SOCIETIES
FP7-ICT
Contract no.: 257493

SOCIETIES

Deliverable D4.2

First Integrated Prototype of the SOCIETIES Platform

Editor:	Alec Leckey (INTEL)
Deliverable nature:	Software Prototype Deliverable
Dissemination level: (Confidentiality)	PU
Contractual delivery date:	31 March 2012
Actual delivery date:	31 March 2012
Suggested readers:	Pervasive and social computing designers, researchers and developers
Version:	1.0
Total number of pages:	
Keywords:	SOCIETIES, Self Orchestrating CommunIty ambiEnT IntelligEnce Spaces, Communications, devices, services, dependability, social networks

Abstract

This document describes the work performed to implement and integrate the various components of the first release of the work package 4 SOCIETIES platform. It documents the features of the software components, how to use it, and how to access it. The release is based on design deliverable D4.1 “CSS platform specification and Design” which documented the technical specification and detailed design for the integrated SOCIETIES Platform.

Disclaimer

Impressum

[Full project title] Self Orchestrating Community Ambient Intelligence Spaces

[Short project title] SOCIETIES

[Number and title of work-package] WP4 SOCIETIES(CSS) Platform

[Document title] D4.2 First Integrated Prototype of the SOCIETIES Platform

[Editor: Name, company] Alec Leckey, INTEL

[Work-package leader: Name, company] Alec Leckey, INTEL

[Estimation of PM spent on the Deliverable] 12 PMs

List of authors

Company	Author
INTEL	Alec Leckey, David McKitterick
SETCCE	Mitja Vardjan
SINTEF	Babak Farshchian
NEC	Miquel Martin
TSSG	Alan Walsh, Lake Marshall

List of Code Contributors

Company
IBM
INTEL
IT SUD
NEC
PORTUGAL TELECOM
SETCCE
SINTEF
SOLUTA.NET
TELECOM ITALIA
TRIALOG
TSSG

Table of Contents

1 Introduction

The SOCIETIES project supports the creation of Ambient Intelligent 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 4 aims to research, design and prototype the SOCIETIES platform infrastructure and the services required to enable the SOCIETIES individual and community experiences (WP5) while realising the concept of an Ambient Intelligent or Pervasive Community. This supports the third party services (WP6) and end user scenarios (WP8) that will be developed within the other work packages.

This document describes the work performed to implement and integrate the various components of the first release of the SOCIETIES platform. It documents the various features of the software components, how to use it, and how to access it. The release implements the 1st version of platform design from the design deliverable D4.1 “CSS platform specification and Design” which documented the technical specification and detailed design for the integrated platform. It is the first in a series of releases for the platform providing underlying functionality to WP5 and WP6.

The release is divided into a section of integrated module groups, each providing its own set of features; Communication Framework provides an open standard wired protocol to allow communication from any client type. Device Management implemented an abstraction layer for all devices/sensors connected to nodes of a CSS. Service Lifecycle & Management provides an infrastructure to support the platform and third party services. Dependability provisioning at the level of service infrastructure and multi-device and multi-network deployment is provided by the Security layer. CSS/CIS Management implements the required functionalities for the creation, management, monitoring and sharing of a CSS and finally, the Social Network connectors implement the required integration for merging the platform with existing SNS systems in order to exchange data and enable interactions between systems.

This document accompanies the software deliverable, D4.2, which comprises of 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 artifacts 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 described in section 3.

[1] https://github.com/societies/SOCIETIES-Platform
[2] http://maven.apache.org/

2 Prototype Features

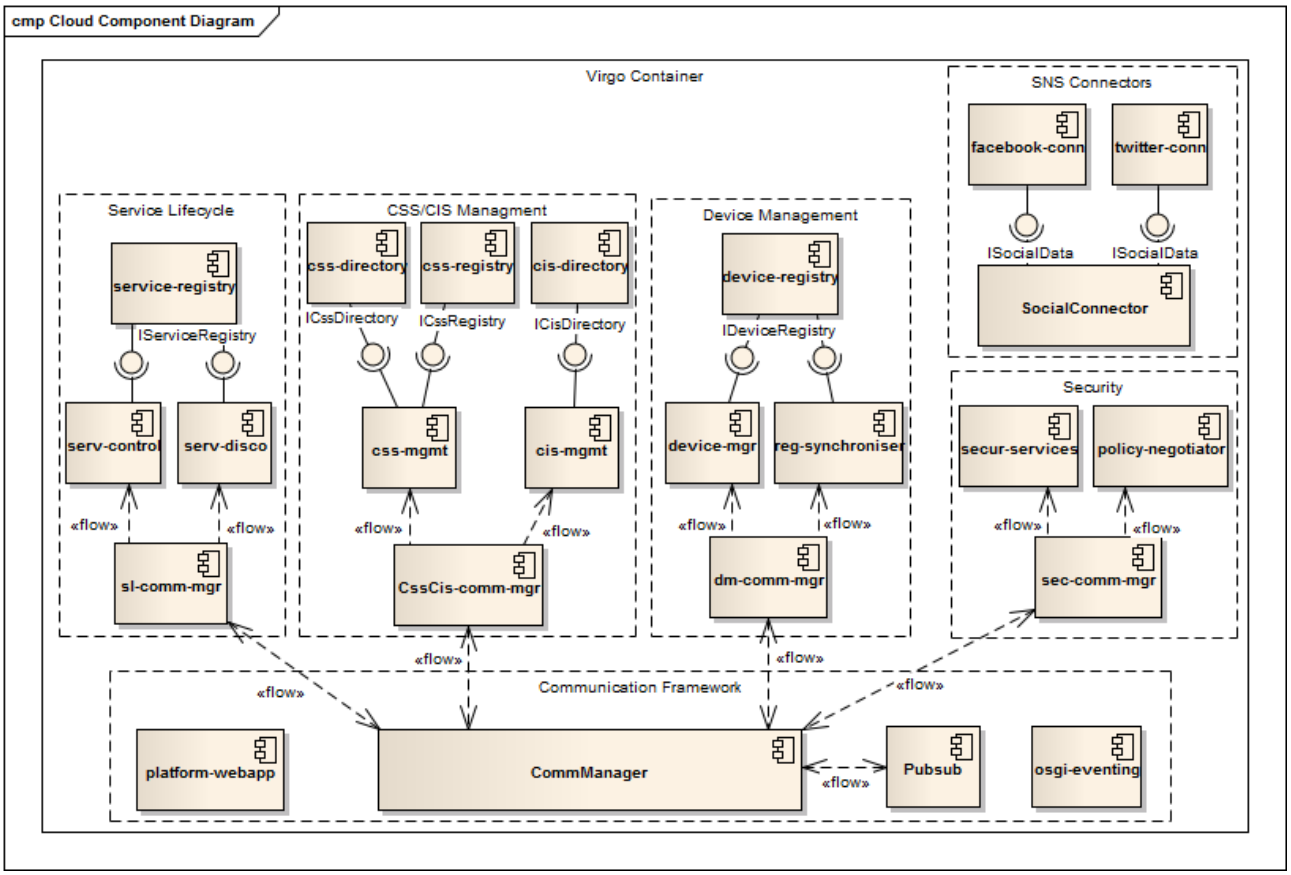
The deployment of the SOCIETIES Platform is divided into several nodes. Each contains different versions of the components as defined by the system architecture:

Cloud Node - This node represents a deployment to a server based device or cloud instance.

Rich Client Node - This node represents a deployment to standard desktop OS. This environment is deemed "rich" in the aspect that the processor, memory, and available disk space is greater than available in a light client node.

Light Client Node - This node represents a deployment to a smart device running a mobile OS. Examples include Windows Mobile 7, Android or iOS.

2.1 Cloud Node/Rich Client Node



2.1.1 Communications Framework

The Communication Framework's goal was to create a Wired Protocol, a documented set of XML communication stanzas. Our solution does not pass Java classes between Client/Server therefore locking our clients to Java only. Now, other development languages can consume our SOCIETIES Cloud node and be part of our solution, eg, a .NET or Python client. This implementation will support adding second mobile OS client later. XMPP gives us a lot of functionality for free that we can leverage: user accounts, authentication, presence status, messaging, information & service querying, etc.

2.1.1.1 Communication Manager

Registered as an External Component with the XMPP server, it represents the communication gateway for sending/receiving messages and information queries to/from other CSS nodes. This component allows the group managers to register their component's namespaces so that the gateway will forward XML messages to the correct component. A factory interface creates new Communication Managers on the demand which are used as gateways for newly created CIS's hosted by this CSS.

2.1.1.2 Pubsub Server

The publish-subscribe eventing functionality provides a framework for a variety of use cases, eg, news feeds, content, presence, geolocation, profile management and any other service that requires event notifications. A person or application publishes information to a node ("topic"), and an event notification is broadcast to all authorized subscribers.

2.1.1.3 OSGI Event Management

This internal interface extends the OSGI eventing system. It is targeted at eventing between bundles running within the same Virgo container. These events are not visible outside of that node, ie, not available between CSS nodes of the same CSS. As we are contained within the Virgo container, developers are free to add their own Java object types as the payload or event info.

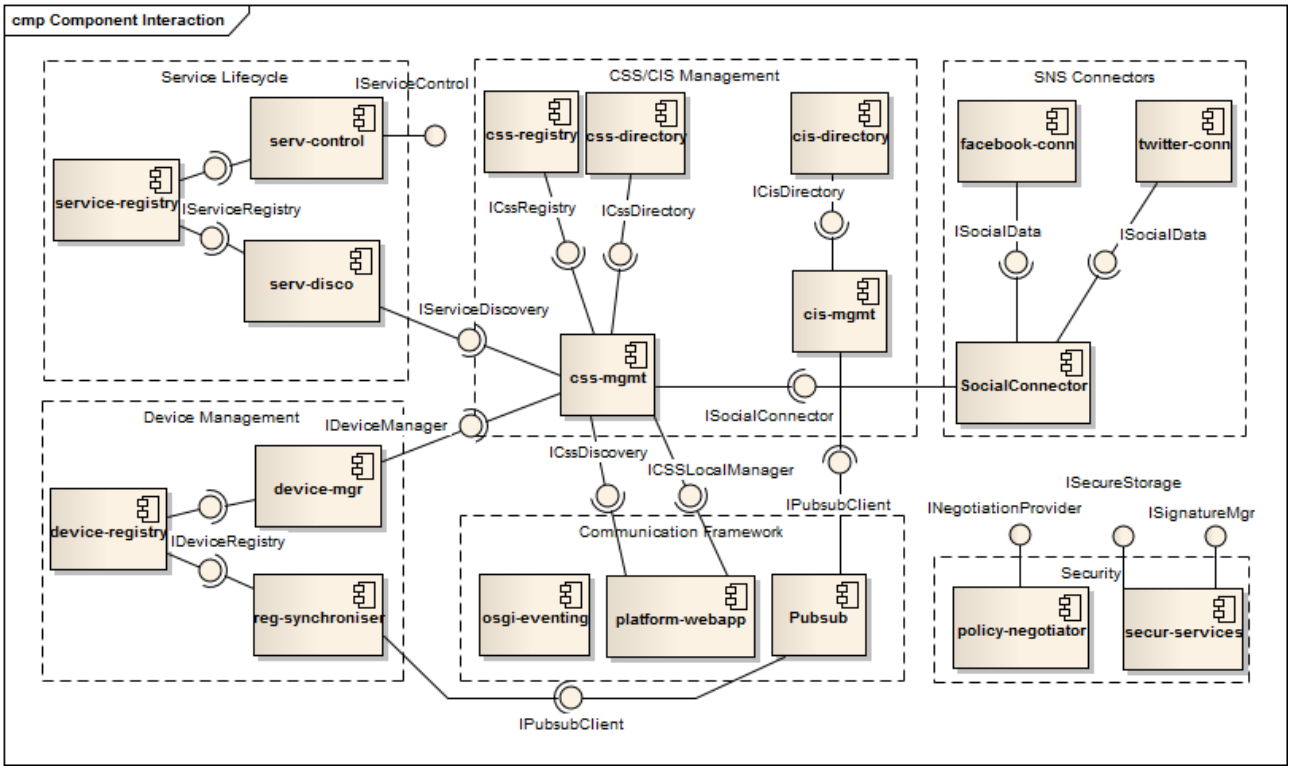
2.1.1.4 Group Communication Managers

Each module group created 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.

2.1.1.5 Platform Web Application

The web application provides a test interface to consume the platform services. Functionality currently supported is:

- Service Browser: list the 3rd party services available within the CSS and their associated metadata
- Each service can then be started/stopped using Service Control.
- CSS Directory: CSS Advertisement Records can be added to the Directory and searched for
- Service Queries: Query for services available from another CSS based on the CSS Advertisement Record



2.1.2 Service Discovery & Lifecycle Management

Service Discovery and Lifecycle Management have developed components 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.

2.1.2.1 Service Registry

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 CIS's they are members of.

2.1.2.2 Service Management

Listens for the installation of new OSGI 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 provides methods for querying for services that are available for both the local or remote CSS's 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.

2.1.3 CSS / CIS Management

CSS management comprises of the the following components which are distributed across the OSGi based components which can run on any operating system (OS) that supports a Java Virtual Machine (JVM) and the Android platform which run on mobile devices such as smartphones and tablets.

2.1.3.1 CSS Manager

This component can run on either a rich or cloud Societies node and allows a user to register and unregister a CSS; log in and out of their CSS; select a domain server and register/unregister a CSS identity; allow a CSS's light and rich nodes to be synchronised with CSS related information from the cloud node, create and modify a CSS's information, its status and the member nodes.

2.1.3.2 CIS Manager

The current version of the Cloud CIS manager is running successfully and it supports listing of CISs and creating CIS. The CISs created there resides on the same bundle, but they use their own communication manager (and have their own Jid). Those CISs have the current services implemented: join, add member and list user.

2.1.4 Device Management

2.1.4.1 Device Manager

This component is the focal point for interaction with the discovery bundles. When a new device is detected it informs this component which assigns it a Device ID. It then creates and maintains the device binding table adding the device to both the device Registry and the Service Registry. If the device is tagged as a "context source", it can raise an internal event to inform the context components of the status of the device (connected, disconnected, updated data).

2.1.4.2 Device Registry

The registry contains a record of all devices connected to nodes of a CSS, storing attributes such as Device ID, Type, description, context Source, etc. When the registry is updated, an event is published to the pubsub node informing the other distributed registries of the CSS to ensure consistency across all nodes.

2.1.4.3 Registry Synchroniser

A distributed component responsible for synchronising the registries on each node, it subscribes to the PubSub node and upon receipt of a new event, updates the local instance of the device registry with the event information.

2.1.5 Security

2.1.5.1 Security Services

The low level services primarily support the Policy Negotiator but exposed also to third parties through the external SOCIETIES API. Digital signature manager has the ability to sign XML elements and embed the signatures in XML files, verify signatures and identities with certificates. Certificates include private keys and are encrypted and stored securely using the secure storage service.

2.1.5.2 Policy Negotiator

The Policy Negotiator bundle offers negotiation or selection of a policy most suitable for the user among various policies offered by some other user. There are always two main sides of negotiation: the requester (who wishes to use a particular service or to join a CIS etc.) and the provider (who is offering the service etc.). The bundle includes both sub-components so each user can have the role of either the requester or the provider in a particular negotiation. Each step of the negotiation is secured with digital signatures, possibly with a valid and accountable identity to enable greater security and trust between parties. The whole process is optimized to minimize number of remote invocations and still provide secure negotiation.

2.1.6 Social Networking Connectors

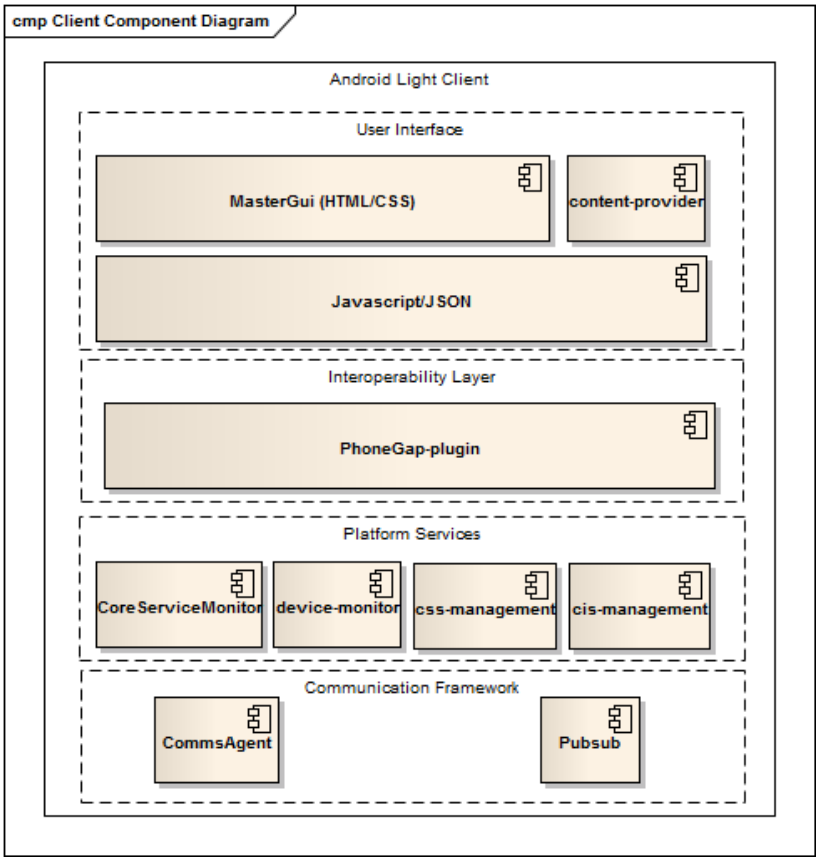
2.1.6.1 Data extracted from Twitter, Facebook

The Social Connectors are able to extract data from numerous existing Social Networks (currently Twitter and Facebook) using their standard APIs. The data is offered as a Container service using the ISocialConnector interface, which provides generic methods for attribute extraction

2.1.6.2 Semantic abstraction of SN extracted data

The vocabulary used in different Social Networks varies in form, but not in content. As such, buddies, followers and friends are all mostly equal terms. This applies to all kinds of social data. In order to operate with this data in a Social Network agnostic way, an abstraction layer is required that maps all such variations into a uniform, common information model. To do so, the SocialData component provides an OpenSocial compatible interface, and aggregates all the information it receives under the OpenSocial model.

2.2 Android Light Client



2.2.1 Communication Framework

CommsAgent: represents the communication gateway. It provides an interface for sending/receiving messages and information queries to/from other CSS nodes by registering via XMPP namespaces

Pubsub: provides the interface for creating event nodes and publishing/subscribing to these nodes that are hosted on the cloud. See Pubsub on client above.

2.2.2 Platform Services

Core Service Monitor: Acts a monitor for the various core services and as a way of detecting Societies compatible 3rd party services starting and stopping.

Device Monitor: Allows various core Android resources to be monitored and act as a central access point for these resources' statuses.

CSS Management: Acts as a peer component to the cloud CSS Manager. It provides similar functionality although for actions such as logging into a CSS and updating a local cache of CSS information it will rely on the CSS cloud node to verify and supply information. This component will also provide the ability to invoke remote methods and allow this and the relevant OSGi peer components to communicate via direct method invocation or events.

CIS Management: This library encapsulates CIS management API into pure java classes so that application code can be reused. The library provides a set of classes such as CIS Manager to the applications and implements content resolver towards the CIS Manager on Android.

2.2.3 User Interface

Master GUI: This HTML5-based GUI allows all of the client's relevant information to be displayed and allow user interaction with the various core components.

Android Client Content Provider: Provides a common access point for external and internal components to access the Android Societies data.

PhoneGap API Reference example: This reference application illustrates the PhoneGap API through the use of an HTML5+Javascript GUI. Every main API functionality can be interacted with or viewed on the GUI. The application also demonstrates the interaction between the HTML GUI and the native Android system using PhoneGap plugins in a bi-directional manner.

3 Installation

This deliverable does not include an end user software distribution and installer, as this will be produced later by WP7. The following instructions describe how to setup your environment, download and build the source code, and then execute the generated binaries.

3.1 Prerequisites Downloads & Versions

- **Java:** Sun/Oracle Java - JDK 6 Update 29 (not Java 7)
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- **Build System:** Maven Version 3.0.3
  http://maven.apache.org/download.html
- **XMPP server:** Openfire 3.7.1
  http://www.igniterealtime.org/downloads/index.jsp#openfire
- **Android:** Android SDK - SDK r12 - v2.2
  http://developer.android.com/sdk/index.html
- **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

3.2 XMPP Server (Openfire) Configuration

After installing Openfire, start the Openfire Server and select "Launch Admin" to lauch 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", eg, "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, ie, 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.

3.3 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

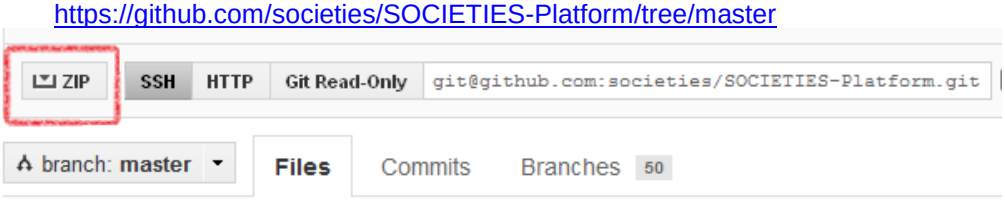
3.4 Android Configuration

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.

3.5 Building the code

The 0.1 release of the SOCIETIES platform can be downloaded from below URL. Unzip the code to a temporary directory

https://github.com/societies/SOCIETIES-Platform/tree/master



Cloud/Rich Client: Change to the /builder folder and issue the command:
mvn clean install –f spring-osgi-projects.xml
Android Client: Change to the /builder folder and issue the command:
mvn clean install –f android-projects.xml

3.6 Running the code

3.6.1 Cloud Node

- Start the Spring container: In the */bin* folder of the Virgo Installation folder, you should execute the command *startup.bat* for windows or *startup.sh* for linux.
- To start the SOCIETIES platform components, you should copy the file *UNZIP_LOCATION/builder/societies-platform.plan* to the *VIRGO_FOLDER/pickup*

This will now start the container and load all the components of the SOCIETIES platform. To view some sample 3rd Party Services in the Service Browser, you can optionally copy *societies-3p-services.plan* to the */pickup* directory also. These are test services that have no user interface.

The Platform web application will allow you to perform some basic functionality provided by the platform. These are test userinterfaces to the components. The long term plan is for Task 6.5 to create a unified and consistent look and feel user interface experience for the trial users.

To view these webpages, you can browse to http://localhost:8080/societies/login.html

3.6.2 Android Node

- Start the Android emulator with the command
  emulator -avd Test22Device -partition-size 128
- Go to the *UNZIP_LOCATION/platform-infrastructure/ClientFramework/AndroidAgent* directory and execute:
  mvn android:deploy
- Go to the *UNZIP_LOCATION/platform-infrastructure/ClientFramework/AndroidPubsub* directory and execute:
  mvn android:deploy
- Go to the *UNZIP_LOCATION/platform-infrastructure/ClientFramework/MasterGUI* directory and execute:
  mvn android:deploy

On your emulator, go to the *Applications* and run the application *MasterGUI* to browse the light client's UI. You will be able to perform some basic tasks such as manage your CSS and edit your CSS Advertisement Record