COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D7.6.1 Integration Plan (Initial)

WP7: Use cases Adaptation, Integration and Experimentation

Version: 1.5
Due Date: 30/06/2015
Delivery Date: 15/02/2015
Nature: Report
Dissemination Level: Public

Lead partner: NTUA
Authors: George Kousiouris, Achilleas Marinakis, Panagiotis Bourelos, Orfefs Voutyras (NTUA), David Jorquera, Jozef Krempasky (ATOS), Paula Ta-Shma (IBM), Adnan Akbar (UniS), Bogdan Târnaucă (Siemens)

Internal reviewers: Juan Sancho (ATOS), Achilleas Marinakis (NTUA),

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	10/7/2014	Achilleas Marinakis	NTUA	Circulating the ToC, adding initial content
v0.2	24/7/2014	Achilleas Marinakis and co-authors	NTUA, ATOS, IBM, UniS, Siemens	Version ready for internal review
v0.3	25/7/2014	Achilleas Marinakis	NTUA	Final version based on the review from ATOS and NTUA
v1.0	25/7/2014	Achilleas Marinakis	NTUA	Version for submission
RE_SUBMIT v1.1	22/1/2015	George Kousiouris	NTUA	Updated ToC
RE_SUBMIT v1.2	26/1/2015	George Kousiouris	NTUA	Updated Strategy and Plan
RE_SUBMIT v1.3	7/2/2015	George Kousiouris	NTUA	Extended content of integration periods
RE_SUBMIT v1.4	12/02/2015	Achilleas Marinakis	NTUA	Internal Review
RE_SUBMIT v1.5	13/02/2015	Juan Sancho	ATOS	Final Review

Table of Contents

Table of Figures

List of Tables

Acronyms

Acronym	Meaning
CBR	Case-Based Reasoning
CEP	Complex Event Processing
D	Deliverable
DoW	Description of Work
GA	Grant Agreement
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IoT	Internet of Things
IP	Integration Point
JSON	Java-Script Object Notation
MS	Milestone
OS	Operating System
REST	Representational State Transfer
SSH	Secure Shell
SVN	Subversion
UC	Use Case
UTM	Universal Transverse Mercator
VE	Virtual Entity
VM	Virtual Machine
WP	Work Package

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 14 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 D7.7.1 “Integration of Results” document.

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 D7.7.1 document, in terms of necessary content and type of information. The initial envisioned integration period details have been included in this document for period 1 (PM10-16). The remaining periods will be further refined in the next iterations of this document.

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.

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.

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.

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

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). During this process it engages a number of roles, whose interaction with the system should be the main focus of integration. 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).

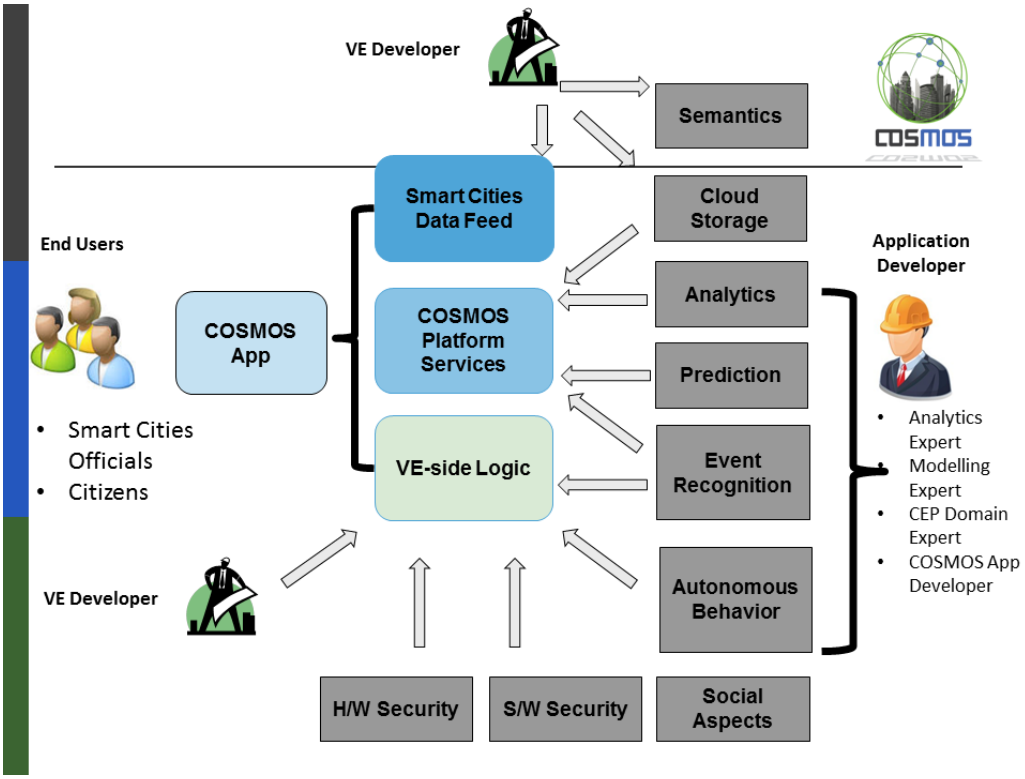

Figure 1: COSMOS Vision. Functionalities and Roles

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. 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 Integration of results) for their incorporation. 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.

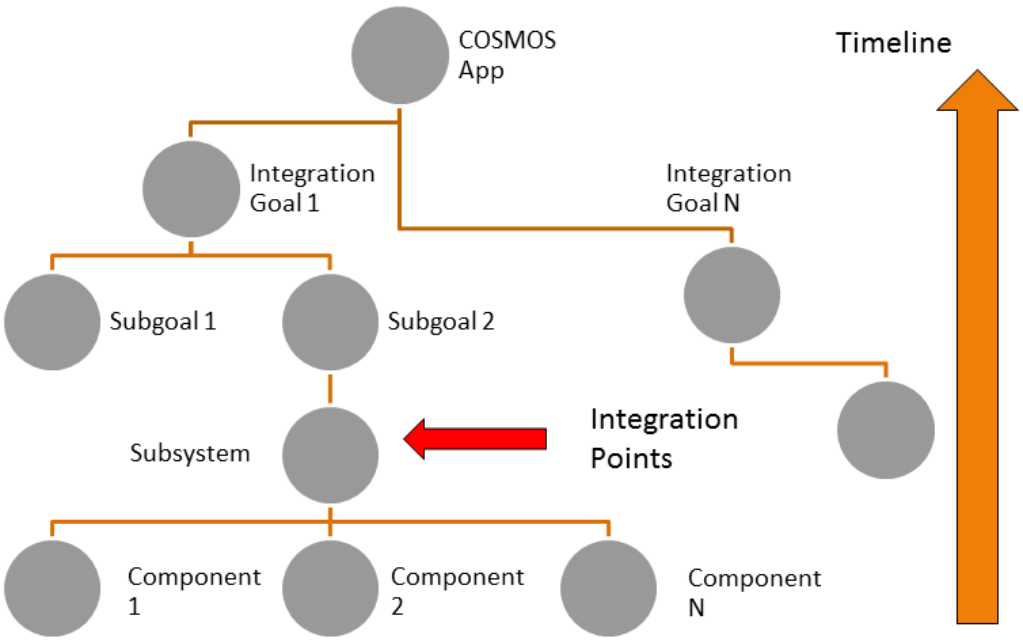

Figure 2: COSMOS Integration Strategy Overview

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 UC 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 that is understandable and usable by the various components and applications. This does not refer to the format 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 dependencies from the UC side, in order for a VE-side COSMOS component to be included directly in the VE. 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 years of the project. However, the ability to have the framework for at least defining preliminary applications should be existent 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 corresponds to Milestone MS8 of the project. This does not include data format aspects, since these have been defined early in the project to be JSON-compatible.
- 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

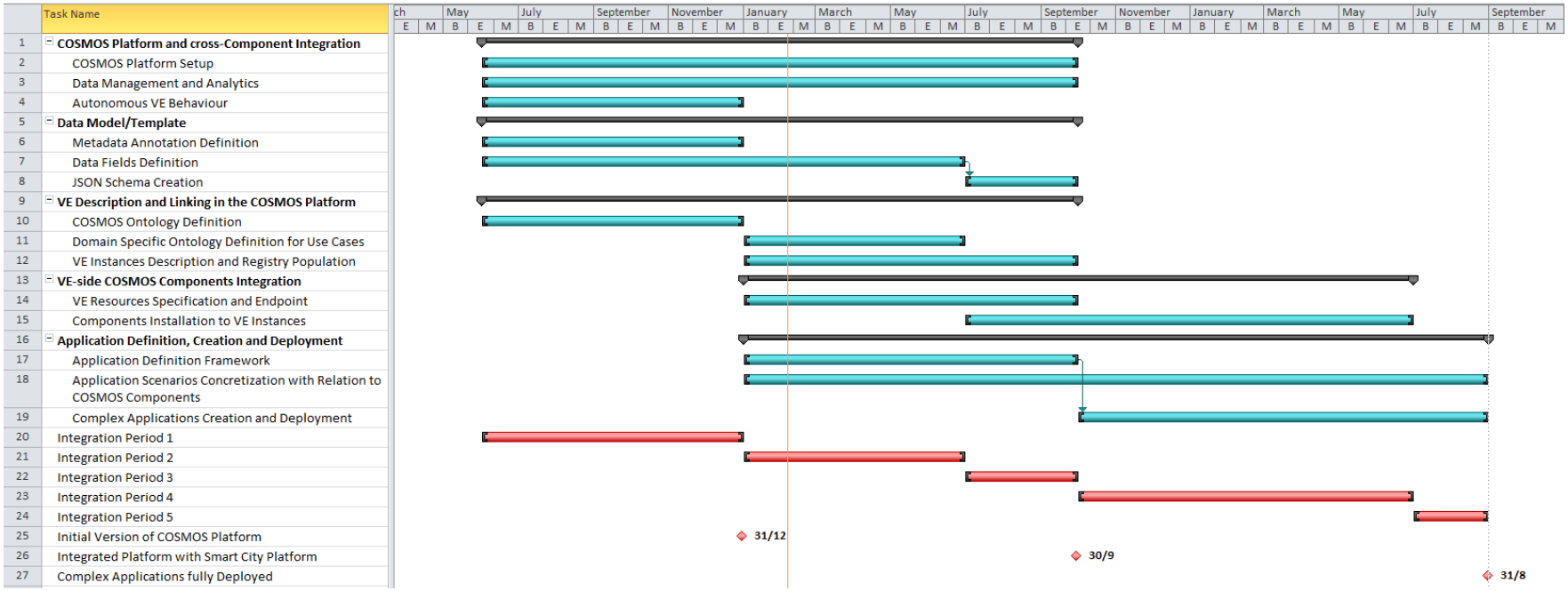

Figure 3: COSMOS Integration Timeline Gantt Chart

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 paragraphs, an analysis of these 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 overall 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. This functionality is expressed mainly as a component diagram (in this document) and as a sequence diagram (in D7.7.1), including the cooperation of more than one components that form a 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

The components to be included in the subsystems examined in the specific period are listed. Before the components are included in the platform, they must be tested in order to verify their behaviour. 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 (D7.7.1) 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

The infrastructure that is needed in order to proceed with the deployment and usage of the components. 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 etc.

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.

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.

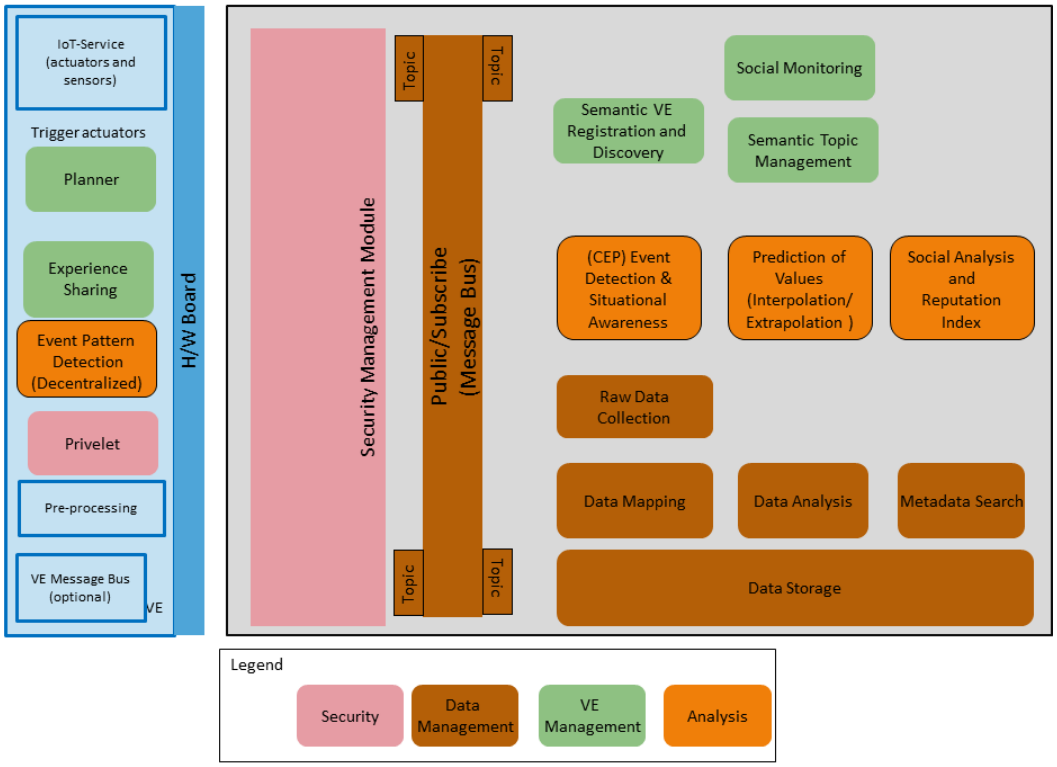

Figure 4: Overall COSMOS Architecture

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.

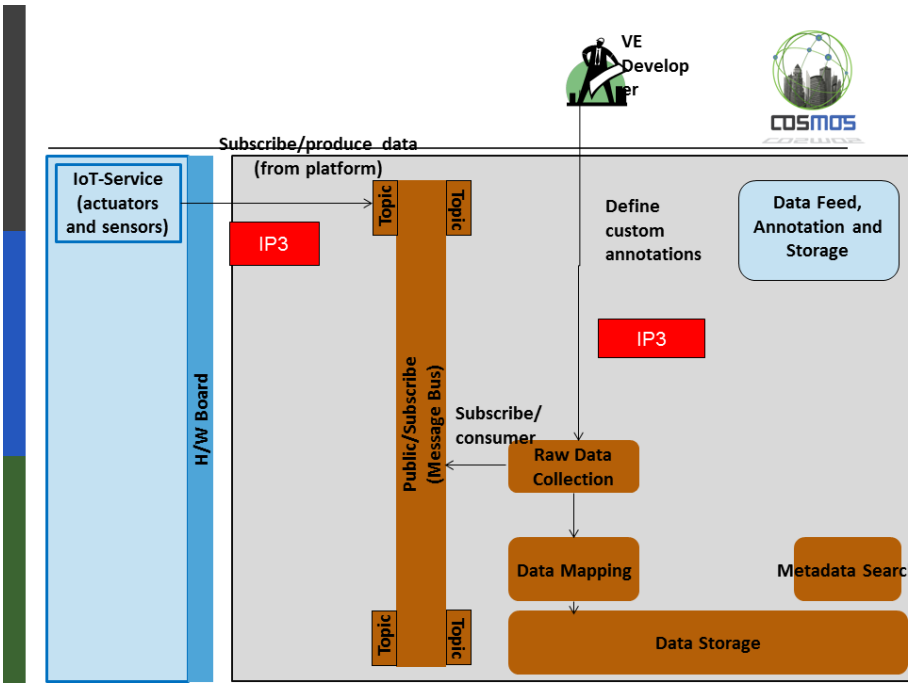

Figure 5: Data Feed, Annotation and Storage Subsystem

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.

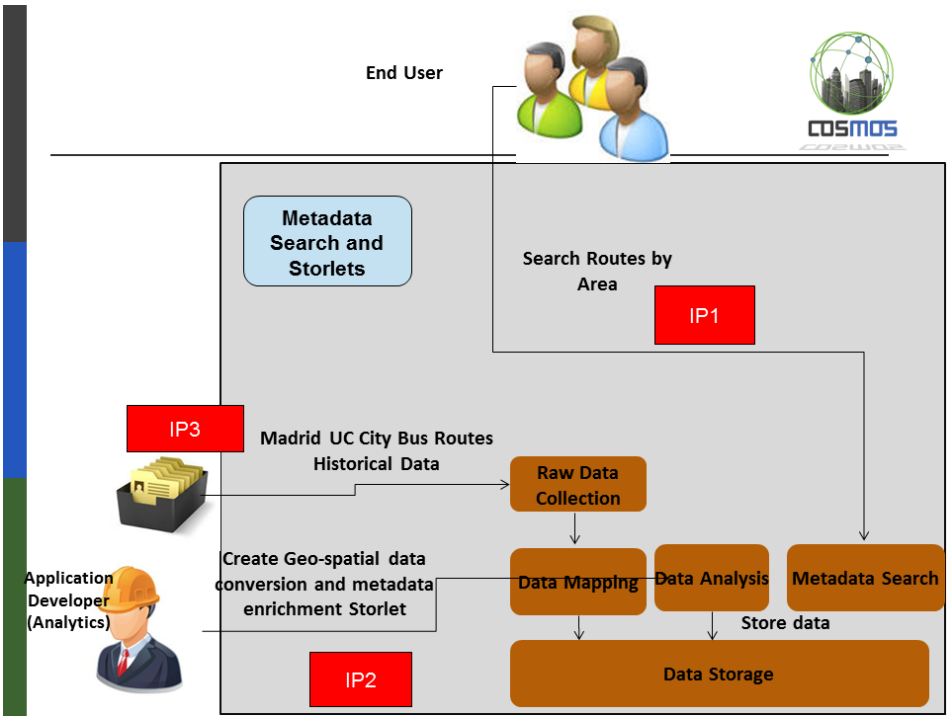

Figure 6: Metadata Search and Storlets Subsystem

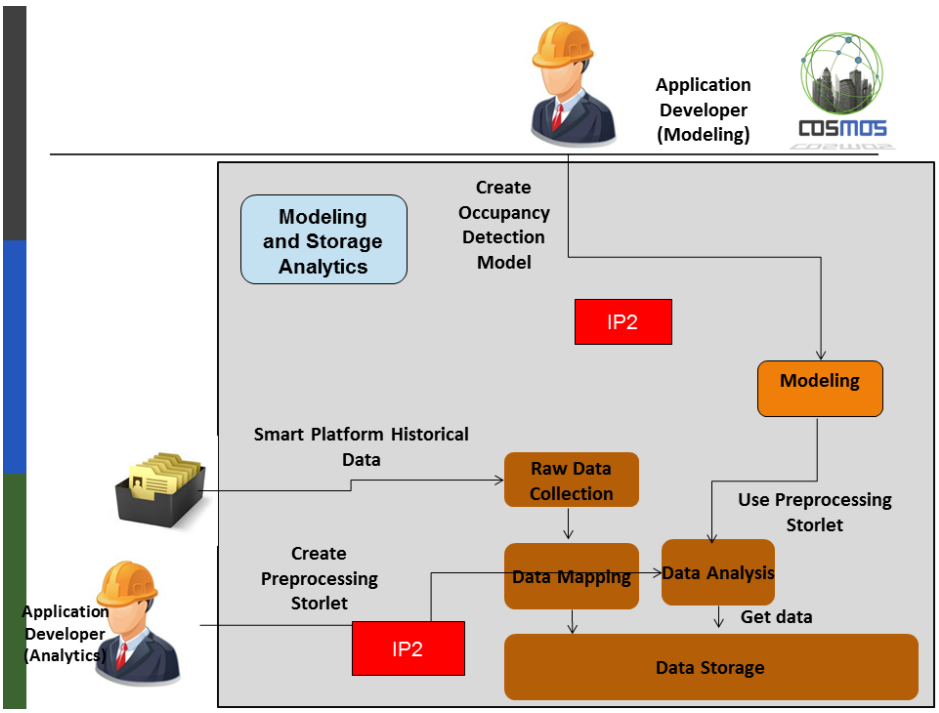

Figure 7: Modelling and Storage Analytics Subsystem

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.

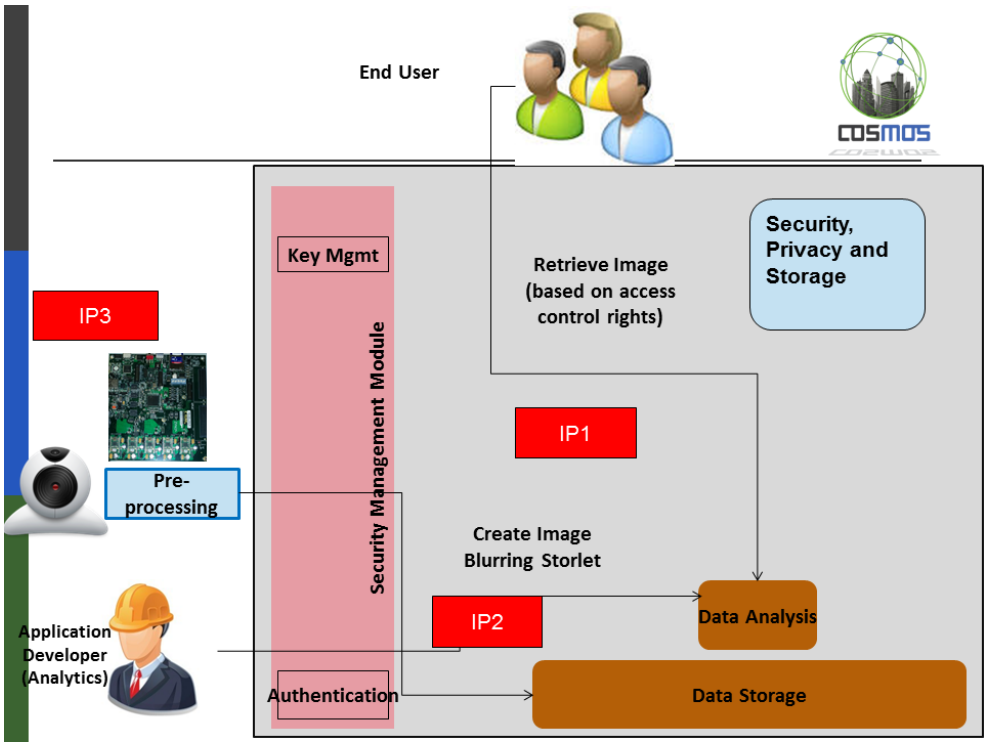

Figure 8: Security, Privacy and Storage subsystem

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).

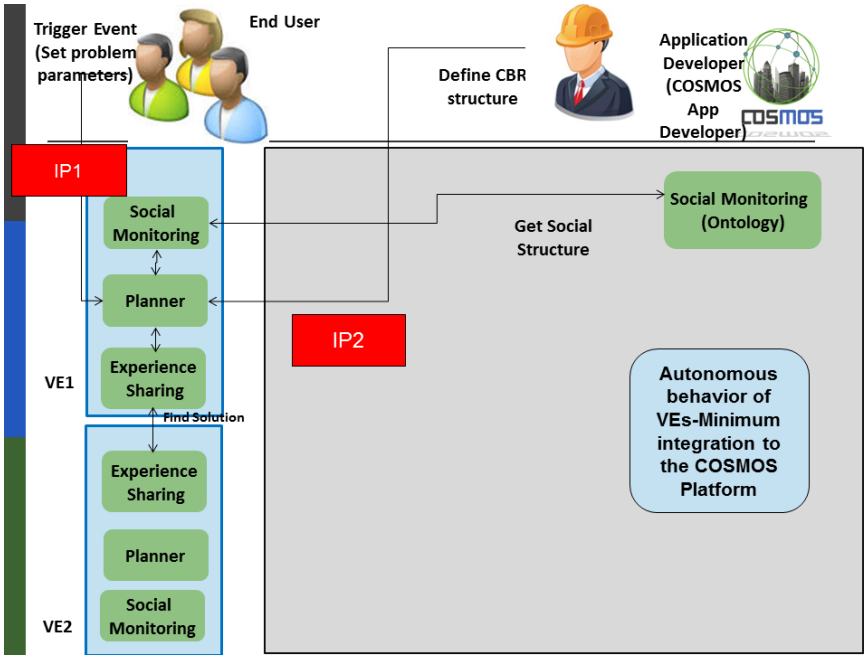

Figure 9: Autonomous Behavior of VEs with minimum platform integration subsystem

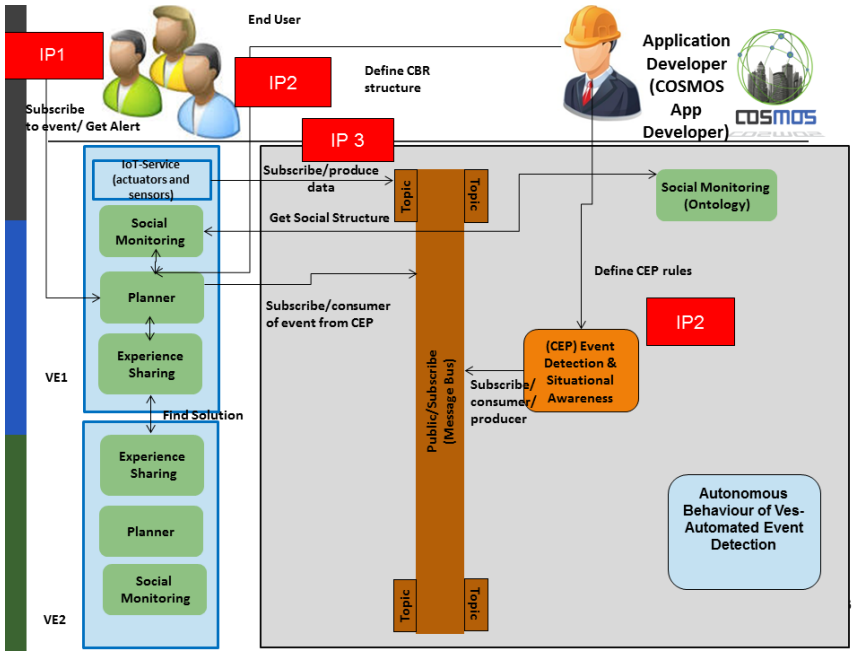

Figure 10: Autonomous Behavior of VEs- Automated Event Detection and Incorporation of COSMOS Platform

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 ^{o}C, the user turned the heating on for a total of 10 minutes (600 seconds) and stopped at 20 ^{o}C, using hot water of 70 ^{o}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.

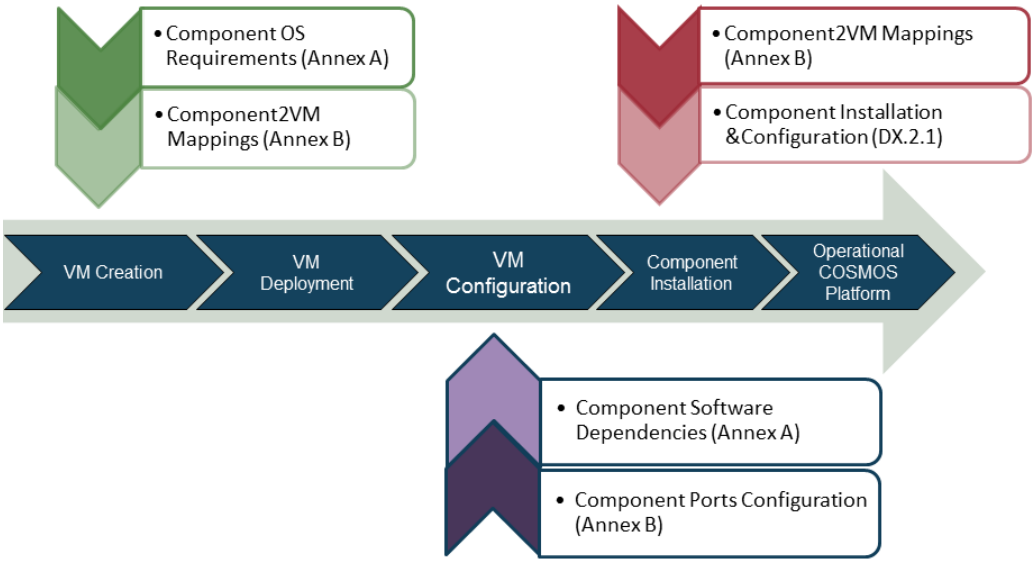

Figure 13: COSMOS Platform Setup Process

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, the main tasks that are expected to be the focus on 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 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 (subsystem)
  - 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 Scenarios concretization
  - This relates mainly to the initial design and development of one or more of the concrete Application Scenarios that have been defined in D7.1.1, in relation to how the available COSMOS enabled functionalities can be used or defined in order to server 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.

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 in case poor alignment is identified.

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.

4.2. Subsystems 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 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).

4.2.2. Application Definition Framework (Data Feed)

The subsystem and roles for this case appear in Figure 15. The main involved entity is the Application Developer, who needs to retrieve relevant endpoints based on their needs (IP2) and create the necessary data channel (e.g. in the MB) from which the various application components will derive the information, based on the application logic.

4.3. Use Case specific aspects involved and concretization

In the second iteration of this document, we expect to have the concrete aspects that are going to be applied in the components in order to enable the realization of the application scenarios (defined in D7.1.1), specifically the ones that are anticipated to be ready for the end of Y2 (M25- September 2015). For these cases we expect to have identified specifically which components 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).

Furthermore, we anticipate to have defined the UC scenario of Taipei and the way to link to the platform in terms of data feeds.

4.4. Testing environment

4.4.1. Testing infrastructure

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.

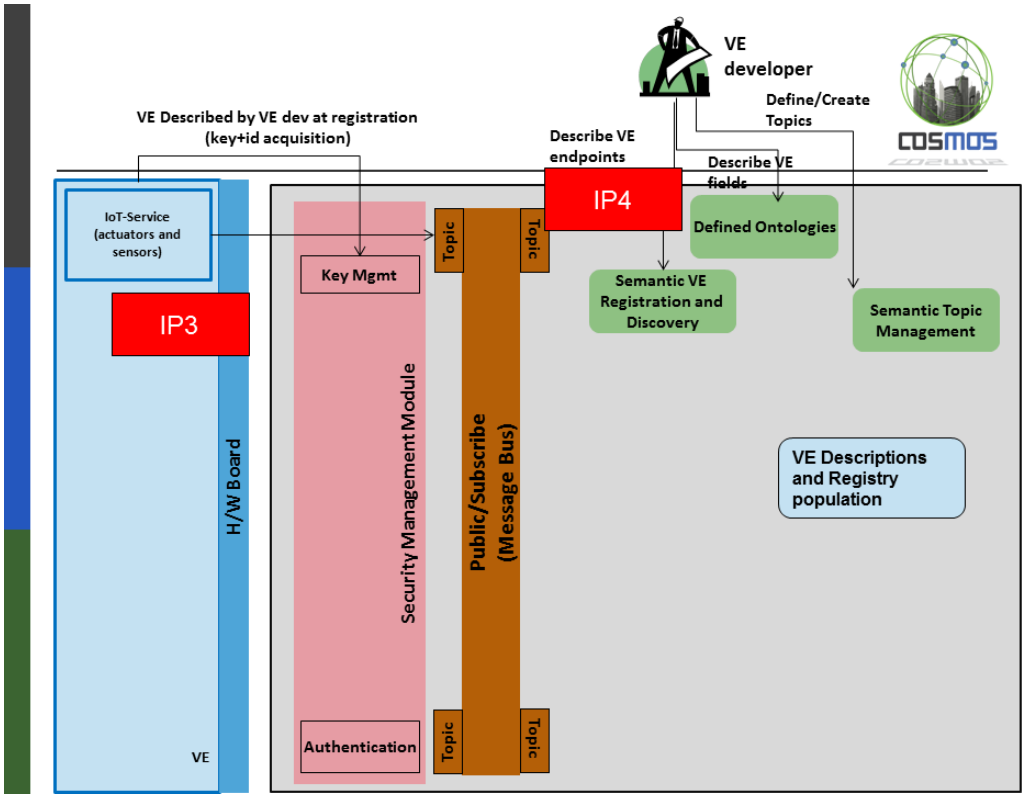

Figure 14: VE Descriptions and Registry population

4.4.2. Involved Tester roles

The involved tester roles at this stage include mainly the VE developers, that are expected to utilized the COSMOS platform components for creating the descriptions of the VEs. 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.

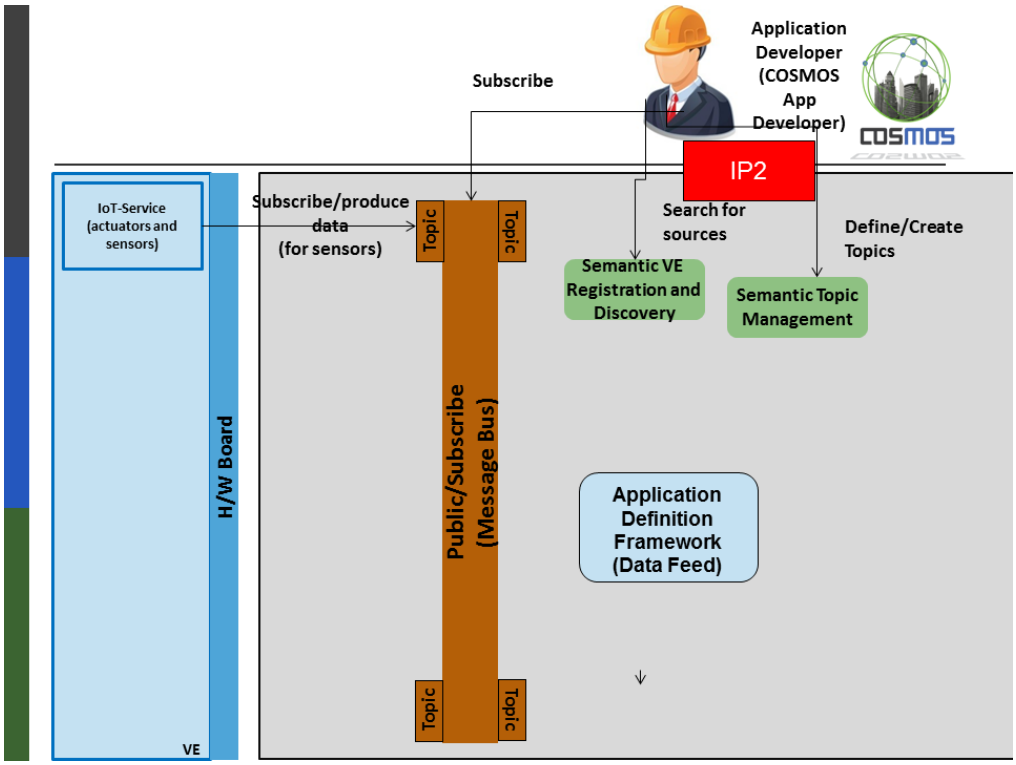

Figure 15: Application Definition Framework Subsystem (Data Feed)

5. Traceability Matrix of Capabilities to Use Cases

A traceability matrix is necessary in order to maintain a mapping between the COSMOS functionalities and how these may be applied to the different UCs (Table 2). Every capability or component should be present in **at least** one of the scenarios, by the end of the project, in order to justify its usefulness and added value.

Table 2: Traceability Matrix of capabilities to UC scenarios

	UC 1 Capital Planning	UC 2 Minimising Carbon	UC 3 Heating Schedule	UC 4 Mobility Assistance
Data Annotation	✓	✓	✓	✓
Data ingestion	✓	✓	✓	✓
Real time feed		✓	✓	✓
Data Analytics	✓	✓		✓
CBR-Planning		✓	✓	TBD
Experience Sharing		✓	✓	TBD
H/W encryption		TBD		✓
Privelets		✓	✓	✓
Predictive Modelling	✓	✓	✓	TBD
CEP	✓	✓	TBD	✓
Semantic Integration	✓	✓	✓	✓

This mapping is defined based on an initial investigation of the application Use Cases, as these are defined through 4 application scenarios in D7.1.1. 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. The exact concretization will be performed in the following integration periods of the project. 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.

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. Therefore this matrix will be more useful in the context of Y2 and Y3 of the project, in which we aim for building the applications that correspond to the realization of these scenarios.

6. 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 initial version of the document, the concrete process of setting up a COSMOS platform provider has been defined, along with the necessary information regarding the practical aspects of this setup. Furthermore, the needs of the immediate period following this plan have been identified, in terms of concrete subsystems for integration. Relevant roles and integration points have been identified.

In the following updates of this document, the upcoming integration periods will be further concretized, based on the technical developments and 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**
  - **Data Mapping**

Table 3: 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 4: Software requirements for the Message Bus

Name	Version	OS
Erlang	R15B	Linux/Windows

  - **Cloud Storage-Metadata Search**

Table 5: 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 6: 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 7: 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 8: 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 9: 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

Table 10: Software requirements for the Privelets

Name	Version	OS
Java Runtime Environment	1.7	Linux/Windows
Apache Tomcat	7.0	Linux/Windows
Java Server Faces	2.2	Linux/Windows

Planner

Table 11: 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 12: Software requirements for the Event Detection at the VE level

Name	Version	OS
ZeroMQ	4.0.4	Linux/Windows

Experience Sharing

Table 13: 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 14, along with the network configuration necessary for the VMs.

Table 14: 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

- **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)