameter exceeded

Available Web Services for on board customers

The following describes each of the available Web services for customers boarding the bus. The answer of the services is modified according to the access level of the user who invoked, so that if the access level is 0 some data may be omitted.

Table 2: Web Services from the Bus SDK

Service content	Service name	Parameters	Response
Real Time Vehicle Data (Gets the ID of the vehicle, the data line being made and	DatosCoche (car data)	stops numeric value with the number of stops to return with information on the distance and estimated time of arrival	DatosCoche XML

position on the line)		(optional)	
Structural data line (Returns information from the structural data of the bus line including position and name of each of the tour stops)	DatosParada (stop data)	No parameters	DatosParada XML

Table 3: Attributes and possible values for DatosCoche XML response

Attribute	Values
vehiculo	numeric value with the identifier of the vehicle
linea	Numeric identifier of the line
sublinea	Numeric identifier of the subline
viaje	Number of trip in the line that is making the bus
sentido	Direction of travel (1 for the round 2 for the back)
estado	Location status. Possible values are: 1 = Not Assigned 2 = Assigned 3 = Incorporation 4 = closure 5 = Located Online 6 = Offline forced by the driver 7 = delocalized Route 8 = load (long) 9 = forward 10 = Fault location (no odometer or GPS) 11 = Located in garages 12 = Position held in a terminal 13 = Out Route Values 14 and 15 = not defined 16 = Performing a (short) limitation
Horario(schedule)	Subnode void. When displayed indicates that the line is being regulated by schedule rather than by frequency.
Desfase (gap)	Offset from the theoretical bus schedule in seconds. A positive value indicates progress on schedule.
desfase_ref	Offset the bus regarding the schedule reference in seconds. A positive value indicates progress on schedule.
Posicion	Position the bus. If the position is located within the route associated to the direction. In any other state the distance traveled since changed this state
GPS	Attributes include: UTM position in meters to the East (X) and North (Y), altitude, UTM Zone, and UTM Band
Paradas (stops)	Table with the following stops requested and estimated arrival. The information for each stop in node attributes appears in Table 4.

Table 4: Stops fields description for estimated arrival

Paradas field Attributes	Description
Codigo (code)	Numeric Code Stop
X	UTM X coordinate in meters from the stop
Y	UTM Y coordinate in meters from the stop
Distancia (distance)	The bus stop distance in meters from the current position
Hora (time)	Estimated arrival time at the stop. It is a numeric value in the format: hhmmss

The <state> subnodes, <times>, <desfase> and <desfase_ref> are only sent when the User access level is greater than 0. The subnode <position> is only sent when the bus is located online. In all other states with the assigned bus is sent when the access level is greater than 0. The subnode <paradas> is only sent when requested more than one stop and the bus is located in line. The attributes of time and distance in the stops are not shown if the stop is after withdrawal or regulatory action. In this case the bus does not arrive at these stops because it has deviated from the normal route of the line. This service does not return any particular failure errors

Example of DatosCoche:

URL: <http://www.bus.local/infoemt/?srv=DatosCoche¶das=2>

Answer:

```
<?xml version="1.0" encoding="UTF-8"?>
<DatosCoche>
<vehiculo>23</vehiculo>
<linea>79</linea>
<sublinea>1</sublinea>
<viaje>1</viaje>
<sentido>1</sentido>
<estado>5</estado>
<horario/>
<desfase>15</desfase>
<posicion>1164</posicion>
```

```
<gps est="60441.000" nor="496361.000" alt="0.000" zone="30" band="S"/>
<paradas>
<parada codigo="5720" x="60498" y="496156" distancia="212" hora="163226"/>
<parada codigo="1132" x="60641" y="495567" distancia="818" hora="163356"/>
</paradas>
</DatosCoche>
```

Figure 27: Example of Bus SDK reply for bus info

The estimated time of arrival at the following stops are estimated from the time planned for the bus in this line. It is assumed that in developing the schedule has taken into account the normal travel time on each route and this portion is taken into account for an estimated use along the line speed. Normally the system sends to each bus control the schedule defining the arrival and departure hours of the headers and some intermediate stops. The software UCP uses these times and the distance between the stops to calculate the time of arrival each stop according to this schedule interpolating function of the distance between the stops and assuming a constant speed. With this information the CPU is able to compute at each instant the gap ahead or delay according to schedule, when interpolating between the output of the previous stop and Check the following from the current position and the distance to these stops. After calculating the offset, the estimated time of arrival at the following stops route is calculated by adding the offset to the arrival time according to schedule.

For the DatosParada operation, it is not expected to be used at this stage, however we include an an example for completeness:

URL: <http://www.bus.local/infoemt/?srv=DatosParada>

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<DatosParada>
<linea>79</linea>
<sublinea>1</sublinea>
<label>79</label>
<sentido>1</sentido>
<secciones>
<seccion codigo="23308" sentido="1" longitud="12027"
nombre="VILLASVERDE ALTO">
<parada codigo="1425" cabecera="true" posicion="0"
X="60714" Y="497154">
PZA.DE LEGAZPI-MAESTRO ARBOS
</parada>
<parada codigo="5718" posicion="732"
```



```

X="60352" Y="496783">
AV.CORDOBA-GTA.CADIZ
</parada>
</seccion>
<seccion codigo="23310" sentido="2" longitud="12607"
nombre="LEGAZPI">
<parada codigo="1197" cabecera="true" posicion="0"
X="59530" Y="492236">
PZA.DE AGATA-DR.MARTIN AREVALO
</parada>
<parada codigo="1195" posicion="355"
X="59698" Y="492293">
DR.PEREZ DOMINGUEZ-AV.REAL DE PINTO
</parada>
</seccion>
</secciones>
</DatosParada>

```

Access records

Each service request is recorded in a log file. Apache server logs will be used to store the following data:

- 1) IP address of the client that requested the service
- 2) User has been authenticated
- 3) Date and time
- 4) Requested service with parameters
- 5) HTTP error code that was returned to the request
- 6) Size of the response sent

Furthermore, during this period information on the final version of the VEProt platform are expected to be available, in terms of APIs for getting directly data from the central platform (to be used in application logic) and information on how to register to VEProt for pushing requested information in the COSMOS MB. Moreover, an abstracted version of the Bus API mentioned above will be provided via the EMT SDK, especially for integration with mobile applications. EMT platform EMting will also be necessary for using the SDK, through which the same methods may be used (e.g. user registration, the contents of the EMT Bus Data Service and API mentioned above etc.). In this case there is no need for implementing REST clients or XML processing functions. This functionality is expected to be available in the following period.

5.2.6.2 Taipei UC data

Examples of Taipei UC data formats have been taken under consideration regarding their format and available information. This data is mainly through excel files, in order to get introduced to the monitored features and thus potential scenarios that can be pursued. An example of the data available appears in Figure 28, while the interface for their acquisition is included in D7.1.2[17].

C	D	E	F	G	H	I	J	K	L
	AppType	REPORTTIME	OLVOLTAGE	OLCURRENT	OLFREQUENCY	OLPOWERFACTOR	OLACTIVEPOWER	OLAPPARENTPOWER	OLMAINENERGY
0D6F00017352A5	0: thermos bottle	2/1/2013 0:00	113.98	0	59.99	1	0	0	82.5
0D6F00017352A5	0: notset	2/1/2013 0:00	113.89	0	59.98	1	0	0	3.1
0D6F00017352A5	0: notset	2/1/2013 0:00	113.91	0	59.98	1	0	0	29.4
0D6F00017352A5	0: vacuum cleaner	2/1/2013 0:00	113.91	0	59.99	1	0	0	3.5
0D6F00017352A5	0: thermos bottle	2/1/2013 0:00	113.98	0	60.05	1	0	0	82.5
0D6F00017352A5	0: notset	2/1/2013 0:00	113.95	0	59.99	1	0	0	3.1
0D6F00017352A5	0: notset	2/1/2013 0:00	113.97	0	60.02	1	0	0	29.4
0D6F00017352A5	0: vacuum cleaner	2/1/2013 0:00	113.93	0	60.05	1	0	0	3.5
0D6F00017352A5	0: thermos bottle	2/1/2013 0:00	113.95	0	60.1	1	0	0	82.5
0D6F00017352A5	0: notset	2/1/2013 0:00	113.84	0	60.1	1	0	0	3.1
0D6F00017352A5	0: notset	2/1/2013 0:00	113.83	0	60.1	1	0	0	29.4
0D6F00017352A5	0: vacuum cleaner	2/1/2013 0:00	113.83	0	60.1	1	0	0	3.5
0D6F00017352A5	0: thermos bottle	2/1/2013 0:00	113.84	0	60.01	1	0	0	82.5
0D6F00017352A5	0: notset	2/1/2013 0:00	113.78	0	60.1	1	0	0	3.1
0D6F00017352A5	0: notset	2/1/2013 0:00	113.86	0	60.06	1	0	0	29.4
0D6F00017352A5	0: vacuum cleaner	2/1/2013 0:00	113.81	0	60.01	1	0	0	3.5
0D6F0001734F12	TV	2/1/2013 0:00	113.69	0	59.98	1	0	0	35.7
0D6F00008DF28C	TV	2/1/2013 0:00	113.61	5.00E-02	59.98	0.37	2.08	5.68	7.2
0D6F000172C336	0: computer	2/1/2013 0:00	113.72	0.14	59.99	0.61	9.58	15.83	70.0
0D6F000172C336	0: screen	2/1/2013 0:00	113.69	0	60.1	1	0	0	1.9
0D6F000172C336	0: sound	2/1/2013 0:00	113.71	0	60.1	1	0	0	3.1
0D6F00008E5C36	TV	2/1/2013 0:01	113.27	0.13	60.11	0.71	10.44	14.73	125.2
0D6F00008E9D9F	TV	2/1/2013 0:01	113.32	0	60.1	1	0	0	41.8
0D6F00017352A5	0: thermos bottle	2/1/2013 0:01	114.03	0	60.1	1	0	0	82.5

Figure 28: Example of Taipei data

For the Taipei case, given the data nature and available fields, including power factor, active and apparent power measurements, interesting aspects to investigate could be power factor analysis and optimization (at the device, flat or entire area granularity) that may act as a guide, through COSMOS platform analytics capabilities, towards what types or sizes of VAR systems to install in similar cases. Involvement of the Planner components could be also investigated in order to coordinate between tenants and suggest the operation of certain devices that would help achieve the desirable power factor across an area. In terms of awareness, in some cases (e.g. vacuum cleaner) the power coefficient is 1, which is indicative that the device is off (if on, it should have an inductive behavior) and the consumption is just resistance (that may be a stand-by light). Thus with the appropriate outlet management, energy savings may be acquired (or even extrapolated to the user to indicate the cost benefits from an optimized behaviour). Occupancy detection through CO2 sensor data and management of devices for reducing footprint can be another possibility, based on COSMOS machine learning capabilities. In this case labelled data would be needed, however through the use of specific rules these labels could be inferred from existing data. Weather correlation with energy demand is also an interesting aspect that can be applied also to both Camden and Taipei. At this stage we are mainly focusing on consuming the data from Taipei, similarly to the process done in Y1 for the Madrid case in Section 3.2.1, that can enable the application of other subsystems (e.g. Modelling and Storage analytics in 3.2.3)

5.3. Use Case specific aspects involved and concretization

Following, information on the UC specific aspects is portrayed, for adapting the scenarios to the used COSMOS services. For the case of Taipei, the main goal of this period will be focused

around getting offline data and including them in the Cloud storage, as well as applying analytics capabilities.

5.3.1. Application scenarios concretization (Madrid Case)

The Assisted Mobility Scenario defined in D7.1.2 can be decomposed into 3 main components:

- **Special Person Mobile App Client:** this component is running on the mobile phone of the SP and needed in order to connect to the VE Bus and obtain the ID of the line and the bus. This information is then sent to the Application Server side, that checks whether the SP has boarded the correct bus. Following, the SP Mobile is used in order to link to the platform MB (via an independent internet access) and publish information on the GPS location of the SP. This is needed in order to compare at the platform side in a real time fashion this location to the bus location (obtained from Rbox) and determine if the SP is on the bus. An independent internet access is needed in case the SP gets off at the wrong stop (in which case it would disconnect from the bus Wifi) and needs to be tracked by the CG.
- **Care Giver Mobile App Client:** this component is running on the mobile phone of the CG and may be used to potentially organize the schedule of the SP and send the related information to the Application Server Logic. Then it may be used to receive notifications from the COSMOS platform, in case of identified events. In case of need, the CG can also monitor in real time the location of the SP, which is relayed through the platform MB.
- **Application Server Logic:** this component is needed to handle organizational aspects of the application, such as topic creation in the MB, maintenance of the different SPs schedules, linking to the platform interfaces (e.g. CEP rules insertion and update), contact to Rbox to register the MB topic to which Rbox should forward the specific bus id's GPS etc. This logic may be created in Nodered, so that it can exploit template flows that will be offered by the COSMOS platform (e.g. template flow for CEP rule update, template flow for topic creation etc.).

Runtime of the scenario

1. The CG inserts the schedule for a specific SP, including bus line, start stop, end stop. This information is inserted and maintained in the Madrid Application Server side logic
2. The SP enters a bus, at which time his mobile phone contacts the Bus Services (through the Bus Wifi and utilizing the Bus API or SDK presented in Section 5.2.6). The Bus ID and line ID are retrieved and sent to the Application Server Logic for validation (or to the MB in case a specific CEP rule is in place). Alternatively the Bus SDK has also a direct call for understanding whether a specific person has boarded the bus.
3. The Application Server Logic retrieves the Bus ID on which the SP has boarded and contacts VEProt in order to enable the redirecting of this bus's GPS data on the application topic created. Now the bus real time GPS data are fed in the MB
4. The SP Mobile App retrieves its GPS and sends it in real time to the Madrid MB topic.
5. Through the CEP platform side component, a continuous comparison is made in the two locations. If a deviation that should not be there is found (e.g. prior to the arrival at the necessary stop), an event is triggered towards the Madrid MB topic, which is also picked up by the CG Mobile App.
6. Real time location monitoring of the SP is activated on the CG's Mobile App, enabling them to be aware of the latter's location. Optionally, messaging may be enabled to instruct the SP what to do.

7. SP's ETA is provided by the application, through the usage of the prediction services of COSMOS that will utilize historical data to create and train a relevant model and then offer the prediction API through a service front-end. This information may be used by the Application Server Logic, along with the real time data fed into the MB (e.g. Bus's GPS location) in order to send updates in the CG Mobile App, regarding updated ETA times.

COSMOS components and their specific concretization involved include:

- MB as a means to publish data from the various actors, as indicated in the previous paragraphs. Topic creation can be designed as a Nodered template flow, that may be configured by the application developer with minimal information
- CEP component to monitor and identify events of importance that regulate application behavior. CEP rules should be adjusted to the specific needs of the application (e.g. checking of bus ID against schedule and location, location of bus against SP etc.)
- Prediction component in order to create and apply a UC specific prediction model that will be used by the actors (in this case the CG) to obtain extended information (e.g. ETA of the SP). Machine learning models may use bus trip history as well as Madrid city traffic flow (density and velocity) and incidents reporting in order to correlate traffic incidents with other factors e.g. weather, time of day/week and use them to make predictions e.g. about traffic flow or traffic incidents

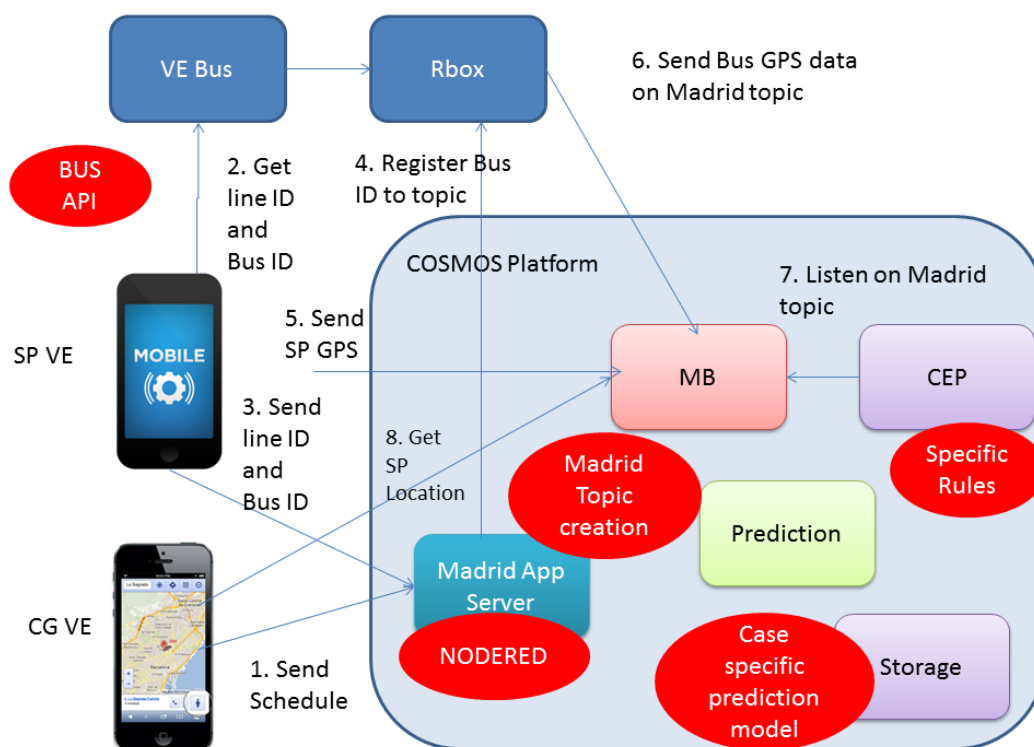


Figure 29: Madrid Assisted Mobility application runtime

As indicated in Section 5.2.6.1, the Bus API returns info on XML formats. Thus it is imperative that this format is processed in order to extract the necessary information and potentially transform it to JSON format for adaptation to the platform. Alternatively, the Bus SDK functionality may be used, in case the exposed functions fulfil the scenario objectives.

5.3.2. Application scenarios concretization (Camden Case)

The Camden scenario we are focusing on, is the Minimising Demand Scenario described in subsection 4.6 of the D7.1.2. Our approach involves the use of the COSMOS components in the flow of a specific COSMOS enabled Application, with the explicit purpose of specifying a Heating Schedule to fulfil the needs of the End User operating the Application.

The main components of the Scenario are:

- COSMOS enabled Application: This Application will be developed by an Application Developer and will run inside the Flat Gateway on top of the VE code. The Application will receive a desired temperature of the End User as well as the time period for the Schedule to be calculated and finally the budget of the End User.
- Flat VEs with relevant Cases: A group of Flat VEs, including the VE code running the actual Application, containing relevant Knowledge in the form of Cases. These Cases could be preprogramed, constructed through historical data, or constructed during the Application lifetime cycle, by utilising the CBR cycle.
- Physical Entities: The physical entities (Flats) containing the Gateways, sensors and the actuators for the Heating Management System.
- COSMOS platform: The COSMOS platform capabilities including Cloud storage and monitoring capabilities for gathering individual sensor readings of temperatures, actuations and energy consumption.

Runtime of the Scenario:

- Taking under consideration his/her budget, a user wants to set a heating schedule for his/her flat.
- The user plans a program, stating which is the desired temperature value for his/her flat for specific time intervals of the day/week etc.
- Because of COSMOS, the flat has a knowledge base of past schedules that can be used and thus it is able to estimate a) how the actuators should be used to achieve the best possible consumption (not necessarily the optimal one) and b) the needed budget.
- This knowledge comes from historical data derived from tracking the energy-behavior of the user.
- Even if the flat does not have a good schedule in its knowledge base, the flat will be able to ask other similar flats for help.
- The application **creates** one or more new **Problems** based on the End User input.
- The Planner searches in the Case Base for Cases with similar Problems and returns the **Solution**. If nothing is found, Experience Sharing is initiated.
- The user accepts the Solution or not and later he/she may provide feedback.
- The user does not have to act in an optimal way but just states his/her desires. Moreover, a prediction regarding the total budget needed for a schedule is provided by COSMOS from the very beginning.

COSMOS components and their specific concretization involved include:

The VE side components of the Planner, Experience Sharing as well as the Social Monitoring are used in order to facilitate the VE2VE communication needed in the flow of the scenario. The Planner is used in the implementation of the CBR technique of Schedule retrieval along with the Experience Sharing for remote Case acquisition. Social Monitoring is used for evaluating retrieved Solutions and providing local ranking for the serving VEs.

The platform based COSMOS Cloud used for the storing of historical data through the accumulation of sensor, actuator and consumption readings. This also includes the use of the Data Mapper component for the aggregation of retrieved data from the Flat VEs through the use of the EnergyHive platform in charge of monitoring Flat sensors of every type.

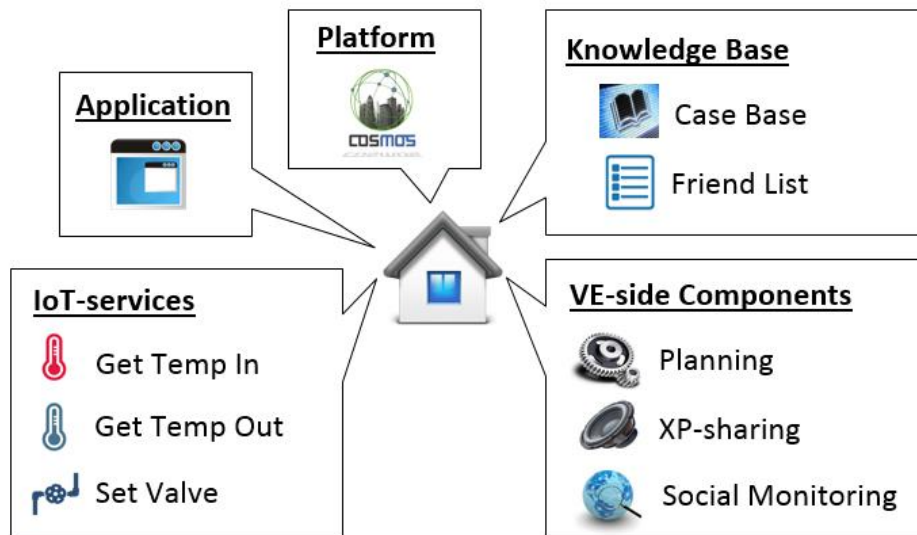


Figure 30: The Flat VE and its connections and characteristics.

5.4. Testing environment and roles

The testing environment at this stage is expected to comprise from COSMOS services, deployed at the COSMOS testbed as in the previous stages, but also from a VEProt test deployment as indicated by the scenario needs. Furthermore, a testing service simulating the Bus Wifi Services should be in place, in order to guarantee the scenario end to end testing.

Involved roles include the VE developers in order to test VE data being redirected to the COSMOS platform, along with application developers in order to create and deploy initial flows. Model developers are also expected to be engaged in order to create specific predictions based on the scenarios. Familiarity with Nodered is expected to kick in during this stage targeting at creating more combined flows in the following months.

In a nutshell the testing environment is expected to comprise of:

- The necessary COSMOS services running at the COSMOS platform side testbed
- The Application Logic servers running at the COSMOS platform side testbed
- An Android development platform for the Madrid Application SP and CG side
- A web based testing service to emulate the Bus SDK
- A raspberry Pi development platform for the planner component and a H/W board as defined in Section 4.2.4.1, as the generic, full fledged COSMOS VE. The planner is also used in the H/W board, however we include some tests to check out applying it to more limited capabilities devices, in case it can be used as standalone.

6. Integration Stage 4 (M26-M34)

6.1. Involved Integration Goals and Refinement

Following the results of the previous integration period, we expect at this stage to strengthen integration, mainly in terms of continuing and enriching previous goals and results. Initially the involved integration goals for this stage include the continuation of the COSMOS platform setup and cross component integration, especially with relation to the following subgoal:

- Integrated Web UI for COSMOS roles: The implementation and integration of a tool that will abstract the various COSMOS stages related to the various roles will be investigated. This tool should provide a front-end for these roles and guide them through the completion of the necessary steps.
- Integration with relation to new functionalities that emerge from the platform side, pending the update in Y3 architecture

With relation to the VE description and linking to the platform the following subgoal will be completed:

- VE Instances Descriptions and Registry population, mainly for the case of DSOs of smart buildings (Taipei case).

With relation to the VE side components integration, the process will continue for the finalization of the subgoal:

- Components installation to VE instances: in this case we will pursue the deployment of all VE side components, potentially through more automated ways (including Nodered flows)

For the Application Definition, Creation and Deployment goal we anticipate extensions with regard to the following subgoals:

- Complex applications creation with reusable flows: in this case the use of pre-existing flows and templates will be pursued in order to enhance and ease application development
- Application archetypes potential extension: in this case, by investigating the remaining application scenarios, we will examine whether the defined archetypes can be extended in order to cover more typologies of applications through the combination of defined subsystems (as done in Section 4.2.5), or the possibility to include new extensions to the platform functionalities.
- Continuation of the concretization of the application scenarios with relation to the COSMOS components, including specific instantiations of components such as the Planner, CEP rules, storlets implementation and prediction modelling.

More details on this period will be gradually included in this deliverable as a living document. The final pending period of M35-36 will be included and refined in the next iteration of this document in Y3.

6.2. Subsystems/Subgoals involved

6.2.1.1 Components installation to VE instances

During this period we will pursue easier VE component instantiation in order to facilitate the extension of functionality similar to how normal software components are installed. Focusing on the existing use-case scenarios we will investigate the deployment mechanisms needed to provide use-case specific VE's given an existing hardware platform.

During this period we will pursue the installation of more VE components such as Pattern detection and Social Monitoring. The final tests will be performed at a platform level where all components will be checked individually but also against the use case/test case scenarios.

6.2.1.2 VE Instances Descriptions and Registry population

Following the preparatory work in the previous period with regard to this subgoal, in this period the instantiation of concrete VEs for Taipei is expected, to complete the VE registration subsystem. The concrete subsystem along with the integration points has already been defined in the previous chapter (Figure 14 of Section 4.2.1.2).

6.2.1.3 Complex Applications creation and deployment- Nodered template flows

This involves the same subsystem identified in Section 5.2.1 (Figure 20), in which however the template flows have been grouped as Nodes, thus hiding and abstracting the internal workings from the developers, as indicated by the red box in Figure 15. Furthermore, the creation of shareable repositories, potentially per category of roles or per type of flows for better grouping, will be examined. In this process the identified integration points per role (1-4 as indicated in Section 2) can also aid in the grouping.

6.2.1.4 Application archetypes potential extension

We chose more complex scenarios for the Y2 integration process in order to investigate the limitations of the COSMOS concepts, potential gaps and necessary improvements in the process that may be identified early and corrected in Y3 iteration. However not in all cases the application templates may be that extensive and complicated, as identified in 4.2.5. For example, application developers that have only the partial roles of Analytics Developer or Model Developer or Domain expert (for CEP), as identified in Figure 1, may need only a subset of the functionalities provided. Thus new archetypes of simpler forms may be created.

Potentially other combinations may be achieved, like decentralized applications that use also application specific logic on the VE side (through a suitable Nodered deployment flow), that may be deployed and operating on the device.

6.2.1.5 Linking external platforms/services (Optional)

One interesting aspect that may be pursued is also the ability to include external services or platforms feeding information or data to the COSMOS platform. This would imply the ability to combine information from multiple sources and thus achieve any possibility for service combination. This is an optional step that may be investigated in case there is a specific scenario to which it can provide added value, that is why it is not included in the main Gantt chart of integration actions.

6.2.1.6 COSMOS Process Web UI

In this period the main functionalities of the COSMOS Process Portal are expected to be implemented, following component maturity and the availability of their specific GUI pages (Figure 31). Potential frameworks for implementation will also be investigated (e.g. Spring, Twitter Bootstrap etc.), focusing on modularity and ease of extension. In this case, the identified integration points per role in this document (IPs 1 to 4) may significantly help us in identifying groups of actions that need to be addressed by each role and per phase (e.g. all IP1 points are for runtime of the application where end users are involved, all IP2 points are for design time for the application developer etc.).

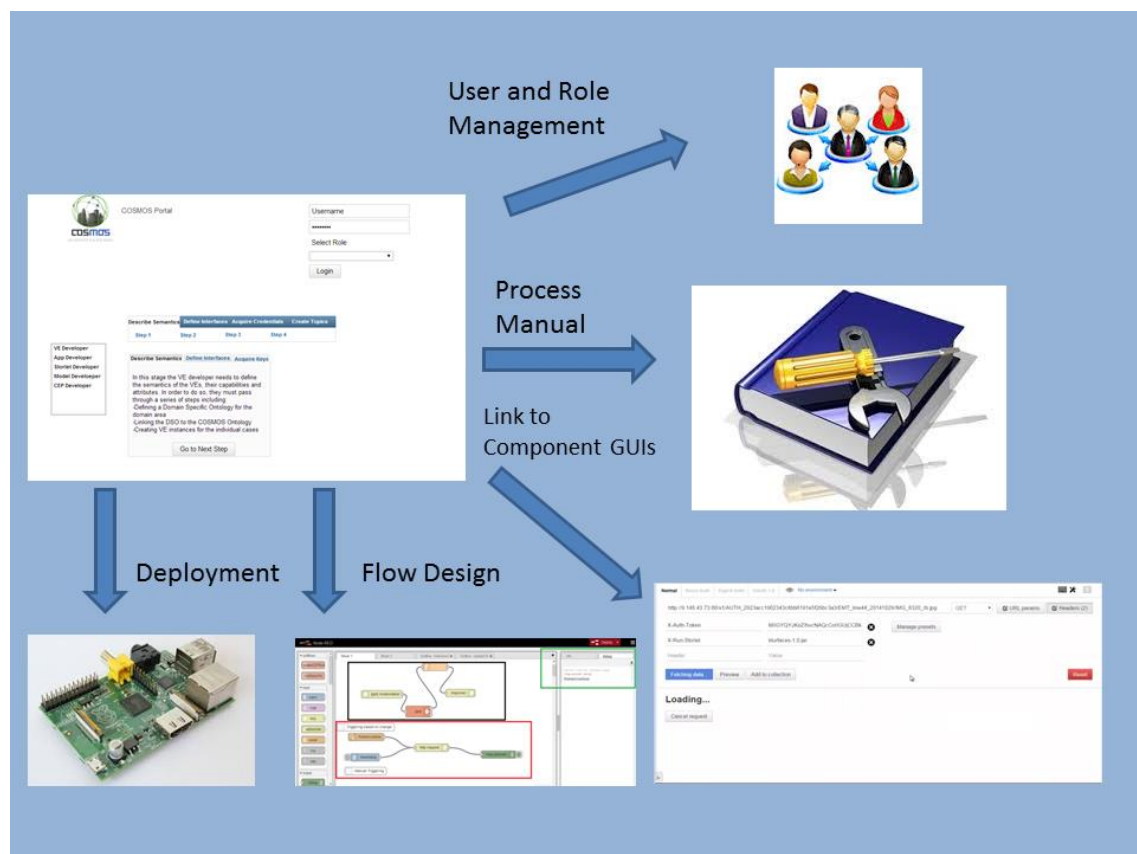


Figure 31: Web UI expected functionalities

6.3. Use Case specific aspects involved and concretization

During this period, the focus will be given to the formulation of the UC concrete aspects that refer to the remaining UC scenarios that will have not been implemented so far. This implies specific CEP rules again for a scenario, specific prediction models that may be necessary based on the scenarios etc, similar to the examples demonstrated in this document for the previous period (Section 5.3).

6.4. Testing environment and roles

Testing roles include application developers in order to test how template flows may be acquired from the repository, instantiated and used during runtime. Furthermore, VE developers will be engaged in the Web UI testing. Towards the end of this period we expect also to involve end users, in order to test the created applications. These tests will mainly center around usability of the interfaces and GUIs, understanding of the operations and clarity of the needed actions.

In a nutshell, testing includes:

- Application developers for interaction with repository and stored flows retrieval, configuration and extension
- Deployed Web UI and relevant component UIs to be tested by Application and VE developers
- The H/W board as defined in Section 4.2.4.1, as the generic, full fledged COSMOS VE.
- End users, at least for application usability, for the applications that have been developed so far. Feedback may be acquired through web simulators, Google Forms questionnaires etc.

7. Traceability Matrix of Capabilities to Use Cases

A traceability matrix is included in order to maintain a mapping between the COSMOS functionalities and how these could be applied to the different UCs (Table 5). This mapping is defined based on an initial investigation of the application Use Cases, as these are defined through 7 application scenarios in D7.1.2. The mapping is based on the current description of the scenarios and on potential capabilities that can be adapted to fit their needs and seem reasonable to be included in the envisioned application. Every capability or component should be present in **at least** one of the scenarios, by the end of the project. The actual applicability will be evaluated in D7.7.X document series.

Table 5: Traceability Matrix of capabilities to UC scenarios

	UC 1 Capital Plannin g	UC 2 Minimis in g Carbon	UC 3/5 Heating Control/Minimiz in g Demand	UC 4 Mobility Assistanc e	UC 7 Identification of Opportunitie s	Taipei Occupanc y Detection
Data Annotatio n and Indexing	✓	✓	✓	✓	✓	✓
Data ingestion with security	✓	✓	✓	✓	✓	✓
Real time feed		✓	✓	✓		
Storlets	✓			✓	✓	
Data Analytics	✓			✓	✓	✓
CBR- Planning		✓	✓	✓		
Experienc e Sharing			✓	TBD		
Privelets		✓	✓	✓		
Predictive Modelling	✓	✓	✓	TBD	✓	✓

CEP+SA	✓		TBD	✓		
Semantic Integration	✓	✓	✓	✓	✓	✓

This information will be updated also in the next versions where application scenarios may be enriched, thus allowing the adaptation of the features, including also the Taipei UC. The entry TBD (To Be Discussed) implies that the specific capability may or may not have meaning to be included in the specific scenario, thus its applicability should be further investigated. Some cases of mappings do not refer specifically to a scenario, for example the semantic integration, once done for a specific VE type, it is automatically existent in all scenarios coming from the specific UC.

For the purposes of Y1, the main focus was given on the integration of the COSMOS platform components and their initial prototypes, along with the ingestion of data from the UCs.

During Y2, the main focus will be on creating the framework that will incorporate mainly the Heating Schedule UC and the Madrid Assisted Mobility UC. Furthermore, initial investigation of ingestion of Taipei historical data will be targeted as well as incorporation of it to the occupancy modelling and identification scenario that was achieved in Y1 with the Surrey Smart building data.

8. Conclusions

In order to proceed with the integration process in the context of the COSMOS project, a necessary plan needs to be in place. The main aspects of this have been highlighted in this document, that will act as a guide also for the results reporting and presentation.

By dividing the process into more fine grained goals and subsystems, we can gain enhanced perspective as to the various aspects that need to be included in the platform, having in mind at all times the progress towards the final target, the COSMOS enabled application. By keeping also track of the UC involvement in each period, we can identify gaps and inconsistencies across the technical WPs and the UC scenarios and optimize their mapping and relationship.

From this updated version of the document, the integration process of COSMOS has proceeded, taking concrete aspects of the application scenarios into consideration. Furthermore, the needs of the immediate period following this plan have been identified, in terms of concrete subsystems for integration and scenario implementations. Relevant roles and integration points have been identified, that may aid us in enhancing aspects of the platform such as usability, end to end functionality. Furthermore, the iteration of the document aided in identifying two subgoals that produce more fine grained division of the tasks. Significant outcomes are also the usage of Nodered for the purposes of the WP as well as the highlighting of actions that are necessary up to the next iteration of this document in Y3. New subsystems have also been examined, including the ones coming from the updated platform architecture, and their combination has aided in creating application archetypes that can serve as templates for application developers. This process is expected to iterate in the following year in order to produce more such indicative usages of the COSMOS components, derived directly from the described application scenarios, as has also been performed during this period.

Thus in the following integration periods we can check and keep track of the technical developments according to the goals highlighted here. It must be stressed that this intends to be a living document, that will be continuously updated based on the technical discussions and the goals within each period.

Annex A Components Involved

- **COSMOS platform**
 - **Nodered Environment**

Table 6: Software requirements for the Nodered Environment

Name	Version	OS
Node.js	0.10.4	Linux
Nodered	0.10.36	Linux
Java Runtime Environment	1.7	Linux/Windows

- **Data Mapping**

Table 7: Software requirements for the Data Mapping

Name	Version	OS
Java Runtime Environment	1.8	Linux/Windows
RabbitMQ server	3.3.1	Linux/Windows
Swift All In One	1.12.0	Linux/Windows

The Data Mapping will be distributed as a NetBeans 8.0 project folder zip.

- **Message Bus**

Table 8: Software requirements for the Message Bus

Name	Version	OS
Erlang	R15B	Linux/Windows

- **Cloud Storage-Metadata Search**

Table 9: Software requirements for the Metadata Search

Name	Version	OS
Ubuntu	13.10	Linux, 64 bit
RabbitMQ server	3.0.2 or higher	Linux
Elastic Search	1.2.0	Linux
Python	2.7	Linux
OpenStack Swift	1.12.0	Linux

- **Cloud Storage- Storlets**

Table 10: Software requirements for the Storlets

Name	Version	OS
Ubuntu	13.10	Linux, 64 bit
LXC	Part of Linux kernel	Linux
Python	2.7	Linux
OpenStack Swift	1.12.0	Linux
Java JDK	7	Linux

▪ Event Detection and Situational Awareness

Table 11: Software requirements for the Event Detection at the COSMOS platform

Name	Version	OS
ZeroMQ	4.0.4	Linux/Windows
Apache Tomcat	7.0.54	Linux/Windows
JDOM	2.0.5	Linux/Windows (JRE)
Jersey	2.10.1	Linux/Windows (JRE)

▪ Prediction

Table 12: Software requirements for the Event Detection at the COSMOS platform

Name	Version	OS
Ubuntu	13.10	Linux
Python	2.7	Linux

▪ Semantic Description and Retrieval

Table 13: Software requirements for the Semantic Description and Retrieval

Name	Version	OS
Ubuntu	13.10	Linux, 64 bit
JBoss Application Server	7	Linux
OpenRDF Sesame	2	Linux
Java JDK	7	Linux

• VE Level

▪ Privelets

Name	Version	OS
Java Runtime Environment	1.8	Linux/Windows
Jetty-Maven-Plugin	9.1.5.v20140505	Linux/Windows
Apache-Jena	2.10.0	Linux/Windows
Pellet-Jena	2.3.2	Linux/Windows
FreeLan	1.1	Linux/Windows

Table 14: Software requirements for the Privelets

▪ Planner

Table 15: Software requirements for the Planner

Name	Version	OS
Java Runtime Environment	1.8	Linux/Windows
Apache-Jena	2.10.0	Linux/Windows
Pellet-Jena	2.3.2	Linux/Windows

Planner will be distributed as a NetBeans 8.0 project folder zip.

- **Event Detection and Situational Awareness**

Table 16: Software requirements for the Event Detection at the VE level

Name	Version	OS
ZeroMQ	4.0.4	Linux/Windows

- **Experience Sharing**

Table 17: Software requirements for the Experience Sharing

Name	Version	OS
Java Runtime Environment	1.8	Linux/Windows
Jetty-Maven-Plugin	9.1.5.v20140505	Linux/Windows

Experience sharing will be distributed as a NetBeans 8.0 project folder zip.

Annex B Project Computing Testbed Details

• Basic Requirements

▪ Accessibility

The COSMOS platform consists of components that span across virtualised infrastructure, framework services and application layers. This makes the components integration a difficult task that requires physical and remote access for the developers to the integration sites and to all platform layers. It is also important, at least for the integration purposes, the developers to have access in the virtual environment to check the correct deployment and execution of application components. To this direction, COSMOS partners are expected to use a Web Client through SSH in order to remotely connect to the VMs.

▪ Security

VMs should be accessible over the Internet through a secured connection. More specifically, access control and firewall need to be incorporated to isolate the infrastructure from the outside world and guarantee its normal operation. To this direction, HTTPS using port 443 can be adopted.

• Components to Virtual Resources Mapping

The component to VM mapping that has been chosen for the COSMOS Platform appears in Table 18, along with the network configuration necessary for the VMs.

Table 18: Components to VMs mapping and VM configuration requirements

Virtual Resource (VM ID)	Components Included	WP(s)	#Cores	RAM	Disk	Connectivity Requirements (opened ports)	Hypervisor requirements (if any)
1	VE1 (Planner, Experience Sharing)	WP5, WP6	1	1GB	8GB	3030, 8080, 5050	None
2	VE2 (Planner, Experience Sharing)	WP5, WP6	1	1GB	8GB	3030, 8080, 5050	None
3	Swift, Storlets requirements for first year of project	WP4	4	8GB	500GB	8080, 9200, 5672, 5000, 6000, 6001, 6002, 873	None
4	Swift, Metadata Search –	WP4	4	8GB	500GB	8080, 9200, 5672, 5000, 6000, 6001,	None

	requirements for first year of project					6002, 873	
5	Event Detection	WP4, WP6	1	1GB	4GB	50100, 8080, 50101, 50102	None
6	Message Bus	WP4	1	2GB	4GB	5672	None
7	Semantic Description and Retrieval	WP5	1	2GB	4GB	8080, 9000, 9990	None
8	Nodered Environment	WP7	1	2GB	4GB	1880	None

• Testbed Description

The hardware infrastructure is provided by Atos. Atos is responsible for assuring a constant access and a continuity of the services that guarantee the correct functioning of the physical infrastructure. The Infrastructure details are the following: 1HP DL180G5 / 2x Intel Quad-Core Xeon L5410 / 24GB RAM/ 4x1TB SATA. These physical resources must accomplish individual work packages needs.

The system is virtualised in order to develop the different functionalities following an isolated VMs strategy. This option has been chosen by the different partners; the argument is that the individual development and deployment of the different modules can be controlled by the developer directly and it does not affect the parallel development of other functionalities e.g. restart VMs, or conflicting requirements. This may also aid in improved packaging and release in the end of the project, in terms of Virtual Appliances with pre-installed COSMOS components.

Annex C Software Packaging and Delivery

This section describes a set of recommendations for developers to ease the compilation and deployment of components in the testbed. The COSMOS project uses SVN [4] as repository to share the software developed within the COSMOS scope. A methodology has been defined to make easier the deployment of software in the different site machines. This methodology includes recommendations related to build tools and packaging software.

• Installation - Execution

The installation of the components in the test-bed may be performed in a variety of ways, however the goal should be to have a, as automated as possible, process. Indicatively, the following ways may be applied:

- Standard package formats may be selected for Linux and Windows:
 - Linux Operating System. For these machines, Ubuntu 13.10 has been chosen as the main COSMOS Linux distribution. For the modules developed for Linux, every module may include a ".deb" package. This will be the installation point of the binary of any component delivered. The naming convention followed should be: eu_cosmos_<wp>_<component-name>.deb. The description of any dependencies will be present during the building of the .deb file so the apt-get command will be able to resolve and download any missing dependencies. [6]
 - Windows Operating System. For these machines, Windows 7 or superior version can be used as COSMOS Windows version. The components developed for Windows must be delivered as MSI (Microsoft Installers) [7] or EXE (Windows Executable Installers) [8] files. Any custom or necessary libraries should be included in the .msi/.exe file by the developers. Again the naming conventions mentioned in the Linux distributions are valid so any msi or exe naming should be: eu_cosmos_<wp>_<component-name>.msi/.exe
- Java-based programs should be provided as executable jar files, including helper scripts that may aid in the configuration of the installation and execution
- Components based on different programming languages should also provide helper scripts for the installation and execution, in the according language of implementation. Additional scripts may be provided for preparing the testbed for the deployment of these components (e.g. installation of dependencies etc.)

When installation has taken place, the packages will have to provide some way for the user to execute them. Whether these packages are COSMOS components or VE_level components, during test-bed testing they should provide some way of seamless execution. Naming conventions may be applied for the aforementioned scripts. For example:

- Prepare_<component-name>.*
- Install_<component-name>.*
- Execute_<component-name>.*

Any form of installation should also necessarily provide a basic configuration file that will be accessed at the start of execution, as a script parameter and provide necessary alterations to the executed program if needed.

Indicative parameters for the scripts may include:

- help: shows help about the usage of the component.

- **configure:** performs any configuration needed prior the execution of the component (optional).
- **start:** starts the component's execution.
- **stop:** stops the component's execution.
- **clean:** frees resources and resets the state of the component after its execution (optional).

Obviously, these scripts will be created, if possible, only for those cases where it makes sense to do it. That is, there could be some components which are started inside a server, in the moment this server starts running, so no start script is needed in this case.

• **Standard Naming Convention**

A standard naming convention will be used for all phases of package development. From source code naming to installation package creation the naming convention is eu.cosmos.<location>.<wp>.<tool-name>.<component-name>, except where any other convention is explicitly mentioned. Further analysis follows:

- **location:** CosmosPlatform, VELevel
- **wp:** security, datamanagement, thingsmanagement, thingsanalysis
- **tool-name:** (optional)
- **component-name:** e.g. Planner, CEP

It is very important that each <> contains **no white spaces**.

• **Standard Readme File**

Every software package should contain a "README.txt" file. This file should contain, at least, the following information:

- Name of the software package.
- Responsible person for the software component.
- Dependencies.
- Configuration instructions.
- Path to the log files produced by the component (if any).

• **Standard License File**

Every software package should contain a "license.txt" file, indicating the applicable license for the specific package and details of usage and distribution permissions.

• **Manuals**

Developers are also strongly encouraged to provide an installation and configuration manual and a user manual for the components they are responsible for. Indicatively, sections for such a documentation can include:

- **Implementation:**
 - **Functional description:** This section describes the overall purpose of the delivered prototype. It must include the context and scope of the prototype; the motivation and main innovations.
 - **Fitting into overall COSMOS solution:** This section describes how the prototype fits into the overall COSMOS chain from a functional point of view. How it is mapped into the COSMOS methodology and how it is related with other components. How it is mapped into the overall COSMOS architecture.
 - **Technical description:** This section describes the technical details of the implemented software.
 - **Prototype architecture:** This section contains a diagram and a description of what is the architecture of components that build up the prototype.
 - **Technical specifications:** This section contains details about programming language, libraries, databases, application servers and so on required for the implementation of the prototype
- **Delivery and usage:**
 - **Package information:** This section describes the structure of the delivered package (folders and files).
 - **Installation instructions:** This section describes the steps that must be followed to install and start up the prototype as well as how to execute the software.
 - **User manual:** This section provides details how to use the prototype.
 - **Licensing information:** This section specifies under which licence the prototype or components inside are delivered.
 - **Download:** This section specifies the path where the source code is available.

• Acceptance Procedure

Developers are advised to follow certain rules to deliver their components to WP7:

1. The source code of the components should be submitted into the subversion repository and tagged with the version of the component (this only applies to the open source code developed during the project). If the code cannot be distributed the binary parts of the component should be available in Subversion or in the relevant testbed facilities for demonstration/integration purposes.
2. The installation scripts should also be available in the repository with instructions to utilize them. Configuration information should also be provided.
3. The requirements for the installation should be clearly defined in the README file of the component.
4. The components should have been passed the unit tests defined inside the WP for these components. A testing report must be available shown the tests performed, especially on interactions with other components, relevant to the test cases defined in Annex A.

- **Integration Tools**

- **Revision Control System**

Revision Control Systems are widely adopted, operating as repositories for storing and maintaining the design and specification documentation, as well as the source code and configuration files of the software under development. The most important advantage of these systems is that they can support multiple partners working simultaneously on different versions of the same document or software. In that way, any bugs and other issues can be easily located and fixed while at the same time new stuff can be added. Additionally, in a project like COSMOS where the development teams are geographically dispersed, revision control improves the development process and the communication between the development teams.

In COSMOS, Subversion (SVN) [4] is used, which is nowadays one of the most popular and complete, in terms of features, open source revision control systems.

- **Alfresco**

Alfresco [5], which is an open source ECM system, is used to manage the project's critical documents like CA, DoW, GA, deliverables released to EC etc. Alfresco has the advantage of providing the COSMOS partners with full access from anywhere and at any time.

- **Wiki**

For the purposes of integration, we have setup a Wiki in which integration information may be included in order to document testbed configuration, mapped IPs, VM names, access credentials etc.

- **Code Quality checks**

COSMOS partners will investigate the possibility to use Sonar [1], which is an open platform to manage code quality and extract useful conclusions. Extracted information will be fed back to developers in order to improve the quality of their code, especially in the case of the code base that is going to be released as open source.

In terms of languages, Sonar supports analysis of Java in the core, but also of Flex (ActionScript 3), PHP, PL/SQL and other languages through plugins (Open Source or commercial) as the reporting engine is language agnostic. A full list of available plugins can be found in [2].

According to [3], Sonar enables to cover quality on 7 axes and so to report on:

- Duplicated code
- Coding standards
- Unit tests
- Complex code
- Potential bugs
- Comments
- Design and architecture

- **Template Adaptation and Validation**

The defined COSMOS template (in JSON format) needs to be validated against the produced data feeds that are expected to be ingested in the platform. Relevant tools for this validation are expected to be used, e.g. as the ones listed in [9].

References

- [1] Sonar tool: <http://www.sonarqube.com/>
- [2] SonarQube Documentation, Plugin Library List, available at: <http://docs.codehaus.org/display/SONAR/Plugin+Library;jsessionid=48B59953A92269D9938CA1751951ED36>
- [3] Method & Tools: <http://www.methodsandtools.com/tools/tools.php?sonar>
- [4] Subversion: <http://tortoisesvn.tigris.org/>
- [5] Alfresco : <http://www.alfresco.com/>
- [6] Debian Packaging: <https://wiki.debian.org/IntroDebianPackaging>
- [7] MSI: <http://msdn.microsoft.com/en-us/library/aa266427%28v=vs.60%29.aspx>
- [8] EXE: <http://msdn.microsoft.com/en-us/library/ff553615.aspx>
- [9] JSON Schema Organisation, Available at: <http://json-schema.org/implementations.html>
- [10] COSMOS Project Deliverable D7.1.1 Use Case Scenarios Definition and Design (Initial)
- [11] Nodered Tool, Available at: <http://nodered.org>
- [12] Extended libraries of Node Red, available at: <http://flows.nodered.org>
- [13] COSMOS Project Deliverable D7.7.1 Integration of Results (Initial)
- [14] COSMOS Project Deliverable D2.3.2 Conceptual Model and Reference Architecture (Updated)
- [15] COSMOS Project Deliverable D3.1.1 End-to-end Security and Privacy: Design and open Specification (Initial)
- [16] COSMOS Project Deliverable D3.1.2 End-to-end Security and Privacy: Design and open Specification (Updated)
- [17] COSMOS Project Deliverable D7.1.2 Use Case Scenarios Definition and Design (Updated)
- [18] RabbitMQ Messaging system: <http://rabbitmq.com>
- [19] Apache Kafka Messaging system: <http://kafka.apache.org>