# Collaborative Open Market to Place Objects at your Service

## D6.4.2
Marketplace integration – Final version

<table>
<thead>
<tr>
<th>Project Acronym</th>
<th>COMPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>Collaborative Open Market to Place Objects at your Service</td>
</tr>
<tr>
<td>Project Number</td>
<td>317862</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP6</td>
</tr>
<tr>
<td>Lead Beneficiary</td>
<td>IBM</td>
</tr>
<tr>
<td>Editor</td>
<td>Benny Mandler</td>
</tr>
<tr>
<td>Reviewer</td>
<td>Fabio Antonelli</td>
</tr>
<tr>
<td>Reviewer</td>
<td>Alex Futasz</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>PU</td>
</tr>
<tr>
<td>Contractual Delivery Date</td>
<td>31/10/2015</td>
</tr>
<tr>
<td>Actual Delivery Date</td>
<td>31/10/2015</td>
</tr>
<tr>
<td>Version</td>
<td>V1.0</td>
</tr>
</tbody>
</table>
Abstract

The final 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 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 an IoT cloud based environment.
Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>01/07/2015</td>
<td>Initial skeleton version</td>
</tr>
<tr>
<td>V0.2</td>
<td>06/10/2015</td>
<td>First draft</td>
</tr>
<tr>
<td>V0.3</td>
<td>10/10/2015</td>
<td>Add demo details</td>
</tr>
<tr>
<td>V0.4</td>
<td>20/10/2015</td>
<td>Incorporate contribution from RETE</td>
</tr>
<tr>
<td>V0.5</td>
<td>27/10/2015</td>
<td>Incorporate contribution from PASSAU</td>
</tr>
<tr>
<td>V0.6</td>
<td>28/10/2015</td>
<td>Incorporate contribution from BSC / FOKUS</td>
</tr>
<tr>
<td>V1.0</td>
<td>31/10/2015</td>
<td>Finalize and integrate all contributions</td>
</tr>
</tbody>
</table>
Table of Contents

Abstract ......................................................................................................................................... 2
Document History ......................................................................................................................... 3
List of Figures ................................................................................................................................ 5
List of Tables .................................................................................................................................. 6
Acronyms ....................................................................................................................................... 7
1 Introduction ................................................................................................................................. 8
2 High level picture – Main components and interactions .......................................................... 8
   2.1 The developers portal ........................................................................................................ 10
      2.1.1 The developers portal environment ........................................................................... 11
   2.2 The cloud run-time ............................................................................................................ 11
      2.2.1 The run-time environment: ....................................................................................... 12
   2.3 Deployment and life cycle management ............................................................................. 13
      2.3.1 Deployment and life cycle management environment ................................................. 14
   2.4 Security ............................................................................................................................... 15
      2.4.1 Identity Management .................................................................................................. 15
      2.4.2 Data Provenance ....................................................................................................... 16
      2.4.3 Reputation Manager .................................................................................................. 16
      2.4.4 Contract Store .......................................................................................................... 17
      2.4.5 Usage Locks .............................................................................................................. 17
      2.4.6 Policy Information Point ............................................................................................ 18
      2.4.7 Policy Decision Point ............................................................................................... 18
      2.4.8 Dynamic Flow Control ............................................................................................... 19
      2.4.9 Static Enforcement and Conflict Resolution .............................................................. 19
2.5 Service Discovery

2.5.1 The service discovery environment

2.6 Data Management

2.6.1 The Data Management environment

3 What is being demonstrated

3.1 Detailed view

3.1.1 Web Dashboard:

4 Integrated platform Installation & configuration

List of Figures

Figure 1: Main COMPOSE platform components

Figure 2: Cloud run-time components

Figure 3: Interactions of the deployment component

Figure 4: Deployment component environment

Figure 5: Service Discovery deployment

Figure 6: Data Management deployment

Figure 7: Lamp Service Object

Figure 8: Shopping Cart Service Object

Figure 9: Smart Watch Service Object

Figure 10: Data Processing Pipe

Figure 11: Queue detector application

Figure 12: Notify cashier application

Figure 13: main demo workflow

Figure 14: Checkout counters models and coordinates
List of Tables

Table 1: Acronyms table ................................................................................................................ 7
## Acronyms

**Table 1: Acronyms table**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CF</td>
<td>Cloud Foundry</td>
</tr>
<tr>
<td>COMPOSE</td>
<td>Collaborative Open Market to Place Objects at your Service</td>
</tr>
<tr>
<td>DPP</td>
<td>Data processing Pipeline</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDM</td>
<td>Identity Management</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>JSON</td>
<td>Java Script Object Notation</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>PIP</td>
<td>Policy Information Point</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SO</td>
<td>Service Object</td>
</tr>
<tr>
<td>SU</td>
<td>Sensor Update</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol And RDF Query Language</td>
</tr>
<tr>
<td>UAA</td>
<td>User Account and Authentication</td>
</tr>
</tbody>
</table>
1 Introduction

This document accompanies the demonstration of the final integrated COMPOSE platform. In this version all of the capabilities envisioned within the platform are in place. Contributions from all the technical WPs have made it into this version of the platform, thus providing the full spectrum of COMPOSE platform capabilities.

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. The most relevant detailed deliverable is D4.1.1 – “Highly scalable runtime environment for the COMPOSE ecosystem”.

2 High level picture – Main components and interactions

Figure 1 presents an overall view 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\(^1\)), making it more suitable as a platform to serve the IoT domain. Thus, as can be seen in the figure, COMPOSE is a cloud platform, with specific capabilities that 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; the main exception being the developers’ portal, which is a crucial COMPOSE component that operates outside the cloud, but interacts tightly with it. 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 include assisted application development, through security, and all the way to automated application deployment into the cloud.

The developers’ portal is the only access point into the COMPOSE platform for end-users and developers as one. 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 forms the integrated COMPOSE platform. Figure 1 shows the core components of the COMPOSE back-end. Data flows from bottom-up through the data management layer and is made available to applications produced by the developers’ portal in a variety of manners. 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 between the cloud platform and 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 cloud applications, enabling the COMPOSE controller to use it, as well as external entities, such as the developers’ portal.

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.

![COMPOSE Platform Components Diagram](image)

**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 multi-tenant one stop shop for developers who wish to create IoT based applications in COMPOSE. This portal serves as the entry point for external users of the platform, whether the users are application developers, smart objects providers, or end-users who consume COMPOSE applications. 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, which are the COMPOSE internal digital counter-parts of real world physical smart 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 applications of their choice, using data coming from the smart objects, or directing commands to these smart objects. The smart object composer integrates capabilities for applications’ discovery, security and deployment. Finally, the automations component allows for sharing of created compositions with other registered users. Developers can choose to share their creations publicly which are persisted in storage. These persisted automations are available for other users for instantiation 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, and access to applications; identify applications with security flaws and recommend corrections.
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. Service recommendation – present the developers with recommendations concerning the choice between different services that can serve the same functionality.
5. Service composition – provide a front-end for the assisted composition services to help guide the developer to the COMPOSE application that will fulfil his requirements.
6. SDKs – to integrate and access various Smart Objects from the developer portal.
7. Service composition – to connect existing building blocks to workflow applications and make them accessible via RESTful APIs.
8. Data management (ServioTicy) for registering service objects and data processing pipes, and displaying their data streams.
9. Reputation: present to the developers the reputation score associated to COMPOSE entities of interest to him.

As of October the final prototype of the developers’ portal has about 320 registered users. Each month the portal has about 340 page views. The top five visiting countries are the United States of America, Germany, Italy, United Kingdom and India, in that order.
2.1.1 The developers portal environment

- Ubuntu 14.04.2 LTS operating system
- Developer portal implemented as Node.js v0.10.33 application
- Apps that are deployed to Cloud Foundry use Node.js v.0.10.33
- The Smart Object Manager uses Angular.js v1.2.10 framework
- The Smart Object Composer component based on Node-RED v0.10.7-git
- Uses MongoDB v2.6.3 for user management
- Service Objects API running in BSC serviced at http://api.servioticy.com/
- Lifecycle Management API (LCM) running in Cloud Foundry (http://docs.composelifecycle.apiary.io/)
- iServe API running in Cloud Foundry (http://compose.bsc.es:9082/iserve/docs/)
- Identity Management API (IDM) running in Cloud Foundry (http://docs.composeidmusers.apiary.io/)

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 discovery 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](image-url)
The main interactions of this component are with:

1. Deployment (COMPOSE controller) – All COMPOSE applications are deployed as Cloud Foundry (CF) applications using a predefined Node-RED template with Node.js runtime. The life cycle management component, which is a part of the deployment component, absorbs its information from the COMPOSE run-time.

2. Service discovery – operates as a part of the cloud environment. This component’s back-end runs as a COMPOSE service which COMPOSE applications can bind to. Internally it uses services offered by the cloud run-time, such as a database.

3. Data Management – operates as a part of the cloud environment. This component’s back-end runs as a COMPOSE service which COMPOSE applications can bind to.

4. Security – COMPOSE security interacts with cloud security mechanisms provided by the UAA which is part of the CF ecosystem, and adds specific security capabilities. The users that are created in COMPOSE are essentially UAA users, so each major action like creating or deleting COMPOSE applications is authenticated by CF itself.

5. Scalable communication infrastructure which is used, among other capabilities, for connecting between the service objects and the COMPOSE applications provides applications with an easy way of sharing data via pub/sub semantics, and is also built from the ground-up with auto-recovery and scalability considerations.

6. Monitoring infrastructure which collects information on the state of service objects, COMPOSE applications, COMPOSE infrastructure, and cloud resources. The monitoring infrastructure taps into CF’s native message passing mechanism (the NATS service) in order to minimize latency and incorrect classifications (false positive/negative).

2.2.1 The run-time environment:

- The COMPOSE cloud is currently comprised of three Machines installed with Ubuntu 10.04.4 LTS:
  - cloud01: runs a Cloud Foundry environment (build 175, installed via nise bosh)
  - abiell: functions as an additional DEA server for the CF environment, and also hosts additional services.
  - Cfdea: functions as the main DEA server as it has 130GB of RAM.
- External services run on cloud01 and abiell, and are introduced to CF apps using service brokers which their backend runs locally and their front-end runs as CF apps.
  - Compose Communication Bus (CSB) runs as a CF application – 3 instances total, each instance embeds a CSB node inside a web application – which is used for auto-discovery of all peer nodes via http requests in order to establish a shared view of external IP addresses and routable ports of all instances, which in turn is published to CSB client applications by having them query any of the 3 instances' web application.
  - MySQL-shared: Binding to the instance creates a database and a new user. Unbinding deletes the user and the database from the DB. The service is available both in cloud01 and in abiell (essentially exposing 2 different services).
  - iServe: The iServe instance runs as a Tomcat servlet on cloud01.
- Redis 2.8.2 – hosted on cloud01, and is used by iServe for caching discovery related entities and compositions.
- MongoDB – Used by the Security Server, installed on abiell and the bind returns simply the MongoDB URI.
- Cloud Foundry instance managed with cf-cli (go executable)
- Service Objects API running in BSC serviced at http://api.servioticy.com/
- The Identity Management service (sec. 2.4.1) runs as a CF app and stores data inside the MySQL service which it is bound to.
- The Security Server is bound to the MongoDB service and uses it to store security policies.
- The COMPOSE controller consists of 2 independent parts:
  - The Life Cycle Manager (LCM) – a Node.js application that is deployed as a CF app and uses the MySQL service for persistent storage. Its functions are tracking registration and deployment of all COMPOSE entities in iServe and in ServioTicy.
  - The Mediator – a java web servlet that has no need of persistence, and simply deploys COMPOSE applications to CF on behalf of COMPOSE users (through the LCM). It receives workflows (JSON objects describing the application logic designed in the Developers’ Portal), prepares a Node-RED runtime for them and deploys them to CF by implementing the Cloud Controller client API.

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 can be deployed, hosted, and managed within ServioTicy or the COMPOSE CF-based cloud platform and added services. In addition, this component tracks and governs the lifecycle management of COMPOSE applications. These COMPOSE entities go inside the lifecycle manager which provides a check for security and policy using the IDM and PDP components, and the registry in the iServe catalogue.

![Figure 3: Interactions of the deployment component](image)

The deployment component works internally with a mediator component which is in charge of the actual deployment to the cloud via the cloud controller. The mediator serves also as an orchestration engine which takes care of run-time aspects of COMPOSE workflows being
deployed to the cloud platform. In addition, the deployment component interacts internally with ServioTicy to deploy Service Object and Data Processing Pipes.

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 – The deployment component interacts with IDM for authentication and authorization of the COMPOSE entities. There are also other interactions to verify that the user can deploy, start, stop or delete an entity.
2. Cloud run-time – For the actual deployment of the application within the COMPOSE cloud run-time.
3. Data Management – For the creation of service objects and data processing pipe.
4. Service discovery – For registering entities.

2.3.1 Deployment and life cycle management environment

The deployment and life cycle management is composed of different CF services and CF applications running in BSC. It further interacts with the cloud controller in order to deploy, monitor, and manage COMPOSE applications.

![Figure 4: Deployment component environment](image)

- Life cycle management: Developed using node.js (0.10.33).
  - The main modules that LCM depends on are async, dot, ecstatic, express, log4js, node-uuid, sequelize and urllib.
There is an API to interact with this component (docs.composelifecycle.apiary.io)

LCM interacts directly with these other COMPOSE Components
- IDM: Identity management
- Mediator: to deploy to Cloud Foundry
- iServe: to register entities
- PDP: to check policy
- ServioTicy: to deploy service objects and data processing pipes.

MySQL – Used by the life cycle management to persist data relevant for the LCM itself. It is used also as a file repository.

Prototype Service Life Cycle: is built using node.js (0.10.33).

The main modules that the LCM Prototype depends on are express, jade, stylus, urllib,

It's a web application used to test the LCM API.

The LCM Prototype interacts directly with these other COMPOSE Components
- LCM.
- IDM.

### 2.4 Security

#### 2.4.1 Identity Management

Identity management deals with authentication of entities, and administration of their identity information. Identity Management is a prerequisite in order to define security policies, since an entity, and its entity information must be referenced, so the policy enforcement framework can choose the right policy for a given entity.

Identity Management is deployed as a cloud foundry application, which can 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 for COMPOSE.

To account for scalability, identity information for Service Objects is replicated in the data management layer, so it can be accessed by the local Policy Decision Point locally. To replicate the identity management information effectively between identity management and the data management layer, a private API exposed by the data management layer is used. Furthermore, to deal with the same problem in the application runtime, identity management uses the CSB messaging system defined and implemented in WP4 to send real-time updates to the local Policy Decision Points running in the application runtime too.

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. It registers a user in the proper Cloud Foundry space with the Cloud Controller to enable him/her to push COMPOSE applications to the cloud.
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 the Data Management layer.

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.

5. **Reputation manager**: Identity management authenticates users providing feedback for COMPOSE entities.

### 2.4.2 Data Provenance

The integrated data provenance module collects precise information about the heritage of data items in the data management layer (WP2) and also in the application layer. This information is retrieved by modifications to the corresponding execution environment. In case of the data management layer this execution environment is a modified Rhino interpreter that analyses the user-defined source code of service-objects (SO) or data processing pipelines. In case of the application layer the execution environment is a modified Node-RED instance.

The provenance data representation is in JSON format, and is part of the security meta-data. It has a unified format throughout the application and data management layer. The provenance data for sensor updates is stored together with the actual data item 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 particular SOs or DPPs. The provenance data in the application layer flows during runtime with the actual messages inside Node-RED.

The data provenance module interacts with the data management layer during the generation of new data item (SU). This can be the case when a new data item is pushed to the platform as well as when data items are dispatched in the data management layer. The integrated provenance module has also the capabilities to transmit and merge provenance data between both layers. This is necessary if a sensor update is read inside the application layer, in this case the provenance data of the SU is merged with the current provenance data of the message in the application layer.

### 2.4.3 Reputation Manager

The reputation manager collects information from the Data Management Layer, and from the application runtime, in order to hold information about popularity and activity of Service Objects, Data Processing Pipes, applications and workflows. Furthermore, it provides aggregated reputation information to the Developers’ Portal, and also to the service discovery. The latter allows users to define trust preferences which include reputation values from COMPOSE entities. Additionally, the reputation API is accessible from COMPOSE applications, in case developers want to build applications leveraging its functionality.

This component interacts with:

1. Application runtime local PDP: to collect events showing interaction between applications. This is used for popularity.

2. Data Management: The data management component also logs events showing whenever a sensor update was delivered successfully or failed to do so, in Service
Objects and Data Processing Pipes. This information is then fed into the reputation manager, in order to calculate popularity and activity scores for entities within the data management layer.

3. Developers’ Portal: The developers portal collects feedback information, and at the same time displays reputation information to users.

4. Service Discovery: The API from the reputation manager is used by the trust component inside the service discovery to influence the results shown to the user by considering the reputation values of COMPOSE application, Service Objects, etc.

5. Identity Management: In order to ensure that only authenticated users can provide feedback, the reputation API uses identity management to verify and obtain identity information about the user providing feedback.

2.4.4 Contract Store

The contract store is used to persistently store and retrieve contracts. This includes the contracts generated by the static analysis as well as developer contracts. Developer contracts are able to refine the automatically generated contracts with the help of additional information provided by the developers of the corresponding entity.

Contracts for all kind of entities can be stored and retrieved by the contract store, this includes among others SOs, DPPs, Node-RED nodes, COMPOSE-applications. Contracts are mainly generated by the static information flow analysis, performed with the modified TAJS. This process is a computation intensive process and therefore this process should not be performed several times for the same entity, highlighting the importance of the contract store.

The contracts for entities are mainly used in the static workflow analysis which analyses also the information flow between entities. This analysis is performed during the verification step of COMPOSE-applications and COMPOSE-workflows. This process triggers also the contract generation of single entities that do not yet have a contract.

This verification step itself is integrated into the life cycle management component and is triggered during deployment. It is also possible to trigger the verification directly in the user interface.

In order to improve the scalability of the contract store it is possible to have a local instance of the contract store that stores the information for this particular instance relevant contracts. The local instances are synchronised with the central contract store with the help of the CSB messaging system. The central contract store is deployed in Cloud foundry and the interaction with it is possible via a REST API. Local instances are directly included in the corresponding environment.

2.4.5 Usage Locks

The policy language Usage Locks, specifically developed for this project, is used in all aspects related to policy information, evaluation, and decision as well as for contract and conflict description and definition. Appropriate processing primitives and user interfaces have been developed and integrated.
2.4.6 Policy Information Point

The policy information point (PIP) is the component used to persistently store and retrieve policies for all kinds of entities. There are local as well as a central PIP, to provide fast access to policies for a high number of requests. The interaction to the central PIP, which is deployed in Cloud foundry is realised via a REST API. The policies in the local PIPs can be directly accessed by its corresponding component, into which the local PIP was integrated. Local PIPs are automatically synchronised with the central PIP in order to propagate policy changes with the help of the CSB messaging system.

This component interacts with:

1. Application runtime: In the application runtime the PIP provides the required policies for the enforcement.
2. Developers Portal: The developers portal provides the interface for developers to specify policies for their entities. These policies are then stored in the PIP.
3. Service Discovery: The filter implemented in iServe to retrieve only entities that are accessible by the person that sends the query uses the PIP to get the corresponding policies.
4. Life cycle: During all steps of the life cycle of entities the PIP provides the required policies for the enforcement.

2.4.7 Policy Decision Point

The policy decision point (PDP) evaluates policies and detects if a certain flow or access is allowed or not. This evaluation has to be performed in several places and components to ensure that information is only accessible by authorized entities.

To provide a scalable policy evaluation, local instances of the PDP are used. These local instances are included in several COMPOSE infrastructure components.

The PDP is integrated into:

1. Data management layer: Inside the data management layer the PDP is necessary to support efficient policy evaluation during enforcement of flows, during dispatching, when interacting with data via its API, and during creation of subscriptions using MQTT or CSB.
2. Application layer: The application layer consists of a modified runtime of Node-RED. During deployment of a flow, the access between data processing pipelines, COMPOSE applications, and users to data or applications is controlled using local policy decision points.
3. Life cycle management: Before the deployment and also before the state changes of a deployed entity, policy evaluation takes place.
4. Discovery: A filter is integrated into iServe to filter query results based on the policies together with the user sending the query.
5. Policy information point: Policy evaluation is required for access control to the PIP.
2.4.8 Dynamic Flow Control

The policy decisions mentioned in the previous section directly trigger policy enforcement. Apart from the integration of the appropriate policy enforcement points, the COMPOSE runtime also deploys dynamic flow control. For this purpose Node-RED has been adjusted as follows:

1. During message exchange inside Node-RED, policy information about each message is maintained and communicated to the next node to be executed. Thus, flow control can be enforced for each single message and its respective fields.

2. Each message also carries further state information generated by security services, e.g. about the user using an application. Messages also maintain the state of security locks which have been opened or closed by security services, e.g. whether a user was authenticated or not.

3. In order to support the conservative propagation of data-centric security policies the execution of the pre-deployed node types in Node-RED uses a contract. The contract specifies how messages are processed and how they change their security policies. For the execution of function nodes which can further execute user-defined JavaScript, a modified version of JSFlow is deployed which performs dynamic policy propagation.

2.4.9 Static Enforcement and Conflict Resolution

In order to simplify the development of secure and accessible applications, we integrated a static analysis component in the marketplace. It processes a workflow generated in glue.things and verifies the correctness of access between nodes and the compliance of the flows of information between nodes and COMPOSE applications. An appropriate user interface has been integrated into glue.things. The latter also processes the results of this analysis. An extra security tab in the sideboard of Node-RED allows the visualisation of various error types.

Further, the static analysis component is supported by a reconfiguration component. It processes the errors found by the static enforcement component and tries to solve the conflicts therein. A rated set of solutions is communicated to glue.things which can display the required changes to a workflow to make it compliant with its security policies. The user can further apply a solution to a workflow to fix it.

2.5 Service Discovery

With the vast amount of applications and service objects anticipated in a platform such as 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.
Each newly registered service object in the platform gets registered with the service discovery component as a part of its registration process. The same holds for each new application created and deployed in the platform. A developer can pose semantic queries from the developers’ portal to look for building blocks he can use for a newly designed application. Once found, such an entity immediately becomes an operational block within the application being designed.

### 2.5.1 The service discovery environment

At the back-end a Tomcat7 servlet hosts iServe as a COMPOSE platform service. iServe in turn is connected to an application-layer load-balancer, running as a jetty application, that redirects SPARQL queries to a PaxDB cluster.

The PaxDB cluster consists of 3 PaxDB nodes (each running on a different physical server) with a single master which receives write requests, while all cluster nodes can service read requests (which are randomly distributed among all nodes). The master and all nodes are periodically probed by the application-layer load balancer, and if a node is considered faulty, it won’t receive requests until it successfully responds to a probe. If the master fails, a new master is elected.

In order to respond to free text search queries, each PaxDB instance uses a Solr cluster (3 nodes in total, 1 for each PaxDB instance hosted in the same physical server to maximise redundancy and minimize network round-trips).

### 2.6 Data Management

The data management service forms an ingestion layer which takes care of the bi-directional communication with the external smart objects. In addition this component takes care of storing the data flowing into the platform from the smart objects, as well as providing real-time data processing, manipulation, and notification capabilities. 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 data 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 6: Data Management deployment

3 What is being demonstrated

The cover story is an application (COMPOSE workflow) that takes place in a supermarket; the same supermarket that is used for the COMPOSE smart retail pilot. The application will identify when a long queue is formed in front of one of the cash registers and will initiate a process to open a new cash register. The cash register to be opened will be signalled by a light bulb being turned on, and an employee will be selected to attend the new cash register. The selected employee will be the one located closest to the cash register to be opened, and he will be alerted by a message sent to his smart watch.

COMPOSE Entities which comprise the demonstration:
- Service Objects: lamp (see Figure 7), shopping carts (see Figure 8), smart watches (see Figure 9), active TAGs
- Data Processing Pipes: geo fence around the cash registers (see Figure 10), carts aggregation.
- COMPOSE applications: Queue Detector (see Figure 11), Closest employee to the cash register, Notification Manager (see Figure 12), Cash register selection, Cash register light switch
- COMPOSE workflows: The combined demo (see Figure 13)

```json
{
    "public": "True",
    "name": "Lamp 01",
    "actions": [
        {
            "name": "status",
            "description": "Set lamp status: on/off"
        }
    ],
    "streams": {},
    "description": "Lamp for the COMPOSE final demo - Unit 01"
}
```

**Figure 7: Lamp Service Object**

```json
{
    "public": "True",
    "name": "Shopping cart 01",
    "actions": [],
    "streams": {
        "position": {
            "channels": {
                "location": {
                    "type": "geo_point",
                    "unit": "degrees"
                },
                "id": {
                    "type": "number",
                    "unit": "identifier"
                }
            }
        },
        "description": "Shopping cart for the COMPOSE final demo - Unit 01"
    }
}
```

**Figure 8: Shopping Cart Service Object**
Figure 9: Smart Watch Service Object

```json
{
  "public": "True",
  "name": "Employee Watch 1",
  "streams": {
    "position": {
      "channels": {
        "location": {
          "type": "geo_point",
          "unit": "coordinates"
        },
        "id": {
          "type": "number",
          "unit": "Identifier"
        }
      }
    },
    "actions": [ {
      "name": "vibrate",
      "description": "Vibrate the Wearable"
    },
    {
      "description": "Wearable for final demo of COPOSE"
    }
  ]
}
```
```json
{
  "name": "Shopping cart aggregation",
  "version": "0.1.0",
  "groups": {
    "AllCarts": {
      "sold": null,
      "stream": "position"
    }
  },
  "streams": {
    "aggregated": {
      "channels": {
        "location": {
          "current-value": "${AllCarts.channels.location.current-value}",
          "type": "geo_point",
          "unit": "degree"
        },
        "id": {
          "current-value": "${AllCarts.channels.id.current-value}"
        }
      }
    },
    "inLocation": {
      "pre-filter": "parseFloat(${AllCarts.channels.location.current-value}.split(',\"\")[0]) > 48.38970 & parseFloat(${AllCarts.channels.location.current-value})",
      "channels": {
        "location": {
          "current-value": "${AllCarts.channels.location.current-value}"
        },
        "id": {
          "current-value": "${AllCarts.channels.id.current-value}"
        }
      }
    }
  }
}
```

**Figure 10: Data Processing Pipe**

![Data Processing Pipe Diagram]

**Figure 11: Queue detector application**
The combined capabilities of all COMPOSE components are demonstrated. Naturally not all capabilities of all components are demonstrated, but rather the major capabilities and interactions are shown.

1. The full flow of registering a new Service Object (or a data processing pipe) into the platform is demonstrated.

   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 (or a workflow) 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 is deployed and runs on the underlying cloud infrastructure.

4. Interaction with the service discovery component, as previously registered entities are located by queries invoked from the developers’ portal, and used in new applications. In addition, workflows are composed by using the underlying assisted composition engine.

5. Scalable communication infrastructure is being used by adding notifications from service objects to COMPOSE applications.

### 3.1 Detailed view

The final COMPOSE demo will showcase most of the main features enabled by the COMPOSE platform. The demo will be based on the smart retail application scenario. The story line will be centered on the supermarket located in Trento, where the localization infrastructure is currently deployed.

Among the COMPOSE capabilities we intend to highlight are:

- Common model for easy integration of heterogeneous smart objects
- Easy creation of application based on object data streams
- Object marketplace, same objects are used under different views (brands, store managers, users, etc.).
- Integrated security aspects
- Multitenancy building applications (many developers, integrating different applications)

Enclosed is a list of COMPOSE components that have been used:

<table>
<thead>
<tr>
<th>Feature</th>
<th>COMPOSE Enabler</th>
<th>Description/Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect sensor information: location</td>
<td>ServioTicy - Service Objects</td>
<td>Location info from carts and employees. Present a unified model to collect information from different sources. Scalable and real time data ingestion and processing.</td>
</tr>
<tr>
<td>Service</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Aggregation of sensor information</td>
<td>ServioTicy - Data Processing Pipeline</td>
<td>Provide developers with consumable data processing capabilities to ease his work. For example, amount of carts in a designated area.</td>
</tr>
<tr>
<td>Login / access control</td>
<td>IDM</td>
<td>Enable login for personalization, access control integrated in the platform.</td>
</tr>
<tr>
<td>Real time notifications</td>
<td>ServioTicy - Pub/Sub mechanisms</td>
<td>Notifications between service objects (DPPs) and a running application</td>
</tr>
<tr>
<td>Service Objects Management dashboard</td>
<td>glue.things - dashboard</td>
<td>Creation of shopping carts description from the interface. A central point of management.</td>
</tr>
<tr>
<td>Discovery of existing capabilities (SOs, applications)</td>
<td>iServe</td>
<td>using glue.thing’s search node</td>
</tr>
<tr>
<td>Integration of existing capabilities</td>
<td>glue.things - composer</td>
<td>Creation of applications and workflows.</td>
</tr>
<tr>
<td>Creation of reusable services (scanthng/evt api/like/dislike products)</td>
<td>glue.things - composer</td>
<td>Applications re-used within a workflow</td>
</tr>
<tr>
<td>Cloud deployment</td>
<td>Deployment infrastructure</td>
<td>All entities deployed within the cloud infrastructure</td>
</tr>
<tr>
<td>Flow Control</td>
<td>Usage Lock Policies and Dynamic Enforcement</td>
<td>User can assign fine-grained policies to individual data items. In this way the distribution of data can be controlled on the data management as well as application layer.</td>
</tr>
</tbody>
</table>

**Demo Flow**

<table>
<thead>
<tr>
<th>Story telling</th>
<th>Action</th>
<th>What is shown</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the store,</td>
<td>Move inside the store with the shopping carts/baskets Employees move around the store as well.</td>
<td>Pictures of the store + video.</td>
</tr>
<tr>
<td>- Every single shopping cart/basket is associated to an active TAG. By doing this, it is possible to localize in real-time where each and every asset is in the store.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Staff members are wearing a smartwatch and carry a smartphone. In addition an active TAG is localising them in real-time inside the store and used to receive real-time notifications on the operations of the store.

Both the tags associated with the shopping cart/basket and the smart watches are represented by a Service Object that captures the object information and receives real-time updates of the location. Further, the objects can also be actuated, as an example for sending notifications.

The scenario is deployed in the supermarket SAIT in Trento.

<table>
<thead>
<tr>
<th>Use case 1: Real-time view</th>
<th>Action</th>
<th>What is shown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show real-time view of the store manager application. In this view, it is possible to monitor the mobility of customers in real-time. This dashboard can be used by the store manager to obtain real-time information on the store operations and on customers’ behavior.</td>
<td>Move inside the store with the shopping carts/baskets.</td>
<td>Web application: we show the real-time web view</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case 2: Queue monitoring</th>
<th>Action</th>
<th>What is shown</th>
</tr>
</thead>
<tbody>
<tr>
<td>The application detects queues forming at the checkout counter and notifies this to all applications registered to it.</td>
<td>Baskets/carts moving all together at the checkout counter.</td>
<td>Web application: we show a red icon when a queue is forming.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case 3: Checkout counter opening/closing</th>
<th>Action</th>
<th>What is shown</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the forming of a queue is detected the application notifies this to store staff members. The notification is received on their smart watch. In particular, only the staff member that is closest to the checkout counter to be opened is notified. It is important to notice that the location of staff members is displayed or not on the web application based on the security / privacy policy applied.</td>
<td>Baskets/carts moving all together at the checkout counter. When a new checkout counter is opened, customers redistribute to the checkout counters which are opened.</td>
<td>Store web application: user moving in the store. Smart watch: a notification is being sent to the smart-watch of the staff closest to the checkout.</td>
</tr>
</tbody>
</table>
Lights: when a new checkout counter is being opened, the light switches to green.

Figure 14: Checkout counters models and coordinates

3.1.1 Web Dashboard:
- The dashboard presents a real-time view of the store status. This includes:
  - all carts moving in the store
  - all staff members moving in the store
  - status of lights which are switched on/off
  - a flag that turns green when a queue is detected

4 Integrated platform Installation & configuration

Deployed in the Barcelona Supercomputing Center (BSC) server farm on 3 servers for hosting the IoT PaaS platform:
- cloud01 (147.83.30.133): 4x2GHz 16GB RAM
• abiell (147.83.42.190): 2x3.2GHz 6GB RAM
• cfdea.pc.ac.upc.edu (147.83.42.216): 16x2.4GHz, 130GB(!) RAM

The OS of all 3 is Ubuntu Lucid 10.04.4 LTS 64bit.

cloud01 functions as the main Cloud Foundry server and hosts all Cloud Foundry components, while abiell and cfdea function as DEA servers (run Cloud Foundry applications) and also host the backend services.

ServioTicy is deployed in a separate server farm of 4 servers also located at the Barcelona Supercomputing Center (BSC), plus additionally 2 VMs for auxiliary services such as domain rewriting and load balancing. The 4 servers devoted to ServioTicy are all identical and have the following specs:

• minerva-1001..1004: 12x2.0Ghz, 192GB RAM, 6TB HDD, 1.6TB SSD (6 SSDs per node configured in RAID-0 to maximize I/O bandwidth), running Ubuntu 12.04LTS.

All nodes in the BSC deployment are connected through a non-blocking 1GbE switching fabric composed of 2xCisco Catalyst 3750X stacked for single logical management.

In terms of software, a service broker is installed on both abiell and cloud01 and consists of two parts:

- The 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.
- The 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:

- The communication infrastructure (CSB) runs embedded in a web application with 3 instances (Cloud Foundry runs each instance on a different server for redundancy). Each instance performs auto-discovery of its peer nodes via http requests to the we application's service endpoint, and gossips the known information to its peer nodes in order to establish a shared view of external IP addresses and routable ports of all instances, and only then the communication infrastructure is spawned with the information of the peer nodes.

- 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. The iServe instance runs as a Tomcat servlet on cloud01.

- MySQL-shared: Binding to the instance creates a database and a new user. Unbinding deletes the user and the database from the DB. The service is available both in cloud01 and in abiell.

- ServioTicy: described in sub-section 2.6.1

- Redis 2.8.2 – hosted on cloud01, and is used by iServe for caching discovery related entities and compositions.
- MongoDB – Used by the Security Server, installed on abiell and the bind returns simply the MongoDB URI.