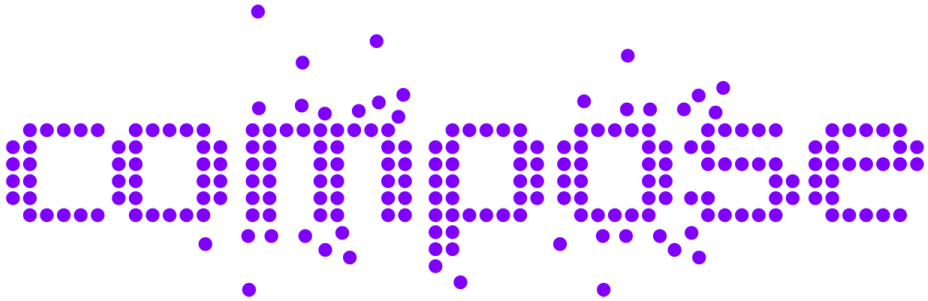

Collaborative Open Market to Place Objects at your Service

D6.4.1

Marketplace integration – First version

Project Acronym	COMPOSE	
Project Title	Collaborative Open Market to Place Objects at your Service	
Project Number	317862	
Work Package	WP6 Open marketplace	
Lead Beneficiary	IBM	
Editor	Benny Mandler	IBM
Reviewer	Robert Kleinfeld	FOKUS
Reviewer	Marko Vujasinovic	INN
Dissemination Level	PU	
Contractual Delivery Date	31/10/2014	
Actual Delivery Date	31/10/2014	
Version	V1.0	

Abstract

The first version of the COMPOSE integrated platform is described. The integrated platform took technological pieces developed in all the technical WPs and connected them into a cohesive solution that can already benefit external stakeholders. The COMPOSE integrated platform offers on the one hand developers the opportunity to easily create applications which are based on IoT smart devices. On the other hand it takes care of data ingestion and making the data accessible to applications. In the middle it takes care of all deployment, hosting, and connections needed in a cloud based environment.

Document History

Version	Date	Comments
V0.1	05/08/2014	Initial skeleton version
V0.2	10/09/2014	Fill in details for run-time, and discovery
V0.3	15/10/2014	Incorporated security info from UNI PASSAU
V0.4	16/10/2014	Added figures, introduction and high level section
V0.5	17/10/2014	Incorporate contributions from BSC and Fokus
V0.6	26/10/2014	Incorporate contributions from Abertis
V1.0	30/10/2014	Finalize and integrate all contributions

Table of Contents

List of Figures

List of Tables

Acronyms

Table 1: Acronyms table

Acronym	Meaning
API	Application Programming Interface
CF	Cloud Foundry
COMPOSE	Collaborative Open Market to Place Objects at your Service
GUI	Graphical User Interface
IoT	Internet of Things
JSON	Java Script Object Notation
PaaS	Platform as a Service
REST	Representational State Transfer
SDK	Software Development Kit
SPARQL	SPARQL Protocol And RDF Query Language
UAA	User Account and Authentication

1 Introduction

This document accompanies the demonstration of the first integrated COMPOSE platform. In this version most of the fundamental capabilities envisioned within the platform are in place. Future releases are expected to enhance various aspects of the platform. Contributions from all the technical WPs have made it into this version of the platform, thus providing a first glimpse of the ultimate capabilities expected by the end of the project.

The main purpose of this document is to accompany the technical demonstration and provide the necessary background information concerning components and their interactions. It is not intended to be a full-fledged design document. More detailed information is provided in the final version of the COMPOSE architecture document (D1.2.2), and individual components are detailed in their own deliverables.

2 High level picture – Main components and interactions

Figure 1 presents an overall view of the first version of the COMPOSE integrated platform. The figure shows the main components which are a part of the platform, and the main interactions between the various components. At its core the COMPOSE platform is a customization of an openly available PaaS infrastructure (Cloud Foundry[1]), to make it better suited as a platform to serve the IoT. Thus, as can be seen in the figure, COMPOSE is a cloud platform, with specific capabilities that should make it easier to develop and deploy IoT based applications. On the left hand side of the figure are the COMPOSE cloud services, while on the right hand side are the COMPOSE components which are deployed as cloud applications.

Most of the components operate within the cloud environment, besides the developers' portal, which is a crucial COMPOSE component that operates outside the cloud. It serves as the connection point between external developers and the COMPOSE platform. It mediates between the platform and the external world and makes it easier to consume COMPOSE offered capabilities. These capabilities are in the realm of assisted application development, through security, and all the way to automated application deployment in the cloud.

The developers' portal is the first access point into the COMPOSE platform for end-users and developers. It integrates various front-end components for IoT application development as well as back-end components which enable the COMPOSE core features. The interworking between all components is the integrated COMPOSE platform. Figure 1Figure 1 shows the core components of the COMPOSE back-end. Data flows from bottom-up through the data management and the service layer to the developers' portal. Integrated components such as data management, service discovery, security and cloud deployment are highlighted in this figure. The developers' portal via the GUI provides direct access to particular features of the back-end components.

[1] http://cloudfoundry.org/index.html

COMPOSE platform components are deployed within the cloud run-time either as cloud applications or cloud services, depending on the requirements and mode of interaction with each such component.

The COMPOSE controller, which is a central point of communication with the developers' portal is deployed as a cloud application, thus making it easily accessible to the external world on the one hand while internally being able to bind to the COMPOSE services it needs to interact with for its proper operation.

Security components, such as the Identity Management, are also deployed as a cloud application, enabling the COMPOSE controller to use it, as well as external entities.

The discovery component is divided into a front-end and a back-end. The front-end is a light-weight cloud application, enabling external entities to interact with it, while the bulk of the work is performed by a back-end, which is deployed as a cloud service. Such a deployment enables the back-end to be state-full as it needs to be.

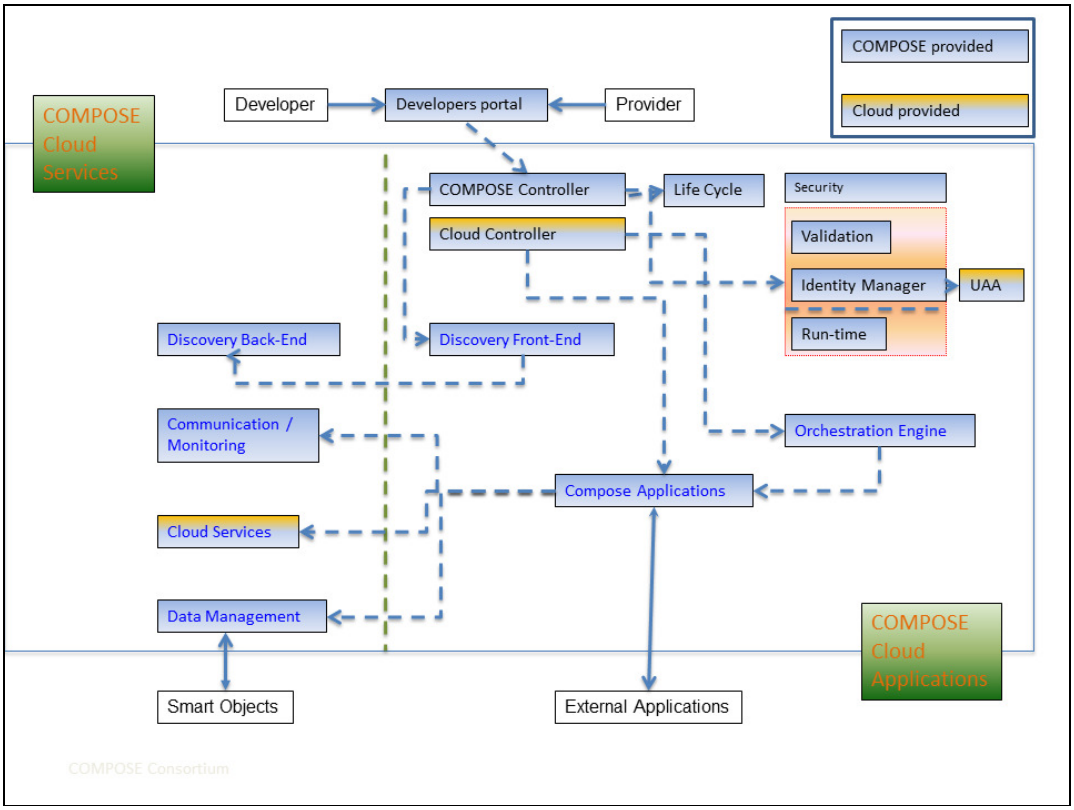

Figure 1: Main COMPOSE platform components

The data management is deployed as a cloud service, enabling it to be state-full, and enables only internal COMPOSE cloud applications to bind to it via its HTTP based API. The communication and monitoring infrastructure is deployed also as a cloud service, for the same reasons, but it enables thin clients to run as or within cloud applications, and bind to the communication servers at the back-end.

2.1 The developers portal

The developers portal is a one stop shop for developers when wishing to create IoT based applications. This portal serves as the entry point for external users of the platform, whether the users are application developers or smart objects providers. The developers' portal provides a user-friendly GUI based interaction mode which helps developers create the application of their dreams.

The developers' portal contains three parts that guide the developers through the process of application creation. The smart objects manager is used to create and manage service objects. It provides features for smart object virtualization, management and policies for authorization and authentication. Once a service object is successfully created end-users are directed to the smart object composer for creating mashup applications of their choice. The smart object composer integrates capabilities for service discovery and deployment.

At its back-end the developers' portal interacts with the cloud infrastructure in order to deploy, run, and manage created applications. The main interactions of the developers' portal are with:

1. Security – to identify and authenticate users; obtain proper tokens for smart object interaction.
2. Cloud deployment – to make the actual deployment of the created application.
3. Service discovery – to locate existing building blocks that developers can use for creating a new application.
4. SDKs – to integrate and access various Smart Objects from the developer portal.
5. Service composition – to connect existing building blocks to mashup applications and make them accessible via REST full APIs.

2.1.1 The developers portal environment

- Ubuntu 10.04.4 LTS
- Developer portal implemented as Node.js v0.10.26 application and is deployed as a CF application
- The Smart Object Manager uses Angular.js v1.2.10 framework
- The Smart Object Composer component based on Node-RED v0.8.1
- Uses MongoDB v2.6.3 for local user management
- Service Objects API running in BSC serviced at http://api.servioticy.com/

2.2 The cloud run-time

The cloud run-time hosts COMPOSE entities and makes applications available to the external end-users. The COMPOSE platform cloud run-time consists of a customized version of the openly available Cloud Foundry PaaS. The cloud run-time hosts COMPOSE application as well as the front-end of COMPOSE specific infrastructure services such as the data management service. In addition the run-time provides binding mechanisms for COMPOSE applications to connect to the infrastructure services they require.

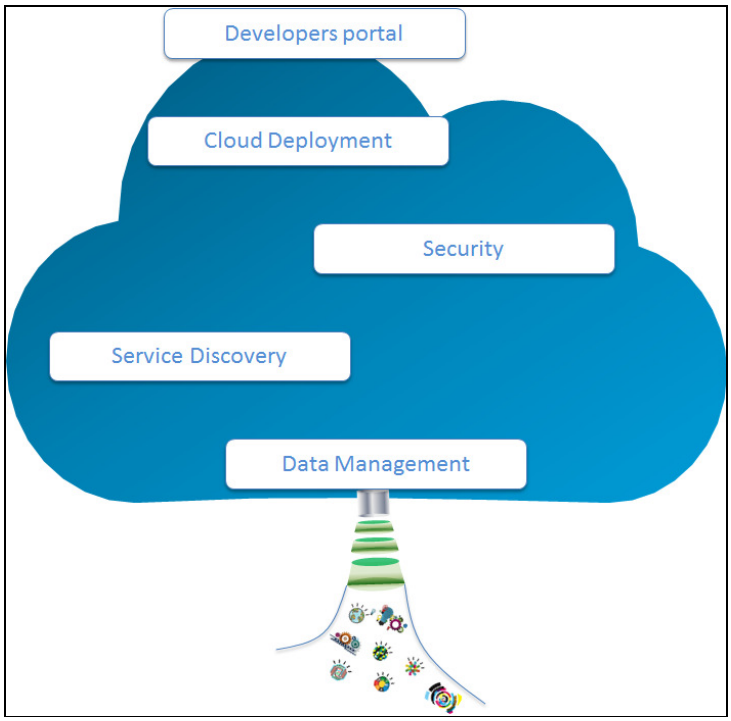

Figure 2: Cloud run-time components

The main interactions of this component are with:

1. Deployment – to close the loop on applications deployment and lifecycle management.
2. Service discovery – operates as a part of the cloud environment. A front-end of the service discovery is hosted as a COMPOSE application, and thus can be accessed by external components, while a back-end is run as a COMPOSE service which is bound to the front-end application.
3. Data Management – operates as a part of the cloud environment. A front-end is hosted as a COMPOSE applications, and thus can be accessed by external components, while a back-end is run as a COMPOSE service which is bound to the front-end application.
4. Security – COMPOSE security interacts with cloud security mechanisms provided by the UAA, and adds specific security capabilities.

2.2.1 The run-time environment:

- Ubuntu 10.04.4 LTS
- Cloud Foundry build 183, single instance using bosh_nise
- Service Broker backend running on the same machine, while frontend running as a Cloud Foundry application.
- Compose Communication Bus running locally
- Cloud Foundry instance managed with cf-cli (go executable)
- Service Objects API running in BSC serviced at http://api.servioticy.com/

2.3 Deployment and life cycle management

The deployment component is in charge of taking the applications designed within the developers portal and transforming them into a COMPOSE entity which will be deployed, hosted, and managed within the COMPOSE CF-based cloud platform and added services. In addition, this component tracks and governs the lifecycle management of COMPOSE applications.

A COMPOSE entity can be one of the following types:

- Service Object
- Data Processing Pipe
- COMPOSE application
- COMPOSE workflow

The main interactions of this component are with:

1. Security – For authentication, authorization, and validation of the application being deployed.
2. Cloud run-time – for the actual deployment of the application within the COMPOSE cloud run-time
3. Data Management – for the creation of a service object.
4. Service discovery – for registering an entity

2.3.1 The Deployment component environment:

The deployment component (COMPOSE controller) is deployed as a CF application within the COMPOSE platform installed in BSC. It further interacts with the cloud controller in order to deploy, monitor, and manage COMPOSE applications.

2.4 Security

2.4.1 Identity Management

Defining principals and how they authenticate with COMPOSE is required before any security mechanism is put in place. However, in addition to managing information about principals, Identity Management must ensure proper access to identity information. This requirement brings a particularly interesting challenge. In a marketplace like COMPOSE, users should be able to define information about entities owned by them (e.g. service objects), but at the same time, every user should decide whether he considers identity information provided by other users reliable or not. For this reason, a generic attribute-based Identity Management scheme was developed for COMPOSE. In this scheme users can tag entities belonging to them with specific attributes defined inside groups. Values assigned to those attributes are only considered valid by the security framework once they have been approved by a group administrator where the attribute was defined.

Identity Management is deployed as a cloud application. It is meant to be used not only by COMPOSE components (i.e. developed by the consortium), but also by COMPOSE application

developers who may want to use Identity Management as a Single Sign On solution. There are two kinds of authentication to interact with Identity Management.

First of all, creation and deletion of principals is done by internal COMPOSE components; therefore, they must use HTTP-digest authentication to keep consistency of the platform. For instance, when a new COMPOSE application is created, it must be ensured that this call is issued by a COMPOSE component to prevent a malicious user from registering non-existing-applications in Identity Management. On the other hand, whenever an API call to Identity Management concerns a particular user or the users' entities, the user needs to be authenticated. To authenticate the user, he must provide a token which has been granted by Identity Management after a successful log-in process.

This component interacts with:

1. Cloud-runtime: Identity Management encapsulates the process of requesting a token for users from the cloud User Account and Authentication server.

2. Data Management: The Data Management component registers service objects once they are created; furthermore, Identity Management is queried to authenticate external devices providing data to Data Management.

3. Developers Portal: The Developers Portal authenticates users with Identity Management.

4. Deployment and life cycle management: This component registers new COMPOSE applications in Identity Management.

2.4.2 Data Provenance

The integrated data provenance module is collecting precise information about the heritage of data items in the data management layer (WP2). Besides the information about the sources it also contains information about the performed operations to generate a specific item. This information is retrieved by a modified execution environment for the user-defined source code of service-objects (SO).

The provenance data is in JSON format, and is part of the security meta-data. It is stored together with the actual data item (sensor update (SU)) in the Couch Base database of the data management layer. To ensure scalability it is possible to activate or deactivate the provenance collection for a SO.

The data provenance module interacts with the data management layer during the generation of a new data item (SU). The creation of such a new data item can have two different causes:

1. A new sensor update is sent to the COMPOSE platform from outside. In this case default provenance information is generated by the provenance module according to the available information (no runtime information).

2. A new sensor update is generated during dispatching. In this case the execution of the source code of the pre-filter, post-filter and the calculation of the current value is monitored by the provenance module, in order to generate precise provenance information.

2.5 Service Discovery

With the vast amount of applications anticipated in a platform such a COMPOSE the chore of a developer would be made much easier if the developer could locate existing building blocks that he can re-use to create his own masterpiece. For this reason COMPOSE contains a service discovery component which holds semantically enhanced service descriptions making it easier to discover services based on various criteria.

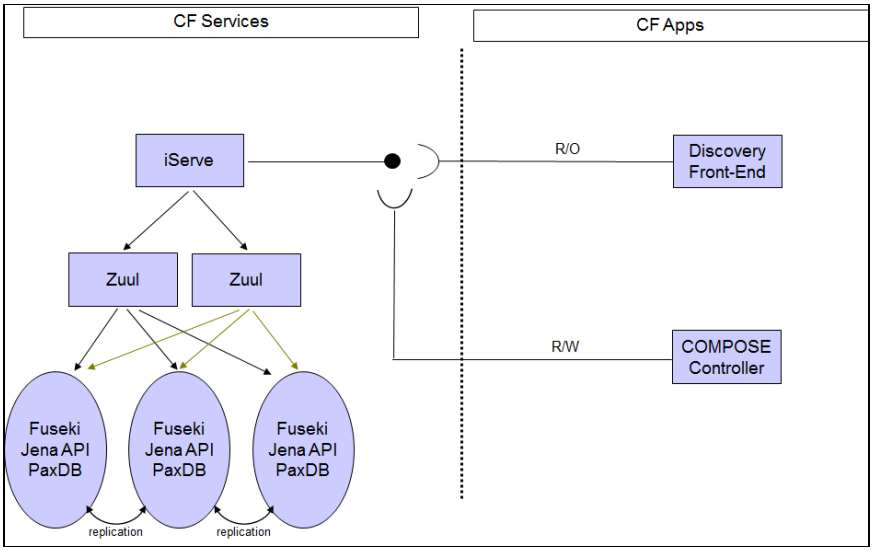

Figure 3: Service Discovery deployment

2.5.1 The service discovery environment

The service discovery front-end is deployed as a CF application, which directs incoming HTTP requests to the iServe service. This application binds to the back-end part of the service using CF mechanisms enhanced by the COMPOSE universal service broker. At the back-end a Tomcat7 container hosts iServe as a COMPOSE platform service. iServe in turn is connected to an application-layer load-balancer that redirects SPARQL queries to a PaxDB cluster. Currently the PaxDB cluster consists of 2 nodes deployed locally with the load balancer.

2.6 Data Management

The current integrated version of the platform incorporates data management services. These services form an ingestion layer which takes care of the bi-directional communication with the external smart objects. In particular this layer stores the information flowing into the platform and provides data processing and manipulation services for the entire platform. This layer consists of a historical data repository in addition to a programmable real-time streams processing unit. In addition a search service over the data is provided as well. The basic internal representation of a smart object is called a COMPOSE service object, whose endpoint exposes a JSON and REST based API for creating, updating, and obtaining date from service objects.

2.6.1 The Data Management environment

The data management components are deployed as a CF service that is exposed and made available to CF applications. The entry point for CF applications is either the HTTP REST API or the MQTT/STOMP bridge that connects the data management environment with the external entities using TCP or WebSockets. The data management service is composed of several components: CouchBase for the registry and data repository; ElasticSearch to index and provide querying mechanisms on the repository; Apache STORM as the event processing and dispatching engine; Jetty and Jersey for the public and internal REST API; Apache Apollo as the multi-protocol message broker; and several NodeJS components to bridge the REST API with external MQTT/STOMP entities. The data management environment interacts with the security component (Identity Management) to enforce Authentication and Authorization. And it provides capabilities to the COMPOSE platform for data provenance and reputation through the STORM topology.

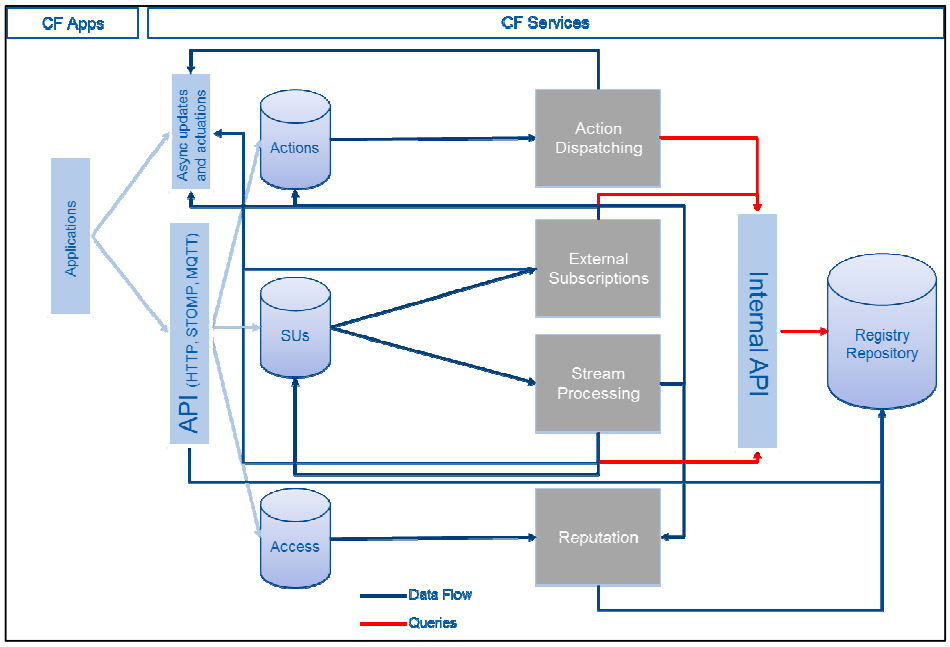

Figure 4: Data Management deployment

3 What is being demonstrated

At this first version of the integrated COMPOSE platform the combined capabilities of most major components will be demonstrated.

1. The full flow of registering a new Service Object into the platform will be demonstrated. This includes the registration of service objects based on existing templates, or the registration of a new kind of s Service Object.

   This flow includes a developer interacting with the developers' portal, while in the background the COMPOSE controller is invoked, which in turn calls the security and

service discovery components. Finally the request is passed into the cloud run-time and to the appropriate service, which in this case is the data management layer.

2. The full flow of interacting with a registered service object. This entails obtaining data stored in the Service Object and sending commands to an object which supports actuation.

   Internal interactions with COMPOSE components resemble the ones described in the first flow.

3. The full flow of creating a new COMPOSE application which makes use of previously registered Service Objects.

   Internal interaction with COMPOSE components resemble the ones described in the first flow, only that at the last stage a new COMPOSE application will be deployed and run on the underlying cloud infrastructure.

4 Installation & configuration

The COMPOSE platform installation is divided into several steps. First in line is the installation and configuration of the basic cloud infrastructure. Second is the installation and configuration of back-end COMPOSE services which are available to be used by applications. In the third stage application front-ends which are hosted by CF are deployed.

- OS environment: Ubuntu 10.04.4 lucid, x86_64 3GHz x2 cores x2threads/core, 6GB memory.
- Cloud Foundry installed with nise_bosh
- Service broker:
  - Backend is a java app with an embedded DerbyDB configured with a simple XML-file. The service catalogue is built dynamically by reading the YAML files given by the service providers.
  - Frontend is a Tomcat servlet hosted on CF itself and is given a user-provided-service in order to know the RMI URL of the backend.
- Services:
  - CSB: Each service instance shares the same coordinator and server nodes. Java clients connect via client library based on Apache Mina; in addition there is a JavaScript thin client available. Upon provisioning of a service instance, a coordinator instance is spawned and 2 coordinator server nodes are pushed to CloudFoundry and connect to the coordinator after start-up. If the whole process succeeds, a new service instance is being added to the configuration and the subsequent provisioning request will be given this configuration.
  - DerbyDB-provisioned: A provision creates a private DerbyDB instance; a bind creates a database within the instance. The instances are managed by the service broker (supported out of the box, with a “broker-managed=true” attribute). When the service broker backend starts up, it also starts the derbyDB instances if they’re down.

- iServe: Binding to the service simply returns the iServe's servlet URL.
- MySQL-shared: Binding to the instance creates a database and a new user. Unbinding deletes the user and the database from the DB. The MySQL server is running on the same host.
- ServioTicy: described in sub-section 2.6.1

To keep track of integration related aspects we have set up a section within the internal COMPOSE web site to keep track of open issues. This web site capability enables creating issues, bug reports, or feature requests and assign them to specific individuals. This tool enables the project to keep track of the overall state of platform integration.

5 Future directions

1. Fully distributed cloud environment – Currently the COMPOSE platform is installed on a limited amount of HW. As the main COMPOSE research is not in the direction of cloud scalability in general, we focused in the initial version of the platform on enhanced capabilities rather than invest time and energy in a large cloud environment installation.

   Nevertheless, towards the end of the project we do intend to support a larger cloud environment and demonstrate COMPOSE in its full glory within such an environment.

2. Orchestration engine – An initial stand-alone version of the orchestration engine, supporting run-time aspects of COMPOSE workflows, already exists. At the next phase we aim to integrate the capabilities of the orchestration engine with the rest of the COMPOSE platform.

3. Enhanced security – As security components are being developed and released they will be integrated in later versions of the integrated COMPOSE platform.

4. Incorporate additional capabilities – as additional COMPOSE related capabilities become available they will be integrated in later versions of the platform. For example, the recommendation and composition capabilities.