SOCIETIES
Deliverable D4.3

SOCIETIES Platform Specification and Design, Revised

<table>
<thead>
<tr>
<th>Editor:</th>
<th>David McKitterick (INTEL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliverable nature:</td>
<td>Design Document</td>
</tr>
<tr>
<td>Dissemination level:</td>
<td>PU</td>
</tr>
<tr>
<td>(Confidentiality)</td>
<td></td>
</tr>
<tr>
<td>Contractual delivery date:</td>
<td>31st May 2013</td>
</tr>
<tr>
<td>Actual delivery date:</td>
<td>5th June 2013</td>
</tr>
<tr>
<td>Suggested readers:</td>
<td>Pervasive and social computing designers, researchers and developers</td>
</tr>
<tr>
<td>Version:</td>
<td>1.0</td>
</tr>
<tr>
<td>Total number of pages:</td>
<td>42</td>
</tr>
<tr>
<td>Keywords:</td>
<td>SOCIETIES, Self Orchestrating Community ambiEnT Intelligence Spaces, Communications, devices, services, dependability, social networks</td>
</tr>
</tbody>
</table>

Abstract

The intention of this deliverable is to outline the revised specification and design of the SOCIETIES platform for the integrated prototype development within work package 4. This work package aims to research, design and prototype the SOCIETIES platform infrastructure and services that will be required to enable the SOCIETIES individual and community experiences while realising the concept of an Ambient Intelligent (AmI) Community, and therefore supporting the third party services and end user scenarios that will be developed within the other work packages.

The SOCIETIES platform design aims to specify a modular, scalable architecture that is compatible with state-of-the-art pervasive and social computing systems but also adaptive to the variety of novel use case scenarios which are to be evaluated in this project. This design deliverable is a revision of D4.1 ‘SOCIETIES Platform Specification and Design’ where the design of the platform has been updated based on technical requirements, system architecture design and technology considerations arising from the deployment of a prototype implementation.
Disclaimer

This document contains material, which is the copyright of certain SOCIETIES consortium parties, and may not be reproduced or copied without permission.

In case of Public (PU):
All SOCIETIES consortium parties have agreed to full publication of this document.

In case of Restricted to Programme (PP):
All SOCIETIES consortium parties have agreed to make this document available on request to other framework programme participants.

In case of Restricted to Group (RE):
All SOCIETIES consortium parties have agreed to full publication of this document. However this document is written for being used by <organisation / other project / company etc.> as <a contribution to standardisation / material for consideration in product development etc.>.

In case of Consortium confidential (CO):
The information contained in this document is the proprietary confidential information of the SOCIETIES consortium and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the SOCIETIES consortium as a whole, nor a certain party of the SOCIETIES consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Impressum

[Full project title] Self Orchestrating Community Ambient Intelligence Spaces
[Short project title] SOCIETIES
[Number and title of work-package] WP4 SOCIETIES(CSS) Platform
[Editor: Name, company] David McKitterick, INTEL
[Work-package leader: Name, company] Alec Leckey, INTEL
[Estimation of PM spent on the Deliverable] 6 PMs

Copyright notice

© 2011-2013 Participants in project SOCIETIES
Executive Summary

The vision of the SOCIETIES project is to research and develop a concept of Ambient Intelligent (AmI) Communities (also known as Pervasive Communities) that extend ambient intelligent or pervasive systems beyond the individual to dynamic communities of users. Driven by context awareness, preference learning and privacy protection, AmI Communities will intelligently connect people & things to communicate, share, consume and organise communities. SOCIETIES will embrace online community services, such as Social Networking, thus offering new and powerful ways of working, communicating and socialising. The project intends developing and trialling use cases for disaster management, university living and conference support.

An AmI Community is a group of, two or more, individuals who have agreed to share some, but not necessarily all, of their pervasive resources – personal information, physical context information, services, and devices – with other members of that community. AmI Communities have the potential to completely transform traditional online social networks, freeing them from web-applications and letting them loose in the real physical world. SOCIETIES supports the creation of AmI Communities by discovering, connecting and organising relevant people and things from both physical and digital environments. SOCIETIES will use pervasive technologies to form adaptive communities, while leveraging social networks and crowd computing techniques.

Within a large integrated project, work package 4 (WP4) aims to research, design and prototype the SOCIETIES platform infrastructure and services that will be required to enable the SOCIETIES individual and community experiences (WP5) while realising the concept of an AmI Community, and therefore supporting the third party services (WP6) and end user trials scenarios (WP8) that will be developed within the other work packages. This deliverable outlines the specification and design of the SOCIETIES platform to be prototyped and evaluated during user trials.

This design deliverable is a revision of D4.1 ‘SOCIETIES Platform Specification and Design’ (1) where the design of various platform subsystems have been updated based on technical requirements, system architecture design and technology considerations arising from the deployment of a prototype implementation. D4.1 included summaries on background technical research that was conducted based on research objectives and perceived technical problems, and design patterns for failure, discovery and interoperability. The SOCIETIES platform design aims to specify a modular, scalable architecture that is compatible with state-of-the-art pervasive and social computing systems but also adaptive to the variety of novel use case scenarios which are to be evaluated in this project.
# List of Authors

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL</td>
<td>David McKitterick, Alec Leckey, Maria Mannion</td>
</tr>
<tr>
<td>TSSG</td>
<td>Alan Walsh, Liam Marshall</td>
</tr>
<tr>
<td>SINTEF</td>
<td>Thomas Vilarinho, Bjorn Magnus Mathisen</td>
</tr>
<tr>
<td>PTIN</td>
<td>João Gonçalves, Sancho Rego</td>
</tr>
<tr>
<td>SETCCE</td>
<td>Mitja Vardjan</td>
</tr>
<tr>
<td>TI</td>
<td>Luca Lamorte</td>
</tr>
<tr>
<td>TRIALOG</td>
<td>Rafik SAID Mansour</td>
</tr>
</tbody>
</table>
# Table of Contents

- Executive Summary.................................................................................................................. 3
- List of Authors.......................................................................................................................... 4
- Table of Contents ...................................................................................................................... 5
- Table of Figures ....................................................................................................................... 6
- 1 Introduction ............................................................................................................................ 7
- 2 High Level Design of the SOCIETIES Platform ..................................................................... 9
  - 2.1 SOCIETIES System Architecture ................................................................................... 9
  - 2.2 Interoperability with the SOCIETIES Platform ............................................................... 11
  - 2.3 WP4 SOCIETIES Platform .............................................................................................. 12
- 3 SOCIETIES Platform WP4 Subsystems ................................................................................. 13
  - 3.1 Communication Framework ............................................................................................ 14
  - 3.2 CSS Management ............................................................................................................ 19
  - 3.3 CIS Management ............................................................................................................. 23
  - 3.4 Activity Feed .................................................................................................................... 27
  - 3.5 Security ............................................................................................................................ 28
  - 3.6 Social Network Integration .............................................................................................. 31
  - 3.7 Device Management ......................................................................................................... 33
  - 3.8 Service Management ....................................................................................................... 35
  - 3.9 Domain Authority & Directory Services .......................................................................... 36
  - 3.10 SOCIETIES Platform Light Client Application .............................................................. 37
  - 3.11 SOCIETIES Platform Web Application .......................................................................... 39
- 4 Conclusion ............................................................................................................................... 40
- 5 List of Abbreviations ............................................................................................................... 41
- 6 References ................................................................................................................................ 42
### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOCIETIES System Architecture</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Functional Viewpoint of the SOCIETIES Platform</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Deployment Viewpoint of System Architecture</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>WP4 Functional Services from System Architecture</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>SOCIETIES Platform WP4 Cloud Subsystems</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>SOCIETIES Cloud Communication components</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>SOCIETIES Communication components</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>Finite State Machine diagram of XMPP Connection</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>High level overview diagram of the CSS Manager component</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>Interoperability of the CSS Management Subsystem</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>CIS Subsystem</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>Creating a new CIS</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Multi CIS Subsystems</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>Activity Feed Subsystem</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>Secure Policy Negotiator</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>Policy Negotiation for Service Sharing</td>
<td>30</td>
</tr>
<tr>
<td>17</td>
<td>Service Shared Installation</td>
<td>30</td>
</tr>
<tr>
<td>18</td>
<td>Social Network Integration Subsystem</td>
<td>31</td>
</tr>
<tr>
<td>19</td>
<td>Device Management Subsystem</td>
<td>34</td>
</tr>
<tr>
<td>20</td>
<td>Using Device Management to integrate drivers and 3rd party services</td>
<td>34</td>
</tr>
<tr>
<td>21</td>
<td>Service Management subsystem</td>
<td>35</td>
</tr>
<tr>
<td>22</td>
<td>Domain Authority Design</td>
<td>36</td>
</tr>
<tr>
<td>23</td>
<td>Component Design of Light Client Application</td>
<td>37</td>
</tr>
<tr>
<td>24</td>
<td>Web Application Deployment</td>
<td>39</td>
</tr>
</tbody>
</table>
1 Introduction

An Ambient Intelligent (AmI) Community (also known as a Pervasive Community) is a group of, two or more, individuals who have agreed to share some, but not necessarily all, of their pervasive resources – personal information, context data, services, devices – with other members of that community. AmI Communities have the potential to completely transform traditional online social networks, freeing them from web-applications and letting them loose in the real physical world. SOCIETIES supports the creation of AmI Communities by discovering, connecting and organising relevant people and things from both physical and digital environments. SOCIETIES will use pervasive technologies to form adaptive communities, while leveraging social networks and crowd computing techniques.

The enabling features of the SOCIETIES platform for discovering, connecting and organising communities are driven by context awareness, preference and intent learning and privacy protection. SOCIETIES will help users discover people and communities relevant to them and their context so they can communicate, share and collaborate. In our university living scenarios, relevant students can discover each other to discuss topics, share study notes and meet up when they are automatically discovered to be nearby. SOCIETIES will enable users to connect to other people and things relevant to them in order to discover information and consume services. In our conference support scenarios, attendees can access personalised agendas and real-time session streams, while utilising indoor navigation and professional networking services. SOCIETIES will empower AmI Communities to be self-organising by utilising the community context and preferences to direct actions, taken on behalf of the community. In our disaster management scenarios, disaster victims will be safely evacuated away from disaster areas by providing individual evacuation instructions, based on the collective community context.

This project builds on previous research from the PERSIST project (2) which defined a context-aware and self-improving personal smart space (PSS) for users to interact dynamically with smart environments and other users’ devices. Building on the PSS concept, SOCIETIES has introduced a set of more community-centric concepts. A Co-operating Smart Space (CSS) represents a single participant (user or organisation), and includes their information, and services within a distributed system of CSS Nodes (user devices/cloud instances). A Community Interaction Space (CIS) represents and provides the interaction mechanisms for an AmI community to exist. Members of an AmI community interact with a CIS via their own personal CSS. The individual members of an AmI community do not need to be human beings but can also be organisations, smart space infrastructures or autonomous agents. AmI Communities can be dynamic in nature, with CISs being formed in an ad-hoc fashion. CISs can also spawn sub-communities or merge with other communities, and will be self-orchestrating, making decisions for their members based on the community content and preferences.

The intention of this deliverable is to outline a revision of the specification and design of SOCIETIES platform for the integrated prototype development within work package 4. This work package aims to research, design and prototype the SOCIETIES platform infrastructure and services that will be required to enable the SOCIETIES individual and community experiences (WP5) while realising the concept of an Ambient Intelligent (AmI) or Pervasive Community, and therefore providing integrated platform features to third party services (WP6) and supporting end user scenarios (WP8) that will be developed within the other work packages. The SOCIETIES platform design aims to specify a modular, scalable architecture that is compatible with state-of-the-art pervasive and social computing systems but also adaptive to the variety of novel use case scenarios which are to be evaluated in this project.

This design deliverable is a revision of D4.1 ‘SOCIETIES Platform Specification and Design’ (1) which reported on background technical research that was conducted based on research objectives and perceived technical problems that this platform would be required to address. This report included summaries of working documents on communication and device abstraction, service infrastructure, dependability and security, CSS and CIS management, and social network (SNS) integration. D4.1 also described some design patterns for failure, discovery and interoperability which are used to guide the design and development of the integrated prototype. Finally, D4.1 outlined the modular design for the SOCIETIES platform including component and sequence diagrams. This deliverable, D4.3, provides a revision to the design of the SOCIETIES platform based on technical requirements, system architecture design and technology considerations arising from the deployment of a prototype implementation.
Chapter 2 describes the high level design of the SOCIETIES Platform, starting with a summary of the SOCIETIES system architecture as specified by WP3 (3), including functional and deployment viewpoints of the architecture. The SOCIETIES platform is described by its key elements in terms of interoperability and the WP4 functional services that contribute to the SOCIETIES platform and provide the backbone and core infrastructure for other parts of the platform to build on.

Chapter 3 outlines the revised design of each WP4 subsystem within the SOCIETIES platform, including high level component diagrams, subsystem module descriptions and interoperability with other platform subsystems.

A glossary of terms used in this document can be found on the SOCIETIES project website (4).
2 High Level Design of the SOCIETIES Platform

This chapter describes the high level design of the SOCIETIES Platform, starting with a summary of the SOCIETIES system architecture as specified by WP3, including functional and deployment viewpoints of the architecture. The SOCIETIES platform is described by its key elements and how it interoperates with other parts of the system such as 3rd party services. Finally, we discuss the WP4 functional services that contribute to the SOCIETIES platform and provide the backbone and core infrastructure for other parts of the platform to build on.

2.1 SOCIETIES System Architecture

The SOCIETIES System Architecture was defined by a process of scenario driven requirements in WP2, functional service definition and informational architecture in WP3, and technology mapping and specification in WP4. The system architecture comprises of 3 key aspects: the SOCIETIES platform, 3rd party services which build on the platform and external software and hardware elements (e.g. social network data and sensor/device integration). Figure 1 depicts a unified viewpoint of the system architecture including all 3 aspects in addition to technology and deployment characteristics.

The SOCIETIES platform is the central part of the system architecture and comprises of the functional services, as depicted in Figure 1, within each system node and the infrastructural elements that support these functional services such as communication and service infrastructure. The platform is designed for deployment to multiple nodes: cloud node for server side capabilities, and the rich/light node for end user client side capabilities. The functional services as defined by the WP3 functional viewpoint and described
in D3.3 ‘System Architecture’ (3) are shown in Figure 2. The services are conceptually grouped as 1) Domain/Infrastructure - Society, 2) CIS – Community, 3) CSS - Participant and 4) CSS – Node. The society services provide federated search and domain administration functions while supporting multiple CIS and CSS instances within each domain deployment. The community services provide support for CIS in terms of community management, community personalisation, community context, and learning and reasoning for communities. The participant services provide user data management and personalisation for each user’s CSS. The node services provide the infrastructural abstractions for communication, devices and services which support the operation of CSSs and CISs.

Figure 2: Functional Viewpoint of the SOCIETIES Platform

3rd party services which build on the SOCIETIES platform are an important part of the SOCIETIES system as they implement specific end user scenarios through applications that utilise the platform functionality in a useful way, thus showing the real value of the platform innovations. The 3rd party service architecture is defined in D3.1 ‘Service Model Architecture’ (5). The final part of the system architecture includes external software/hardware services and data sources that are integral parts of the system in order for the system to provide any intelligent functionality or to support any end user scenario. These services and data sources include existing social networks, external software services, sensor networks and embedded sensors, which provide rich contextual sources to the platform.

The deployment viewpoint of the SOCIETIES system architecture, as shown in Figure 3 taken from D3.4 ‘Interoperability Architecture Revision’ (6), depicts various system nodes for a cloud environment and a user devices level. The cloud nodes include a domain node which supports domain federated search and administration services and a CSS user cloud node which represents each user’s CSS instance in the cloud. The user device nodes include a rich node and light node, where these support user interaction and device integration on laptop and smartphone device ranges, respectively.
2.2 Interoperability with the SOCIETIES Platform

The SOCIETIES platform supports various points of interoperability with external elements such as 3rd party services, device integration and user interaction:

- The 3rd party service interoperability is provided through open and standardised APIs. 3rd party developers can use localised invocations of Java APIs within a standard enterprise application container technology supporting OSGi services and using the Spring (7) framework for dependency injection and inversion of control. For light client integration, Android platform services are exposed on the device for local invocation from 3rd party Android applications (see section 3.10.2). 3rd party services can also integrate with the platform via remote XMPP based communications where the 3rd party API is exposed with XMPP endpoints.

- Device integration is supported by a device management and context source management components, where embedded or integrated sensors or other devices can be detected, registered as context sources and used to capture a variety of user data for inference purposes.

- User interaction with the platform is provided by a web application interface integrated with the user’s cloud node (see section 3.11) and an Android client application (see section 3.10). Both interfaces provide features to discover relevant users, communities and services that are available on the platform. 3rd party services can provide additional user interfaces to support application specific scenarios.
2.3 WP4 SOCIETIES Platform

Figure 4 gives an overview of the "core services" provided by the SOCIETIES platform as specified by the WP3 System Architecture. The highlighted core services are those that are relevant to the WP4 SOCIETIES platform. The remaining core services are specified by the WP5 user and community experience designs of the SOCIETIES Platform. The services in the diagram are grouped according to the major concept they manipulate or operate on. For example, services that operate on a single CIS are grouped together, as are those that operate on a CSS, and those found on every node in a CSS.

Figure 4: WP4 Functional Services from System Architecture

The WP4 platform services provide infrastructural support for communication between nodes and services, for device and node management, for service discovery and management. On behalf of a user, services are provided for the management of each CSS, integration with social networks and deployment of services to a CSS. At the community level, the lifecycle of CISs and related data are supported by management services. Within each platform deployment federated search and administration services are provided by WP4 subsystems, where these and the other supporting subsystems of the WP4 platform functionality are described in Chapter 3.

The SOCIETIES platform comprises of functionality from both WP4 and WP5 to enable the support for user and community experience, and generally WP5 features build on WP4 functionality. Although in some cases WP5 features are consumed by WP4 subsystems, such as privacy protection, context brokering and user feedback. For example, any 3rd party API exposed from a WP4 subsystem must be privacy aware to ensure all user data is protected at all levels of the platform.
3 SOCIETIES Platform WP4 Subsystems

This chapter outlines the design of each WP4 subsystem within the SOCIETIES platform, including high level component diagrams, subsystem module descriptions and interoperability with other platform subsystems. Figure 5 shows the SOCIETIES platform WP4 subsystems, components and some dependencies, as part of Cloud node deployment of the platform.

![Cloud Component Diagram]

Figure 5: SOCIETIES Platform WP4 Cloud Subsystems

The design of the WP4 subsystems, as previously specified in D4.1, were revised based on technical requirements, system architecture design and technology considerations arising from the deployment of a prototype implementation. Those subsystems included are:

- Communication Framework
- CSS & CIS Management
- Activity Feed Service
- Security Components
- Service Management
- Device Management
- Domain Authority & Directory Services
- User Interfaces

Subsystems which were not revised nor considered for the prototype implementation, such as the service marketplace, are not covered in this chapter.
3.1 Communication Framework

The Communication Framework provides a network communication API to other platform components and third party services, allowing them to communicate with their counterparts in other network nodes. It has an embedded identity model that allows addressing other nodes directly using the SOCIETIES identity scheme. Besides its basic message patterns, namely request-response and asynchronous message, it also supports publish-subscribe interactions. The Framework fundamentally relies on XMPP (8) (9) and XML serialization to allow remote components to send and receive Java Beans (data objects) transparently. The Communication Framework consists of multiple subsystems: Cloud node, Rich client & Light client (Android). The cloud node subsystem includes the domain authority and user CSS cloud components. The rich client subsystem is similar by design and thus not defined further in this section. The other subsystem is the light client which is design for the Android platform and prototyped as a communications app.

SOCIETIES platform communications are based on XMPP protocols which involve logging into a relevant XMPP server and sending/receiving different types of XMPP stanzas. The XMPP server can relay the sent messages to and from other known users and can optionally send server-to-server messages and so allow a federated approach to XMPP messaging. Since the chosen XMPP server is Openfire (10) based, the Smack library is used to expose the client API of the Openfire platform to the Android client. In particular, an Android variant of this library, aSmack, is used. While it is possible to construct appropriate stanzas and send/receive them to relevant recipients another approach was taken. The Cloud Node APIs are communicated with via known classes which are serialised via Simple into XMPP stanza payloads. A small set of stanza messages are formed within the Android Communication, for example, SendIQ and SendMessage and the serialised payloads added to form the complete XMPP message. A similar but reverse re-serialisation process also takes place for XMPP messages received by the Android device.

The SOCIETIES Platform Android Client application utilises the SOCIETIES Communications app in order to use one or both forms of supported communications:

- Invocation of remote methods on other CSS nodes
- Send and/or receive Pubsub node events

All communications between the SOCIETIES Client and the other nodes of its CSS (user) and other CISs (communities) are channelled through the SOCIETIES Android Communications app. Since the app exposes two main APIs to the SOCIETIES Client, there is no impediment to other 3rd party apps using the communications components provided that they use known XMPP namespaces. In order to hide any implementation details from 3rd party apps, the preferred route to providing communications between 3rd party apps and the SOCIETIES platform is to expose SOCIETIES Client services to these other apps and allow the relevant services to handle the implementation details. The hybrid option of using both forms of interaction is also a possibility.
3.1.1 Design of Communication Framework Cloud Subsystem

![Diagram of Communication Framework Cloud Subsystem]

**Figure 6: SOCIETIES Cloud Communication components**

3.1.1.1 Communication Framework Commons

The Communication Framework Commons aggregates the functionality that is common to the various implementations, namely Identity model utilities. It exists with the purpose of being re-used across different Communication Manager implementations.

3.1.1.2 XC Communication Manager

The XC Communication Manager implements the Communication Manager interface by relying on an XMPP External Component connection to the XMPP server. It provides the core functionality of the Communications Framework and acts as a messaging gateway to other platform nodes and components deployed to those nodes.

3.1.1.3 PubsubClient (Virgo)

The Virgo (11) version of the PubsubClient relies on the Communication Framework to implement the API for the use of publish-subscribe functionality by other components.

3.1.1.4 PubsubService

The PubsubService relies on the Communication Framework to provide, over the network, publish-subscribe broker functionality. It keeps track of subscriptions and published items, and is responsible for triggering notifications over the network.

3.1.1.5 Cloud Communication Utilities

A Cloud Node may host CISs, which are identity-addressable entities of SOCIETIES just like CSSs. In order to allow the creation and destruction of CISs, the dynamic instantiation of communication endpoints must be supported. Cloud Communications Utilities implements this communication management functionality by relying in the dynamic instantiation of Communication Manager implementations.
3.1.1.6 Cloud Communication Bundle

The Cloud Communication Bundle is a bundle component. It packs together the required components and configures them in order to provide the Communications Framework functionality for the Cloud Node. The required components are all the above: Communication Framework Commons, XC Communication Manager, PubsubClient, PubsubService and Cloud Communications Utilities.

3.1.1.7 Domain Authority Communication Bundle

The Domain Authority Communication Bundle is a bundle component. It packs together the required components and configures them in order to provide the Communications Framework functionality for the Domain Authority Node. The required components are Communication Framework Commons and XC Communication Manager.

3.1.2 Design of Communication Framework Light Client (Android) Subsystem

![Composite structure AndroidCommunicationsComponents](image)

Figure 7: SOCIETIES Communication components

The main components are the Communications and Pubsub Android services, the XMPP Connection Manager and the Helper classes used by the client components to expose a simpler API. It is possible for client components to use the remote services directly but the Helper classes hide the binding and connection details and provide a unified, tested and stable method of interaction. All communications are effectively channelled through the XMPP Communication Manager in the form of a pipe which works on a First In First Out (FIFO) principle. The services that form the basis of the communications component are fully
asynchronous to allow non-blocking communication to take place while allowing the SOCIETIES Client app to be as responsive as possible for user interactions.

3.1.2.1 XMPP Communication Manager

The central component is an Android service which exposes an API to allow an Android component to register their message payload namespace (to allow serialisation to take place), send messages and optionally receive replies. This mechanism is the basis for remote invocation.

---

**Figure 8: Finite State Machine diagram of XMPP Connection**

---

The XMPP Connection Finite State Machine describes the various states that can occur when creating and maintaining an XMPP connection with a Sociology CSS Domain XMPP server.

1. **Disconnected** is the initial state and the first event is generated by a user attempting to log in.
2. **Wait For Network** is where a check for the availability of valid network, i.e. broadband, WiFi etc. is made. The check either fails resulting in the No Network Found event or, if successful, the Valid Network state is reached.
3. **Valid Network** indicates that a valid network to attempt an XMPP connection has been found and the Attempt Connection event is fired.
4. **Waiting To Connect** is the next state and if an XMPP connection is successfully created and authenticated with the user’s credentials, the final state of Connected is reached. If the credentials are invalid or any other connection exception prevents the connection being made the Attempted Connection Failure event is fired and the Disconnected state is reached.
5. **Connected** is the final destination state. If a user wishes to log out the Disconnect event is fired and the Disconnected state is reached.
6. **Reconnecting** state can be reached if the XMPP connection is disconnected as caused by the Attempt Auto Reconnection event. When this happens, the XMPP re-connection mechanism attempts several re-attempts to re-establish the connection. In the event that the re-connection is successful, the Auto Reconnected event is fired and the Connected state is reached. If the connection cannot be re-established the Disconnect event is fired and the Disconnected state is reached.
3.1.2.2 Pubsub Manager

The Pubsub service is the second main component which registers the Pubsub namespace and exposes an API to allow other XMPP components in any other app on the device to subscribe to known Pubsub nodes and receive events on these nodes and, optionally, publish events to existing nodes and create new nodes. The Pubsub component utilises the basic communications component to send and receive the relevant Pubsub stanzas.

3.1.2.3 XMPP Connection Manager

This auxiliary component manages the XMPP connection as a Finite State Machine (FSM) and provides a reliable XMPP connection for the service. For example, in the event of temporary loss of the connection, the FSM will attempt to re-connect on a periodic basis. As can be seen from the FSM diagram (Figure 8) a real device such as an Android device cannot assume a stable connection but must actively manage the connection and handle any connection events in a structured manner so that the device user is aware of any connection problems but is not required to manage it manually.

3.1.2.4 Communication Helpers

The Helper classes are not components but APKLIBs (APK libraries) that can be incorporated into a target application and allow the application components to use the Helper classes to bind with and interact with the SOCIETIES Communication services.

3.1.3 Interoperability with other SOCIETIES Subsystems

The communications framework provides a dynamic factory method for the on demand creation of communication managers for other platform components. This is used to provide a unique endpoint for communication for CISs without routing the messaging through the CIS owner communication manager thus masking the identity of the CSS.

Each SOCIETIES subsystem has their own group communication manager which registers with the Communication Manager to receive messages and information queries based on their registered namespaces. These are then forwarded to the relevant service for remote invocations of their service methods. Group managers provide an abstraction to platform services thus removing tight coupling with the networking technologies and messaging protocols.

All components in the SOCIETIES Client application can use the exposed SOCIETIES Communications services when the need for inter-CSS node communication is required.
3.2 CSS Management

The primary functionality that the CSS Management subsystem provides in the SOCIETIES platform is the ability to create and maintain a user CSS. It achieves this by allowing the user to create a CSS by interacting with the XMPP server to create an account. Once this is in place, it provides the ability to log into the system, modify the CSS Profile information and store this data by pushing the profile information into the context management subsystem, from WP5. Another main aspect to the CSS Manager provides the functionality to add and remove additional CSS Nodes to a user's CSS, such as Android light client nodes or other rich nodes such as laptops, PCs etc. The CSS Management subsystem also provides the user with the ability to discover relevant Friends (i.e. other CSS Users) and send friend requests. The suggestion of relevant friends is based on personalising the list of other users on the system. This personalisation is inferred based on existing social network relationships, similar context attributes (e.g. users have the same location) and community membership.

3.2.1 Design of CSS Management Subsystem

![CSS Management Subsystem Diagram](image)

Once the CSS has been created and logged into the domain server the CSS Manager pushes the contents of the CSS Record to be persisted within the context database. This record contains the private details relating to the CSS.

Both CSS A and CSS B contact the CSS Directory on the DA node through the Comms framework components to advertise their existence with a CSS Advertisement. This advertisement contains the public details relating to the CSS, such as the CSS Name and the CSS Identity. Each user CSS can get a list of all other CSS’s currently visible on that DA node and therefore be available for friend invitations.

**Figure 9: High level overview diagram of the CSS Manager component**
3.2.1.1 CSS Manager

Once the CSS has been created and logged into the domain server the CSS Manager pushes the contents of the CSS Record to be persisted within the context database. This record contains the private details relating to the CSS. Both CSS A and CSS B contact the CSS Directory, which resides on the Domain Authority (DA) node, and through the Communication Framework components to advertise their existence with a CSS Advertisement. This advertisement contains the public details relating to the CSS, such as the CSS Name and the CSS Identity. Each user CSS can get a list of all other CSSs currently visible on that DA node and therefore be available for friend invitations.

3.2.1.2 CSS Directory

The CSS Directory component is responsible for managing all of the CSS Advertisements within the system. Each CSS once created will advertise its existence by means of creating a CSS advertisement. This advertisement is sent to the CSS Directory. The advertisement contains details of the CSS such as the CSS name and identity. This information is publically available and can be accessed by any CSS in order for them to discover other CSSs and potentially send friend requests to them. The directory is accessed by CSSs by the use of a CSSDirectoryRemoteClient component which allows the CSS to make simple local method calls to retrieve the directory records. This component then interacts with the CSS Communication Manager to contact the CSS Directory on the DA node and gather the requested information.

3.2.1.3 CSS Communication Manager

The CSS Communication Manager (CommsManager) bundle is responsible for handling all the communications to and from the CSS Manager component. It acts as the central focal point for interactions between the CSS Manager and the communications framework. The commsManager contains 3 classes, the commsClient and the commsServer classes, it also provides a callback class. The CSS Manager interacts directly with the commsClient which creates the message bean to send, then gets an instance of the commManager and uses this to send the outgoing message. The commsServer in turn, uses the commsFramework component to receive data from remote components or CSS nodes again through the use of message beans. The commsServer receives the incoming messages or queries from the commsframework and extracts the relevant details from the bean and uses this to call the appropriate methods on the CSS Manager component to complete the transaction.
3.2.2 Interoperability with other SOCIETIES Subsystems

The CSS Manager component integrates with a number of other SOCIETIES components from both WP4 and WP5. Below is a list of the components where integration takes place and a brief description of the functionality that is provided by this integration.

![Interoperability of the CSS Management Subsystem](image)

**Figure 10: Interoperability of the CSS Management Subsystem**

### 3.2.2.1 Context Broker

Integration with the Context Broker (a WP5 component) allows the CSS Manager to persist data within the context database. This data includes the CSS Profile (private CSS record containing user details such as name, identity, workplace, position, email, etc.). Also persisted are the Friends list and the CSS Node information.

### 3.2.2.2 Social Network Connector

Integration with the Social Network (SNS) Connector component provides the ability for the CSS to connect to their chosen social network and pull information from that network for use within the SOCIETIES platform. In particular the CSS Manager retrieves the list of social network friends from these sites and uses this as a basis for providing the user with suggested friends that they might want to become “CSS Friends”.

### 3.2.2.3 Activity Feed Service

Integration with the activity feed service allows the CSS to post activities that have been carried out by the system to inform the user that the CSS has carried out certain actions. These activities include recording
when the CSS has been published, when friend requests have been received or acted upon such as accepted or rejected or when the CSS profile has been modified in some way.

3.2.2.4 Communication Framework

Integration with the communication framework component provides the ability for the CSS to communicate with other components and with other CSSs across the network. It is based on the XMPP protocol and it interacts with the framework through its dedicated communication bundle called the CSS commsMgr.

3.2.2.5 SOCIETIES WebApp

Integration with the SOCIETIES WebApp component provides the ability for the user of the CSS to make use of the features within the CSS Manager through the user interface. Here the CSS Record can be modified, connection to the SNS connectors can be established, and CSS Advertisements can be modified to change the CSS advertisement name or Identity. Here also the list of suggested friends can be requested and friend requests can be sent or accepted, rejected etc. It also provides the ability to view the list of current friends and remove any of them if required.

3.2.2.6 3rd Party Services

Integration with 3rd party services comes in the form of providing an external API (ICSSManager) to allow the 3rd Party service to make use of some of the methods on the CSS Manager. The methods exposed for integration by 3rd Party services are:

- `getSuggestedFriends()` - this allows 3rd Party services to get a list of suggested friends for this particular CSS and filter based on whether or not they are members of particular social networks.
- `getSuggestedFriendsDetails()` - again this provides the 3rd Party service with a list of suggested friends with the filtering option but this time it returns a list of the CSS Advertisements for the relevant CSSs instead of just the identities.
- `sendCSSFriendRequest()` - this allows the 3rd Party service to send a friend request to selected CSSs from within their own service.
3.3 CIS Management

A Community Interaction Space (CIS) represents and provides the interaction mechanisms for an AmI community to exist. Members of an AmI community interact with a CIS via their own personal CSS. The individual members of an AmI community do not need to be human beings but can also be organisations, smart space infrastructures or autonomous agents. AmI Communities can be dynamic in nature, with CISs being formed in an ad-hoc fashion. CISs can also spawn sub-communities or merge with other communities, and will be self-orchestrating, making decisions for their members based on the community content and preferences. The CIS management subsystem offers capabilities for the management of CISs which includes:

- Creation, hosting and management of CISs;
- API for interacting with CISs hosted on other containers

3.3.1 Design of CIS Management Subsystem

Each CSS has an instance of the CIS Manager which is responsible for the creation, hosting and management of communities that are owned by the hosting CSS user. All communication by the user’s CSS devices to manage their community is routed through the Communication Manager of the CSS. However, for other community members of the CIS, a unique end point is provided to handle this CIS specific communications e.g. membership queries, join requests, activity posting. Every time a new CIS is created, the hosting CIS Manager requests a new communication manager for that CIS, using the Comms Factory Manager. This separates the communication channels for the CIS Manager (accessed via the CSS identity through the CISManager schema) and each newly created community (accessed via a unique CIS identity through the CIS Community schema).

During the creation of a CIS, an Activity Feed is also created for the CIS (by calling the Activity Feed subsystem). That activity feed resides on the Activity Feed Service component, though it is accessed XMPP-wise through the CIS that owns the feed and through the activityfeed schema. The CIS Manager also creates objects for each CIS which the CSS subscribes to. Those CISs are represented as CISSubscribedImpl objects. However, it is just the CIS Manager and CIS objects which are responsible for persisting CIS data for the user. The first component stores data regarding the relationship between the CSS and its CISs, while the second one stores data regarding the CIS itself.

Joining a community is a negotiated activity. Upon receiving the Join request, the CIS Manager will respond with the list of membership criteria for that CIS. The joining CSS will then query its local context engine for these parameters and forward these values as part of the request. Should these qualifications match the criteria, then membership is accepted for this community and a successful join response is sent to the client.

3.3.1.1 CIS Manager Interfaces

The CIS Manager bundle can also be accessed inside the Virgo Container through the ICisManager interface. That interface provides an API to the CIS Manager and for retrieving interfaces to both remotely hosted and local CISs. In total, the CIS Manager functionality can be accessed by 3 different java Interfaces: ICisManager, ICisOwned and Icis

ICisManager: exposes methods to manage a CIS present in the container (such as "create CIS", "delete CIS", "listing CIS") and methods used to change the user subscription to a CIS hosted in another CIS Manager (such as "join CIS" and "leave CIS"). That interface is implemented by the CisManager object.

ICisOwned: exposes methods on an owned (and consequently locally hosted) CIS object. It corresponds to the interface to retrieve information from a CIS such as listing Members, getting CIS data, getting a handler to the CIS activity feed (IActivityFeed interface) and managing members from the CIS. That interface is implemented by the Cis class and it can be retrieved through methods within ICisManager such as getOwnedCis.
ICis: corresponds to a sub-set of the ICisOwned interface which offers the same methods of ICisOwned apart from the ones to manage the membership (adding and removing member). This interface is aimed for interfacing CISs in which the CSS is not a member, and therefore, cannot manage their members. That interface is implemented by the CisSubscribedImp object (though it can be also casted into a CIS) and it can be retrieved through methods within ICisManager such as getCis.

A picture illustrating such interfaces can be seen below:

![Figure 11: CIS Subsystem](image_url)

As presented in Figure 11, both CISs and CisManager implements the IFeatureServer interface which handles incoming XMPP messages from both other Virgo components (such as CIS managers on other nodes) and android clients. Figure 12 exemplifies the creation of a CIS, where the trigger for the CIS creation could come as 1A) a local API call to the ICisManager and coming from another component in the same container (such as the WebApp), or 1B) a create CIS XMPP message sent by an Android Client (following the cis.manager schema). In both cases, the CIS Manager will react by creating a new CIS Object which will be accessible through an ICisOwned interface or through the XMPP id assigned to the new communication manager belonging to the new CIS.
While Figure 12 shows interaction in a single container, Figure 13 shows one CIS Manager interacting with a CIS that was created in another container. The picture illustrates a "join" which is again triggered to the CIS Manager either through ICisManager API or through a XMPP message (not represented in the image). The CISManager then uses the CSS Communication Manager to send the join request over the internet to the CIS hosted in another container. If the join is successful, an ICis interface will be returned so it can be used directly (though still through CSS Communication Manager) to get the list of members of that CIS.

Figure 12: Creating a new CIS

Figure 13: Multi CIS Subsystems
3.3.2 Interoperability with other SOCIETIES Subsystems

A summary of the integration between the CIS Manager and other SOCIETIES components can be seen in the list below, which describes all the interfaces used by the CIS Manager component and how the CIS Manager uses them.

- **Communication Factory Manager**: Used to create new communication managers for the new CISs. This allows direct communication with the CIS without needing to go through the hosting user’s CSS.

- **ICommManager**: Interface to the communication manager belonging to the CSS which owns the CIS Manager.

- **ICisDirectoryRemote**: Used by CIS Manager to register and unregister CISs in the CIS Directory.

- **IEventMgr**: Send internal events to the container regarding CIS subscription.

- **IPrivacyPolicyManager**: Called to register/update the privacy policy of CISs.

- **IPrivacyDataManager**: Called when one tries to access data from the CIS in order to check if it has rights to access it.

- **ICtxBroker**: Used to retrieve the CSS qualifications which are needed in the join message. These are compared against the membership criteria for the community. If the qualifications pass then membership is accepted for the CIS.

- **INegotiation**: Called for negotiating the joining of a CIS.

- **IUserFeedback**: Called to return feedback to the user regarding the joining of a CIS.

- **IActivityFeedManager**: Invoked to retrieve/create/delete a CIS feed.

3.3.2.1 Eventing and Activity Publishing

The CIS Manager component also publishes 2 types of events related to CISs. The CIS Manager object publishes internal events on behalf of its CSS and regarding the CSS activities related to CISs. On other hand, events happening inside a CIS are registered as activities in the CIS activity feed.
3.4 Activity Feed

The activity feed service provides activity feeds to the SOCIETIES platform and 3rd party services. An activity feed provides a way to emit, store and fetch activities. The main usage for activity feeds within the platform is for CISs to store activities for a community and for CSSs to store user focused activities. CIS activities can include any community activities such as members joining, services being shared and user posts by community members. CSS activities can include actions taken on behalf of the user such as learning of preferences and context updates. These activities provide the user with some level of inspectability by generating a log of system activities on behalf of the user. The activity feeds can also be used by third party developers for storing any kind of application specific activities.

3.4.1 Design of Activity Feed Subsystem

![Diagram of Activity Feed Subsystem]

3.4.1.1 Activity Feed Manager

The Activity Feed Manager service provides two main methods for feed manipulation and access through its interface IActivityFeedManager. GetOrCreateFeed will fetch an existing activity feed or create a new activity feed based on a provided feed ID. DeleteFeed will delete a feed with the provided feed ID.

3.4.1.2 Activity Feed

After acquiring an activity feed through the activity feed manager the activity feed interface IActivityFeed provides standard methods for manipulation of the feed itself: create, read, update and delete (CRUD). On top of the CRUD methods the interface provides more fine grained operators, to filter the activities you apply the CRUD methods on. These latter methods are needed as the number of activities can grow quite large.
3.5 Security

The security subsystem is not strictly separated from the other parts of the platform. Most of the components designed in the scope of security subsystem utilize not only some security functionality, but also seamlessly integrate other non-security functionality. The WP4 components in scope of the security subsystem are Security Services, Policy Negotiator and Domain Authority REST Server. Other security components within the SOCIETIES platform include privacy protection and privacy access control, which are provided by WP5.

3.5.1 Security Services

The “Security Services” component is the only dedicated security component. It is a low-level utility, used to digitally sign data and verify signatures. The data can be raw, or XML. In the latter case, the signature is embedded in the same XML document, together with its corresponding X.509 (12) certificate. The key or certificate to use can be specified explicitly. However, the Security Services component is designed with multiple SOCIETIES identities in mind and the certificates are mapped to SOCIETIES identities. In order to be able to sign data or verify a signature without explicit usage of digital keys or digital certificates, only the selected identity is needed, assuming a 1:1 mapping between SOCIETIES identities and digital certificates.

![Figure 15: Secure Policy Negotiator](image)

3.5.2 Domain Authority REST Server

One of the main parts of the Domain Authority REST server is the web application that serves the installation of files for 3rd party applications, as well as any other files like images, videos and other resources that can be downloaded by the 3rd party service after it is installed and started. The webapp implements a RESTful server and various HTTP methods to upload and download files. There is no restriction on file type, length, or name, but any upload request must be accompanied by service ID and X.509 certificate (without the private key) of the service provider or the file uploader. The REST server internally stores all certificates and associates every service ID with the corresponding certificate and list of files. The files are saved to a local file system and not directly accessible by the world. The files can be accessed through HTTP GET requests. The required parameters passed with the HTTP GET request are filename, service ID, and digital signature of the file name. When a request is received, the server checks its local registry for the given service ID and corresponding digital certificate. File download is authorized only if the given signature is successfully verified against the public key from the certificate, i.e., if the client presents the service provider’s signature of the resource to be downloaded.
The security approach is prone to abuse by distributing the provider’s signature to others once the signature is obtained. However, once the client obtains the files, it is impossible to prevent the client from unauthorized distribution of files anyway. On the other hand, the approach allows convenient update of any files without the need to redistribute credentials to file downloaders. And the client can download files from multiple locations multiple times, again without additional distribution of credentials, i.e., without additional involvement of the service provider. This enables also small pervasive devices with intermittent access to the Internet to be service providers.

3.5.3 Policy Negotiation for Service Sharing and CIS Joining

The Secure Policy Negotiator negotiates a policy between two parties in a secure manner. The policy is either Service License Agreement (SLA) in case of negotiation for service consumption, or CIS agreement in case of negotiation for joining a CIS. The details of the core negotiation process are described in Deliverable D4.1 [pp. 15-16, 47-48] (1)

When Policy Negotiator is used to negotiate policy for service sharing, there are two phases to complete:

1. When a user chooses to host or provide a service to others, Policy Negotiator on the provider side downloads any files associated with the service. The files do not need to be on the Internet; they can be on a local file system or restricted LAN, etc. Policy Negotiator automatically uploads these files to the Domain Authority REST server, configured in the container (it may or may not be on the Domain Authority), together with the service ID and user’s X.509 digital certificate. Then the service is ready to be shared.

2. At any later time, any number of other users can request to use the service. A request triggers policy negotiation on the requester side (1). After the negotiation is completed successfully, separate privacy policy negotiation is invoked and upon success, the Secure Policy Negotiator on the provider side returns URLs with all required credentials (digital signatures) to Secure Policy Negotiator on the requester side. The requester node can then install the service if applicable (if the service does have a client), and the service can later use the URLs to download any additional resources.

When Policy Negotiator is used to negotiate policy for joining a CIS, the process does not involve any file uploading and is shortened to core negotiation between requester CSS and provider CIS, as described in D4.1 (1). Figure 16 shows the integration of components used to support policy negotiation for service sharing, while Figure 17 shows a sequence diagram for the installation of a shared service.

3.5.4 Interoperability with other SOCIETIES Subsystems

Secure Policy Negotiator and Domain Authority REST (13) server, integrated with Privacy Negotiation Management and Service Management, provide the basis for a distributed Service Marketplace. The Domain Authority REST server is a lightweight server that strongly depends only on Security Services and is designed to be installed on any node, even outside the SOCIETIES platform. However, it is distributed with the Domain Authority node as this node is a convenient default location for service providers who do not require using their own REST server instance. CIS Manager is integrated with Secure Policy Negotiator to initiate negotiation of any policy for CIS membership. Secure Policy Negotiator is integrated with Personalization Manager to personalise and score policy options according to the user during negotiation.
Figure 16: Policy Negotiation for Service Sharing

Figure 17: Service Shared Installation
3.6 Social Network Integration

The Social Network Integration provides a proxy between the SOCIETIES Platform and the user’s social network communities. Using this subsystem, the platform can connect to one or more social networks and fetch most of the user’s profile information, plus his/her behaviour on the social network. It also provides an API to push data through one or more social channels. The component hides the details of each social network API and it provides a common model to describe and store the social data in order to make it available to the other modules of the platform.

3.6.1 Design of Social Network Integration

![Figure 18: Social Network Integration Subsystem]

3.6.2 Social Connector Modules

Social Connector is the component responsible to interface with the social network sites. Due to the fact that each social network has its own API (design, protocol and implementation) a specific component for each social network is required. The generic interface ISocialConnector provides a unified definition to call each social network, providing a common way to group together the information. SOCIETIES handles profiles, friend’s contact list, the list of followed groups, and the social activities stream. To connect a specific social network with the platform it is necessary to implement a specific Social Connector that implements the required API to fetch and push data through the social network. The component should provide as output the raw response of the social network API (as JSON String) of each of the previously discussed information types (Profile, Friends, Groups, and Activities). It might also provide an API to post data to the social network. The minimum input parameter required to initialize a Social Connector is a social network token, a string parameter required by each social network API.
3.6.2.1 Oauth Module

It is required to implement the authentication process to get a social network token. Generally it is a web flow where the user provides their own credentials, but only the first time. Once authenticated, the user grants access to the data required by the (SOCIETIES) connector. At the end of the process a temporary/complete token is forwarded to the public callback URL defined by this module in the social network. The token then is stored in the platform until the user manually disconnects the social network.

3.6.2.2 Social Data

Social Data is the core of the Social Network Integration. It contains the list of active connectors, and provides a set of APIs for the platform to utilise the social network data. The connection is persisted, that means that every time the client is restarted all the connections are restored. Social Data provides:

- Private set of APIs
  - getSocialProfiles() that provides the list of SN Profiles data
  - getSocialFriends() the list of each SN contacts
  - getSocialGroups() the list of belonging/followed/liked groups
  - getSocialActivities() the list of performed activities in the SN
- Public set of APIs to publish content on the SN
  - sendPost(Message) to post a textual message on the SN
  - sendPost(Checkin) to make a Checkin on the SN (where applicable)
  - sendPost(Event) to create an Event on the SN (where applicable)
- Management set of APIs to utilise the connectors.

The most valuable work done by Social Data is the data abstraction. Every time the data is updated, each connector connected is called, and the raw data provided by them is passed to a converter which is called to translate the specific social network language into a common one, extensively used in the platform and moreover on the net. The data model chosen has been defined by Open Social, a consortium that tries to gather together all the different social network data models into a single one by social data specification (14). The converter translates the raw data fetched by the connector into an OpenSocial object, in particular in Person, Group and ActivityEntry. In this way all the data has the same format and meaning, also for the client that will use those data. Some but not all the data retrieved is stored into the CSS Context Broker, to improve the SOCIETIES Profile data, and moreover make those data available to other components of the platform. An external API has been created to post data by 3rd party services.

3.6.2.3 SNS Communication Manager

This module implements the logic to connect Social Data with the communication framework. The module is used to keep the data across all CSS nodes aligned. It is possible to use both the platform web application and the platform Android application to connect/disconnect social network with the platform. A schema that maps the Social Network connector has been defined and passed through the communication framework to propagate the information between the various nodes.

3.6.2.4 Social Data Light

This module is part of the SOCIETIES platform Android application and it is responsible to receive or send data from/to the Light Client. Whenever a connection/disconnection with a social network is done on the Light Client, an event is created by the Social Data Light and sent back to the CSS Social Data module by the Communication framework. In the same way it receives the list of available connections that are stored.
3.7 Device Management

The component diagram in Figure 19 depicts the set of sub-modules of the Device Management subsystem. The goal of the device management is to provide a set of technology independent capabilities for a CSS in the discovery and management of the CSS associated devices. The types of the device include internal sensors of a smart phone (e.g. a GPS sensor) and external devices such as multimedia devices (e.g. a scanner). A feature of the device management is that the device may be attached to several CSS nodes. Therefore a synchronization mechanism is required to get the device registry of a CSS updated. It uses the capabilities of the low layer modules to access to the actual device. The following features are supported by Device Management:

- manages the device registry
- provides a device API
- shares devices to the CIS
- provides management functions (e.g., add/remove, start discovery, configure devices)
- provides elements for GUI

3.8.1 Design of the Device Management subsystem

3.8.1.1 Device Manager

The Device Manager is the central component, providing the external API of the module. It manages each component device which represents any physical hardware connected to a CSS node through different ways (internal sensors accessible via OS API, external device connected to a LAN, etc).

3.8.1.2 Device Driver Simulator

This part represents the software to abstract the OS dependent driver. The goal of this module is to provide a generic API per device type following the SOCIETIES Device Description. This module is provided by the SOCIETIES platform, but is OS dependent due to the OS Device API connection and is implemented for each OS.

3.8.1.3 Device Registry

The Device Registry components aim at maintaining the list of devices selected by the user to be part of its own CSS. The storage will include all information on the device (its type, its location (node), its configuration, etc). This registry does not include devices physically present on the same network but not selected by the user. This module provides an API available for a GUI to manage the selection, addition, removal of devices within the list.
3.8.2 Interoperability with other SOCIETIES subsystems

The Device Management driver component acts as an intermediate layer between sensors and actuators attached to a CSS node and the Device Management (DM) component. A device contains one or more services, so the driver will register the OSGi service objects under IDriverService interface. Figure 20 shows the design of integrating devices (RFID and TV Screen) and 3rd party services.
3.8 Service Management

The Service Management Subsystem provides for service discovery and lifecycle management. Its components are designed to support service lifecycle control, such as discovery, deployment, activation, monitoring and execution. The service infrastructure consumes the communications framework providing a common service environment for multiple devices and platforms.

3.8.1 Design of Service Management

3.8.1.1 Service Management

Service Management supports 2 key features of service discovery and service control. Service discovery listens for the installation of new OSGi compliant services being installed in the container and checks the associated metadata of the service. If it matches a SOCIETIES 3rd party service (based on certain metadata tags), it will then register this service in the Service Registry. It also provides methods for querying of services that are available for both the local or remote CSSs provided the remote CSS’s identity is known. Based on this service listing, methods are available for controlling these 3rd party services. The Service Registry is then updated with the new service status. This component also provides support for service control where the lifecycle of the OSGi service can be controlled (started, stopped, etc) by the platform user interfaces (Android and Webapp) and programmatically via the XMPP based remote service control API.

3.8.1.2 Service Registry

This is a database residing on the cloud node that persists the list of 3rd party services that this CSS node and other nodes are hosting and potentially sharing within CISs they are members of. Service Sharing is provided by the CIS Manager and this component uses the service registry to discover and configure the services which are available for sharing within a CIS.
3.9 Domain Authority & Directory Services

The domain authority is a centralised node that hosts the XMPP communication server and the directory services. Only one domain authority is required per platform instance and is responsible for managing an identity domain. The directory services hold the list of CSSs registered in the system and the list of communities that are available to join. Figure 22 shows the components provided by the domain authority node.

3.9.1 Design of the Domain Authority

Figure 22: Domain Authority Design

3.9.1.1 XMPP Server

The XMPP server provides an XML-based open standard for real-time communication, instant messaging, presence and generalized routing of XML data (9). In addition to the core functionality, a number of XMPP Extension Protocols (XEPs) exist that continuously add new features. Authentication is required before sending any messages. SOCIETIES administrators can choose to federate with any other open XMPP server domain expanding the list of registered CSS users and CISs. This feature of the protocol allows fine-grained federation control.

3.9.1.2 CSS Directory

The CSS Directory component is a simple database of registered users within the system. It supports standard features of search, retrieve and update of a user’s public profile. The information stored is considered the publicly available information of a CSS and can be extended to hold as little or as much data that the user is comfortable sharing.

3.9.1.3 CIS Directory

The CIS Directory component is a registry of all communities that have been created within system. This centralised component allows users to search for communities of interest and request to join them. By design the actual CIS is hosted by the owning CSS, this directory simply offers the functionality to publicise its existence.
3.10 SOCIETIES Platform Light Client Application

The SOCIETIES Light Client application’s main purposes are to provide a user interface to the SOCIETIES platform and expose the SOCIETIES platform to 3rd party applications on the device.

3.10.1 User Interface

It is more than likely that a user’s main interface to SOCIETIES will be through their mobile device(s) reflecting the general trends in how users are interacting with social media sites and other user-oriented media. The SOCIETIES Light Client allows a user to create their identity, create and update their public profile, login and out of the platform, view their current CSS devices and activities; interact with friends and communities both on SOCIETIES and other social network sites, and receive notifications.

3.10.2 Third Party Applications

In order to promote an ecosystem of add-on functionality it is important that the SOCIETIES platform provides access to 3rd party applications and services to allow user and community data and events to be accessible in a controlled manner. The SOCIETIES Client exposes several of its Android service components and content providers to other apps on the same device. Although it is possible for 3rd party apps to communicate directly with the SOCIETIES platform via the SOCIETIES Communications app, the preferable route is to leverage the exposed SOCIETIES components keeping the communication implementation hidden from 3rd party apps.

3.10.3 PhoneGap/Cordova

The SOCIETIES Client is an HTML5+JS+CSS based app rather than the more conventional fully native app. The PhoneGap/Cordova framework (15) has been used to provide a Javascript API to common device sensors and resources and to utilise its Javascript/Java bridge abstraction which could allow the app to be ported to other supported Mobile OSs, if required. As can be seen from Figure 23, the UI of the app is completely composed of HTML+CSS and Javascript libraries such as JQueryMobile and custom code providing the dynamic behaviour of the UI. The interaction between the UI Javascript and the app’s native components can be either synchronous or asynchronous, although the latter is preferred.

Figure 23: Component Design of Light Client Application
3.10.4 Native Components

The native components present in the application consist of a small number of Android activities, several Android services and one Content Provider. A subset of the services is exposed for use by other 3rd party apps that wish to use the SOCIETIES platform. Due to the number of native components present in the application, it was not practical to allow a number of developers to develop these components in the same application. As a result, the native components were developed as APKLIBs and tested independently before inclusion into the SOCIETIES Client app.

3.10.5 Interoperability with other SOCIETIES Subsystems

Since the SOCIETIES Client app acts as a container for all relevant SOCIETIES components, there is no differentiation in how WP4 and WP5 components are treated although WP5 components do rely on the WP4 components for basic platform functionality.
3.11 SOCIETIES Platform Web Application

The SOCIETIES platform web application provides a user interface to manage the user’s CSS. Figure 24 shows the deployment of the platform web application which routes user sessions through a web server to the user’s cloud node hosted in an application container.

![Diagram of Web Application Deployment]

Figure 24: Web Application Deployment

3.11.1 Apache HTTP Server

The Apache HTTP server is configured to handle all SOCIETIES http requests and relay these requests to appropriate user containers. It receives http responses from user containers and relays them back to the user’s browser. This design removes the need to open multiple ports for external access on a server for each container. Each end user does not need to know the port of their container in order to access their platform webapp. They simply enter a standard URL with their identity. This design works with all webapps running in user containers (includes platform and 3rd party services).

3.11.2 Web Application Controllers

Each controller is created per http request. They contain no local data and all controllers have reference to a user session bean. Every controller must check that the current user is authenticated and logged in on each call (hence the reference to the session bean). If not set, then the user is not actually logged on and the request must be denied and/or the user redirected to a home page. Each webapp controller is deliberately designed to have little or no logic. It is simply used as a gateway to the relevant platform component.

One user session bean is instantiated per http session. It is created when the session starts and the user successfully logs in and destroyed when the session expires or when the user logs off. When the user logs in, a flag is set in the User Session Bean. It is this flag that each controller checks prior to accepting and processing the request.
4 Conclusion

The aim of this deliverable is to outline the revised specification and design of the SOCIETIES platform for the integrated prototype development within work package 4. This work package aims to research, design and prototype the SOCIETIES platform infrastructure and services that will be required to enable the SOCIETIES individual and community experiences while realising the concept of an Ambient Intelligent (AmI) Community, and therefore supporting the third party services and end user scenarios that will be developed within the other work packages. The design builds on technology research conducted within WP4 and design patterns outlined for failure, discovery and interoperability use cases. The design details of the SOCIETIES Platform include specifying subsystems and components, guided by the output provided by the WP3 system architecture and the WP2 technical requirements, and addressing the requirements from the other ‘consumer’ work packages, WP5 and WP6.

This deliverable outlines the revised design of each WP4 subsystem within the SOCIETIES platform. A high level design of the SOCIETIES Platform, including a summary of the SOCIETIES system architecture as specified by WP3, was discussed. This was followed by a detailed description of the WP4 functional services that contribute to the SOCIETIES platform and provide the backbone and core infrastructure for other parts of the platform to build on. These descriptions included high level component diagrams, subsystem module descriptions and interoperability with other platform subsystems. The described subsystems are the Communication Framework, CSS & CIS Management, Activity Feed Service, Security Components, Service Management, Device Management, Domain Authority & Directory Services, and User Interfaces. All subsystems are loosely coupled and modular by design and can be deployed independently of other modules once the platform infrastructure can be utilised to maintain all required interactions.

The next deliverable from WP4, D4.4 ‘Second Integrated Prototype of the SOCIETIES Platform’, will document the second prototype implementation of the designed subsystems and platform functionality available for other platform subsystems from WP5 and 3rd party services from WP6. The design and prototype implementation will be further evaluated during user trials based on scenarios for disaster management, university living and conference support.
## 5 List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmI</td>
<td>Ambient Intelligence</td>
</tr>
<tr>
<td>PSS</td>
<td>Personal Smart Space</td>
</tr>
<tr>
<td>CSS</td>
<td>Co-operating Smart Space or Cascading Style Sheets</td>
</tr>
<tr>
<td>CIS</td>
<td>Community Interaction Space</td>
</tr>
<tr>
<td>SNS</td>
<td>Social Network</td>
</tr>
<tr>
<td>XMPP</td>
<td>Extensible Messaging and Presence Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Mark-up Language</td>
</tr>
<tr>
<td>Pubsub</td>
<td>Publish/subscribe</td>
</tr>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>OSGI</td>
<td>Open Services Gateway Initiative</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOTA</td>
<td>State-of-The-Art</td>
</tr>
<tr>
<td>JIDs</td>
<td>Jabber Identifications</td>
</tr>
<tr>
<td>XC</td>
<td>External Component for XMPP</td>
</tr>
<tr>
<td>3P</td>
<td>Third Party</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>DM</td>
<td>Device Management</td>
</tr>
</tbody>
</table>
6 References

5. —. D3.1 Service Model Architecture. 2012.