SmartSociety

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

Grant Agreement No. 600584

Deliverable 8.4 Working Package 8

Final platform prototype and validation

Dissemination Level (Confidentiality):[1]	*PU*
Delivery Date in Annex I:	*30/06/2016*
Actual Delivery Date	*August 1, 2016*
Status[2]	*D*
Total Number of pages:	23
Keywords:	*Platform, Prototype, Integration, Validation*

[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
This document contains material, which is the copyright of *SmartSociety* Consortium parties, and no copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights. The commercial use of any information contained in this document may require a license from the proprietor of that information. Neither the *SmartSociety* Consortium as a whole, nor a certain party of the *SmartSociety*s Consortium warrant that the information contained in this document is suitable for use, nor that the use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information. This document reflects only the authors' view. The European Community is not liable for any use that may be made of the information contained herein.

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:	Final platform prototype and validation
Work-package leader:	Iacopo Carreras, UH
Deliverable owner:	Daniele Miorandi, UH
Quality Assessor:	Luc Moreau, SOTON

List of Contributors

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

Executive Summary

This report describes the version 2.0 of the SmartSociety platform, which 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 revised architectural view, plus a description of the platform runtime and of the software components integrated. A brief description of the validation and testing steps carried out by developing applications on top of the platform concludes the report.

Table of Contents

List of Acronyms

Acronym	Full Name	Description
CM	Context Manager	System component in charge of monitoring the context the agent represented on the platform by a peer is in.
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.
PS	Provenance Service	System component in charge of logging actions and information flows occurring on the platform.
RM	Reputation Manager	System component in charge of handling the reputation of any system resource, including peers.
SmartCom	Communication Middleware	System component in charge of managing communication channels between the platform and the peers.
TEM	Task Execution Manager	System component in charge of monitoring the execution of tasks and trigger corrective actions if needed.

1 Introduction

This report describes the version 2.0 of the SmartSociety platform. The corresponding software components can be accessed at the following integrated repository: `https://gitlab.com/smartsociety/`[3].

The version 2.0 builds upon the first version, which was delivered as D8.2 [1]. During the period M25-M30, the integration work continued along the roadmap included in D8.2 [2] leading to the integration of the provenance service, to the development and integration of the monitoring component, to the integration of the new version of the peer manager and to a refactoring of the runtime in order to align with the development of the programming model in WP7. Furthermore, some lab experiments were carried out to test and validate the platform functionalities and its ability to support the complete application life-cycle.

The report is organized as follows. Sec. 2 includes a revised description of the platform architecture. Sec. 3 describes the artifacts that constitute the platform prototype. Sec. 4 describes the validation and testing activities and their outcomes. Sec. 5 concludes the report providing a roadmap for activities between M30 and M48.

[3]A login/password has been created for reviewers: SmartSocietyReviewer/sm@rts0c13tyr3v13w3r

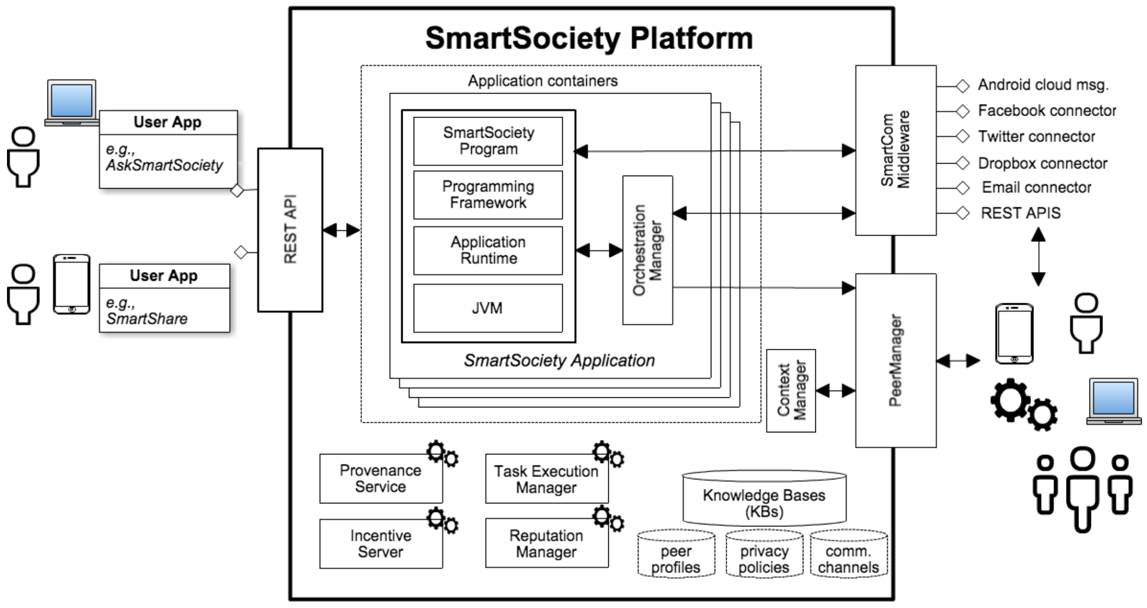

Figure 1: SmartSociety platform users and architecture.

2 Platform Architecture

This section describes the SmartSociety platform architecture at M30. In line with the iterative approach pursued by the Consortium, the description is a snapshot of the current status of project activities in terms of integration of technology and software modules developed, and will evolve during the remainder of project lifetime according to the roadmap described in Sec. 5.

A simplified, high-level view of the SmartSociety platform architecture is presented in Fig. 1 [3]. The rectangle boxes represent the key platform components. All components expose RESTful APIs. The deployment can be distributed. In order to ease inclusion of peers into the platform, interaction via popular commercial protocols and communication software is also supported.

User applications contact the platform through the REST API component. All incoming user requests are served by this module that dispatches them to the appropriate platform application, which will be processing and responding to them. The platform applications run sandboxed in Docker [4] containers. The applications can be deployed on different (virtual) machines. Applications request from the Peer Manager a set of permissions for accessing and manipulating personal private data of peers. The Peer Manager

[4]`https://www.docker.com/`

can shortly be described as the central information system of the platform, managing all peer and application information, and allowing privacy-aware access and sharing of the data among platform components. The platform application makes use of SmartSociety platform's programming libraries, allowing the developer to execute collective-based tasks on the platform. Each platform application features a dedicated *Orchestration Manager (OM)* component. The OM is the component in charge of preparing and orchestrating collaborative activities among peers. Performing these functionalities requires the OM to heavily use the Peer Manager and SmartCOM Middleware components.

A brief description of the functionality of the components integrated is reported here for the sake of completeness.

- **Application Runtime**: this is the component providing the support logic enabling peers to execute tasks. The runtime provides handlers for interacting with the various internal components provided by the SmartSociety platform. In the current version (see 3 for more details on this and on the impact of the programming framework) the Application Runtime provides the main endpoint for external applications to interact with the SmartSociety platform.

- **Orchestration Manager**: it provides two key functionalities: composition and negotiation [4]. Composition takes task requests as inputs and generates tasks by solving constraint satisfaction problems specific to each application, potentially interacting with the peer manager (so that peer with certain capabilities can be identified) and the reputation service (so that the reputation of the capable peers can be obtained). The 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. 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.

- **Peer Manager**: it is responsible for managing peer-related information [5]. This includes their profile, as well as any other information which will be used by the Orchestration Manager for identifying possible peers that can execute a given 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 is used by the Orchestration Manager to create compositions. Besides individual peers, the Peer Manager also manages collectives, which are groups of peers characterized by specific properties. The Peer Manager includes a set of mechanisms for ensuring the user's privacy [5].

- **Task Execution Manager:** The TEM takes as input a description of the task under execution and data about the actual status of peers (mediated by the PM) to understand the level of completion and to trigger, if needed, appropriate actions by the OM.
- **Context Manager**: it is responsible for monitoring the context the agent represented on the platform by a peer is in [6]. For an individual human agent, this includes, e.g., tracking the location and recognizing the activity the agent is currently involved in. This can be further used to trigger context adaptation in applications.
- **Reputation Manager**: it handles the reputation of any system resource, including peers [7]. It computes the reputation of a given peer based on feedback from other peers. Reputation information can be exposed directly to application users or used by the PM in the selection of suitable peers for a given task.
- **Provenance Service**: it logs descriptions of actions performed by platform components and peers, as well as information flows between them, according to the W3C PROV recommendation [8]. In particular, the provenance store is the component responsible for keeping a record of how the overall compositions are being created, executed and how the data managed by the platform is being transformed.
- **Monitoring and analysis 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 manages incentives and interventions that can be used to achieve higher quality results. Incentives can be used to stimulate the participation of a peer (or a collective) in the execution of a given task or can be used to define specific strategies for community engagement.

3 Components and APIs

At the moment the platform mainly consists of a runtime allowing a Smart Society application to manage the submission of 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, Provenance Service). The code is in a private repository at `https://gitlab.com/smartsociety/appruntime`.

The final version of the application runtime will be heavily dependent on the Programming Framework, whose model has been defined in [9]. The current prototype has been developed with the following aims:

- To identify what the runtime should expect from a generic application;
- To define a first high level API to be provided to application developers;
- To test the interaction of the platform with the different components.

A Smart Society Application has its own identifier, generated during the registration phase. Currently, the application code has to be written by a developer directly in Java; in the final version of the platform, the code will be generated dynamically through the programming framework.

3.1 Runtime

Each application is expected to run in its own process by the SmartSociety runtime. The application will be provided by runtime with a SmartCom and a Orchestrator instances. In order to interact with external entities (such as peers or user applications), the runtime exposes three kinds of resources:

- **POST:/task/:applicationId/** To submit tasks, the runtime will ask the application to create an *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 instance creation. The runtime associated to the instance (hence the task) creates an identifier that is returned as response for further usage.

- **GET:/task/:applicationId/** This resource is used to query the status of one or multiple tasks. Query parameters can be used to specify a filter on the tasks of interest. The status format for each task is just required to be a valid json node (either a text or more complex structures) and it is completely domain–dependent.

- **GET:/task/:applicationId/:taskId** Used to retrieve the status of a specific task. As for the previous point both query parameters and the response are domain–dependent.

- **POST:/message/: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, which is passed as a specific field in the message body, is used by the runtime to dispatch the message to the correct application instance. The content of the message is then handled by the task runner instance that changes its state according to the information received.

In v.2.0 of the platform, the application developer is expected to provide certain functionalities. This includes methods for:

- Creating a new task runner that will take care of setting up and carry out the task submitted to the application;

- Retrieve the active task runners according to some query parameters, this will be used by the query endpoint.

The task runner is the instance of an application-specific class, which implements a simple interface allowing the runtime to start the execution and to ask for its status.

3.2 Component Library

The component library allows the application development to be abstracted from the actual implementation of the components. Component wrappers provide a simple interface to the component integrated with the runtime, so that the developer is shielded from minor changes in components.

In this section we briefly describe the functionality provided by the component library.

3.2.1 SmartCom

The developer does not need to interact directly with SMARTCOM: a method is provided for sending a message to a given collective, the programmer is required to provide a specific handler for handling answers to the message. The runtime will take care of receiving answers through the specific REST endpoint and, by using the conversation ID, it will route the answer to the correct handler. The platform provides also an adapter for sending out Android notifications or for contacting peers trough a REST endpoint. Other supported adapters (Dropbox, Email, ...) can be added easily.

3.2.2 Orchestration Manager

When the application is launched an orchestration manager (OM) is also executed. Communication with the OM works trough a REST API (a generalized version of the one presented in [4]). The library supports the following requests:

- Submit a task request for composition;
- Retrieve a specific task request or task;
- Wait for a negotiable task;
- Accept a task on behalf of a peer;
- Perform a trivial negotiation with explicit agreement;
- Wait for an agreed task.

3.2.3 Peer Manager

The library allows the easy creation of a collective given a collection of peers. The identifier of the collective is the only information needed by SmartCom to carry out communications with all the relevant peers.

3.2.4 Provenance

The runtime provides easy methods for logging on the provenance store the generic (i..e, independent from the application domain) part of the provenance graph. This includes actions such as, e.g., creation of a task, creation of a collective etc. The domain-dependent part of the provenance graph shall be specified by the application developer.Some helper function is provided for binding specific data whose model cannot be known a priori. The communication with provenance store (happening through a REST API) is hidden to the developer by the library.

3.3 Monitoring Framework

The monitoring framework is responsible for the monitoring of the overall SmartSociety platform and components. It acts as a central collection point for any information which is considered important to ensure the proper functioning of the platform. More details are reported in App. A.

3.4 Expected evolution

In the future version of the platform each application and its runtime will run in separate containers. This will provide isolation among different applications and will support faster application deployment and portability across machines. A service for routing the requests from general endpoints to the right application runtime will be included.

The way the runtime will evolve and its interaction with applications are highly dependent on the outcome of the programming model and programming framework specification efforts currently ongoing within WP7.

In terms of integration of other components, the roadmap foresees to integrate in the upcoming months the context manager (which will interact strictly with the peer manager), the reputation maanager, the task execution manager and the incentive server.

4 Expected Usage and Examples

The final goal of validation activity is to test the ability of the platform to meet the high-level requirements described and analyzed in detail in [1]. In Table 1 we summarize the current status of the platform development in terms of ability to meet the requirements identified by the Consortium. As it can be seen, not all requirements have been met yet, in most cases as they refer to components which are still not fully completed within the respective workpackage. Yet, the platform in its current version already provides the minimum level of functionality required to handle a rather large range of social computations, as demonstrated by the available demo applications.

We now move to briefly describing the validation and testing activities carried out within the scope of Task T8.4 ('Lab experiments and platform validation'). These were based on the flexible AskSmartSociety! application developed in year-2 and presented in [2] and included the development of a number of small-scale prototypes aimed at testing the flexibility and extensibility of the platform. In particular, the following two results were achieved:

- **Image tagging application:** based on the AskSmartSociety! application, a simple image tagging application was carried out. By means of such an application participants can post an image through the Ask! mobile application and ask other participants to provide tags annotating the image content. Other participants can tag the image using the Reply! application or the SmartSociety twitter peer. Tags are collected in the AskSmartSociety! peer, where they are disambiguated (using a third-party semantic analysis engine), ranked and finally the most likely tags are reported back to the user. The development required minimal modifications of the Ask! and Reply! application, integration with a third-party service for entity extraction [5] and a third-party service for image storage [6].

- **Facebook Peer and AskSmartSociety! Integration:** we developed a Facebook peer, in the form of a Facebook application connected to the SmartSociety platform. The integration was carried out for the AskSmartSociety! application. In this case, as it was for the Twitter peer, questions asked through the Ask! application get posted on the Facebook app page. Users who subscribed to the application can

[5] https://dandelion.eu/

[6] http://imgur.com/

ID	Chapter	Description	Met?	Remarks
CR-1	Computational Requirements	The platform should be able to execute human-based computations	Yes	See SmartShare and AskSmartSociety! demo
CR-2	Computational Requirements	The platform should be able to execute machine-based computations	Yes	
CR-3	Computational Requirements	The platform should be able to support hybrid computations	Yes	See AskSmartSociety! demo
PP-1	Peer Profiles and Peer Profiling	Peers will be characterized by a static and a dynamic profile.	Partially	See [10, 11, 5], integration with context manager ongoing
PP-2	Peer Profiles and Peer Profiling	Peer profiles data storage	Partially	See [5] for integration with PPL
PP-3	Peer Profiles and Peer Profiling	Peers can have multiple profiles	Partially	See [11, 5]
PP-4	Peer Profiles and Peer Profiling	Platform will support the profiling of peers	Partially	Integration with reputation service completed, full profiling missing
PU-1	Platform Usability	The SmartSociety platform should be accessible through a set of open APIs	Partially	Final set of APIs will depend on the programming framework [9]
PU-2	Platform Usability	Ease for developers to create and manage applications	No	Depends on release of programming framework
PU-3	Platform Usability	Support application development and deployment life-cycle automatically	Partially	A first version of the runtime is included in v2.0
PU-4	Platform Usability	Management GUI	Yes	The monitoring component has been integrated in v2.0 and includes a management dashboard
HT-1	Platform Components and Interactions Heterogeneity	Number of different channels to interact with human peers	Yes	See [12] and AskSmartSociety! demo.
HT-2	Platform Components and Interactions Heterogeneity	Platform components will run on a variety of heterogeneous devices	No	Currently tested only on servers.
HT-3	Platform Components and Interactions Heterogeneity	SmartSociety applications will be accessible via a variety of devices, online via the web and on mobile devices	Yes	See SmartSmare and AskSmartSociety! demos.
HT-4	Platform Components and Interactions Heterogeneity	SmartSociety will support a diversity of user interfaces for user engagement and recruitment	Partially	See [13]
SEC-1	Privacy and Security	Access Control	Partially	See [5]
SEC-2	Privacy and Security	Trust and Reputation	Partially	Based on integration of the reputation service
SEC-3	Privacy and Security	Informed Consent	Partially	See [5]
SEC-4	Privacy and Security	Peer Defined Usage Control Policies	No	
SEC-5	Privacy and Security	Secure Collection and Storage	Yes	See [5]
SEC-6	Privacy and Security	Comprehensive explanations of security and privacy issues	No	
GOV-1	Governance	Platform should support the collection of data related to governance	Partially	Provenance service integrated in v.2.0
GOV-2	Governance	Platform should support the implementation of governance policies	No	
PR-1	Performance Requirements	Scalability	No	No scalability tests performed so far.

Table 1: Features of platform 2.0 against requirements.

reply to the question using the ‘comment’ Facebook functionality. Additionally, users can ‘like’ comments posted by other users, an information which may be used to rank answers. In the AskSmartSociety! peer, answers from the Facebook peer are collected and integrated with those coming from the Reply! mobile app and the Twitter live feed.

The work carried out to realize the two prototypes highlighted a number of issues in the current version of the platform, in particular in terms of functionality and ease of use of the APIs exposed by the Peer Manager; a new set of APIs was released by WP4 accordingly.

5 Conclusions

The vision of the SmartSociety platform is to become an 'IFTTT [7] for social computation', i.e., a platform that allows the easy integration of hybrid (human and machine) computational elements within the scope of a well-defined application workflow. While various steps have been taken in this direction, the overarching goal will require intense activities for the remainder of the project.

Version 2.0 of the SmartSociety platform integrates seven key components: peer manager, orchestration manager, communication middleware, provenance service, reputation service, monitoring and analysis service and application runtime. This configuration is sufficient for developing a number of different applications using various types of social computation. Some limitations in terms of the single components are expected to be overcome with refinements from the relevant technical WP.

In terms of the upcoming steps, four major components to be integrated are the context manager (which will interact strictly with the peer manager), the reputation maanager, the task execution manager and the incentive server. Major changes in the platform configuration will come from the delivery of the programming framework by WP7 (foreseen at M36), which will have a major impact on how platform functionality can be accessed by application developers. Interactions with the virtual gamified environment in WP9 may also lead to a refactoring of some of the features exposed by the platform. Another major release of the platform is foreseen at M36.

[7] `https://ifttt.com/`: IFTTT (IF This Then That) is a Web-based service allowing non-technical users to create chains of conditional statements (called 'IF recipes') and actions (called 'DO recipes') involving popular Web services.

References

[1] I. Carreras, "Platform requirements analysis and system design," SmartSociety FP7 Project Deliverable D8.1, Dec. 2013.

[2] I. Carreras, D. Miorandi, and T. Schiavinotto, "Platform prototype: early results and progress report," SmartSociety FP7 Project Deliverable D8.2, Dec. 2014.

[3] O. Scekic, D. Miorandi, T. Schiavinotto, D. I. Diochnos, A. Hume, R. Chenu-Abente, H.-L. Truong, M. Rovatsos, I. Carreras, S. Dustdar, and F. Giunchiglia, "SmartSociety – a platform for collaborative people-machine computation," in *Proc. of SOCA 2015*, Rome, IT, 2015.

[4] M. Rovatsos, "Static social orchestration: implementation and evaluation," SmartSociety FP7 Project Deliverable D6.2, Dec. 2014.

[5] A. H. Daniele Miorandi and R. Chenu, "Peer modeling and search," SmartSociety FP7 Project Deliverable D4.3, Jun. 2015.

[6] G. Kampis, "Models for human/machine symbiosis: final results," SmartSociety FP7 Project Deliverable D3.2, Oct. 2014.

[7] M. Rovatsos, "Computational models and validation," SmartSociety FP7 Project Deliverable D2.3, Jun. 2015.

[8] L. Moreau, "Provenance, trust, reputation and big data," SmartSociety FP7 Project Deliverable D2.2, Dec. 2014.

[9] O. Scekic and H.-L. Truong, "Programming models and languages," SmartSociety FP7 Project Deliverable D7.2, Jun. 2015.

[10] A. Hume, "Peers and micro-task profiling: early results and progress report," SmartSociety FP7 Project Deliverable D4.1, Dec. 2013.

[11] ——, "Peer search in smart societies," SmartSociety FP7 Project Deliverable D4.2, Dec. 2014.

[12] H.-L. Truong and O. Scekic, "Virtualization techniques and prototypes," SmartSociety FP7 Project Deliverable D7.1, Dec. 2014.

[13] M. Pompa, "Prototype version 1, testing scenarios and initial evaluation," SmartSociety FP7 Project Deliverable D9.3, Jun. 2015.

A Monitoring component

In the period M25-M30, the monitoring component was developed and integrated within the scope of WP8.
The monitoring framework is responsible for the monitoring of the overall SmartSociety platform and components. It acts as a central collection point for any information which is considered important to ensure the proper functioning of the platform. Target users of the monitoring components are system administrators who need to check the status of the system and the liveness of the services.

The monitoring framework allows to:

- Dynamically collect diverse information relevant for monitoring the proper functioning of the platform and its performance. Such information is permanently stored in a non-relational database and available at any time for inspecting specific platform behaviors.

- Visualize such information in real-time by means of interactive dashboards, or query the collected data via dedicated APIs.

The high-level architecture of the monitoring component is represented in 2.

The monitoring framework is based on Logstash[8], an open-source tool for managing events and logs. Logstash can be used to collect logs, parse them, and store them for later use (e.g., search and visualization). It is possible to define what information shall be permanently stored and processed by the monitoring framework. In particular, it is possible to integrate:

- System logs: these logs correspond to logs, which are generated by the various system components such as, e.g., web servers, application servers.

- Application logs: specific logs that are produced by applications, and require a constant integration for debugging and monitoring purposes.

[8] `http://logstash.net/`

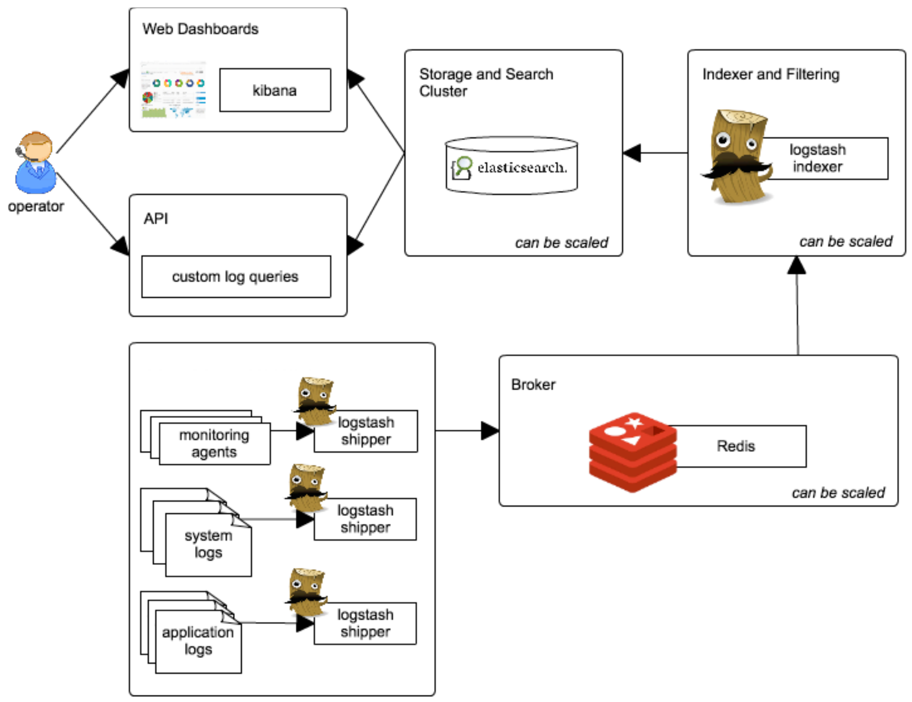

Figure 2: Monitoring framework architecture.

- Monitoring information: any agent that can be configured to deliver data to the Logstash infrastructure.

In all three cases, a Logstash shipper is used to connect the specific source of data to Logstash. Specific shippers already exists for some widely used system components such as, e.g., web servers, databases, etc., while custom shippers can be created for specific cases. In the case of SmartSociety, we created a dedicated shipper to collect the events produced by the various platform components.
The following component is a Redis Broker. This is an optional component that can be used in order to scale the system to large volumes of events and data. Based on Redis, data is indexed in order to prepare it for optimal searching and querying. Once the data is indexed, it is stored in an ElasticSearch cluster for storage and search. Starting from the data stored in ElasticSearch, it is possible to build queries on scale to explore the collected data. We decided to use Kibana[9] as the tool to create and visualize queries on the collected data. Kibana is fully integrated with ElasticSearch, and allows to easily explore and analyze large volumes of data. A sample visualization is shown in 3. The

[9] `https://www.elastic.co/products/kibana`

metrics and specific charts can be configured dynamically by the administrator of the platform.

In addition, ElasticSearch provides APIs for querying and extracting the data stored in the platform. This can be helpful in the case aggregated views such as, e.g., monthly reports, are needed.

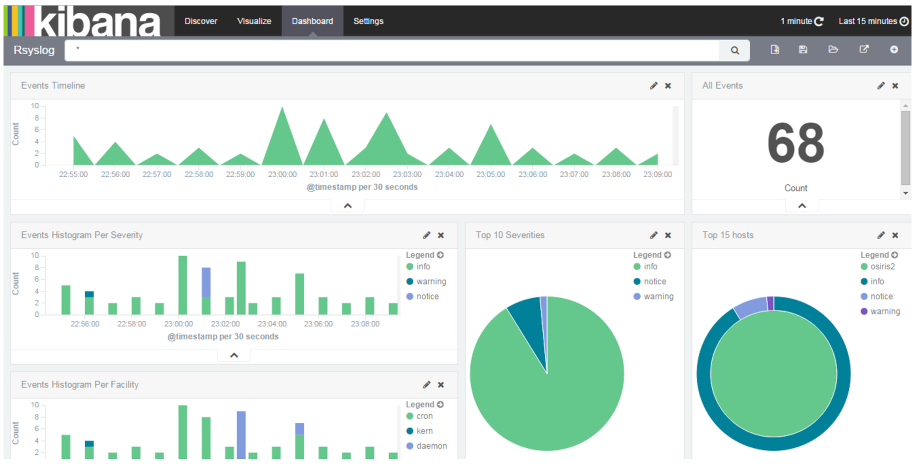

Figure 3: Example of the monitoring framework web interface.