Abstract

This document describes the work performed to implement and integrate the various components of the second release of SOCIETIES Work Package 5 - CSS Individual and Community Experience. It documents the delivered software components and their provided features. Details on their access and usages are also provided. The release is based on the five design deliverables, D5.1, D5.2, D5.3, D5.4 and D5.5 which documented the detailed designs for each of the five key subsystems that support the SOCIETIES CSS Individual and Community Experience and is an update to D5.6 which accompanied delivery of the First Integrated Prototype of WP5.
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] WP5 CSS Individual and Community Experience
[Editor: Name, company] Nick Taylor, HWU
[Work-package leader: Name, company] Nick Taylor, HWU
[Estimation of PM spent on the Deliverable] 12 PMs

Copyright notice
© 2013 Participants in project SOCIETIES
## List of authors

<table>
<thead>
<tr>
<th>Company</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWU</td>
<td>Nick Taylor, Eliza Papadopoulou, Patrick Skillen</td>
</tr>
<tr>
<td>TSSG</td>
<td>John McGovern</td>
</tr>
<tr>
<td>ICCS</td>
<td>Ioanna Roussaki, Nikos Kalatzis, Nicolas Liampotis, Pavlos Kosmides</td>
</tr>
<tr>
<td>TRIALOG</td>
<td>Olivier Maridat, Bruno Jean-Bart</td>
</tr>
<tr>
<td>ITSUD</td>
<td>Zhiyong Zu, Daqing Zhang</td>
</tr>
<tr>
<td>SETCCE</td>
<td>Mitja Vardjan</td>
</tr>
<tr>
<td>AMITEC</td>
<td>Yiorgos Bouloudis</td>
</tr>
<tr>
<td>NEC</td>
<td>Piotr Szczytowski</td>
</tr>
<tr>
<td>PTIN</td>
<td>Christopher Viana Lima</td>
</tr>
<tr>
<td>IBM</td>
<td>Guy Feigenblat</td>
</tr>
<tr>
<td>SINTEF</td>
<td>Bjørn Magnus Mathisen</td>
</tr>
</tbody>
</table>

## List of Code Contributors

<table>
<thead>
<tr>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWU</td>
</tr>
<tr>
<td>TSSG</td>
</tr>
<tr>
<td>SINTEF</td>
</tr>
<tr>
<td>ICCS</td>
</tr>
<tr>
<td>AMITEC</td>
</tr>
<tr>
<td>DLR</td>
</tr>
<tr>
<td>IBM</td>
</tr>
<tr>
<td>TRIALOG</td>
</tr>
<tr>
<td>ITSUD</td>
</tr>
<tr>
<td>PTIN</td>
</tr>
<tr>
<td>SETCCE</td>
</tr>
<tr>
<td>NEC</td>
</tr>
</tbody>
</table>
# Table of Contents

List of authors........................................................................................................3
List of Code Contributors.......................................................................................3

Table of Contents ................................................................................................4
1  Introduction.........................................................................................................6
2  Prototype Features.............................................................................................7
2.1  Intelligent Community Orchestration .............................................................7
   2.1.1  Community Recommendation Manager .................................................7
   2.1.2  Model Analyser .....................................................................................9
   2.1.3  Data Collectors ....................................................................................10
   2.1.4  Suggested Community Analyser ..........................................................10
2.2  Context ..........................................................................................................11
   2.2.1  User and Community Context DB Mgmt ...............................................14
   2.2.2  Context Broker ....................................................................................14
   2.2.3  Context Source Management ..............................................................15
   2.2.4  Location Adapter .................................................................................15
   2.2.5  User and Community Context History Mgmt ........................................15
   2.2.6  Context Event Management ................................................................15
   2.2.7  CIS and CSS monitoring ......................................................................15
   2.2.8  User and Community Context Inference Mgmt .................................16
   2.2.9  User Context Prediction ......................................................................16
   2.2.10 Community Context Prediction .........................................................16
   2.2.11 User and Community Context Refinement ..........................................16
   2.2.12 User and Community Context Inheritance .........................................16
   2.2.13 User Context Similarity Evaluation ....................................................16
   2.2.14 Community Context Estimation .........................................................17
   2.2.15 Context GUI ......................................................................................17
   2.2.16 Geofencing Proxy ...............................................................................18
2.3  Personalisation ..............................................................................................19
   2.3.1  Personalisation Manager ....................................................................19
   2.3.2  User Preference Management .............................................................19
   2.3.3  User Preference Learning ....................................................................20
   2.3.4  User Preference GUI ..........................................................................20
   2.3.5  Community Preference Management ...............................................21
   2.3.6  DIANNE ..............................................................................................21
   2.3.7  Context Aware User Intent (CAUI) Task Manager ..............................21
   2.3.8  Context Aware User Intent (CAUI) Discovery ......................................22
   2.3.9  Context Aware Community Intent (CACI) Discovery ...........................22
   2.3.10 Context Aware User Intent GUI ...........................................................22
   2.3.11 Context Aware User Intent (CAUI) Prediction ......................................22
   2.3.12 CRIST User Intent Discovery ...............................................................23
   2.3.13 CRIST User Intent Task Manager ........................................................23
   2.3.14 CRIST User Intent Prediction ..............................................................23
   2.3.15 CRIST Community Intent Discovery ...............................................23
   2.3.16 Social Profiler .....................................................................................23
2.4  Privacy & Trust .............................................................................................24
   2.4.1  Privacy Data Manager .........................................................................26
   2.4.2  Data Obfuscation Manager ..................................................................26
   2.4.3  Privacy Policy Manager .......................................................................26
   2.4.4  Identity Selection Manager ..................................................................27
   2.4.5  Privacy Preference Manager, Privacy Preference Learning, Privacy Preference Condition Monitor 27
   2.4.6  Privacy Negotiation Manager ...............................................................28
   2.4.7  Privacy Assessment Manager ................................................................28
   2.4.8  Privacy Assessment GUI .....................................................................28
2.4.9 Direct, Indirect and User Perceived Trust Engine .......................................................... 31
2.4.10 Trust Repository ........................................................................................................ 31
2.4.11 Trust Evidence Repository and Collector ................................................................. 31
2.4.12 Trust Broker and Trust Event Manager .................................................................. 31
2.5 User Agent .................................................................................................................... 32
  2.5.1 Decision Maker ......................................................................................................... 33
  2.5.2 Conflict Resolution .................................................................................................. 33
  2.5.3 User Feedback .......................................................................................................... 33
  2.5.4 User Action Monitor ............................................................................................... 36

Appendix: Installation Guide ................................................................................................. 37
  Prerequisites, Downloads & Versions ............................................................................... 37
  XMPP Server (Openfire) Configuration ............................................................................ 37
  Build System (Maven) Configuration ............................................................................... 38
  Android Configuration .................................................................................................... 38
  Building the code ........................................................................................................... 38
  Running the code ............................................................................................................ 38
  Cloud Node ..................................................................................................................... 38
  Android Node .................................................................................................................. 38
1 Introduction

The SOCIETIES project supports the creation of Ambient Intelligence 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. Work package 5 covers five distinct areas in which we are researching, designing and prototyping supporting technologies for the SOCIETIES CSS Individual and Community Experience. They are: Intelligent Community Orchestration, Context, Personalisation, Privacy & Trust and User Agent.

This document describes the functionalities and features of the second prototype deliverable of the SOCIETIES CSS Individual and Community Experience. The release is divided into module groups, one for each WP5 area. Intelligent Community Orchestration provides support for identifying new, relevant communities and recommending them to the user. Context provides extensive support for user context management, including context inference and the management of external context sources (e.g. sensors). Support is also provided for community context. Personalisation provides support at an individual user level. This includes the automatic learning of user preferences and user intentions (for service adaptation) from monitored user behaviour and context. Privacy & Trust provides support for individual privacy protection (including data obfuscation, privacy policy negotiation and privacy preferences) as well as trust (including direct and user perceived trust). Finally, User Agent provides support for decision making between the various intelligent mechanisms within the platform and acts as the application point for service adaptations. It also provides support for behaviour monitoring and user feedback on platform behaviours.

Work package 5 functionality is supported by the SOCIETIES Platform enabling technologies (WP4) and in turn supports the third party services (WP6) and end user trials (WP8) that are being delivered by other work packages.

This document accompanies the software deliverable, D5.7, which comprises the open source software. The software is publicly available on our source repository, hosted by Github.com\(^1\), and may be downloaded or browsed. The repository includes Java source files, XML configuration files and other artefacts that are managed by our integrated build system supported by Maven\(^2\). The build and installation instruction details for the software components targeted at the Spring-OSGi and Android deployments are included as an Appendix.

Note that this deliverable is a software deliverable and this document is merely a description of that software. The first prototype was submitted as D5.6 and this deliverable is the second prototype which is thus an update to D5.6. This document attempts to provide a stand-alone description of the second prototype but reference to the document accompanying D5.6 might be required in places.

\(^1\) https://github.com/societies/SOCIETIES-Platform
\(^2\) http://maven.apache.org/
2 Prototype Features

2.1 Intelligent Community Orchestration

2.1.1 Community Recommendation Manager

The Community Recommendation Manager (CRM) is responsible to provide a list of communities’ suggestions for the CSS users or a 3rd party service. The CRM component uses the CIS Management to retrieve the remote communities which the CSS user still not participates, presenting the results according to a set of selected filters.
The CRM enables the CSS users or a 3rd party service, through an API, to search for communities by defining a set of primary or secondary filter. The primary filter allows querying accurate results, thus only results that exactly match the filter will be presented. The secondary filter allows queries which will present precise or close results.

The communities have by default a number of membership criteria associated which expose the purpose of their creation. The membership criteria present their information using context attributes available in the Context Management component. The attributes can be numeric or string values that are used a posteriori for comparisons in the CRM.

Each filter uses a set of operators provided by the CRM which are used for comparisons. The filters compare the context attributes in the membership criteria with the data provided by the user or service. The filter operators possible are: equal, not equal, greater, greater or equal, less, less or equal and in. For instance, a filter can be defined as: location, equal, Tramore, while a second filter can be defined as: age, greater or equal, 25. The results would present communities located in Tramore with people above or equal to 25 years.

In addition, the CRM results are shown ranked from most to least relevant, based on the filters defined for search. In addition, there is a limit by default to five results. The limit can be changed at any time by requestor.

The figure below presents a more detailed the internal structure class of the CRM component.
2.1.2 Model Analyser

The premise behind the Model Analyser is to use context updates to provide a real time interaction between an end-user’s pervasive environment and their interaction within the platform. The Model Analyser is a collective of dependant components the function of which is converting raw data into context state models.
2.1.2.1 Context State Models (CSM) Analyser

CSM Analyser provides modelling of the context data into roles that will be used to clean the data in preparation for the mining algorithm. Event updates received from an end-user trigger the process, events can be of a number of formats such as:

- Adding Context attributes, for instance an end-user adds ‘Experience’ to their professional profile.
- Updating Context attributes, some attributes such as location will be regularly updated.
- Manually selecting new communities, an end-user wishes to join a specific community from a GUI of selectable communities.

As event updates are received, the impact is analysed and then checked if there is any consequential ripple effects caused by the event. For instance, an updated context event would have the immediate effect of potentially removing an end-user’s acceptable criteria for joining a community. The ripple effect would check if the end-user’s new update allows them to join additional communities, or does this event trigger a tipping point for a new community suggestion.

- Available on Virgo. No need on Android.

2.1.2.2 Group Identifier

The Group identifier provides the mechanism to identify how changes to the CSM can aid in the identification of theoretical groups that may form the basis for a community and the corresponding community management action.

- Available on Virgo. No need on Android.

2.1.3 Data Collectors

There are two further components in ICO namely the ‘CIS Data Collector’ and the ‘CSS Data Collector’. These are simply there to aid in good software design principles; they are task 5.1 common listeners and publishers of activity data within the SOCIETIES platform. For instance, Collaboration Pattern Analyser and Group Identifier components subscribe to CIS Data Collector rather than both components developing the similar subscribers.

- Available on Virgo. No need on Android.

2.1.4 Suggested Community Analyser

The Suggested Community Analyser (SCA) is responsible for constantly analysing information about other CSSs a CSS comes in contact with either by joining a CIS or using services shared by other CSSs. Based on the information it receives from multiple sources such as activity feeds, profile and context information as well as other components in the ICO system, it can suggest that the user join or leave a CIS, create a new CIS or create a sub-community of a CIS. Moreover, when the Suggested Community Analyser suggests that a user creates a community, it will be able to classify that CIS as temporary or on-going which will affect the lifecycle of that CIS.
2.2  Context

The Context Management system (CMS) acts as an intermediate layer between platform/3P context-aware services and the sources of context information. The CMS functionality is realised through a multitude of components available for two main distributions: (i) the Virgo CSS container (Rich Client) and (ii) the Societies Android Application (Light Client). The CMS Rich Client consists of two bundles, namely, Context Management and Location Management. The overall deployment of the CMS architecture is illustrated in the following figure.

The diagrams that follow illustrate the components comprising each distribution along with the required/implemented interfaces. It should be noted that the components of the Context Management bundle have been grouped according to the major concept they manipulate or operate on. More specifically, components that operate on user level (single CSS) are grouped together (see first diagram below), as are those that operate on community level (see second diagram below). Components that operate on either user or community level are depicted in both diagrams. The third diagram below illustrates the overall CSM light client architecture deployed on Android devices. Finally, the fourth diagram below illustrates the high level architecture of the Location Management bundle.
In the following sections, brief descriptions of the main CMS components are provided.

### 2.2.1 User and Community Context DB Mgmt

User Context DB Mgmt component is responsible for managing and maintaining context data on each node of the CSS. Additionally, it supports the instantiation and the management of data classes for modelling context information and context metadata such as Quality of Context. The Context Model includes all the classes that model the context information to be retrieved, exchanged, maintained and managed in general in the CSS. This component utilises Hibernate, which is an object-relational mapping (ORM) library, in order to map the Context Model objects into records in a traditional relational database. Finally, the User Context DB Mgmt provides information to the Context Event Management, about any changes that are made in the

The functionality of Community Context DB Mgmt component is similar to the User Context DB Management but it is deployed only on CIS level. Hence it is responsible for instantiating the context model and managing community context data referring to groups of users.

### 2.2.2 Context Broker

The Context Broker manages the interaction between the components that gather contextual data and the components or services that request the retrieval of context information from the Context DB. The Context Broker enables the exchange of context data among different CSSs and CISs through the WP4
Communications Framework. In addition, it acts as a gateway to the Context History DB. Context Broker is also providing the ability to register or unregister for context changes notifications from the Context Event Management. Finally, it triggers the intelligent functionality provided by the inference management. A light version of the Context Broker is also available on Android (light user client). It supports a subset of features that are related to the interaction with the Context DB.

2.2.3 Context Source Management

The Context Source Manager component serves as intermediate layer between context sources, and the context database. These sources include external sensors, as well as, context-providing services and devices. The underlying model assumes a certain level of intelligence of context sources or respective driver modules. They have to be able to find the access point to the context source manager and provide information about which information they provide. Concluding, context provisioning requires a context source registration process to allow only supported parties to supply such information and to be able to distinguish different providers for the same context attribute type.

2.2.4 Location Adapter

The Location Adapter component, which is the core of the Location Management component of the CSM, exploits IBM’s Presence Zone Server (PZS) functionality in order to determine the location of the user. The component acts as a wrapper for the PZS and handles the registration process of the CSS nodes to the PZS. Thus, it consists of two sub-components: the wrapper/adapter and the configurator. It acts as a secondary context source and therefore, it exploits the Context Source Manager to forward its location updates to the context database.

2.2.5 User and Community Context History Mgmt

The User and Community Context History Management component provides the necessary mechanisms for persistent storage of historical context data (History of Context - HoC) for individuals and communities. Historic context data are maintained on a database that resides on the cloud node of each CSS or CIS. The component also provides data management functionality in order to store, retrieve and delete historic context data. HoC data are modelled by the HistoricAttribute class which is a reduced version of the ContextAttribute, containing only the necessary data to be used for learning processes. ContextAttributes contain a flag in order to indicate if the data should be stored in the HoC DB or not. The same context history management methods are used for both user and community context. Depending on the context identifier that the context history consumer is utilizing the request is routed to the appropriate user or community history database.

2.2.6 Context Event Management

The context management component provides methods for registering CtxChangeEventListeners in order to listen for context change events. There are two ways to subscribe for context change event notification: based on a context identifier and based on a context type. The subscriber's implementation of the CtxChangeEventListener interface allows reacting to the following event notifications: CREATED, MODIFIED, REMOVED, UPDATED. The mechanism is common for both user and community context data objects.

2.2.7 CIS and CSS monitoring

This class provides the necessary mechanisms in order to update user and community context based on CSS and CIS related events. Received events are translated according to context model and persisted in context database. Examples of such events are CSS owner profile changes, new CSS connections, CIS membership changes, etc.
2.2.8 User and Community Context Inference Mgmt

User and Community Context Inference Mgmt component coordinates and controls a set of individual methods for inferring and improving the quality of existing context data. This component provides the intelligence in order for the right inference process to be selected. User and community context inference methods include the prediction of context data, the refinement of existing context data, as well as, the inheritance of context data from the members of the community that the user belongs. In addition, community context estimation techniques are provided for communities of users while mechanisms for context similarity evaluation are provided for individual user context.

2.2.9 User Context Prediction

The User Context Prediction component provides functionality that allows the estimation of current context values but also supports the prediction of future context. More specifically, this component is responsible for performing both long term and short term context predictions for individual user context. The prediction algorithm is based on neural networks statistical methods.

2.2.10 Community Context Prediction

The Community Context Prediction component provides functionality that allows the prediction of future community context, in cooperation with the User Context Prediction component. Community Context Prediction can be achieved directly, using History of Community Context and special algorithms for special cases of community context, or indirectly, feeding User Context Prediction results into Community Context Estimation methods. This component is to be demonstrated in the second trial.

2.2.11 User and Community Context Refinement

The User and Community Context Refinement component’s functionality is transparent for the context consumer and supports both on demand and continuous inference. It builds on a plug-in architecture supporting the dynamic inclusion of refinement rules and refinement Bayesian algorithms. Refinement rules are represented in a formal way, specifying its output and input context attributes and the inference algorithm which is capable of evaluating it, together with algorithm specific information. Based on the general information, the reasoning manager handles the refinement rules and forwards them for evaluation to the dedicated algorithms.

2.2.12 User and Community Context Inheritance

The Context Inheritance component provides functionality for context inheritance in hierarchical communities that will enable passing-on of context information from parent communities to child communities when specific conditions apply. More precisely, the child community will examine the list of parent communities it has originated from, and try to find usable existing pieces of context to fill in the gaps, either directly or after appropriate estimation. Furthermore, the collection of user context data from the community members will be supported and enhanced. For example, a user belonging to a community might inherit some context info from this community and not need to fill up all the info manually.

2.2.13 User Context Similarity Evaluation

An application may need to group users according to similarity in one or more aspects. These aspects are examined by the values of a user’s context attributes. The User Context Similarity Evaluation component provides the functionality to quantify the amount of similarity shared in each of these attributes. Each evaluation consists of two or more users and one or more attributes as required by the referencing application. The context broker is used to obtain the needed context attribute data for each user.

The evaluateSimilarity method of User Context Similarity Evaluation takes the following input criteria:

- List of users;
- List of attributes to be used in the evaluation.
This evaluation result returns three levels of similarity data.

- **Overall Estimation – Boolean true/false**
  - False indicates there was no similarity bond between this group of users under the context attributes assigned, this will allow a requesting App to waste no further processing on this and move on.
  - True indicates we have found something and data from levels 2 & 3 result will give further insight into what it is.

- **Attribute Summary – Each of the attributes that were examined will provide an individual metric of its bond to similarity. Again, a requesting App can tell from examining this level which attributes led to the positive similarity evaluation**

- **Attribute Breakdown – a more detailed breakdown of the similarity based on the taxonomy of each attribute. For example in a requested list of attributes e.g. “Books & Food” we can tell that a group of users like the genre Horror in their reading tastes and all like pizza food.**
  - Available on Virgo. No need on Android.

### 2.2.14 Community Context Estimation

The Community Context Estimation component allows the extraction of new context data that apply to an entire community of users. The component offers various mechanisms that support the estimation of common context values based on common features, mean value, and majority of preferences. These mechanisms range from simple majority rules, up to complex multidimensional calculations, with methods for the calculation of alphanumerical and geometrical types of community context. The mean value, an example of simple calculations on a set of values, and the convex hull, an example of a more complex type of community context calculation, were demonstrated in the first trial. In the second trial this component will be used as the basis for more complicated actions on community context, such as inheritance and prediction.

### 2.2.15 Context GUI

The context GUI component provides a simple and easy to understand way to present context information related with a user, a community of users or other entities such as devices, services, etc. It provides a user interface that allows the management and navigation of available context through the different attributes and associations. This component demonstrated its functionality in the first trial based on a simplified GUI (depicted in the following figure). A more complete version is used for the second user trial.
2.2.16 Geofencing Proxy

The Geofencing Proxy provides access to the Geofencing service hosted and maintained by NEC. The Geofencing service models geographic localities as polygons (geofences). Each of such defined geofences corresponds to Societies Platform CIS. Geofencing Proxy registers at Context Broker for CSS location updates and uses Geofencing Server to evaluate the CSS membership in CIS corresponding to underlying geofence. Geofencing Proxy triggers notification to subscribers about events associated with entering or exiting geofences. The Geofencing Proxy can also be used to create new and query existing geofences.

In the following diagram, the dependencies of the Geofencing Proxy with the Geofencing Server and the Context Broker are illustrated.
2.3 Personalisation

2.3.1 Personalisation Manager

The PersonalisationManager component orchestrates the prediction functionality of the following components: DIANNE, UserPreferenceManagement, CRISTUserIntentPrediction and CAUIPrediction. The Personalisation Manager registers for two types of events, context update events and user action events. It registers for action events upon initialisation but for context events, it provides a method that allows any of the above components to notify that they are interested in context events that signal changes in specific context attributes. Then it registers with the Context Broker for the events that notify changes to the specific context attribute. Upon receiving an event (user action or context), the Personalisation Manager requests the preference outcomes of the DIANNE and the User Preference Management components and the user intent predictions from the CAUI Prediction and CRIST Prediction components. The outcomes of DIANNE and User Preference Management are compared and resolved to reach non conflicting result and the same is performed for the predictions retrieved from the CRIST Prediction and CAUI Prediction components. The resolved outcomes are sent to the User Agent - Decision Maker component for implementation.

2.3.2 User Preference Management

The User Preference Management component is an umbrella component for all the different functionality that is performed on the user preference models that includes managing, monitoring, evaluating and merging preferences.

The management part of this component handles the storage and retrieval of the user preferences to and from the context database which it uses as a storage medium. It maintains a registry of the user preferences that can be used to retrieve a user preference object directly from the database in order to avoid performing time consuming searches in the database. It maintains two caches for preferences and context values to reduce the number of calls to the context database which are updated appropriately by listening for context update events.
The merging part of the User Preference Management component listens for user action events and triggers a learning cycle when a counter has been reached. As soon as a new preference is learnt, the merging algorithm runs to merge the existing preference model with the newly learnt model. The result of the merging is stored in the context database.

The monitoring part of the User Preference Management is responsible for registering for context events with the Personalisation Manager for context attributes that affect the preferences of currently active services. When changes in the context of the user are pushed from the Personalisation Manager or changes to the preference models are pushed by the merging algorithm, a re-evaluation of the preferences based on the new input is performed and its outcome is returned to the Personalisation Manager.

Finally, the User Preference Management component is monitoring the user’s activities such as joining a CIS or sharing a service with a CIS. In the case of joining a CIS, the User Preference Management requests community preference information from the Community Preference Management component. It then uses the User Feedback component to ask the user which community preferences they want to download and merge with their own user preferences. Based on the response of the user, the component downloads the selected preferences and merges them with the existing user preferences. In the case of sharing a service with a CIS, the User Preference Management component will ask the user which of the preferences of the service they shared with the CIS they want to anonymously upload to the CIS. The selected preferences are then uploaded. In order to keep the user’s preferences and the community preferences updated, the User Preference Management component schedules uploading updated user preferences and downloading updated community preferences once a day.

2.3.3 User Preference Learning

The User Preference Learning component implements Quinlan’s C4.5 decision tree learning algorithm. It operates on a request basis. When a request for a preference learning cycle is requested (by User Preference Management) an asynchronous thread is executed to retrieve behaviour history from context, process the history and extract preferences. The result is returned to requestor via asynchronous mechanisms.

2.3.4 User Preference GUI

The User Preference Graphical User Interface is provided by the SOCIETIES web-app component and allows the user to add a new preference and to view, edit or delete existing preferences. Context conditions can be added manually and they can be nested to form complex conditional expressions visualising them as tree structures. The leaves of the tree can only be preference outcomes (i.e. actions that indicate an action to be implemented) while the branches of the tree can only be context conditions.
2.3.5 Community Preference Management

The Community Preference Management component is responsible for managing the community preferences of the CISs that a user administrates. It provides a merging algorithm that merges existing preferences with preferences uploaded by the members of a CIS. The algorithm makes sure to remove any references to identities or identifiers that point to a specific CSS so as not to cause issues with privacy or have a preference of a CSS depend on context information of another CSS. Each CIS maintains its own set of community preferences. The component exposes an interface that allows CIS members to upload their user preferences or download community preferences on demand as well. The interface also provides a method to query what community preferences a CIS has without downloading the preference objects.

2.3.6 DIANNE

The Dynamic Incremental Associative Neural NEtwork (DIANNE) is a single layer, feed-forward neural network that learns associations between context and user behaviour in an incremental and temporal fashion. It is a real-time compliment to the offline batch preference learning algorithm (C4.5) and can respond rapidly to changes in user behaviour that indicate a change in user preferences. It is a continual process running in its own thread, constantly listening for context and behaviour updates and accommodating them into internal knowledge. Output is returned when requested by the Personalisation Manager.

2.3.7 Context Aware User Intent (CAUI) Task Manager

The CAUITaskManager component provides the necessary classes that allow the instantiation of user behaviour model. Additionally the component provides methods that allow the construction of the CAUI Model by creating and adding the necessary values to the appropriate UserIntentAction and UserIntentTask classes along with the respective transition probabilities among actions and tasks. Additionally this interface provides access to the CAUI model based on various criteria such as action or task ID, type, name.
2.3.8 Context Aware User Intent (CAUI) Discovery

The CAUIDiscovery component is responsible for performing learning procedures aiming to create a user intent model. Upon request this component will retrieve data stored in Context History Database and based on learning algorithms will extract common behaviour patterns of the user. The current implementation of the discovery algorithm calculates the number of transitions among subsequent actions and associates context data to each action. The process results in a user intent model reflecting the most frequent user actions sequences and escorting context data.

2.3.9 Context Aware Community Intent (CACI) Discovery

The CACIDiscovery component is responsible for performing learning procedures aiming to create a community intent model. When learning is triggered this component contacts all contained CSS nodes that are part of a CIS and requests respective individual CAUI models. These models are processed in order to extract common characteristics and to build a common intent model that is appropriate for user action prediction. Finally the extracted model is distributed to individual CSSs and is used in conjunction with CAUI model aiming to maximise user prediction accuracy.

2.3.10 Context Aware User Intent GUI

The Context Aware User Intent GUI allows the user to view and manage his/her personal intentions model which is learned based on his/her past actions. The GUI also allows the management of inherited community models which is used in combination with user’s personal prediction model in order provide secure predictions. The GUI also provides a log of performed user actions and respective predictions. A screenshot of the GUI is depicted in figure xx.

2.3.11 Context Aware User Intent (CAUI) Prediction

The CAUPIPrediction component will provide the necessary functionality for evaluating the CAUI behaviour model in order to predict the most probable User Action. The prediction algorithm considers the last
performed action, the current context that describes user’s situation and previous prediction. Additionally the component triggers the user behaviour model discovery process.

2.3.12 CRIST User Intent Discovery

The CRIST User Intent Discovery component is responsible for constructing the user intent model from an action history list. A record in the list is composed of an action and the accompanying situation which describes the environment in which the action occurred. The intent model stores user action patterns and corresponding frequency of these patterns exhibited in the behaviour of the user. The CRISTUserIntentManager requests the CRIST User Intent Discovery to generate a new intent model when the model is out of date or to initialise the first model when enough information about the user has been accumulated.

2.3.13 CRIST User Intent Task Manager

This component is responsible for providing all necessary data structures and methods to maintain a prediction enabled environment. The data structures include an action history list, an intent model, a current user action, a current user situation, and a list of registered contexts. The methods called by CRIST User Intent Prediction include: predicting user intent based on the current user context and current user action, and retrieving the current intent action of a certain service and parameter. The algorithm combines input such as the current user action and the current user situation and matches it to the current intent model. More specifically, the current context of the user is retrieved from the context Broker and situations are created to describe the status of the environment. The resulting output provides a list of possible actions. In order to implement the above functions, the intent model learning method of CRIST User Intent Discovery might need to be called to generate a new model if the current one is getting out of date. Depending on the new model, the CRIST User Intent Task Manager might also need to register with the Personalisation Manager for additional context events.

2.3.14 CRIST User Intent Prediction

The CRIST User Intent Prediction component exposes two methods that allow the Personalisation Manager to request a CRIST user intent prediction based on the user performing an action and on the context of the user changing. A third method is also provided that allows a component to request the current intent action that should be performed on a specific parameter of a service. The CRIST User Intent Task Manager is called to process the request and return the appropriate user intent prediction. The basic idea of the prediction algorithm revolves around finding out in the user’s action history, which action is performed most of the time under certain situation and prior user action.

2.3.15 CRIST Community Intent Discovery

The CRIST Community Intent Discovery component is responsible to create a community intent model, which is used in conjunction with CRIST user intent model aiming to improve user intent prediction accuracy. When a CSS creates or joins a CIS, in one hand, it uploads its individual intent model, so the CIS intent model can be updated by including this new individual model; in the other hand, the CSS can download the new community model, so its individual model is updated by referring to other members’.

2.3.16 Social Profiler

The Social Profiler is a bundle that elaborates user social activities and interactions in order to derive preferences, interests and profiles. Those profiles are defined by the main actions that a user performs on their account in the social networks they are members of. Currently, the component provides a set of pre-configured profiles (narcisist, photomaniac, egocentric, etc.), essentially generated by a generalization of the main "trackable" actions from Facebook, Twitter and Foursquare (such as likes, picture posting, commenting, location check-in, etc.). The component doesn't have a direct interaction with the social networks but it uses the SocialData bundle as a proxy (a societies platform bundle). That component provides all the required functionalities required to connect, read and update social information. One very
interesting thing of both the components is that they both are using opensocial social data specification ([http://opensocial-resources.googlecode.com/svn/spec/trunk/Social-Data.xml](http://opensocial-resources.googlecode.com/svn/spec/trunk/Social-Data.xml)) as a common data model which maps proprietary data structure into a common one. The SocialProfiler takes advantage of this specification by using Person and ActivityEntity Objects to map on a graph database (based on neo4j) actions performed by the user on any social network. The algorithm generates a rank value for the relationship of the graph nodes that describe each social action. Using these values, it generates a percentage that defines how much the user belongs to each pre-defined profile of the bundle, and which one is the dominant. Moreover it provides user’s social categories and interests that the user likes most, based on the pages/likes he/she has voted more.

### 2.4 Privacy & Trust

The Privacy & Trust Protection layer provides components in order to protect personal data. It also aims to help the user understanding his personal exposition to the world, and taking the right decision about his protection. The following diagram depicts the modules provided by the Privacy & Trust task, both on the rich/cloud node and the light node.

![Privacy & Trust Overall Deployment](image)

The process is divided into several steps starting from a learning phase where the user’s privacy preferences are collected and the privacy policy for third party services and CIS are defined. Five steps can be described:

- managing continuous learning of the user’s privacy preferences,
- creating and manage privacy policies that will be associated to a CIS or a third party service,
- negotiating an agreement between the privacy preferences of a user, and the privacy policy of a CIS or a third party service before to use them,
- managing access control over personal data each time another user, CIS or third party service try to access them,
- collecting trust and assessment evidence to provide relevant feedback to the user.

The architecture on the light and the rich clients are similar, but the light version performs a subset of functionality and requires the rich node. The following paragraphs describe each component based on the rich client architecture, and provide details about the light client one when necessary.
The “Privacy & Trust” architecture is divided into three main components: “Assessment Management”, “Privacy Protection” and “Trust Management”. Each component manages its own remote interface.

This architecture provides a good modularity level, and avoids cyclic dependencies.

In terms of implementation, the rich client platform is based on Virgo; these components are implemented as OSGi bundles for this platform. The light client is an Android application; these components are packaged as Android services for this client.
2.4.1 Privacy Data Manager

The Privacy Data Manager performs access control over personal data from SOCIETIES each time another user, a CIS or a third party tries to access them. When such a request is received, the “Privacy Data Manager” uses the “Privacy Preferences Manager” to evaluate the most relevant preference according to this request and generate an access control result. This result is then stored using Hibernate in order to retrieve it quickly when the same request is received later.

At the moment, SOCIETIES Context data, and some CIS data are protected by the “Privacy Data Manager”. All SOCIETIES data providers (like the “Activity feeds” or the “CSS Manager”) should integrate with privacy aspects. Enforcing this integration is part of the “Privacy by Design” principle. This integration progress is followed by our “Privacy Internal Audit” process.

- Available on Virgo (rich client) and Android (light client).
- Communication available.

2.4.2 Data Obfuscation Manager

The Data Obfuscation Manager performs obfuscation of some data in order to reduce their personal content. This is the second part of privacy access control, and it allows privacy protection to perform in very fine granularity. Each type of data requires a specific obfuscation algorithm even though some similarity and classifying can be determined. This component provides an architecture to handle a specific algorithm for each data type, but to trigger evenly the obfuscation or the explanation to the user in a GUI. It implements two obfuscation algorithms: GPS coordinates, and user name.

- Available on Virgo (rich client) and Android (light client).
  - No need of remote access for most obfuscation algorithms. Communication can be used to perform heavy obfuscation algorithms on the rich node.

2.4.3 Privacy Policy Manager

The Privacy Policy Manager is responsible for performing CRUD actions for privacy policies and privacy agreements in order to associate a privacy policy to a new CIS or a new third party service, and to store the agreement documents after a privacy negotiation. These data are stored using the Context broker.

The privacy policy of a third party service lists the personal data needed by the service to complete its actions. It is the same for a CIS, except that these actions are defined by the SOCIETIES platform. The privacy agreement is the result of a privacy negotiation and describes the manner in which personal data will be handled in a CIS or by a third party service.

- Available on Virgo and Android.
- Remote access available.
A privacy policy management GUI is provided on the webapp, to browse a specific privacy policy. Furthermore, as depicted below, both on the webapp and Android, another GUI allows the user to visualize a CIS or a third party service privacy policy.

2.4.4 Identity Selection Manager

The Identity Selection Manager selects the most relevant identity in terms of privacy. The SOCIETIES platform handles only one identity hence the current version of this component always returns the main identity of the user, support for multiple identities is not implemented. Because the privacy layer architecture and API can manage multiple identities, implementing support of multiple identities should be almost transparent for the privacy layer. However, the impact in other SOCIETIES modules would require significant effort to be corrected and potentially delay more important project goals.

2.4.5 Privacy Preference Manager, Privacy Preference Learning, Privacy Preference Condition Monitor

The Privacy Preference Manager manages all the aspects of the privacy preferences of all types (“Privacy Policy Negotiation and Access control preferences”, “Identity Selection preferences” and “Data Obfuscation preferences”). Its basic responsibility is storing and retrieving the preference objects into and from the context database in which they are permanently stored. It also provides preference evaluation functionality which is performed when a privacy preference outcome is requested. User decisions regarding the handling of privacy such as disclosing data and agreeing to disclose data under specific conditions are captured, using the “User Feedback” component, and used to create privacy preferences. New decisions are also merged with existing preferences. The context of the user and the user's trust towards other CSSs and services in the system are monitored and when changes occur, the privacy preferences are re-evaluated and their result is implemented appropriately.

- Available on Virgo. No need on Android.
- No need of communication.
2.4.6 Privacy Negotiation Manager

The Privacy Negotiation Manager conducts a privacy policy negotiation between the owner of the CSS and a third party service provider or the administrator of a CIS. The negotiation involves retrieving the privacy policy of the service or CIS and generating a corresponding privacy policy for the user on the fly based on his privacy preferences. This process involves the user consent, which is collected using the “User Feedback” component. After a successful negotiation, a negotiation agreement is agreed by both parties and stored for auditing purposes as well as for access control purposes. A Negotiation Agent and a Negotiation Client are the two parties to the negotiation. This component provides the functionality for both as a CSS can act both as a Negotiation Agent when it shares a service or administers a CIS and a Negotiation Client when it attempts to consume a service or join a CIS.

- Available on Virgo. No need on Android.
- Remote access available.

2.4.7 Privacy Assessment Manager

The Privacy Assessment Manager component manages the assessment part of the privacy through two components: “Privacy Assessment Engine”, and “Privacy Logger”. The “Privacy Logger” logs events where private data is accessed or possibly shared or transmitted. The “Privacy Assessment Engine” parses the privacy log and assesses when, how often and which individual components and identities are possibly sending and receiving user’s private data. Even when such disclosure of private data is compliant with privacy agreement, it may still not be in user’s best interest.

- Available on Virgo.
- No need of communication.

2.4.8 Privacy Assessment GUI

The results of Privacy Assessment are exposed internally by Privacy Assessment Manager and displayed by its graphical user interface (GUI). The GUI is web based and integrated into the new SOCIETIES webapp. It aggregates numerical metrics retrieved from Privacy Assessment Manager and creates visual bar-charts from the metrics values. The charts enable the end user to quickly get the following information:

- which software components are accessing his personal data,
- which software components are transmitting data,
- which identities are possibly receiving the data, and
- estimated correlation between data transmission and data access for each software component.

Some of the result types are shown in the following screenshots. The user can browse the results by clicking left and right arrows at the top left. If needed, the user can also limit the time interval of interest by setting start and end times.
Privacy Assessment

Remote and local identities data has been sent to

Show results from ___ to ___ or leave blank for no limit.

Update
While all these results help the user assess the actual privacy practices of the installed services, it is the correlation results that are most interesting, as well as most complex. For every SOCIETIES identity, OSGi bundle, and Java class that has ever transmitted any data, the correlation between the relevant data transmissions and data access is shown. From the above figure, the user can see that:

- Five bundles have transmitted some data.
- Three bundles (the ones with non-zero blue bars) could have in theory transmitted user’s personal data that has been gathered locally in the past from the context by any bundle.
- Two bundles (the ones with non-zero red bars) have accessed the user’s data themselves, and later transmitted some data.
The value of a blue or red bar represents a measure of estimated possibility and frequency of transmission of local user data by a particular software component (SOCIETIES identity, OSGi bundle, or Java class).

### 2.4.9 Direct, Indirect and User Perceived Trust Engine

The Direct Trust Engine is responsible for evaluating the trust evidence that result from direct interactions among the trustor (CSS owner) and the trusted entities (users, communities, or services), in order to estimate the trust level of the latter. More specifically, trust evidence includes information such as, trust ratings, social connections with other individuals, interactions with services, as well as, community membership data. This component is then responsible for processing such information so as to estimate the direct trust in the entities involved. However, trust assessment cannot be based solely on the experiences and evaluation of a user's own interactions, but also on those of other trustworthy individuals. The Indirect Trust Engine employs collaborative filtering algorithms in order to make automatic predictions with regards to the trust evaluations of a user based on the trust opinions originating from that particular user's trusted connections. The similarity of users with regards to their trust preferences is also taken into account during indirect trust evaluation, hence, the derived trust values are fully personalised. The User-perceived Trust Engine is then responsible for fusing the direct and indirect trust values of an entity in order to assess the aggregate value as perceived by the CSS owner. The direct trust value generally outweighs the indirect one in this fusion process. However, the weight of each factor also depends on the confidence level with which it has been estimated. For instance, when the direct trust evidence is not sufficient, the opinions from other CSSs have a greater effect in assessing the aggregate trust value.

- Available on Virgo.
- No need of communication.

### 2.4.10 Trust Repository

The Trust Repository component provides a trust query interface on top of the underlying DBMS that actually controls the storage, management and retrieval of trust data. More specifically, this component is responsible for translating trust queries (w.r.t. direct, indirect and user-perceived trust information) into standard SQL queries that can be executed in the platform's DBMS. In this context, it utilises Hibernate which is an object-relational mapping (ORM) library, in order to map Trust Model objects, i.e. TrustedCsss, TrustedCiss and TrustedServices, to records in a traditional relational database.

- Available on Virgo.
- No need of communication.

### 2.4.11 Trust Evidence Repository and Collector

The evaluation of direct and indirect trust in an entity is based on trust evidence. The “Trust Evidence Collector” is responsible for obtaining such information and storing it in the “Trust Evidence Repository”. This information can be of various forms and originate from various sources. More specifically, trust evidence includes locally collected data from direct interactions with services, CSSs and CISs (direct trust evidence), as well as, trust opinions from other CSSs (indirect trust evidence).

- Available on Virgo.
- No need of communication.

### 2.4.12 Trust Broker and Trust Event Manager

The Trust Broker acts as a gateway to the trust calculations maintained either locally or in other CSSs. In this respect, it provides a query interface through which trust information consumers can specify various criteria, such as, the ID or the type of a particular entity in order to retrieve the evaluated direct, indirect or user-perceived trust values. It should be noted that trust queries can be performed either synchronously or
asynchronously. In the latter case, the consumer can be notified upon trust value update events. The facilities required for subscribing and publishing trust-related events are provided by the Trust Event Manager. This component serves as an abstraction layer to the platform’s underlying eventing mechanisms, i.e. WP4 Event Management and PubsubClient components.

- Available on Virgo and Android.
- Remote access available.

A subset of the Trust Broker functionality is exposed through a GUI available for the SOCIETIES Webapp. More specifically, this GUI allows the CSS owner to access the trust values evaluated by the system on their behalf. The displayed results can be filtered by ID, while the user can specify a sort order based on the assessed value or time of evaluation. Assigning trust ratings to the users, communities or services evaluated by the system is also supported.

![GUI screen showing trust relationships and details](image)

### 2.5 User Agent

![Diagram of User Agent composite structure](image)
2.5.1 Decision Maker

The Decision maker provides decision making functionality and makes trade-off from received preference objects and intent objects. The entry of Decision Maker is the service interface “IDecisionMaker” and port “makeDecision”. The arguments for input are two lists; the list of preferences and the list of intents respectively sent by the Personalisation Manager component. The DecisionMaker and AbstractDecisionMaker provide concrete implementations of the IDecisionMaker interface with the DecisionMaker extending the AbstractDecisionMaker. The AbstractDecisionMaker controls the flow of conflict detection and resolution. When the conflict cannot be resolved, the user Feedback component is used to present the actions to the user and ask them to select one to implement or neither. To further improve the efficiency of conflict resolution and personalisation, a close-looped learning algorithm is processing the feedback to learn from the way users resolve the conflicts and incorporate this into the decision making process in future decisions. The DecisionMaker is responsible for looking up 3p services and sending the message to corresponding services. An Android implementation of the DecisionMaker is also provided.

The current status of designing and implementation is:
- Communication available.

2.5.2 Conflict Resolution

The Conflict Resolution component is an internal component of the User Agent block. It receives a pair of Intent and Preference objects from the DecisionMaker and attempts to resolve conflicts between these objects. Multiple trade-off strategies are adopted, including preference-priority, intent-priority, high-confidence-priority and tree-based integration of above strategies. The user is given the choice to select which resolution strategies they would like to apply. Moreover, a tree-based rule composition mechanism is provided to present the complicated rule for resolution, e.g., the composed rule for context-aware trade-off between preference-priority or intent-priority policies. However when the conflict cannot be resolved, conflict resolution may query the user feedback by sending request to User Feedback Manager.

The current status of designing and implementation is:
- Available on Virgo. No need on Android.
- Communication available.

2.5.3 User Feedback

The User Feedback component provides interaction and notification mechanisms for use by platform components. It provides two modes of operation: explicit and implicit. Explicit mode requires interaction and feedback from the user before platform behaviour can continue. Implicit mode presents the user with a notification for a timeout period. If the user does not reject the notification within the timeout period it is taken as positive feedback and platform behaviour resumes accordingly.

An extension of the explicit notification GUI is also provided for the purposes of asking the user informed consent during the Privacy Policy Negotiation process which takes place when a user joins a CIS or installs a service provided by another CSS. The details of the negotiation requests are shown to the user who can indicate a DENY or PERMIT, change the actions that can be performed individually on each piece of requested data and alter the conditions for the request. The screenshots below provide examples of such a notification request on the SOCIETIES webapp and android.
Since the previous release, User Feedback has been extended to persist requests in the user’s database. This has two implications: first, if the user’s platform is restarted, requests are recalled and aren’t lost. This also allows the user to view a history of their responses. Second, if a user is not logged in to their webapp or Android app, requests are no longer lost, because they are recalled from the database when the user first logs in.
**sd Component Requests User Feedback**

- Requestor
- UserFeedback
- Database
- XMPP PubSub
- Client (Android / Web)
- User

User Feedback - Request feedback() - Store UF request() - Send PubSub event() - PubSub Event() - See notification() - Decision() - Respond() - UF response() - PubSub response() - Update UF record() - Return response() - Return response()

**sd User Login on New Devi...**

- UserFeedback
- Database
- XMPP PubSub
- Client (Android / Web)
- User

User Login - Login() - Request outstanding events() - Recall records() - Records returned() - Return events() - Send to client() - Display to user()
2.5.4 User Action Monitor

The User Action Monitor (UAM) receives user behaviour updates from services when the user performs some personalisable action. The UAM stores the action in behaviour history with a snapshot of the user's current context. This history is retrieved and processed by preference learning and intent discovery components in due course. The UAM also keeps track of the user's current interaction device (in CSSs with more than one client device) so that platform notification and interactions occur on the most appropriate device. An Android implementation of the UAM is available.
Appendix: Installation Guide

This deliverable does not include an end user software distribution. The following instructions describe how to setup your environment, download and build the source code, and then execute the generated binaries.

Prerequisites, Downloads & Versions

- **Java**: Sun/Oracle Java - JDK 6 Update 29 (not Java 7)
- **Build System**: Maven Version 3.0.3
- **XMPP server**: Openfire 3.7.1
  - http://www.igniterealtime.org/downloads/index.jsp#openfire
- **Android**: Android SDK - SDK r12 - v2.2
- **Application Server**: Virgo Tomcat Server 3.0.2
  - http://www.eclipse.org/virgo/download/
- **(Optional) Developer IDE**: Eclipse 3.7 Indigo comes with the required eGit
  - http://www.eclipse.org/downloads

XMPP Server (Openfire) Configuration

After installing Openfire, start the Openfire Server and select "Launch Admin" to launch the admin website. The first time you do this you will need to make some config choices. You can select defaults for most screens, but please make the following edits:

- "Server Setup": Domain name defaults to your machine name. Change to "societies.local"
- "Database Config": Choose "Embedded Database"

When complete, login to the admin website and do following tasks:

- Under "Server" -> "Server Settings" -> "External Components". In the "Service Enabled" box, click the "Enabled" checkbox and enter a "Default shared secret", e.g. "password" and click "Save Settings"
- In the "Allowed to Connect" box, create a new subdomain (which represents a new External Component)
  - Subdomain: xcmanager
  - password: password.societies.local
- Click "Add Component" to finish adding the component. You do not browse to any .JAR file or currently running component. This simply represents the connection info of a component that will connect later, i.e., the xcmanager component.
- Edit your local hosts file to ensure that "societies.local" maps to 127.0.0.1. Confirm this by pinging societies.local from command prompt.
Build System (Maven) Configuration

Some custom profile settings are required to be added to your /.M2/settings.xml file. These are set as a separate profile so should not conflict with any existing settings you have. These can be found in the UNZIP_LOCATION/example/settings.xml file.

Android Configuration

The current parent Android POM file specifies an Android Virtual Device (AVD) called "Test22Device". You will need to create an Android emulator on your machine with this name and it should be configured as an Android version 2.2 device.

Building the code

The release 1.1 of the SOCIETIES platform can be downloaded from the project website –

http://www.ict-societies.eu/open-source/downloads/

The code should be built with the following commands –

Cloud/Rich Client:

mvn clean install -f spring-osgi-projects.xml

Android Client:

mvn clean install -f android-projects.xml

Running the code

The 1.1 release is configured with three different containers, hence the instructions below provide information on how to start three CSS cloud nodes and android nodes.

Cloud Node

- Spring container (multiple nodes configuration):
  - Starting the first CSS container: In the /bin folder of the Virgo Installation folder, you should execute the command startup.bat for windows or startup.sh for linux.
  - Starting the second CSS container: In the /bin folder of the Virgo Installation folder, you should execute the command startup-user1.bat for windows or start_user1.sh for linux.
  - Starting the third CSS container: In the /bin folder of the Virgo Installation folder, you should execute the command startup-user2.bat for windows or start_user2.sh for linux.

This will now start the container and load all the components of the SOCIETIES platform and the CSS Individual and Community Experience.

Android Node

- Start the Android emulator with the command emulator -avd Test22Device -partition-size 128
• Navigate to the required directory, e.g. UNZIP_LOCATION/user-agent/Android-user-agent and execute: \texttt{mvn android:deploy}

On your emulator, go to the \textit{Applications} and run the application (e.g. User Agent) to browse the User Agent test GUIs.