SmartSociety

Hybrid and Diversity-Aware Collective Adaptive Systems
When People Meet Machines to Build a Smarter Society

Grant Agreement No. 600584

Deliverable 8.2 Working Package 8

Platform Prototype: Early Results and System Design

Dissemination Level (Confidentiality):[1]	*CO*
Delivery Date in Annex I:	*31/12/2014*
Actual Delivery Date	*December 16, 2014*
Status[2]	*D*
Total Number of pages:	21
Keywords:	*Platform, Prototype, Integration*

[1]PU: Public; RE: Restricted to Group; PP: Restricted to Programme; CO: Consortium Confidential as specified in the Grant Agreeement
[2]F: Final; D: Draft; RD: Revised Draft

Disclaimer

Full project title:	SmartSociety: Hybrid and Diversity-Aware Collective Adaptive Systems: When People Meet Machines to Build a Smarter Society
Project Acronym:	SmartSociety
Grant Agreement Number:	600854
Number and title of work-package:	8 Architecture and Integration
Document title:	Platform Prototype: Early Results and System Design
Work-package leader:	Iacopo Carreras, UH
Deliverable owner:	Iacopo Carreras, Daniele Miorandi
Quality Assessor:	Luc Moreau, SOTON

List of Contributors

Partner Acronym	Contributor
UH	Daniele Miorandi, Iacopo Carreras, Tommaso Schiavinotto

Executive Summary

This report briefly describes the context and components of the first version of the SmartSociety platform.

The SmartSociety platform integrates the instantiation of the components designed and developed by the Consortium members within the framework of the technical workpackages WP2-WP7.

The report at hand includes a set of revised architectural diagrams, developed by the Consortium building upon the results in Deliverable D8.1. The revised architecture is now fully aligned with the specification of components as developed within WP2-WP7, and has been developed according to the high-level requirements identified in WP1 and further analysed in D8.1.

The report then describes the interfaces of the platform components integrated up to M24 (the peer manager, the orchestration manager and the communication middleware). The software produced is then presented.

The report concludes with a roadmap detailing the integration milestones for the third year of project activities.

Table of Contents

List of Acronyms

Acronym	Full Name	Description
CM	Communication Middleware	System component in charge of managing communication channels between the platform and the peers.
IM	Incentives Manager	System component in charge of managing the implementation of incentive schemes.
KB	Knowledge Base	System component in charge of storing and managing the knowledge in the platform.
OM	Orchestration Manager	System component in charge of orchestrating the lifecycle of tasks on the SmartSociety platform.
PF	Programming Framework	System component in charge of exposing appropriate primitives and interfaces to application developers.
PM	Peer Manager	System component in charge of managing peers.

1 Introduction

This report is a short accompanying document whose aim is to describe the components developed and integrated within a coherent platform by the Consortium members during the second reporting period. The actual deliverable is a prototype of the SmartSociety platform, which can be found at: ...

Starting from the analysis of requirements and initial platform architecture in D8.1, WP8 undertook an intense dialogue with technical workpackages (WP2-WP7) in order to ensure full alignment of the top-level platform architecture and of the scientific and technical outcomes of the single WPs. The results of this first phase was a revised architecture of the SmartSociety platform, which is presented in Sec. 2. During the second half of the year the actual integration of components started, resulted in a 'minified' version of the platform, which integrates the Peer Manager (PM), the Orchestration Manager (OM) and the Communication Middleware (CM). For such aforementioned components, the interfaces used are summarised in Sec. 3. which is described in Sec. 4. The missing components will be integrated during the third year according to the roadmap outlined in Sec. 5.

2 The SmartSociety Platform Revised Architecture

During the second year of project activities integration activities started. In the integration process, the architecture initially presented in Deliverable D8.1 was deeply revised. This process was required in order to align the activities carried out within the project's technical work packages (WP2-WP7) and to ensure interoperability among the components developed by the various partners. In this section we briefly present the architecture as it currently stands (at M24). No major changes are currently foreseen, even if —given the research-oriented nature of the SmartSociety project— this cannot be guaranteed. In this sense, the architectural specifications of the SmartSociety platform have to be seen as a live document, which reflects the actual progress of the research activities carried out by Consortium partners.

2.1 Logical View

The logical view of the SmartSociety platform is presented in Fig. 1.

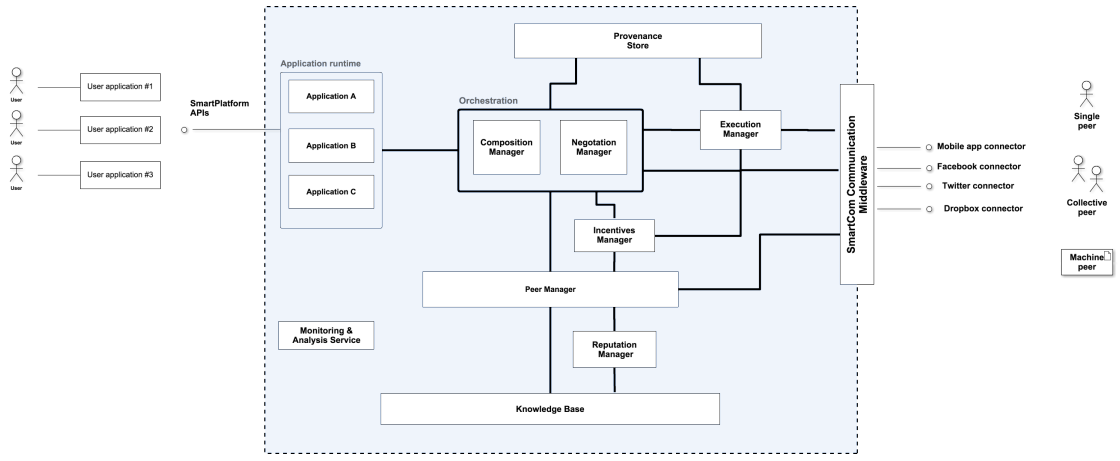

Figure 1: SmartSociety logical view.

The core components which are part of the revised version of the SmartSociety are the following:

- **Application Runtime**: it is the component where the applications created by developers are executed. The runtime provides all the necessary programming abstractions for interacting with the various internal components provided by the SmartSociety platform.

- **Orchestration Manager**: it provides two key functionalities: composition and negotiaton. Composition takes as inputs tasks and interacts with the peer manager to find suitable peers for completing the task. The negotiation manager is in charge of handling the negotiation process with peers in order to ensure that the services and resources required to carry out the task are actually present. This includes the initial creation of a *composition* which can fulfill a given computation task, whereas a composition determines (i) the peers which can potentially execute such task, and (ii) the necessary interactions and/or external services which are needed to execute such task.

- **Peer Manager**: it is responsible for managing all peer-related information. This includes their profile, as well as any other information which will be used by the Orchestration Manager for identifying possible peers which can execute a certain task. It maintains a profile of each peer, which represents a model thereof in terms of knowledge, resources and capacity. It provides a peer search functionality that allows other components to find for the most appropriate peers for a given tasks. Besides individual peers, the PM also manages collectives, which are groups of peers characterized by specific properties (see Deliverable D4.2 for details).

- **Context Manager**: it is responsible for monitoring the context the peer is in. For an individual human peer, this includes, e.g., tracking the location and recognizing the activity the peer is currently involved in. This can be further used to trigger context adaptation in applications.

- **Reputation Manager**: it handles the reputation of peers. It computes the reputation of a given peer based on feedback from other peers and users. It uses data from the provenance store in order to carry out the computation. Reputation is based on various metrics that the system will be able to compute, starting from peers'

execution of tasks. Reputation information can be exposed directly to application users or used by the PM in the selection of suitable peers for a given task.

- **Knowledge Base**: it is shared among the various components of the platform, and contains an agreed ontology about peers, task, workflows, incentives, etc. It provides the domain terminology, as well as all the semantic relations between terms. In particular, the knowledge base is responsible for ensuring that all component share a common understanding of the inputs/outputs between any two parties of the platform. The overall knowledge base will be divided into domains, which allow to capture the diversity of the entities being part of SmartSociety. A semantic reasoner will allow to perform semantic queries over the knowledge base.

- **Provenance Store**: it logs actions performed by platform components and peers according to the W3C PROV recommendation. In particular, the provenance store is the component responsible for keeping track of how the overall compositions are being created, executed and how the data managed by the platform is being transformed. It will play a fundamental role in the evaluation of peers reputation, as well as in ensuring that the system as a whole enforces the proper privacy constraints imposed by the peers.It further supports an auditing service which is able to reconstruct and visualize provenance trails.

- **Execution Manager**: the execution manager deals with the different interaction patterns that might be required to interact with peers.

- **Monitoring and analytics service**: it logs and monitors the platform jobs and can be used by platform administrators to perform root-cause analysis and to extract analytics on the performance of the system.

- **Incentives manager**: it provides insight on incentives and interventions that can be used to achieve higher quality results.

2.2 Deployment View and Network Diagrams

The SmartSociety platform has been designed around a REST architecture with the aim of supporting flexibility in the deployment model. This means that the platform shall seamlessly support both single-tenant as well as multi-tenant deployment models. Also, the platform component can be centralised on a single infrastructure or can be distributed

across different servers. The choice of the specific deployment model to be used depends on technical as well as business considerations. In the remainder of this section we present, as a use case, the current deployment utilised for integration, testing, validation and experimentation purposes. This is by no means to be considered the only model supported, but it provides an actual example of the supported configuration.

A network diagram is reported in Fig. 2. The end user in this case interacts with a User Application running on her smartphone. The mobile app interacts with the Application residing in the SmartSociety Platform Core Deployment. Some of the components reside on different servers; in this case the PM is hosted by partner DISI and the PS and RM are hosted by partner SOTON. The Core Deployment further manages interactions with peers (through the CM). In this case we have a human peer which interacts through a Peer Mobile Application, a machine peer (Google) and a machine peer (Twitter) which then reaches out to human peers interacting with it through any Twitter client.

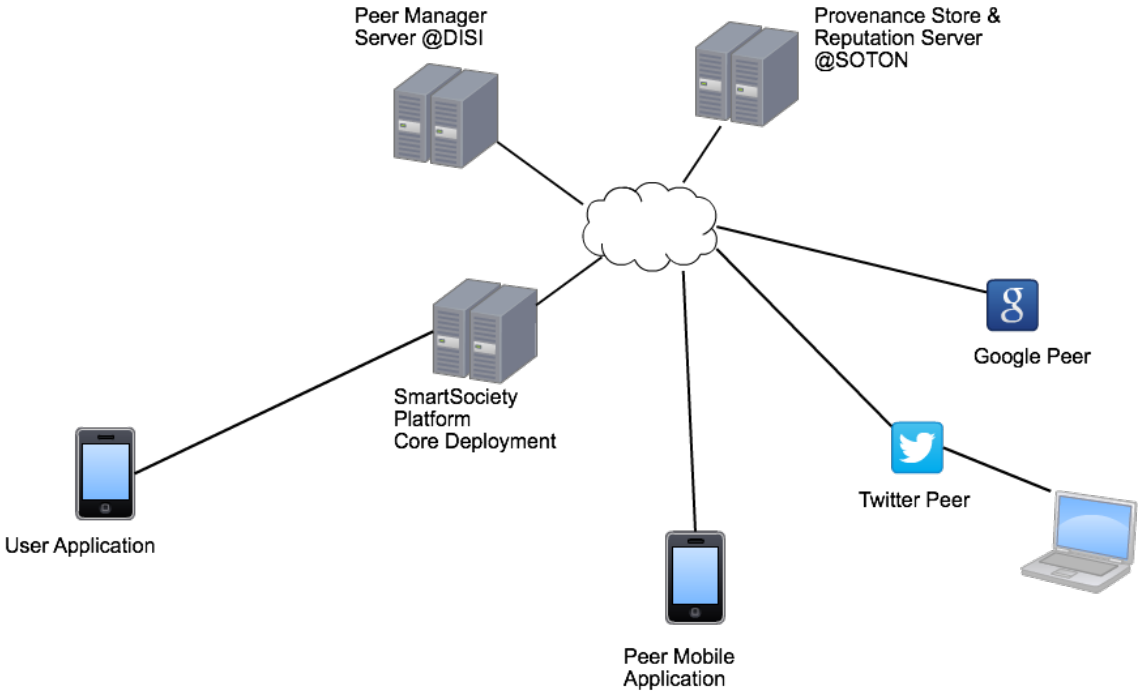

Figure 2: SmartSociety platform: network diagram

The corresponding deployment diagram is reported in Fig. 3. Here it can be seen that the core deployment server hosts four key components, i.e., the incentives manager, the communication middleware, the orchestration manager and the application container

(where the application is executed). On the end user smartphone, besides the user application (client) also the client of the context manager is deployed. This interacts with the corresponding server component on DISI server, where the PM is also hosted. The PS and RM are on the SOTON server. A third-party server is used to host both Google and Twitter peer, which then connect to the corresponding web service.

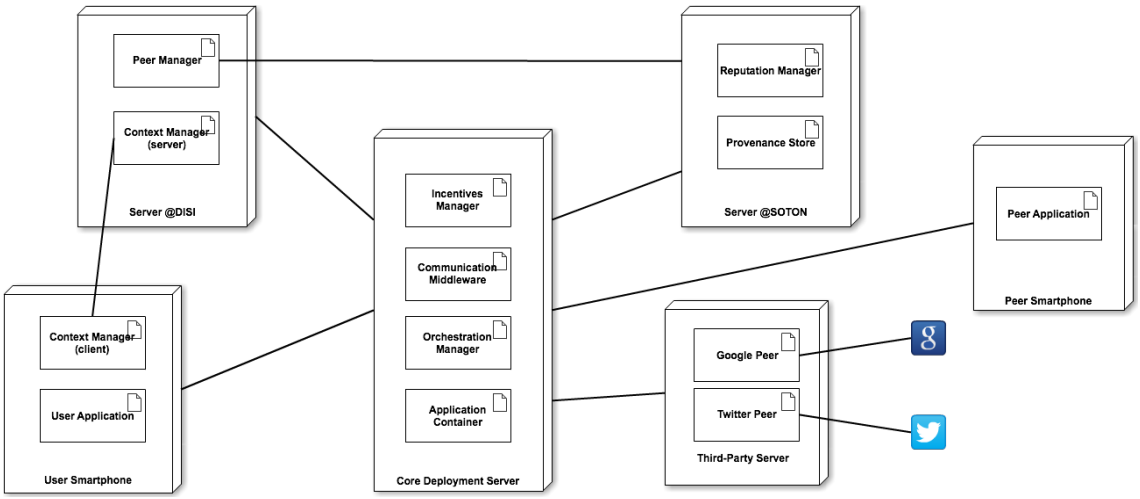

Figure 3: SmartSociety platform: deployment diagram

2.3 Dynamic View

Fig. 4 provides a sample dynamic view of how the various components of the SmartSociety platform interact with each other, when running an application based on SmartSociety.[3]

The 'template' sequence diagram can be summarized in the following steps:

- Platform invocation: the starting point is a user application (e.g., a mobile application) which is exploiting the SmartSociety platform for running a human computation task (refer to Sec. 2.4 for a more concrete example). The request is directed towards an application deployed in the SmartSociety application container and providing the necessary support for executing the requested task. The invocation will contain all the necessary metadata, including a complete description of the task, according to the APIs offered by the application running in the platform.

[3]For the sake of clarity, and as these were developed while integrating the first version of the platform, only interactions among the user application, application, orchestration manager, peer manager, communication middleware and human peer are reported.

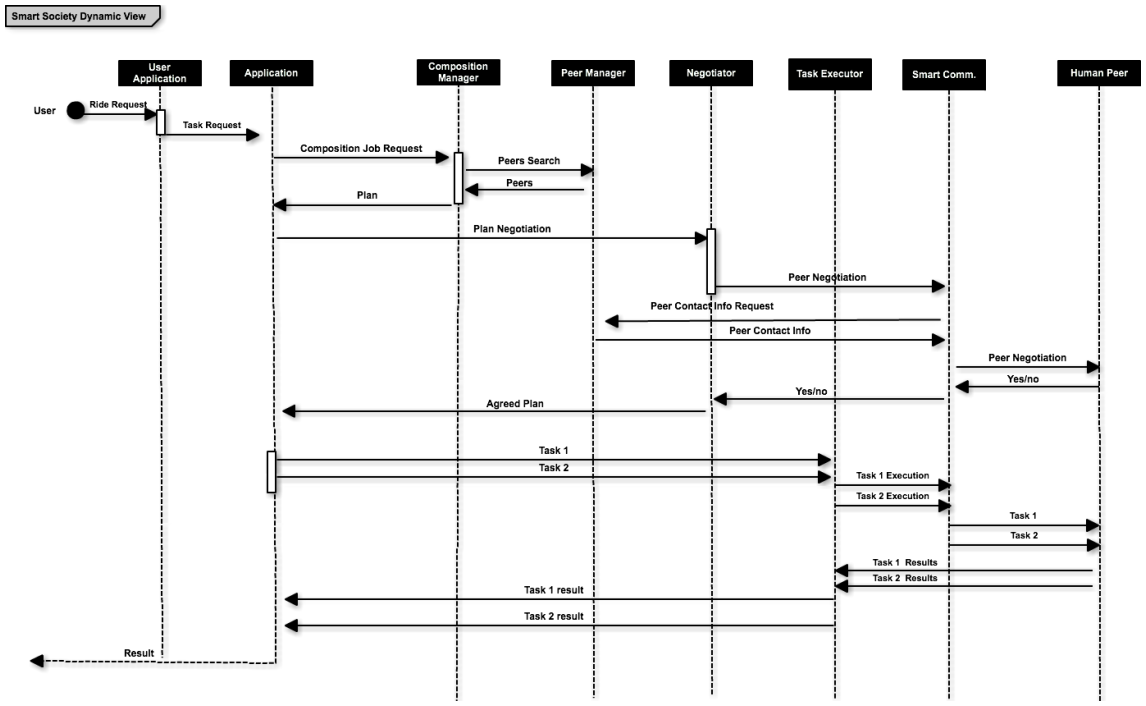

Figure 4: SmartSociety dynamic view.

- Composition request: the application, after receiving the request from the user application, converts it into a *composition job request*. A composition is the request to create a composition of peers/collectives capable of executing the task. The composition job request happens through the programming abstractions provided by the runtime, which allow to fully characterize the task that needs to be executed. The composition job request is received by the OM.

- Peer search: the OM queries the PM in order to identify suitable peers for carrying out the intended task. The list of identified peers is then returned to the OM.

- Negotiation: the OM uses the list of peers provided by the PM to start negotiations with them. It reaches out to them using the functionality provided by the CM.

The SmartSociety platform is meant to support a rather wide range of social computation patterns (or templates). In order to provide insight into the flexibility of the platform and the actual interworking of components, we have developed sequence diagrams for two

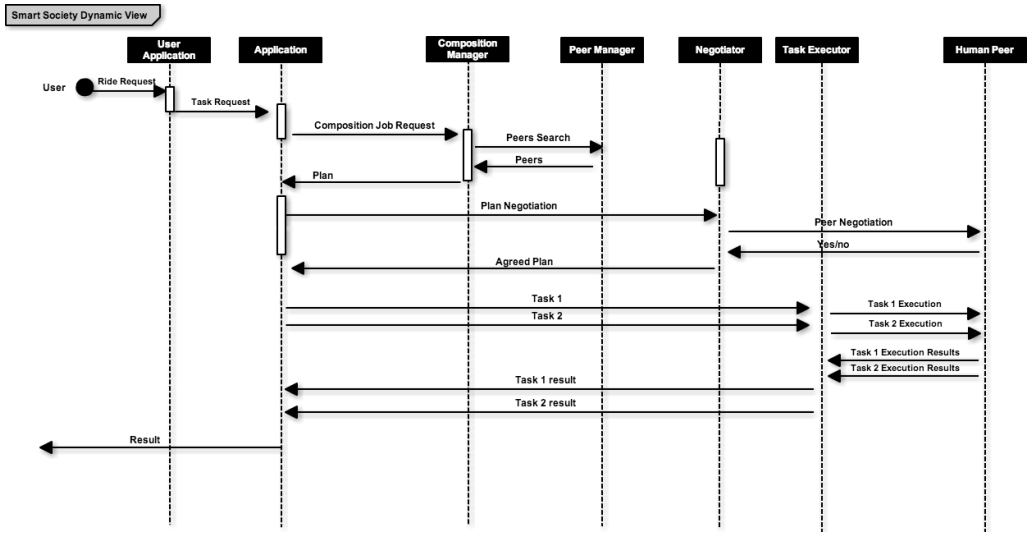

Figure 5: Sequence diagram of the SmartShare application.

'extreme' applications:

- SmartShare is a ridesharing system able to account for user's preferences and to compute recommendations based on the feedback provided by other service users. It is what we call a 'full negotiation' scenario, in which the computational task of finding an agreement on the rides is left to individuals and collectives. The platform in this case is used to carry out administration tasks, in particular keeping track of the rides and ride requests, their status and to maintain reputation of drivers and passengers.

- AskSmartSociety! is a Q&A service supporting hybridity. The computational pattern here is that typical of micro-tasking applications (á la Mechanical Turk, roughly speaking), where the task in this corresponds to a question to be answered. The service supports hybrid computation in that questions can be transparently provided by machine peers or human peers. Quality criteria can be specified in order to define when a chosen answer has to be presented to the user.

In the following we present details about the two aforementioned applications.

2.3.1 Example: SmartShare

2.4 Example: Ask SmartSociety

Ask SmartSociety is a simple Questions and Answer service enabled by the SmartSociety platform which has been used as the benchamark for implementing the initial version of the SmartSociety platform. It focus on tourism, which is the reference domain to be used for validating the SmartSociety vision.
Ask SmartSociety! will be a service where users can post questions in natural languages and peers can provide answers. Peers providing answers can be humans (individuals or collectives) as well as machines (intelligent software agents). Peers can compose (forming collectives, hybrid or not) to provide answers. Answers can be ranked based on the reputation of the peers or on community ranking (similarly to stackoverflow). In some instances the user issuing the question can select an answer and provide feedback on the peer providing it. Two examples (grounded in the tourism application scenarios) can help in understanding the features of the Ask SmartSociety! service:

Next week Peter will fly to Venice. He will be busy in meetings during the day but wants to explore some hidden places at night. He could well explore various online tourist sites but he prefers to ask experts and local people. He could also google for relevant content, but he does not actually need an answer right away, he just needs to get it in one week. And having one system which allows him to query local experts, web-based recommendation services and incoming tourism institutions looks definitely appealing to him! Alice is visiting Milano during the next week. It is his first time in Milan, and she is looking for a restaurant in the city centre. Since it is spring time, she would love eat outside and therefore find a restaurant with a garden. Alice is also celiac, and she needs to find restaurants, which do have gluten free menus. She relies on the Ask SmartSociety! application for retrieving some suggestions on where to have dinner during her stay in Milan. She is looking for unconventional recommendations.

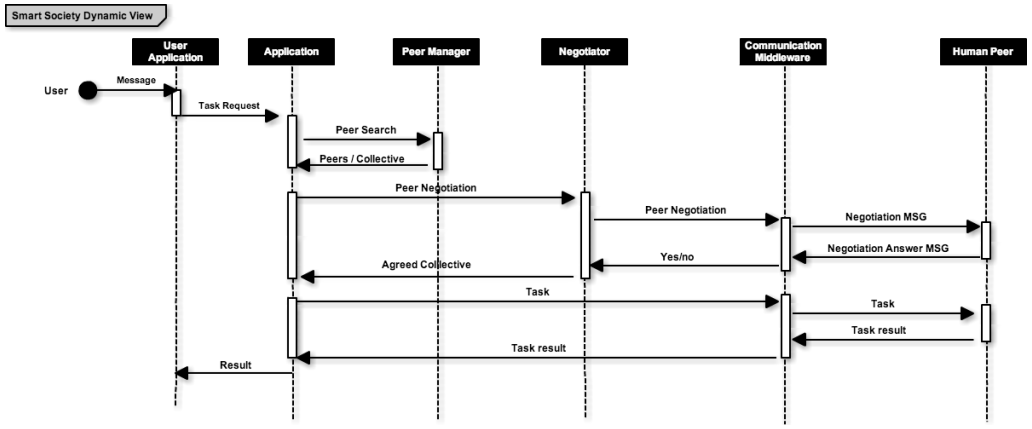

Figure 6: Sequence diagram for AskSmartSociety! applications.

3 Interfaces Specifications

In this section we will list the components integrated in the current instance of the SmartSociety platform and the key APIs thereof used for supporting the two scenarios described in the previous section.

3.1 Peer Manager

The Peer Manager (PM) API specifications can be found at `http://demos.disi.unitn.it:8080/smartsociety/` and are further described in Deliverable D4.2. The PM is used for:

- **Retrieving peers satisfying some requirements**: This request is made by the Orchestration Manager in order to identify potential peers composing a suitable collective for carrying out a given task. Endpoint used:
  `???`

- **Creating collective**: This request is made by the Application/Application Runtime at the end of the negotiation. Endpoint used:
  `/peers/collectivePeer` (POST)

- **Retrieving peers in a collective**: This request is made by the Smartcom, so that it will be able to retrieve peer information later. Endpoint used:
  `/peers/collectivePeer???` (GET)

- **Retrieving peers info**: This request is made by Smartcom, that requires the information about the communication channel to be used. Endpoint used: `/peers/collectivePeer???` (GET)

3.2 Communication Middleware

The SmartCom Communication Middleware APIs specifications can be found at `https://github.com/tuwiendsg/SmartCom` and are further described in Deliverable D7.1. The Communication Middleware is mainly used for:

- **Application to peer communication**: The Smart Society application sends messages to the peers belonging to a collective through Smartcom, that will deliver it using the preferred communication channel declared in the peer profile.
- **Peer to application reply communication**: Peers can reply to messages from the application. The platform provides an endpoint to which the peers can send a reply to a given message. The message is identified by the conversation id, this attribute is also used to know which application. originally created the message, so that the platform can correctly dispatch it.

3.3 Orchestration Manager

The Orchestration Manager APIs specification can be found at `https://bitbucket.org/rovatsos/smartsociety-internal/wiki/PeerManager/APIBGU` and are further described in Deliverable D6.2. The OCM is mainly used for:

- **Posting a new task request**: In order to post a task request the following endpoint is used:
  `/applications/:app/taskRequests` (POST)
- **Querying task requests:** The Smart Society application polls the orchestration manager to know the composition/negotiation state of the request. Endpoint used: `/applications/:app/taskRequests/:request_id` (GET)
- **Querying tasks:** The Smart Society application retrieve the negotiable tasks to know which peer to contact for the negotiation. Endpoint used: `/applications/:app/tasks/:task_id` (GET)

- **Updating tasks:** The Smart Society application update the task according to whether the peer accepted or rejected it. Endpoint used: `/applications/:app/negotiations` (PUT)

4 Platform description

At the moment the platform mainly consists of a runtime allowing the Smart Society application of managing the submission tasks by the user application. The platform provides also some library (against which the application is compiled) to operate with the current components (SmartCom, Orchestration Manager, Peer Manager).

A Smart Society Application has its own identifier, generated in registeration phase. At the moment the application code is written by a developer directly in Java, in the final version the code will be generated through the programming framework. Each time a task is submitted to the application a new application instance is created, each instance has it's own state.

4.1 Runtime

When the runtime is run three endpoint are provided:

/task/:applicationId/ To submit tasks, the runtime will ask the application to create a *instance initializer* that will take care of the setup phase of the task. The application is expected to carry out all the operation that can be performed before interacting with the peers. The posted data is domain dependent, and it will be serialized and managed by the application at the moment of the instance creation. The runtime associate to the instance (hence the task) an identifier that is returned as response.

/task/:applicationId/ To retrieve the status of all or part of the tasks of the application, query parameters can be used in the query and they will be managed by the application that might filter which tasks to show based non them. Note that the status format for each task is just required to be a valid json node (either a text or more complex structures), but it will be completely domain dependent, in fact the smart society application can send whatever information the user application requires.

:applicationId/:taskId To retrieve the status of a specific task, as for the previous point both query parameters and the response are domain dependent.

essage/:applicationId/ Used by the peer applications to communicate back with the application. To use this endpoint the application must have previously contacted the peer with a message containing a given conversation identifier. Such identifier is used then by the runtime to dispatch the message the correct application instance, the content of the message

is then handled by the application instance that can change it's state according to the information received.

4.2 Component Library

The component library provides allows the application development to be abstracted from the real components, allowing easier testing. Furthermore the component wrappers will give a simple interface to the component that is integrated with the runtime.

4.3 Expected evolution

In the future version of the platform each application and its runtime will be containerized. A service for routing the requests from general endpoints to the right application runtime will be provided.

The way the runtime will evolve, and its interaction with applications, is highly dependent on the outcome of the programming model and programming framework definition efforts.

5 Discussion and Further Integration Steps

In this deliverable we have described the core structure of the SmartSociety platform. At the moment the platform integrates three key components: peer manager, orchestration manager and communication middleware. This 'minified' version of the platform is sufficient for developing applications and run experiments. It further represents the core around which the other components will be integrated and deploy in order to realise the full-fledged version of the platform.

In line with the DoW, the actual internal release of the first version of the platform (including user modules) represents MS18 and is due at M27. By M27 the integraiton roadmap foresees the integration of the provenance store and of the reputation manager. It will further include a first version of the execution manager and of the monitoring and analysis service. Security and privacy aspects will be integrated during the third year of the project, in particular within the scope of WP4 (where all personal information is stored). The programming framework development will be strictly aligned with the platform functionality enhancement and will support the easy registration and deployment of applications. The context manager will be integrated with the peer manager during year-3. The incentives manager will be integrated in a later stage of the project, the actual integration date depending on the progress of research activities within WP5. Further work will be carried out in a dialogue with WP1 in terms of the governance of the platform. Last, but not least, the development of the platform will also reflect the work on exploitation plans and potential business models carried out within the scope of WP10.

The second platform prototype (including validation results) will be delivered as D8.3 at M30.

References