COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D7.6.1 Integration Plan

WP7 Use cases Adaptation, Integration and Experimentation

Version:	1.0
Due Date:	30/06/2014
Delivery Date:	25/07/2014
Nature:	Report
Dissemination Level:	PU
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:	Jozef Krempasky (ATOS), Orfefs Voutyras, Vassilis Psaltopoulos (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
0.1	10/7/2014	Achilleas Marinakis	NTUA	Circulating the ToC, adding initial content
0.2	24/7/2014	Achilleas Marinakis and co-authors	NTUA, ATOS, IBM, UniS, Siemens	Version ready for internal review
0.3	25/7/2014	Achilleas Marinakis	NTUA	Final version based on the review from ATOS and NTUA
1.0	25/7/2014	Achilleas Marinakis	NTUA	Version for submission

Table of Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Meaning
CA	Consortium Agreement
CBR	Case-Based Reasoning
CSV	Comma-Separated Values
DoW	Description of Work
EC	European Commission
ECM	Enterprise Content Management
GA	Grant Agreement
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IoT	Internet of Things
OS	Operating System
SSH	Secure Shell
SVN	Subversion
VE	Virtual Entity
VM	Virtual Machine
WP	Work Package

1. Executive Summary

The integration is an essential activity, especially on an Integrated Project with a wide range of research areas and functionalities developed. This Work Package (WP) is in charge of keeping the coherence of all the developments, ensuring the several components converge to the same goals, performing the integration of the software and setting up the testbed where the demonstrators will be installed and executed.

The integration process is compartmentalised in two different activities. The first one is a more generic planning in which the definition of the demonstrators and the further alignment with the technical Work Packages is performed. The other integration activity is more focused on the actual integration of the software components, testing and execution of the platform components and the demonstrators' applications.

The initial integration plan foresees a series of remote and face to face integration workshops that will conclude with the construction of the Project Demo by October 2014.

A subversion repository, which is the basic tool to support the integration activities, is introduced. The subversion repository will contain the components' software and the installation packages and will provide a central point for updated information about the components, their requirements and their relations with other parts of the project.

During the initial months of the project some activities have been performed that allow the definition of the initial layout of the testbed structure (in terms of hardware and network), identifying the general components that are part of COSMOS and the applications, and their requirements towards the testbed.

This document also provides a set of recommendations about the internal structure of the software components and the creation of the software packages and related documentation.

Finally, an acceptance procedure is defined. This procedure requires that the software must be available in the Subversion repository, the components installation packages must be ready with the associated documentation and the components must have successfully passed their unit tests.

2. Introduction

The aim of this document is to describe how the integration of the different parts developed by the WPs is going to be performed to build the testbed and the final demonstrators, and the process put in place to reach this goal.

The integration and demonstration purposes of the COSMOS project are both formally defined in WP7. The Project Partners are expected to showcase the outcomes of the research and development WPs of COSMOS, verifying their applicability through two 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 two use-cases will be implemented, the experiments will be prepared and the evaluation of the experimentation will follow.

We will functionally test the capabilities of the COSMOS technologies under real-life smart city conditions and exercise them in a realistic context, in order to verify the applicability of the implemented technology in different application domains. In addition, real-life showcases will be performed aiming at demonstrating and disseminating the achievements of the project.

To keep the consistency of all the developments and to ensure that all the components follow the same direction, an integration plan has been defined and put in place, covering the definition of the demonstrators, the coherence of the development activities inside the technical WPs and the integration and testing of the software components, including the available tools for software integration and collaboration.

Based on the requirements imposed by the applications and by the platform itself, an initial description of the testbed is provided in section 6. The way to access the testbed from the different locations of the organisations, constituting the COSMOS Consortium, is also mentioned in this section.

Finally some recommendations in terms of software packaging, internal structure, installation and execution of components are given in section 7. 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.

2.1. Objectives

The 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

3. Integration Process

This section explains the steps needed to perform the integration activities and the integration time scheduling. This section also describes the tools used to support the integration.

3.1. General Description of the Integration Process

3.1.1. General Integration Planning and Demonstrators Definition

In the scope of WP7, the demonstrators which are going to be implemented in the first year of the project are defined. COSMOS has different WPs that are performing research and development on different areas. Therefore, a coordination among the implementation of functionalities of WPs that can rely on or interact with capabilities of other WPs is needed. To achieve this degree of coherence among WPs, a plan has been defined consisting of the following steps:

- Describe the demonstrators from the application perspective, including detailed descriptions of the two scenarios (subsection 5.1 and 5.2).
- Identify, inside the technical WPs (WP3, 4, 5 and 6), the set of key innovations and features that are going to be implemented for each demonstrator and thus identify the dependencies between the technical WPs and refine and align the development plan for each individual WP, jointly with the rest of the project.

3.1.2. Component Integration Process

At a lower level, the integration process consists of the following steps:

- Identify the functional components and their dependencies with other ones (especially on different WPs).
- Identify the requirements and set up of the testbed.
- Define and operate the integration tools.

3.1.3. Identification and Analysis of the Functional Components

The first step to perform the integration of the components is to have a complete view of the building blocks and functionalities that are going to be developed and of their software requirements too. For detailed information on that, please see section 4.

3.1.4. Identification of the Requirements and Set Up of the Testbed

The hardware infrastructure of the testbed must be prepared to accommodate the set of requirements coming from the functional components and the applications that are described in chapters 4 and 5 respectively. For more information regarding the testbed structure, please see section 6.

3.1.5. Test Cases

It is expected that the developers will perform a set of activities to ensure the correct behaviour of the components and the proper interactions with the other components of the project.

3.1.5.1 Types of Test Cases

Three types of test cases are expected to take place:

- **Unit/Component Test Cases:** Unit test cases concern the interfaces within the components and are quite low-level. They concern how components should react to an input and whether they provide the appropriate output.
- **Integration Test Cases:** Integration testing will be based on extracted and combined sequence diagrams from D2.3.1 [1], and based on the anticipated demo scenarios component combinations that are described in section 5. Integration testing will be performed in an iterative way, in order to initially check component interfaces.
- **System Test Cases:** System testing will include end to end validation of COSMOS functionality and is not expected to be performed in full scale in Y1.

3.1.5.2 Test Cases Results Template

The results for each test case will be presented using the template in Table 1. This applies also to the unit tests. In order to reduce table creation, one test case may contain more than one unit tests. However the tested aspects should be explicitly mentioned in the respective fields.

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.

Table 1: Template for the Test Cases Results

3.1.6. Development Planning

The functional components that are involved in the application scenarios will be the integration target for the first year of the project, in the described combinations. For further information please see section 5.

The following figure depicts the general process for development and integration activities:

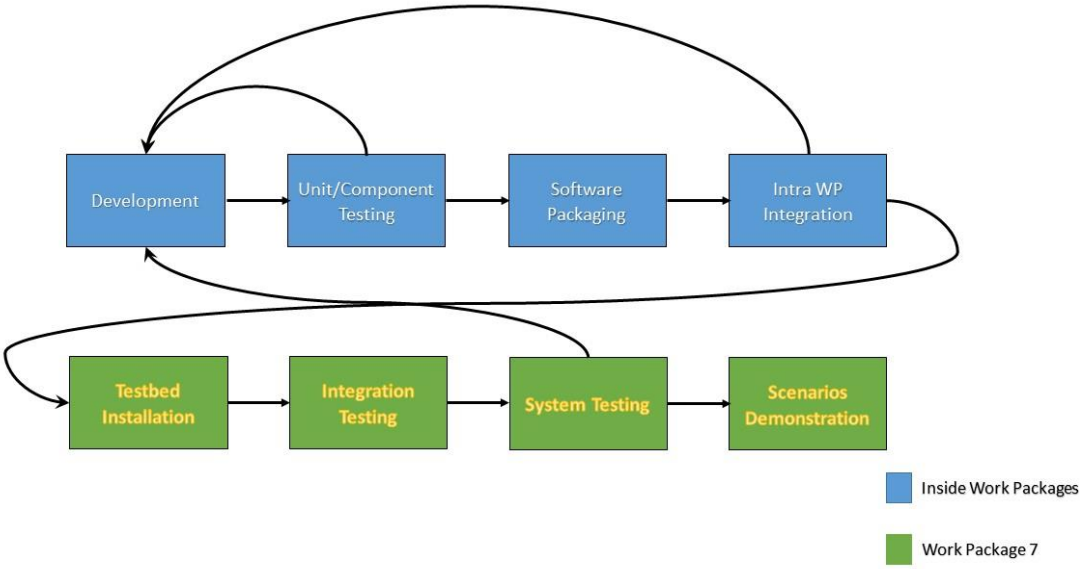

Figure 1: Integration Process

WP7 expects to have available the set of components and software packages associated with them, to perform the integration following the planning presented in the following Gantt chart:

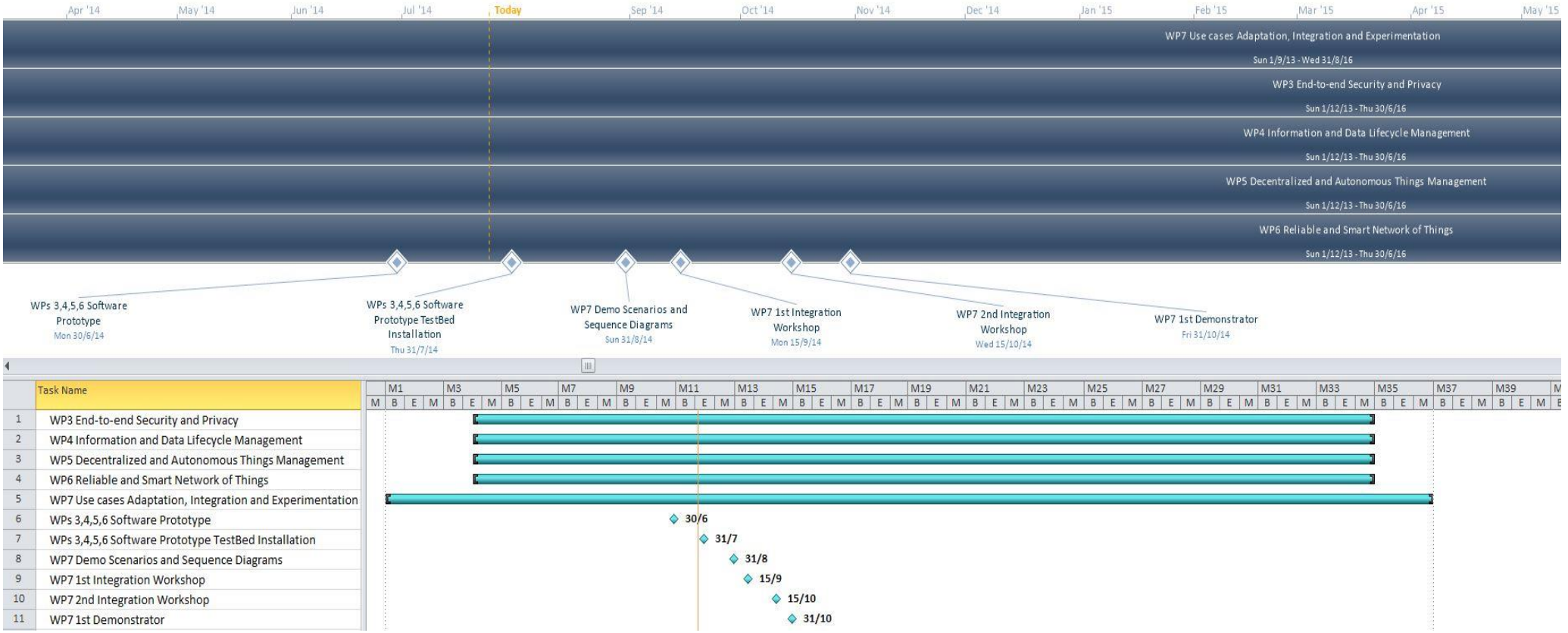

Figure 2: COSMOS Integration Timeplan for Y1

The main milestone is the demonstrator day of release in October 2014. The timeline includes delivery days for components, which are planned in the COSMOS DoW and described as WPs prototypes that need to be integrated into the testbed.

For the demonstrator there will be two integration workshops in September 2014 and October 2014. To ensure the correct integration of all the components the following process has been defined:

- The components will be available as packages in the Subversion repository.
- The components will be installed and deployed in the testbed by the developers, following the respective instructions (in the README file).
- When problems are detected during the integration testing the developers of the involved components will be responsible to solve the problem in a timely manner.

The final test will assure the correct execution of all the steps of the defined scenarios in the demonstrators.

3.1.7. Integration Tools

3.1.7.1 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) [5] is used, which is nowadays one of the most popular and complete, in terms of features, open source revision control systems.

3.1.7.2 Alfresco

Alfresco [6], 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.

3.1.7.3 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.

3.1.7.4 Code Quality checks

COSMOS partners will investigate the possibility to use Sonar [2], 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 [3].

According to [4], 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

4. COSMOS Functional Components Software Dependencies

4.1. COSMOS Platform

This section contains the dependencies of the components which are deployed at the COSMOS platform.

4.1.1. 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

Table 2: Software requirements for the Data Mapping

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

4.1.2. Message Bus

Name	Version	OS
Erlang	R15B	Linux/Windows

Table 3: Software requirements for the Message Bus

4.1.3. Cloud Storage – 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

Table 4: Software requirements for the Metadata Search

4.1.4. Cloud Storage – 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

Table 5: Software requirements for the Storlets

4.1.5. Event Detection & Situational Awareness

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)

Table 6: Software requirements for the Event Detection at the COSMOS platform

4.1.6. Prediction (Interpolation/Extrapolation)

Name	Version	OS
Ubuntu	13.10	Linux
Python	2.7	Linux

Table 7: Software requirements for the Event Detection at the COSMOS platform

4.1.7. 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

Table 8: Software requirements for the Semantic Description and Retrieval

4.2. VE Level

This section contains the dependencies of the components which are deployed at the VE level.

4.2.1. Privelets

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

Table 9: Software requirements for the Privelets

Privelets will be distributed as an Eclipse Kepler 4.3.0 project folder zip.

4.2.2. 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

Table 10: Software requirements for the Planner

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

4.2.3. Event Detection & Situational Awareness

Name	Version	OS
ZeroMQ	4.0.4	Linux/Windows

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

4.2.4. Experience Sharing

Name	Version	OS

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

Table 12: Software requirements for the Experience Sharing

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

5. Application Scenarios

This section contains descriptions of the demo applications that are developed in the context of WP5 and 6.

5.1. Intelligent Transportation scenario

1. The travel time between any two bus stops along the route will be calculated using historical data.
2. The bus transmits its coordinates with a specific event id when reaching bus stops.
3. A bus may or may not stop at a particular bus stop. Additionally, to calculate travelling time between consecutive bus stops, the bus should stop at both stop ids. Hence, only those readings are extracted from the raw file which fulfil both requirements.
4. The travel time between every two consecutive bus stops is averaged using historical data.
5. The travel time is saved in different contexts, such as busy rush hours and quite night time.
6. Average travel time between two bus stops will act as weight and bus stops will act as nodes.
7. The scenario will calculate the total travel time by adding up the weights from starting node to the destination node by taking input in the form of (Source Bus Stop id, Destination Bus Stop id).
8. Graph theory can be further explored to find the optimum route from source to destination.

5.2. Efficient Heating Scenario

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.

10. The case is evaluated based on its results and the Trust of the friend that shared its case is recalculated.

5.3. Potential Component Combinations for Y1 Demonstration

For Y1 demonstration, we are currently investigating the possibility of showing partial demonstrations by combining the functionality of more than one components. So far, two potential paths have been investigated that will be the basis for driving the integration processes:

- **Data management** which mainly includes components that are deployed at the COSMOS platform. Potential combinations include the IoT Service Sensor Data, Message Bus, Data Collection, Data Mapping, Cloud Storage and Prediction, which are going to be shown in the Intelligent Transportation Scenario (section 5.1).
- **Autonomous behaviour of VEs** which mainly includes components that are deployed at the VE level. Potential combination includes Planner, Experience Sharing, VE Registry, Social Monitoring and Social Analysis, which are going to be shown in the Efficient Heating Scenario (see 5.2).

In case some components have not reached the necessary level of maturity, their functionality will be based on simple interface usage or their input will be simulated for Y1.

5.4. Use Cases Data

For the first year of the project, use cases have provided historical data in CSV format. These data are used as input for the application scenarios. For the next years of the project, use cases are going to provide both historical and real data as well their application scenarios, covering different environments and contexts of a smart city.

6. Testbed Structure

6.1. Basic Requirements

6.1.1. 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.

6.1.2. 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.

6.2. Components Specific Resource 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 – requirements for first year of project	WP4	4	8GB	500GB	8080, 9200, 5672, 5000, 6000, 6001, 6002, 873	None

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

6.3. Testbed Description

The hardware infrastructure is provided by Atos, since this is the commitment stated in the DoW. 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.

7. 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 [5] as repository to share the software developed within the COSMOS scope (section 3.1.7.1). 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.

7.1. 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. [7]
    - 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) [8] or EXE (Windows Executable Installers) [9] 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.

7.2. 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**.

7.3. 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).

7.4. 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.

7.5. 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.

7.6. 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 Section 3.1.5.

8. References

[1] COSMOS Project D2.3.1 Conceptual Model and Reference Architecture

[2] Sonar tool: http://www.sonarqube.com/

[3] SonarQube Documentation, Plugin Library List, available at: http://docs.codehaus.org/display/SONAR/Plugin+Library;jsessionid=48B59953A92269D9938CA1751951ED36

[4] Method & Tools: http://www.methodsandtools.com/tools/tools.php?sonar

[5] Subversion: http://tortoisesvn.tigris.org/

[6] Alfresco : http://www.alfresco.com/

[7] Debian Packaging: https://wiki.debian.org/IntroDebianPackaging

[8] MSI: http://msdn.microsoft.com/en-us/library/aa266427%28v=vs.60%29.aspx

[9] EXE: http://msdn.microsoft.com/en-us/library/ff553615.aspx