Project Acronym:	SUPERSEDE
Project Title:	SUpporting evolution and adaptation of PERsonalized Software by Exploiting contextual Data and End-user feedback
Call identifier:	*H2020-ICT-2014-1*
Topic:	ICT-09-2014 Tools and Methods for Software Development
Type of Action:	Research and Innovation action
Grant agreement no.:	644018
Starting date:	1 May 2015
Ending date:	31 April 2018

D1.5: Comprehensive monitoring techniques, v2

WP1	Feedback and Data Collection
Tasks 1.3	Development of comprehensive monitoring techniques.
Due Date	30/04/2018
Submission Date:	30/04/2018
Deliverable Responsible:	UPC
Version:	1.0
Status:	Final
Author(s)	Marc Oriol, Jordi Marco, Quim Motger (UPC), Ronnie Schaniel (FHNW).
Reviewer(s):	Sergi Nadal (UPC), Angelo Susi (FBK).
Deliverable Type	DEM
Dissemination Level	PU

Version History:

Version	Author	Date	Description
0.1	Marc Oriol (UPC)	15/01/18	ToC + initial contributions
0.2	Marc Oriol (UPC)	23/02/18	Update on the structure and general document
0.3	Marc Oriol, Jordi Marco (UPC)	14/03/18	Update on sections 1 - 2
0.4	Marc Oriol (UPC)	10/04/18	Update on section 3
0.5	Marc Oriol (UPC), Quim Motger (UPC), Ronnie Schaniel (FHNW)	11/04/18	Update on section 4 and section 5.
0.6	Marc Oriol (UPC)	13/04/18	Update on last sections and final deliverable
0.7	Sergi Nadal (UPC), Angelo Susi (FBK)	18/04/18	Internal reviews
1.0	Marc Oriol (UPC)	27/04/18	Final version

Table of Contents

Acronyms

API	Application program interface
CPU	Central processing unit
DBMS	Database-management system
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
QoS	Quality of Service
RAM	Random Access Memory
RESTful	Representational state transfer
SOAP	Simple Object Access Protocol
SSH	Secure Shell

1 Executive Summary

The deliverable D1.5 “Comprehensive monitoring techniques, v2” is the second deliverable of Task T1.3 “Development of comprehensive monitoring techniques”, and provides the results of the activities conducted in this task at M36. The key artefact of D1.5 is the final version of the monitoring system and its constituent monitoring tools.

The implemented monitoring system and its monitoring tools provides the mechanisms to gather, at runtime, data from several sources and from different nature, such as Quality of Service (QoS), user events, logs, infrastructure related metrics (e.g. CPU, disk consumption, memory consumption), messages from social media (e.g. tweets) or marketPlaces (e.g. appStore, google Play). The monitoring system has been integrated with the Feedback mechanisms developed in T1.2. The result of such integration is a unified framework that combines both monitoring and feedback gathering tools, named FAME.

Although the monitoring system is fully integrated with the SUPERESEDE framework, it can also be used as a standalone artifact as it has been implemented following a loosely-coupled Service-Oriented Architecture.

The monitoring system has been used in all three SUPERSEDE use cases.

This deliverable reports on the main innovations of the monitoring system, the architecture and the monitoring tools implemented, an installation guide and a user manual, as well as the licensing information.

2 Introduction

2.1 About this deliverable

Deliverable D1.5 is the second and final deliverable of Task T1.3. This deliverable provides the results at month 36 of the monitoring system of the SUPERSEDE project.

This document provides an overview of the delivered monitoring system, the overall architecture and the technical details of each monitoring tool implemented. This deliverable provides also the details and artifacts to install and use the monitoring system.

2.2 Document structure

This document is structured as follows: Section 1 provides the Executive Summary. Section 2 gives a brief introduction of the deliverable and introduces its main innovations. Section 3 provides the details regarding Innovations, including the scientific foundations and functional description. Section 4 describes the technical details, including the architecture and the components description. Section 5 provides the information on how to install and use the software. Finally, section 6 provides the conclusions, and section 7 the references.

2.3 Main innovations

This is the final software release of the monitoring system, whose main functionality is to obtain runtime data from different sources and of different nature in real time for the subsequent analysis in the SUPERSEDE loop. The main innovations that the presented monitoring system contains are:

- **Combining explicit feedback and monitoring:** At M18 we provided the unified framework that combines both monitoring and feedback gathering tools. From M18 until M36 we have validated such combination in one of the use cases, where the feedback provided by the end-users were combined with the actions (i.e. events) generated.
- **Heterogeneous extensible distributed monitoring system**: At M18 we provided an initial set of monitoring tools, namely, for Social Networks and MarketPlaces. We have extended the monitoring system with more monitoring tools, to gather QoS (e.g. Response Time, availability), infrastructure related metrics (e.g. CPU, RAM,...), user events and logs.
- **Reconfigurable monitoring system with different reconfiguration capabilities per each type of monitor:** At M18 we provided a monitoring system capable of being reconfigured at runtime with different reconfiguration capabilities. We have extended the reconfiguration capabilities to the new monitors implemented.

3 Innovations

3.1 Scientific foundations

The scientific foundations of the proposed monitoring system have resulted in several publications in international conferences and journals. Below we describe the list of publications that resulted from the scientific advances of the presented monitoring system.

- M. Oriol, M. Stade, F. Fotrousi, S. Nadal, J. Varga, N. Seyff, A. Abello, X. Franch, J. Marco, O. Schmidt (submitted to RE18). “FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring”.

- O. Cabrera, X. Franch, J. Marco. “Ontology-based context modeling in service-oriented computing: A systematic mapping”. Data & Knowledge Engineering, Vol. 110, pp. 24 - 53. 2017.
- O. Cabrera, X. Franch, J. Marco. “3LConOnt: a three-level ontology for context modelling in context-aware computing”. Software & Systems Modeling, pp 1–34. 2017
- E. C. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini, M. Stade. “The Crowd in Requirements Engineering: The Landscape and Challenges”. IEEE Software 34(2): pp. 44-52. 2017
- N. Seyff, M. Stade, F. Fotrousi, M. Glinz, E. Guzman, M. Huber, D. Muñante, M. Oriol, R. Schaniel. “End-user Driven Feedback Prioritization”. First Workshop on Requirements Prioritisation and Enactment (PRIORE 2017), at REFSQ 2017, pp. 1-7. 2017
- M. Stade, M. Oriol, O. Cabrera, F. Fotrousi, R. Schaniel, N. Seyff, O. Schmidt. “Providing a User Forum is not enough: First Experiences of a Software Company with CrowdRE”. 25th IEEE International Requirements Engineering Conference (RE’17) pp. 164-169. 2017.

3.2 Functional description

The main functionality of the monitoring system is to effectively monitor different metrics and events within, or related to, the software systems supervised by the SUPERSEDE framework. The motivation of the monitoring system is to provide SUPERSEDE users with an extensible system to manage multiple monitors easily, providing also a common API that allows the automatization of the management of the different monitors, which is a critical aspect to support the automatic reconfigurations performed in the later phases of the SUPERSEDE lifecycle loop [D4.9].

Most of the monitors integrated in the monitoring system are not developed from scratch but rather exploit existing tools and wrap them in a common RESTful interface to be managed using a common framework. In this sense, the monitoring system is composed by a set of monitors implemented as RESTful web services, which provide an API with the necessary RESTful methods to start a new monitoring process, reconfigure it or stop it, specifying the necessary parameters. Each of these monitors uses third-party components - libraries or APIs - as tools to perform the monitoring. In this regard, the monitoring system may include more than one monitoring tool to monitor the same element.

At the M18 the types of monitors being supported were for social networks and market places, aiming at obtaining comments and ratings of users of the system under analysis, also known as explicit indirect feedback [HKV08].

- **Social Network monitors:** these types of monitors aim at collecting data that users give in social networks (e.g. opinions, feedback, etc.). In particular, we support the monitoring of Twitter (i.e. tweets for a specific software system).
- **Market Place monitors:** these types of monitors aim at collecting reviews (i.e. comments and ratings) that users give in Market Places for the apps they have in their device. In particular, we support the monitoring of both Google Play and Apple’s App Store.

At M36, we implemented several new monitors in order to support (more) dynamic adaptation scenarios (see [D4.6]) and reconfiguration scenarios (see [D4.9]). With these new monitors, we provided the capability to gather QoS metrics (e.g. response time, availability), user events, logs and infrastructure related metrics (e.g. CPU, disk consumption, memory consumption).

- **QoS monitors:** these types of monitors aim at collecting QoS data (e.g. response time, availability). In this regard, we have implemented an HTTP Monitor able to obtain the response time and response code of HTTP-based technologies. In this regard, it can be used to monitor web pages, RESTful services, SOAP services, etc.
- **User events monitors**: these types of monitors aim at collecting the events generated by end-users in an application. In this regard we have implemented a monitor to collect events in HTML-based applications. The monitor is able to collect clickstreams and the navigation path
- **Infrastructure:** these types of monitors aim at measuring metrics related to the infrastructure of the servers where an application is deployed. We have implemented a monitor that collects the disk-consumption.
- **Logs**: these types of monitors aim at collecting the logs. The monitor we have implemented obtain the logs of java applications from log4J

Table 1 summarizes the list of monitors supported at M18 and M36 with the monitoring tools provided and the monitored metrics .

Table 1. *Implemented monitors at M18 with the monitoring tools that obtains the data*

	Monitoring type	Monitor	Monitoring tool(s) used	Metric(s) monitored
M18	Social Networks	Twitter	• Twitter Streaming API	tweets
M18	Market Places	Google Play	• Google Play Developer API • AppTweak	comments and ratings
M18	Market Places	Apple's App Store	• iTunes API • AppTweak	comments and ratings
M36	QoS	HTTP Monitor	• *from scratch*	Response time and Response code
M36	User events	HTML Monitor	• *from scratch*	Clickstreams and navigation path
M36	Infrastructure	Disk Monitor	• *from scratch*	Disk consumption

M36	Logs	Logs Monitor	• log4J	logged events.

Each implemented monitor offers the capability of performing concurrent monitoring processes, with different configuration parameters and using different tools to perform the collection of data. In this regard, it is possible to run multiple instances of the same monitor with different configurations. The monitors can be easily configured and reconfigured at runtime by invoking its RESTful services using a common JSON structure.

For each of these monitoring processes, once it has been configured and initialized, data is collected at specific time intervals - specified in the input configuration - and it is sent to Apache Kafka (the unique entry point for data in WP2) for subsequent analysis.

All the implemented monitors include a list of common parameters for their configuration (see Table 2).

Table 2. *Common parameters for all monitors*

Parameter name	Mandatory /optional	Parameter description
toolName	Mandatory	Name of the monitoring tool to be used. When a monitor has more than one monitoring tool available, this parameter allows to select the one that the user prefers.
kafkaTopic	Mandatory	An identifier used by Apache Kafka to identify the messages it receives. In this regard, each running monitoring instance have a unique kafkaTopic.
kafkaEndpoint	Optional	The endpoint of the Apache Kafka where the monitoring data should be sent. If no kafkaEndpoint is set, the default Apache Kafka endPoint will be used. This parameter allows to send the monitoring data to a different Apache Kafka is needed.
timeSlot	Mandatory	The time slot in which the monitor should send the collected data to Apache Kafka.

Besides the common parameters, some of the presented monitors have their own specific parameters (see Table 3).

Table 3. *Specific parameters for each monitor*

Monitor	Parameter name	Mandatory /Optional	Parameter Description
Twitter	keywordExpression	Mandatory	Search String to gather the tweets. The keyword expression allows the usage of different operators like AND, OR, parentheses, etc.
	accounts	Optional	To limit the search of tweets to a specific list of twitter accounts
Google Play	packageName	Mandatory	The identifier of the Application to monitor.
Apple's App Store	appId	Mandatory	The identifier of the Application to monitor.
HTTP Monitor	url	Mandatory	The URL of the web service or web page to monitor.
	method	Mandatory	The method to invoke the url (e.g. GET, POST)
	body	Optional	If the selected method is POST, this parameter includes the body of the message to be sent to the URL.
Disk Monitor	host	Optional	If the disk to monitor is not localhost, the host identifies the IP/address of the server to monitor.
	user	Optional	If the disk to monitor is not localhost, the user is the username to be used when logging to the external server (this requires to have installed SSH credentials).
	instructions	Mandatory	The linux commands to be used to monitor the disk (e.g. du, df, etc.)

4 Implementation

This section reports about the technical specifications of the monitoring system. It introduces how it fits into the overall SUPERSEDE solution, it presents the architecture and provides technical details of each component.

4.1 Fitting into overall SUPERSEDE solution

Figure 1 presents how the monitoring system fits into the overall SUPERSEDE functional perspective. Monitoring, in conjunction with feedback gathering, is the first activity carried

out in the SUPERSEDE lifecycle loop. Monitoring is responsible to obtain, at runtime, real time data related to the target system for the subsequent analysis in WP2.

The integration of this tool in SUPERSEDE is twofold. On the one hand, the monitors have been integrated with the systems using SUPERSEDE (i.e. the systems under analysis that require adaptation and evolution); and on the other hand, it has been integrated with the component of SUPERSEDE.

- Integration with the systems under analysis: The monitors have been installed within, or have access to the required data concerning the target system. The integration requirements are different depending on the type of monitor to be installed. For instance, to monitor infrastructure metrics of the system (e.g. CPU, memory, etc.), the monitoring tools need to have access to such infrastructure. By contrast, to monitor the tweets related to a system, there is no need of integration with the target system. The integration requirements for the monitors that require additional integration activities are described in Table 4.
- Integration with other SUPERSEDE components: The collected monitored data is sent to the Analysis in WP2 through a publish-subscribe messaging system. Moreover, the monitors offer also self-adaptation capabilities. To accomplish such self-adaptation, it uses the same SUPERSEDE framework. In this regard, the monitors are connected to WP4 components to enact the adaptations.

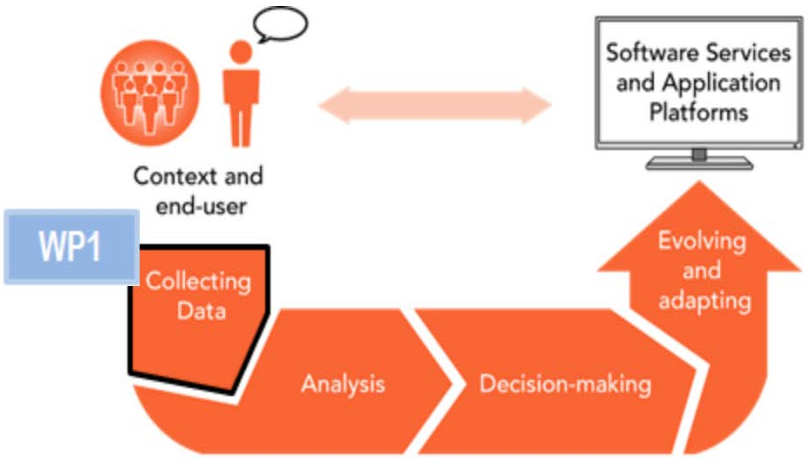

***Figure 1**. SUPERSEDE lifecycle.*

***Table 4.** Integration requirements for the monitors that require additional integration activities.*

Monitor	Integration requirements
HTML Monitor	A javascript snippet needs to be installed for each web page being monitored.
Disk Monitor	The Disk Monitor needs to be installed in the same server it is monitoring or have access to the server through SSH.
Logs Monitor	A log4J custom appender needs to be installed to the target application.

4.2 Prototype architecture

The monitoring and feedback gathering unified architecture is depicted in Figure 2 and is intended to support both the tasks of monitoring and feedback gathering of the SUPERSEDE lifecycle loop. In the Figure 2, we have highlighted the components that are relevant for monitoring. Feedback gathering components are detailed in [D1.3].

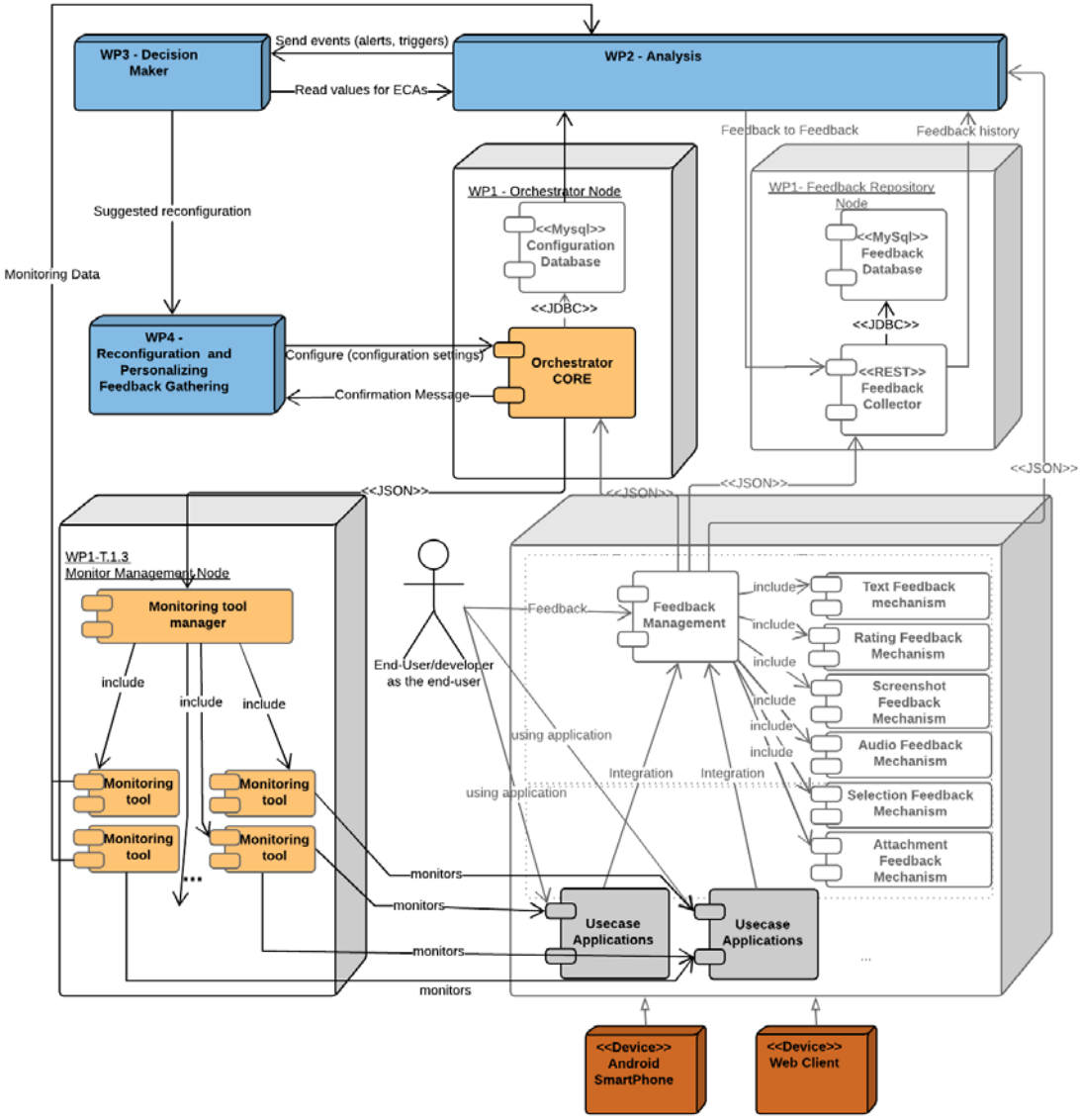

Figure 2. *Monitoring general architecture*

The architecture provides the capabilities of replacing, adding, re-configuring and adapting monitoring tools by adopting a decoupled design based on services that can be orchestrated. In the next section we describe each component in detail.

4.2.1 Components description

The components of the monitoring and feedback gathering architecture depicted in Figure 2 are described below:

Monitoring and feedback gathering orchestrator. This component is responsible for orchestrating the group of monitors and feedback gathering tools integrated in the architecture. For this purpose, the orchestrator makes available a RESTful API (the Orchestrator CORE) from which a user can register and integrate monitors and feedback gathering mechanisms into the architecture. The orchestration can execute actions over the monitors and mechanisms that have been integrated into the architecture, such as assigning, changing, configuring and reconfiguring the monitors.

Monitoring tool manager. This component is a broker responsible for dispatching a call at run time to a specific monitoring tool that is able to fulfill the required monitoring task. This process starts when the orchestrator receives a request from the SUPERSEDE user for monitoring a specific characteristic related to a service or application. Then, the orchestrator dispatches this monitoring request to the monitoring tool manager which in turn dispatches a monitoring request to the specific monitoring tool that can retrieve the information required.

Monitoring tools. This element in the architecture represents the monitoring components that have been registered by means of the orchestrator component and are responsible to gather the defined metrics.

4.2.2 Technical specifications

Here we describe the technical specifications of the components of the architecture related to monitoring. That is, the orchestrator, the monitoring tool manager and the monitoring tools. The programming language used for all these components is Java 8, and they run under Apache Tomcat 8.5.

The most important libraries of the orchestrator and the Monitoring tool manager are:

- **Google gson**: A library for serialization/deserialization of json files (v2.7)
- **Google guice**: A dependency injection framework for managing dependencies (v4.1)
- **guice-servlet**: An extension for google guice for the configuration of java servlet (v4.1)
- **Apache httpclient**: HttpClient for testing of the API (v4.5.2)
- **mockito-all**: Library for mocking of classes in unit testing (v1.9.5)
- **Apache commons-fileupload**: Library for facilitating uploading/downloading of files (v1.3.2)
- **JUnit 4:** Java unit test framework (v4.12)

The most important libraries of the monitors are:

- **Org-Json:** A library for serialization/deserialization of json files (v1.5)
- **Oracle Jersey framework:** A set of libraries used for implementing the RESTful APIs (v2.0)
- **Apache kafka:** Used for communication with kafka endpoints (v0.9)
- **Junit 4:** Java unit test framework (v4.2)

A summary of the technical aspects for the Orchestrator and Monitoring manager is shown in Table 5. Regarding the monitoring tools, as they differ depending on the type of monitor, we show in Table 6-11 the technical aspects of the particular monitors.

Table 5. *Technical specification of the Orchestrator and Monitoring Manager*

Programming language	Java 8
Libraries used	Apache Tomcat v7 library, Apache HttpClient 4.5, Apache HttpCore 4.4, Apache commons IO 2.5, Apache commons lang 3.4, apache commons logging 1.2, mysql connector 5.1, Google Guice 4.1, Google Guice Servlets 4.1, Google reflections 0.9, Google gson 2.6, Junit 4, Mockito 1.9
DBMS	MySQL Distrib 5.5.41
Application server	Tomcat 7
Type of web service	RESTful web service
Other	-

Table 6*. Technical specification of the Social Networks Monitors*

Programming language	Java 8
Libraries used	Apache Tomcat v7 library, Junit 4, Javax WS API 2.0, Json, Apache Kafka 0.9, Oracle Jersey 2.0, Apache Log4j 1.2, Twitter4j 4.0
DBMS	-
Application server	Tomcat 7
Type of web service	RESTful web service
Other	-

Table 7. *Technical specification of the Market Places Monitors*

Programming language	Java 8
Libraries used	Apache Tomcat v7 library, Junit 4, Javax WS API 2.0, Json, Apache Kafka 0.9, Oracle Jersey 2.0, Apache Log4j 1.2, Google Protobuf 2.4
DBMS	-
Application server	Tomcat 7
Type of web service	RESTful web service
Other	-

Table 8. *Technical specification of the QoS Monitors*

Programming language	Java 8
Libraries used	Apache Tomcat v7 library, Junit 4, Javax WS API 2.0, Json, Apache Kafka 0.9, Oracle Jersey 2.0, Apache Log4j 1.2, Apache commons httpclient 3.1
DBMS	-
Application server	Tomcat 7
Type of web service	RESTful web service
Other	-

Table 9. *Technical specification of the User events Monitors*

Programming language	Java 8, javascript
Libraries used	Apache Tomcat v7 library, Junit 4, Javax WS API 2.0, Json, Apache Kafka 0.9, Oracle Jersey 2.0, Apache Log4j 1.2, javax.jms 1.1, javax servlet 3.0, Apache geronimo 1.1, andes-client 3.1.1, wso2 Carbon 4.4.1, wso2 securevault 1.0.
DBMS	-
Application server	Tomcat 7
Type of web service	RESTful web service
Other	The monitor includes a server side and a client side. The server side is implemented in Java as a RESTful service running under Tomcat. The client side is implemented in Javascript and must be imported in the web pages being monitored.

Table 10. *Technical specification of the Infrastructure Monitors*

Programming language	Java 8
Libraries used	Apache Tomcat v7 library, Junit 4, Javax WS API 2.0, Json, Apache Kafka 0.9, Oracle Jersey 2.0, Apache Log4j 1.2, Apache commons httpclient 3.1, jsch 0.1.54
DBMS	-
Application server	Tomcat 7
Type of web service	RESTful web service
Other	The monitor can be deployed either remotely or locally with respect to the server to be monitored. When deployed remotely, it uses SSH to connect to the server to monitor.

Table 11. *Technical specification of the Log Monitors*

Programming language	Java 8
Libraries used	Apache Tomcat v7 library, Junit 4, Javax WS API 2.0, Json, Apache Kafka 0.9, Oracle Jersey 2.0, Apache Log4j 1.2, Spring Boot Web Starter
DBMS	-
Application server	Tomcat 7
Type of web service	RESTful web service
Other	The monitor includes a server side and a client side. The server side is implemented in Java as a RESTful service running under Tomcat. The client side is implemented in Java and must be imported in java applications that requires monitoring.

5 Delivery and usage

5.1 Package information

The delivered package contains the following structure of folders and files relevant for monitoring:

orchestrator: Contains the RESTful service and required libraries to manage the monitors and feedback gathering tools. The structure is as follows:

- root directory
  - deployment - sql dumps for deployment
  - migrations - database migrations
  - src - source code - servlet class and guice modules
    - main - source code of the orchestrator
    - test - API- and unit-tests

monitormanager: Contains the RESTful service and required libraries to manage the monitors. The structure is as follows:

- root directory
  - src - source code
    - model - *data and parameter model files*
    - services - *REST API implementation*
  - test - *monitor testing java files*

monitoring: Contains classes and functions that are common and shared for all the monitors. The structure of this package is as follows:

- root directory
  - src - *source code*
    - main - *contains main source code*
      - model - *generic data and parameter model files*
      - kafka - *communication with Apache Kafka*
      - controller- *generic controller of the monitor*

monitors: Contains the implementation of the monitors. Each monitor has been implemented by applying the same architecture. The following example shows the structure of folders and files:

- MonitorName - *monitor project*
  - src - *source code*
    - main - *contains main source code*
      - monitoring.model - *data and parameter model files*
      - monitoring.tools - *tools implementation*
      - monitoring.controller- *controller of the monitor*
    - test - *monitor testing java files*

The package includes other folders that are relevant for feedback gathering but not for monitoring. Details about these other folders are described in [D1.3].

5.2 Installation instructions

In this section we provide the installation instructions of the components required for the monitoring system. They are the Orchestrator together with the Monitoring Manager and the list of monitors needed. To install the whole unified framework with the feedback gathering tools, it is required to install the feedback gathering framework, installation instructions at [D1.3].

- **Orchestrator**
  To install and start up the orchestrator server please check the deployment instructions at https://github.com/supersede-project/monitor_feedback/tree/master/orchestrator.

- **Monitors**
  To install and start up the monitors please check the deployment instructions at https://github.com/supersede-project/monitor_feedback/tree/master/monitors.

5.3 User Manual

The prototype provides a RESTful API in the Orchestrator to configure and manage the list of available monitors. Also, the monitors can be used standalone as they also provide a RESTFul API. It is important to clarify that different types of monitor have different parameters to be configured, and therefore the methods provided in the API are flexible in order to accept different types of parameters according to the monitors being configured. The manual describing the API and the parameters for the configuration of the monitors are described in the links provided in the README.MD on the github of each of the monitors.

5.4 Licensing information

The prototype and its components are released as Open Source Software under the license Apache License 2.0. Licensing information for the third party subcomponents, libraries and services used are listed in Table 12.

Table 12. *Licensing information for the monitoring prototype subcomponents.*

Component	Technology	Third party Subcomponent	Subcomponent License
Orchestrator and Repository	Java	Apache HttpClient 4.5	Apache License 2 (ASL2)
		Apache HttpCore 4.4	Apache License 2 (ASL2)
		Apache commons IO 2.5	Apache License 2 (ASL2)
		Apache commons fileupload 1.3	Apache License 2 (ASL2)
		Apache commons lang 3.4	Apache License 2 (ASL2)
		Apache commons logging 1.2	Apache License 2 (ASL2)
		mysql connector 5.1	General Public License (GPL2)

		Google Guice 4.1	Apache License 2 (ASL2)
		Google Guice Servlets 4.1	Apache License 2 (ASL2)
		Google reflections 0.9.9	Berkeley Software Distribution 4 (BSD4)
		Google gson 2.6	Apache License 2 (ASL2)
		JUnit 4	Common Public License (CPL) / Eclipse Public License (EPL)
		Mockito 1.9	Massachusetts Institute of Technology (MIT) license
Monitor (common libraries needed for all monitors)	Java	javax.ws.rs API	Common Development and Distribution License (CDDL)
		Json	Massachusetts Institute of Technology (MIT) license
		Apache Kafka	Apache License 2 (ASL2)
		JAX RS Provider For JSON Content Type	GNU Lesser General Public License 2.1 (LGPL 2.1)
		Jersey Container Servlet Core	Common Development and Distribution License (CDDL)
		Jersey Core Common	Common Development and Distribution License (CDDL)
TwitterAPI	Java	Twitter4j	Apache License 2 (ASL2)
		log4j	Apache License 2 (ASL2)
Google Play	Java	Protobuf	Apache License 2 (ASL2)
		Google Play Developer API	Others
		AppTweak API	Others

Apple's App Store	Java	iTunes API	Others
		AppTweak API	Others
HTTP Monitor	Java	Apache commons httpclient 3.1	Apache License 2 (ASL2)
HTML Monitor	Java + Javascript	javax.jms 1.1	Common Development and Distribution License (CDDL)
		javax servlet 3.0	Common Development and Distribution License (CDDL)
		Apache geronimo 1.1	Apache License 2 (ASL2)
		andes-client 3.1.1	Apache License 2 (ASL2)
		wso2 Carbon 4.4.1	Apache License 2 (ASL2)
		wso2 securevault 1.0	Apache License 2 (ASL2)
Disk Monitor	Java	Apache commons httpclient 3.1	Apache License 2 (ASL2)
		JSCH 0.1.54	Berkeley Software Distribution (BSD)
Logs Monitor	Java	Spring Boot Web Starter	Apache License 2 (ASL2)

5.5 Download

The software can be downloaded from the github repository. The path for the unified framework that includes the orchestrator, the monitors and the feedback gathering tools is, as specified in the installation instructions: https://github.com/supersede-project/monitor_feedback

To download specific monitors as standalone services without the unified framework, the reader can go directly to:
https://github.com/supersede-project/monitor_feedback/tree/master/monitors/

6 Conclusions

In this document we provided a technical description of the final SUPERSEDE Monitoring system released at M36. The source code of the Monitoring system and this report are the two complementary parts of the deliverable D1.5, which is the final deliverable of task T1.3.

We described here the functional and design specification of the monitoring system, which enables end-users to monitor data of the targeted systems from different sources. Specifically, the implemented monitors are for Twitter, Google Play, Apple's App Store, Quality of Service, infrastructure, user events, and system's logs.

In this report we also highlighted the main innovations achieved with the activities conducted in T1.3, such as *combined explicit-feedback and monitoring* mechanisms that support *reconfigurable monitoring system including different reconfiguration capabilities* and an *heterogeneous extensible distributed monitoring* system.

7 References

[D1.3] M. Stade, R. Schaniel, N. Seyff, M. Oriol, D. Muñante. D1.3: Direct multi-modal feedback gathering techniques, v2. SUPERSEDE Project Public Deliverable. 2018

[D4.6] J. Gorroñogoitia, M. Oriol, Q. Motger, S. Stevanetic: D4.6 Methods and tools to enact software adaptation and personalization v3. SUPERSEDE Project Public Deliverable. 2018

[D4.9] D. Muñante, M. Oriol., A. Perini, R. Schaniel, J. Gorroñogoitia. D4.9: Feedback-gathering and monitoring reconfiguration techniques, v3. SUPERSEDE Project Public Deliverable. 2018

[HKV08] Y. Hu, Y. Koren, C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets" *8th IEEE International Conference on Data Mining*, 2008.