SOCIETIES
FP7-ICT
Contract no.: 257493

SOCIETIES

Deliverable D5.6

First Integrated Prototype - CSS Individual and Community Experience

Editor:	Sarah Gallacher
Deliverable nature:	Software Prototype Deliverable
Dissemination level: (Confidentiality)	PU
Contractual delivery date:	31 May 2012
Actual delivery date:	31 May 2012
Suggested readers:	Pervasive and social computing designers, researchers and developers
Version:	0.1
Total number of pages:	21
Keywords:	SOCIETIES, Self Orchestrating Community ambiEnT IntelligeEnce Spaces, Intelligent Community Orchestration, Context Management, Personalisation, Privacy, Trust, User Agent

Abstract

This document describes the work performed to implement and integrate the various components of the first release of the work package 5 SOCIETIES 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.

Impressum

[Full project title] Self Orchestrating Community Ambient Intelligence Spaces

[Short project title] SOCIETIES

[Number and title of work-package] WP5 SOCIETIES CSS Individual and Community Experience

[Document title] D5.6 First Integrated Prototype - CSS Individual and Community Experience

[Editor: Name, company] Sarah Gallacher, HWU

[Work-package leader: Name, company] Sarah Gallacher, HWU

[Estimation of PM spent on the Deliverable] 12 PMs

List of authors

Company	Author
HWU	Sarah Gallacher, Eliza Papadopoulou
TSSG	John McGovern
ICCS	Ioanna Roussaki
TRIALOG	Olivier Maridat, Bruno Jean-Bart
ITSUD	Haoyi Xiong, Daqiang Zhang
TI	Luca Lamorte
SETCCE	Mitja Vardjan

List of Code Contributors

Company
HWU
TSSG
SINTEF
ICCS
AMITEC
DLR
TELECOM ITALIA
IBM
TRIALOG
ITSUD
PORTUGAL TELECOM
SETCCE

Table of Contents

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 first 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 scenarios (WP8) that will be developed within other work packages.

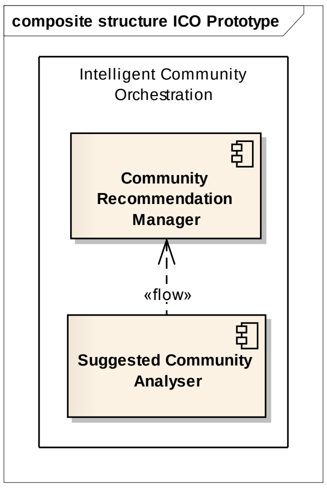
This document accompanies the software deliverable, D5.6, which comprises of open source software. The software is publicly available on our source repository, hosted by Github.com[1], and may be downloaded or browsed. The repository include Java source files, XML configuration files and other artifacts 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 included as an Appendix.

[1] https://github.com/societies/SOCIETIES-Platform

[2] http://maven.apache.org/

2 Prototype Features

2.1 Intelligent Community Orchestration


composite structure ICO Prototype
Intelligent Community Orchestration
Community Recommendation Manager
«flow»
Suggested Community Analyser

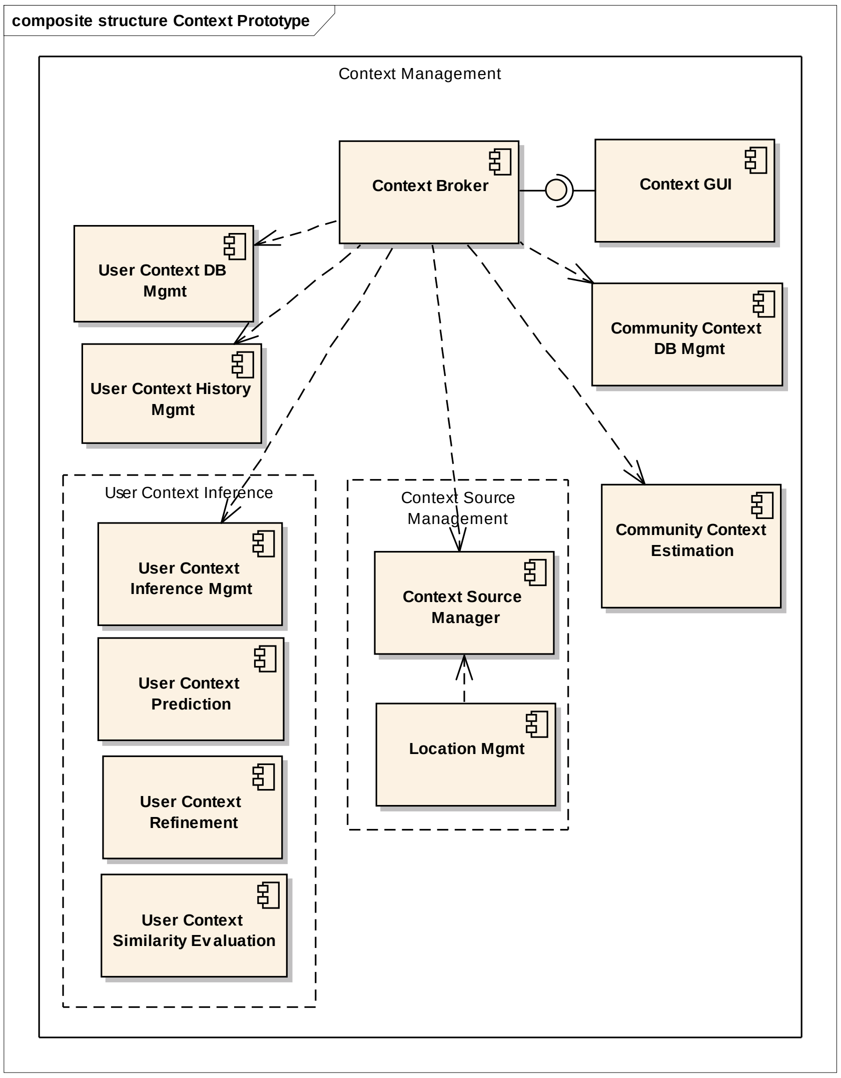
2.1.1 Suggested Community Analyser

The Suggested Community Analyser (SCA) is capable of taking recommendations from sub modules of 5.1 namely the Egocentric Community Analyser (ECA), CSCW and CSM Analyser and can determine based on criteria such as privacy, preferences and context similarity, whether these suggestions warrant some action. Actions currently supported are creating a CIS, joining a CIS, leaving a CIS and creating a sub CIS.

2.1.2 Community Recommendation Manager

The Community Recommendation manager acts on the suggestions of the Suggested Community Analyser (SCA) and determines what action(s) should be taken regarding the creation of new communities and their configuration. Actions currently supported follow those of the SCA namely creating a CIS, joining a CIS, leaving a CIS and creating a sub CIS.

2.2 Context


composite structure Context Prototype
Context Management
Context Broker
Context GUI
User Context DB Mgmt
User Context History Mgmt
Community Context DB Mgmt
User Context Inference
User Context Inference Mgmt
User Context Prediction
User Context Refinement
User Context Similarity Evaluation
Context Source Management
Context Source Manager
Location Mgmt
Community Context Estimation

2.2.1 User 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. This component will demonstrate its full functionality in the first trial.

2.2.2 Community Context DB Mgmt

The functionality of Community Context DB Mgmt component is similar with 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. This component will demonstrate a limited part of its functionality in the first trial.

2.2.3 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. Finally, it triggers the intelligent functionality provided by the inference management. This component will demonstrate almost its full functionality in the first trial.

2.2.4 Context Source Management

The Context Source Manager component serves as the interface from external sensors and context providing services and devices, the context sources, to the context database. 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. This component will demonstrate its full functionality in the first trial.

2.2.5 Location Mgmt

The Location Mgmt component 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, 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. This component will demonstrate part of its functionality in the first trial.

2.2.6 User Context History Mgmt

The User Context History Mgmt component provides the necessary mechanisms for persistent storage of historical context data (History of Context - HoC) for individuals. Historic context data are maintained on a database that resides on the cloud node of each CSS. The component also provides data management functionality in order to store, retrieve and delete historic context data. HoC data is 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. This component will demonstrate its full functionality in the first trial.

2.2.7 User Context Inference Mgmt

User Context Inference Mgmt component coordinates and controls four individual methods for inferring and improving the quality of existing context data. These processes 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. Finally a context similarity evaluation process will also be provided by the Context Management architecture. The User Context Inference Management component will provide the intelligence in order for the right inference process to be selected. This component will demonstrate only part of its functionality in the first trial.

2.2.8 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 both individual and community context. The prediction algorithm is based on neural networks statistical methods. This component will demonstrate only part of its functionality in the first trial.

2.2.9 User Context Refinement

The User 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. This component will demonstrate most of its functionality in the first trial.

2.2.10 User Context Similarity Evaluation

The User Context Similarity Evaluation component that provides functionality for the estimation of the similarity of context data based on context values, semantics, quality of context and comparison algorithms. This component will demonstrate only part of its functionality in the first trial.

2.2.11 Community Context Estimation

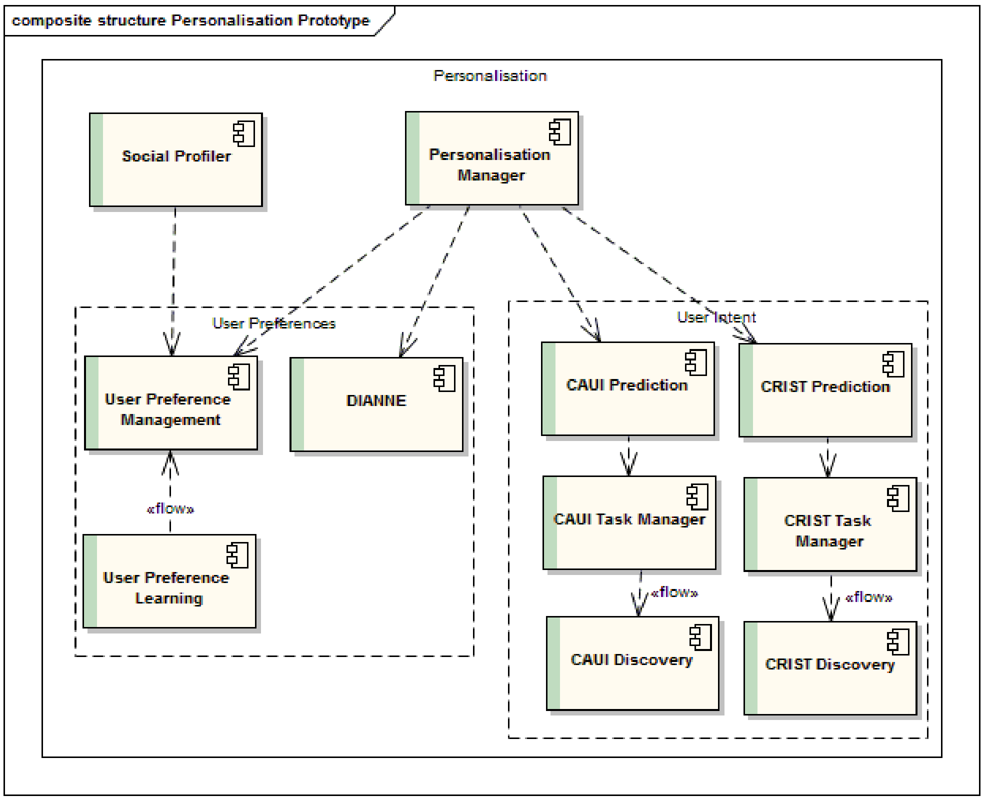
The Community context estimation component allows the extraction of new context data that apply to an entire community of users. The component offers various mechanisms that support the estimation of common context values based on common features, mean value, and majority of preferences. These mechanisms range from simple majority rules up to complex multidimensional calculations. More specifically, the first trial version of this component includes methods for the calculation of alphanumerical and geometrical types of community context. The mean value is an example of simple calculations on a set of values, while the convex hull (based on the quick hull algorithm) is an example of a more complex type of community context calculation.

2.2.12 Context GUI

The context GUI component provides a simple and easy to understand way to present the context of a device/User. It provides a simple User Interface to show the available context and navigate through the different attributes and values. This component will also demonstrate part of its functionality in the first trial.

In addition to the above functional components, there is also the Context Model, which represents the context information in a structured and consistent manner. It will demonstrate its full set of features in the first trial.

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.

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 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.

2.3.5 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.

2.3.6 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 behavior 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.

2.3.7 Context Aware User Intent (CAUI) Prediction

The CAUIPrediction 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.

2.3.8 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.

2.3.9 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.

2.3.10 CRIST User Intent Prediction

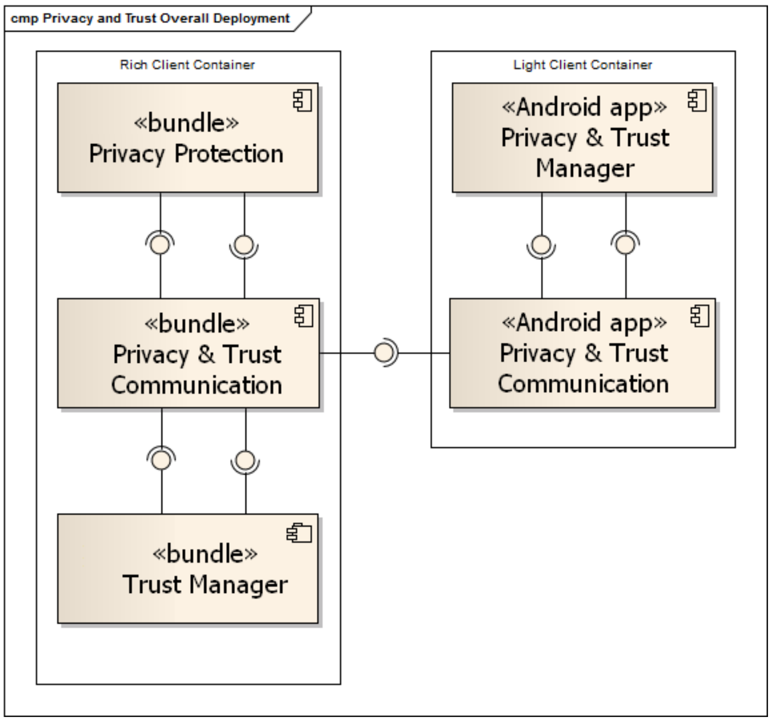
The CRIST User Intent Prediction component exposes two methods that allow the Personalisation Manager to request a CRIST user intent prediction based on the user performing an action and on the context of the user changing. A third method is also provided that allows a component to request the current 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.

2.3.11 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, ego-centric...), 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) 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

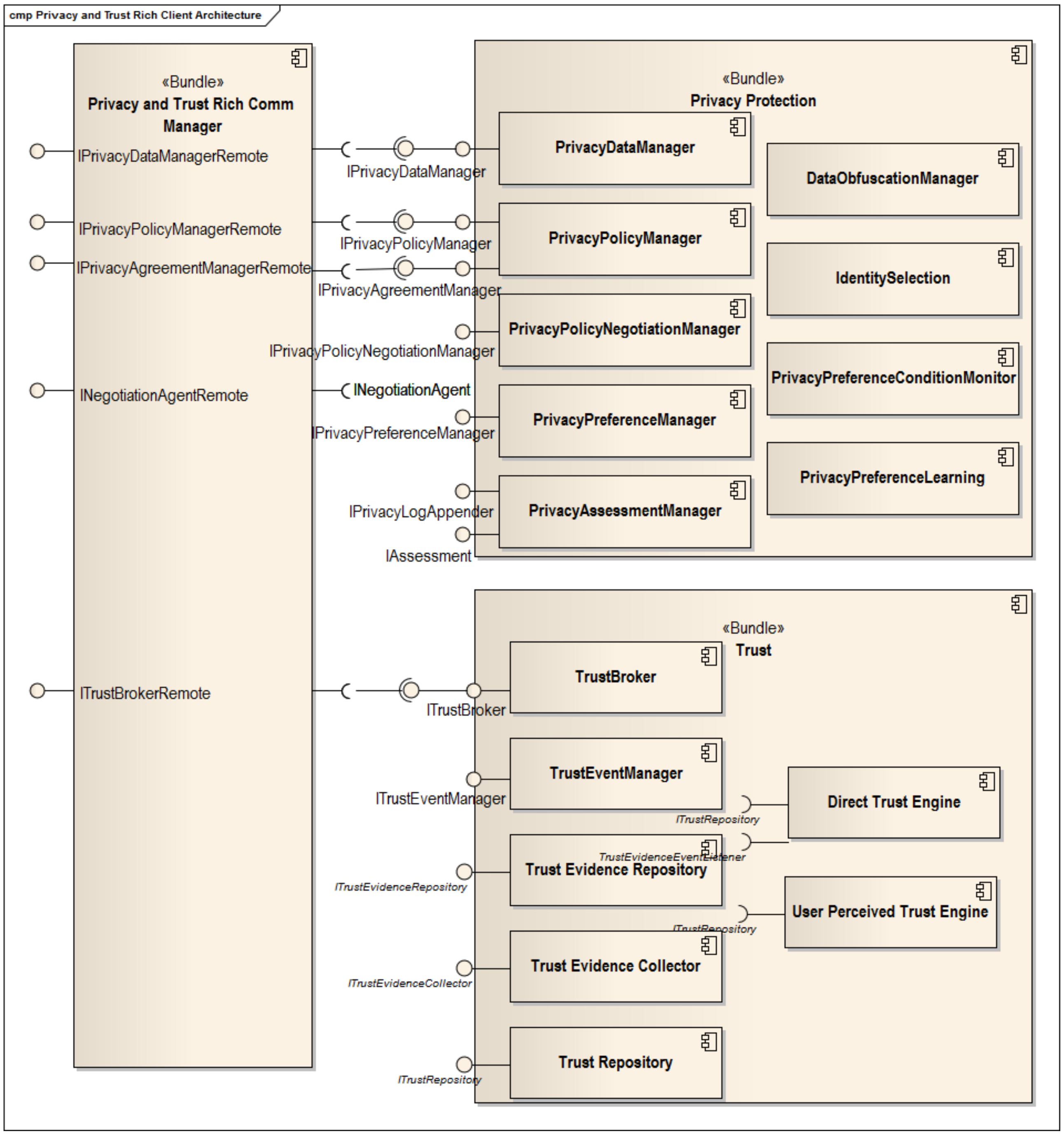
The following diagram depicts the modules provided by the Privacy & Trust task, both on rich/cloud node and light node.


cmp Privacy and Trust Overall Deployment
Rich Client Container
«bundle» Privacy Protection
«bundle» Privacy & Trust Communication
«bundle» Trust Manager
Light Client Container
«Android app» Privacy & Trust Manager
«Android app» Privacy & Trust Communication

The Privacy & Trust protection provides components in order to protect personal data. The process is divided into several steps starting from a learning phase where the user privacy preferences are collected and the privacy policy for service 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 a personal data from the SOCIETIES context each time another user, CIS or third party try to access them,
- collecting trust and assessment evidences to provide relevant feedbacks and to the user.

The architecture on the light and the rich clients are similar, but at the moment, the light version only performs a subset of functionality and requires the rich client. The following paragraph described each component based on the rich client architecture.



2.4.1 Privacy Data Manager

Manages 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 Preference 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 quicker when the same request is received latter. At the moment, SOCIETIES Context data are protected by the “Privacy Data Manager”, and other data providers (like the Device Manager, or the activity feeds) will be integrate with privacy aspects.

- 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. Currently, this component handles two types of data: GPS coordinates, and user name.

- Available on Virgo and Android.
- No need of communication.

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. A privacy policy includes the personal data that will be used in a CIS or by a third party service, whereas a 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.
- Communication available.

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 at the moment hence the current version of this component always returns the main identity of the user. When the functionality for supporting multiple identities will be available in future SOCIETIES release, the changes to this component will be transparent for other components, since SOCIETIES the architecture already supports multiple identities.

- Available on Virgo. No need on Android.
- No need of communication.

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 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. 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.
- Communication 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 Direct and User Perceived Trust Engine

The “Direct Trust Engine” is responsible for evaluating the trust evidence that result from direct interactions among the trustor and the trusted entities, in order to estimate the trust level of the latter. More specifically, this component retrieves such information from the Trust Evidence Repository, processes it, estimates the direct trust and stores the estimated value in the Trust Repository. 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 (trustor).

- Available on Virgo.
- No need of communication.

2.4.9 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.

2.4.10 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). Note that the functionality related to the collection and processing of the latter will be available in the next release.

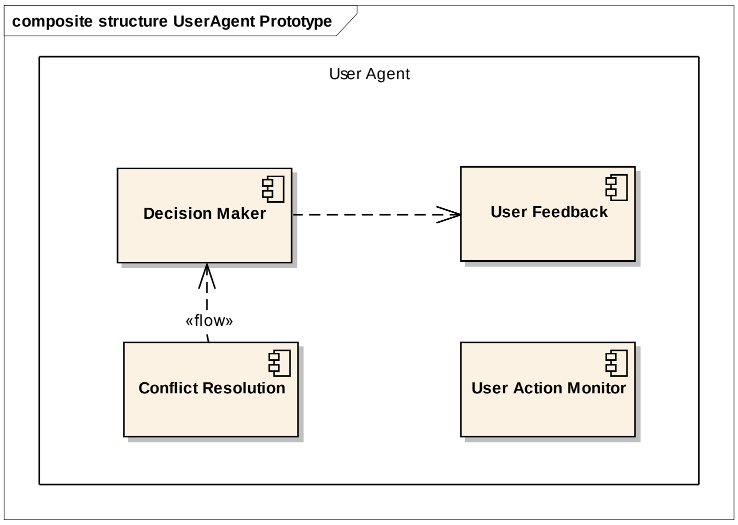
- Available on Virgo.
- No need of communication.

2.4.11 Trust Broker and Trust Event Manager

The Trust Broker and Trust Event Manager acts as a gateway to the trust calculations maintained in the Trust Repository. In this respect, it provides a query interface through which trust information consumers can specify the ID of a particular entity in order to retrieve its user-perceived trust value. 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 by subscribing to the trust event manager. In addition, the Trust Broker enables the exchange of trust information among different CSSs via the platform's Communications Framework.

- Available on Virgo.
- Communication available.

2.5 User Agent



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 DecisonMaker extending the AbstractDecisionMaker. The AbstractDecisionMaker controls the flow of conlict detection and resolution, while 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.

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. However when the conflict is irresolvable, conflict resolution may query the user feedback by sending request to User Feedback Manager.

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.

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 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.

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

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.

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

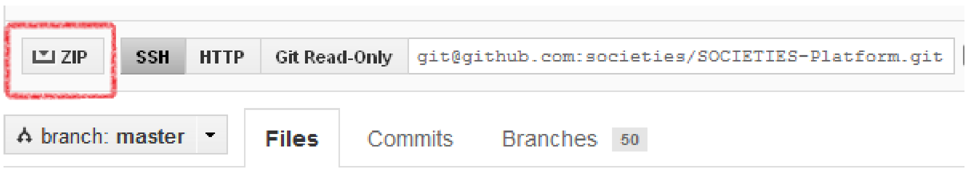
Android Configuration

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 0.2 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

Running the code

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 WP5 platform components, you should copy the file *UNZIP_LOCATION/WP5.plan* to the *VIRGO_FOLDER/pickup*

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: *mvn android:deploy*

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