

symphony

Project full title: *Orchestrating Information Technologies and Global Systems Science for Policy Design and Regulation of a Resilient and Sustainable Global Economy*

Contract no.: 611875

D5.1

Definition of System Architecture

Workpackage:	5	
Editor:	Anna Triantafillou	ATC
Author(s):	Nikos Dimakopoulos	ATC
	Anna Triantafillou	ATC
	Panagiotis Kokkinakis	ATC
	Kostas Giannakakis	ATC
	Richard Mark Brown	ATC
	Miha Papler	JSI
	Luis Rei	JSI
	Roni Bulent Ozel	UJI
	Efthimios Bothos	ICCS
	Lex Robinson	PLAYGEN
	Linda Ponta	UNIGE
Authorized by	Silvano Cincotti	UNIGE
Doc Ref:	D5.1	
Reviewer	Inna Novalija, Miha Papler	JSI
Dissemination Level	PU	

SYMPHONY Consortium

No	Name	Short name	Country
1	UNIVERSITA DEGLI STUDI DI GENOVA	UNIGE	Italy
2	INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS	ICCS	Greece
3	PlayGen Ltd	PlayGen	United Kingdom
4	GCF - GLOBAL CLIMATE FORUM EV	GCF	Germany
5	Athens Technology Center S.A	ATC	Greece
6	UNIVERSITA POLITECNICA DELLE MARCHE	UNIVPM	Italy
7	INSTITUT JOZEF STEFAN	JSI	Slovenia
8	Germanwatch Nord-Sued-Initiative e.V.	GW	Germany
9	UNIVERSITAT JAUME I DE CASTELLON	UJI	Spain

Document History

Version	Date	Changes	Author/Affiliation
V0.1	5/05/2014	First draft with ToC	ATC
V0.2	9/06/2014	Second draft with initial input from partners	ATC
V0.3	16/07/2014	Third draft including 1 st draft of Use Cases	ATC
V0.3	25/07/2014	Forth draft including 2 nd draft of Use cases and sequence diagrams	ATC
V0.4	08/08/2014	Fifth draft including changes based on partners feedback	ATC
V0.5	15/09/2014	Sixth version including revised input of partners	ATC
V0.6	26/09/2014	Review of compiled version	JSI
V1.0	30/09/2014	Accepted and addressed comments by JSI and Technical partners	ATC

Executive Summary

SYMPHONY is a diverse project with several work packages working in parallel in order to achieve common objectives. In order to start this work with a mutual sense of direction, we needed a high-level blueprint of the overall technology development within the project. The role of this document is to be a common point of reference throughout the project.

The structure of this document is based on best practices for architecture descriptions, showing complementing views with different abstraction levels. The scenarios and requirements contained within these views have been collectively defined and revised through several iterations in this first project phase.

The document presents a coarse-grained collection of technical components that provide the necessary functionality and defines the principles for their integration in a manner that ensures completeness, safety and efficiency. To this end, we propose a Service-oriented approach, where each functional unit is implemented as a stand-alone service, communicating in a standardized manner with other components. In order to preserve the openness of the architectural design, the SYMPHONY system will use an orchestration mechanism, i.e. the distinct services will communicate and be synchronised via a central component the SYMPHONY Orchestrator. This decision allows the replacement and removal of services without significant overhead, as the new service will have to satisfy just the communication aspects with the SYMPHONY Orchestrator and not with every other component.

We consider the architecture to be a live document that will evolve and be aligned with the ongoing work in the project. An updated version of this deliverable will be delivered in M33 together with the final version of the SYMPHONY prototype.

TABLE OF CONTENTS

EXECUTIVE SUMMARY	3
1 INTRODUCTION	7
1.1 PURPOSE AND SCOPE	7
1.2 METHODOLOGY AND STRUCTURE OF THE DELIVERABLE.....	7
1.2.1 <i>Architectural Design Principles</i>	7
1.2.2 <i>Structure of the Deliverable</i>	8
2 SYSTEM USE CASES.....	9
2.1 OVERVIEW OF USER REQUIREMENTS.....	9
2.2 THE SYMPHONY “ACTORS”	10
2.3 THE SYMPHONY “USE CASES”	11
2.3.1 <i>Overview of the Use Cases for the Simple User</i>	11
2.3.2 <i>Overview of the Use Cases for the Advanced User</i>	12
2.3.3 <i>Overview of the Use Cases for System Administrator</i>	16
2.3.4 <i>List of SYMPHONY Use Cases</i>	18
2.4 SYMPHONY FUNCTIONAL REQUIREMENT	18
2.4.1 <i>Social Media Mining</i>	19
2.4.2 <i>Information Market</i>	19
2.4.3 <i>Agent Based Model</i>	20
2.4.4 <i>Gamification</i>	20
2.4.5 <i>User Management</i>	20
2.5 SYMPHONY NON-FUNCTIONAL REQUIREMENTS	20
3 SYMPHONY SYSTEM ARCHITECTURE	22
3.1 HIGH LEVEL OVERVIEW	22
3.2 SYMPHONY ARCHITECTURE DESIGN	22
3.3 SYMPHONY COMPONENTS.....	27
3.3.1 <i>SYMPHONY Front-End</i>	27
3.3.2 <i>SYMPHONY Back-End</i>	30
3.4 DATA COMMUNICATION.....	44
3.4.1 <i>Dashboard’s Communication</i>	44
3.4.2 <i>Gamification Engine’s Communication</i>	45
3.5 SYSTEM INTERACTIONS & INFORMATION FLOW.....	46
3.5.1 <i>Accessing the platform</i>	46
3.5.2 <i>Viewing Information</i>	47
3.5.3 <i>Acquiring Estimations</i>	50
4 DATA DESCRIPTION	53
4.1 SYMPHONY DASHBOARD	53
4.2 GAMIFICATION ENGINE	55
4.3 AGENT BASED MODEL	56
4.4 INFORMATION MARKETS	58
4.5 SOCIAL MEDIA MINING	58

5	IMPLEMENTATION PLAN	61
5.1	UI DESIGN PRINCIPLES	61
5.1.1	<i>Human-Centered Design Methodology.....</i>	<i>61</i>
5.2	INTEGRATION PLAN	64
5.2.1	<i>Agile Development Methodology</i>	<i>64</i>
5.2.2	<i>Agile Practices.....</i>	<i>66</i>
5.2.3	<i>Continuous Integration process</i>	<i>67</i>
5.2.4	<i>Development Infrastructure.....</i>	<i>68</i>
5.2.5	<i>Development Schedule.....</i>	<i>69</i>
6	REFERENCES	71
	APPENDIX A. 1ST TECHNICAL QUESTIONNAIRE	72
	APPENDIX B. 2ND TECHNICAL QUESTIONNAIRE	77

LIST OF FIGURES

FIGURE 1: USE CASES FOR THE SIMPLE USER.....	11
FIGURE 2: USE CASES FOR THE ADVANCED USER	13
FIGURE 3: ACCESS PLATFORM USE CASES.....	13
FIGURE 4: VIEW INFORMATION USE CASES.....	14
FIGURE 5: ACQUIRE ESTIMATION USE CASES	15
FIGURE 6: ADMINISTRATOR'S USE CASES	17
FIGURE 7: COMPONENTS IN THE SYMPHONY PLATFORM WITH PUBLIC DATA INTERFACES, FUNCTIONS, INPUTS AND OUTPUTS	24
FIGURE 8: LAYER-BASED CONCEPTUAL REPRESENTATION OF THE SYMPHONY ARCHITECTURAL DESIGN	24
FIGURE 9: A LOGICAL VIEW REPRESENTATION OF THE SYMPHONY ARCHITECTURAL DESIGN	26
FIGURE 10: SYMPHONY DASHBOARD ELEMENTS.....	27
FIGURE 11: UML CLASS DIAGRAM OF THE SYMPHONY DASHBOARD	28
FIGURE 12: SOFTWARE COMPONENTS THAT COMPOSE THE SYMPHONY PLATFORM	31
FIGURE 13 - UML CLASS DIAGRAM FOR THE GAMIFICATION ENGINE	32
FIGURE 14: UML CLASS DIAGRAM FOR THE INFORMATION MARKET COMPONENT	34
FIGURE 15: UML CLASS DIAGRAM FOR THE SOCIAL MEDIA MINING COMPONENT.....	36
FIGURE 16: UML CLASS DIAGRAM FOR THE AGENT BASED MODEL COMPONENT	39
FIGURE 17: AN EXAMPLE TOPOLOGY FOR ESB (PLEASE REFER TO [8])	43
FIGURE 18: UML CLASS DIAGRAM FOR THE ORCHESTRATOR PLATFORM	44
FIGURE 19: REQUIRED INPUTS FOR THE "DASHBOARD SOFTWARE COMPONENT"	45
FIGURE 20: REQUIRED INPUTS FOR THE "GAMIFICATION ENGINE SOFTWARE COMPONENT"	45
FIGURE 21 - USER REGISTRATION SEQUENCE DIAGRAM.....	46
FIGURE 22 - USER LOGIN SEQUENCE DIAGRAM.....	47
FIGURE 23 - SIMPLE USERS' EXPECTATIONS FROM SOCIAL MEDIA & VIEW SENTIMENT ANALYSIS OF PUBLIC OPINIONS FOR A DOMAIN SEQUENCE DIAGRAM	48
FIGURE 24 - VIEW EXPERTS EXPECTATIONS ON SPECIFIC EVENTS SEQUENCE DIAGRAM.....	49
FIGURE 25 - GET A GRAPHICAL VIEW OF CITIZENS EXPECTATIONS SEQUENCE DIAGRAM.....	50
FIGURE 26 - ACQUIRE AN ESTIMATION OF EVENT EVOLUTION IN THE FUTURE SEQUENCE DIAGRAM	51
FIGURE 27 - ACQUIRE AN ESTIMATION OF HOW SOCIAL MEDIA SIGNALS ALIGN WITH MACROECONOMIC TRENDS SEQUENCE DIAGRAM	52

FIGURE 28 - ACQUIRE AN ESTIMATION OF THE IMPACT OF THE DIFFERENT ECONOMY SCENARIOS SEQUENCE DIAGRAM.....	53
FIGURE 29: AGILE METHODOLOGY WORKFLOW	66
FIGURE 30: AGILE PROJECT MANAGEMENT PROCESS FRAMEWORK.....	66
FIGURE 31: CONTINUOUS INTEGRATION WORKFLOW	68
FIGURE 32: CONTINUOUS INTEGRATION WORKFLOW (2)	69

LIST OF TABLES

TABLE 1: SUMMARY OF USER REQUIREMENTS.....	9
TABLE 2: SYMPHONY ACTORS	10
TABLE 3: LIST OF SYMPHONY USE CASES.....	18
TABLE 4: SYMPHONY NON-FUNCTIONAL REQUIREMENTS	21
TABLE 5: DESCRIPTION TABLE OF THE SYMPHONY DASHBOARD	30
TABLE 6: DESCRIPTION TABLE OF THE GAMIFICATION COMPONENT.....	33
TABLE 7: DESCRIPTION TABLE OF THE INFORMATION MARKET COMPONENT	35
TABLE 8: DESCRIPTION TABLE OF THE SOCIAL MEDIA MINING COMPONENT	38
TABLE 9: DESCRIPTION TABLE OF THE AGENT BASED MACROECONOMIC ENGINE COMPONENT	42

1 Introduction

The purpose of this document is to define the flows and architecture of the SYMPHONY system in terms of the supported functionalities, the respective processes and the components that realise them.

1.1 Purpose and Scope

WP5 aims at providing the architectural and implementation aspects for the delivery of the SYMPHONY tools and their integration in a unified platform. The design of the SYMPHONY tools is driven by the conceptual architecture and user requirements defined in WP1 and will drive the design and implementation of the various components produced in the context of WPs 2, 3 and 4. The decisions presented in this deliverable are a subject to refinements and modifications, based on the progress of the technical work packages, as well as the validation and evaluation phases.

1.2 Methodology and Structure of the Deliverable

1.2.1 Architectural Design Principles

The presentation of the SYMPHONY system architecture follows a logical path, from the use cases supported by the system, to the functional conceptualisation of the needed components and ultimately to the software modules and tools that will be used for the implementation of these functionalities.

Using the Use Case Scenarios, we have then defined the System Use Cases, i.e. the actions and sequences that implement the processes necessary for serving the Use Case scenarios, while providing an abstraction of the independent functionalities that are combined in order to form the Use Case. These System Use Cases reveal the degree of complexity and needs for modularity, communication and orchestration of the various components that will be integrated within the SYMPHONY system. The architectural design aims to cover all these intricacies, while maintaining the openness of the system and ensuring that it will be scalable and easily modifiable. Furthermore, all the design and implementation decisions are grounded in established technology and industry standards.

The aforementioned characteristics of the system led to the adoption of a Service-oriented Architecture, with each module being implemented as an independent Web Service. The modules do not communicate with each other, but rather they all interact via sharing data. The sequence of execution is determined by a separate component, which orchestrates the invocation of the appropriate modules for carrying out the tasks defined by the Use Cases. This architecture, as explained in Section 3 of this document, satisfies the functional and non-functional requirements of the SYMPHONY system in a concise, clear and easily adjustable manner.

As the SYMPHONY system aims to support policy making by employing analysis on web content, experts contribution as well as citizens feedback, our initial focus is not to define the specific technologies that will be used by the various SYMPHONY components. Instead, we define the functionality that is necessary for exploiting the online information efficiently. The application of specific methods and software tools will be analysed in the respective technical Work Packages (WPs 2, 3 and 4).

To this end, the design of the SYMPHONY architecture focuses on the conceptual acquisition, processing and presentation phases and (a) specifies the system use cases relevant to the overall SYMPHONY vision;

(b) proposes a modular approach that realises all these system use cases; (c) specifies the way that the independent modules will be orchestrated (d) defines the information exchange flows and paths and provides an indication on how the results of the system use cases can be presented to the end-user in order to support him/ her on his policy design activities.

1.2.2 Structure of the Deliverable

The deliverable is organised as follows:

Section 2 presents the SYMPHONY System Use Cases and summarises the functional and non-functional requirements of a system supporting these cases.

Section 3 provides a description of the SYMPHONY components in two levels; (a) a conceptual level, where the components are examined with respect to the actions performed by them; and (b) a design level, where the actual software entities are presented. The architecture of the system and its data flow as well as the user activities and information flow are also described.

Section 4 gives an overview of the data description that will be exchanged by the various components of SYMPHONY.

Section 5, provides an overview of the integration planning including details regarding the implementation methodology to be used by the technical partners.

Appendices, the technical questionnaires used to collect partners' input are presented in the appendix

2 System Use Cases

Below we present an overview of the user requirements as derived from D1.1. The main actors of the system and the relevant use cases as derived from the analysis are also presented in the following sections.

2.1 Overview of User Requirements

A summary of user requirements based on the deliverable D1.1 is presented in the following table.

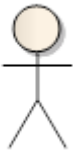
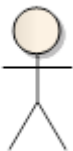
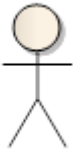
Table 1: Summary of User Requirements

ID	User Requirements
UR1	Monitor citizens' expectations on key policy variables in near real time;
UR2	Process information shared in social media;
UR3	Analyse the text and sentiment of the social media information and derive beliefs and expectations regarding the economy;
UR4	Search based on selecting different keywords;
UR5	See the trends related to key variables of the economy with respect to time (how the beliefs of social media users varied in time)
UR6	Visualise social media data in different perspectives
UR7	Create / initiate a new information market
UR8	Place / initiate some questions for getting the citizens expectations;
UR9	Invite participants to join a game session;
UR11	Have a graphical representation of the IM contracts / outcomes
UR12	Get a graphical view of the European economy
UR13	Initialise agents expectations on the economy with values
UR14	Pressing "Execute Model" to start the simulation
UR15	Get the simulation result
UR16	Be able to select a role
UR17	Having a chat room access
UR18	Having an easy registration process for the end users, as well as personalized access to the system

2.2 The SYMPHONY “Actors”

In order to identify the main ‘Actors’ relevant to the SYMPHONY platform, different user types are represented in this section as actors. An actor is anyone who exchanges data with the platform. The basic actors (user categories) of the platform are presented as follows:

Table 2: SYMPHONY Actors

Actors	Role in the System	Description
 Citizens / Industry / Civic society	Simple User	<p>The “Industry” are actors who influence the opinion of policy advisors and policy makers through e.g. lobbying.</p> <p>The “Civic Society” are institutions and organizations that manifest interests and will of citizens.</p> <p>The “Citizen” are actors that need to understand and influence policies;</p> <p>These users will be able to interact with policy makers and policy advisors through a gamified agent based model of the economy. These stakeholders will be able to interact with the gamified agent based model and understand the impact of the policy alternatives and their choices.</p>
 Policy Maker / Policy Advisor	Advanced User	<p>Advanced User extends the functions of Simple Users and further to it uses the following functions. The “Policy Maker” and/or “Policy Advisor” are the end users of the system that will be able to initiate an advanced search regarding a Policy domain / topic as well as acquiring estimations and feedback on citizens’ expectations (through Social Media Mining), experts’ opinions (through Information Markets), and simulation results (through Agent Based Modelling).</p>
 System Administrator	Administrator	<p>The “System Administrator” is any authorized person who monitors the system from the technical point of view, ensure its proper operation and handles users and roles and statistics. The platform as a system that needs to be maintained and monitored shall have a person or organization which will be responsible for the overall administration of the system, user management (rights, permissions, accounts) and solving problems. The system administrator is also responsible for the overall operation of the platform and especially for the ways the content is provided to the users and can be made available through different communication channels in an easy and effective way.</p>

2.3 The SYMPHONY “Use Cases”

A description of the use-case view of the SYMPHONY system is provided in this section. It mainly describes the set of the use cases that represent some significant, central functionality. It also describes the set of use cases that have a substantial architectural coverage or that stress or illustrate a specific, delicate point of the architecture.

The Use Cases of the SYMPHONY platform and related applications are listed below.

2.3.1 Overview of the Use Cases for the Simple User

- Create Account – register to platform
- Login
- Participate in the Information Market
- Participate in the Game;

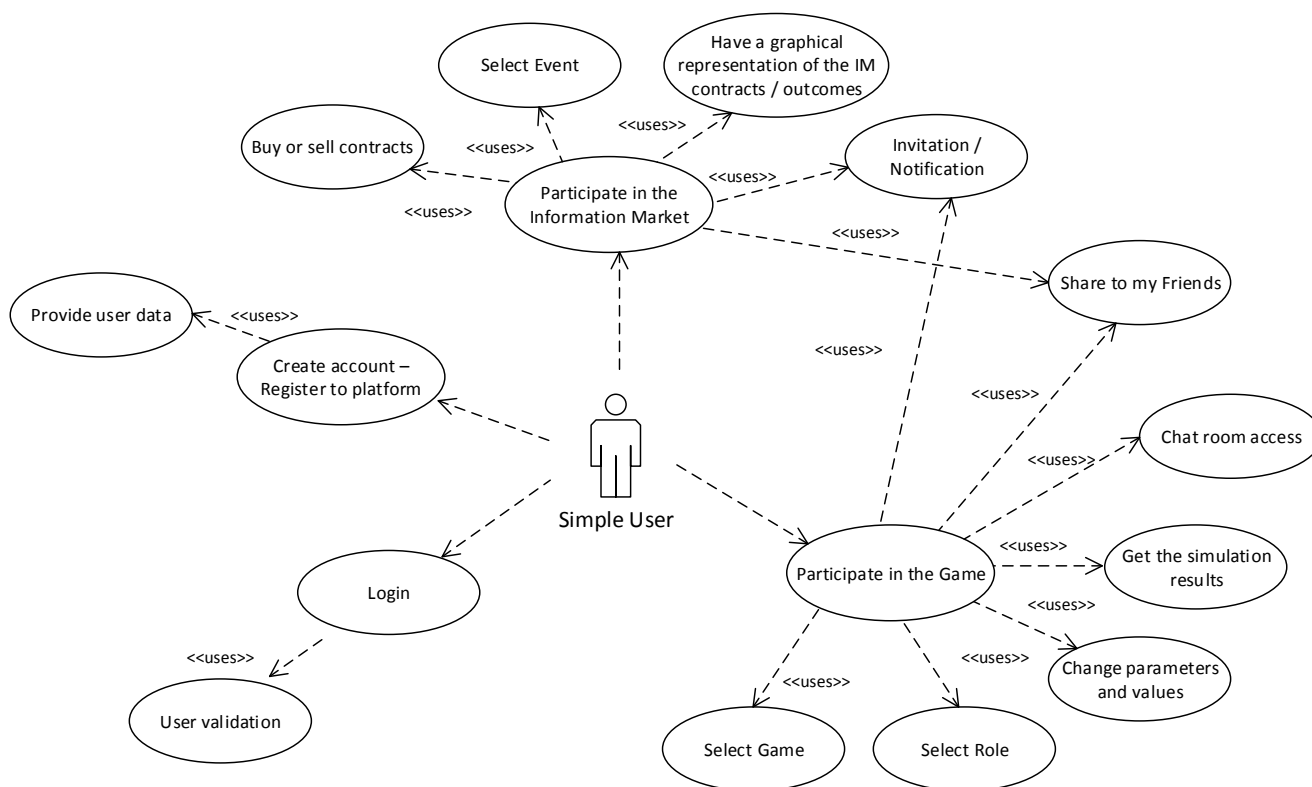


Figure 1: Use Cases for the Simple User

- UC1: Create account – register to platform

The user initiates a request for registration to the system by providing his/her personal data. For registration the user provides the following data:

- first name,
- last name,
- job title,

- email address,
 - preferred username and password
- **UC2: Login**

The users will have to login to the system by providing their username and password, in order to access the appropriate system's web pages and functionalities
- **UC3: Participate in the Information Market**

The user will be able to access the SYMPHONY platform and participate in the 'Information Market'. In order to do so the users will:

 - Receive invitation/notification;
 - Select event;
 - Buy or sell contracts
 - Have a graphical representation of the IM contracts / outcomes
 - Share to my Friends;
- **UC4: Participate in the Game**
 - Receive invitation / notification when a game starts though email;
 - Select 'Game'
 - Select a role in the game (household, firm (CGP - Consumer goods producers), commercial bank, central bank, government, Retail sellers (malls) and Investment goods producers);
 - Change parameters and values;
 - Get the simulation results
 - Have a chat room access
 - Share to my Friends (SM or Email);

2.3.2 Overview of the Use Cases for the Advanced User

Access System

- Create account – register to platform
- Login – User validation

View Information

- Citizens expectations from Social Media
- View sentiment analysis of public opinions for a domain
- View experts expectations on specific events;
- Get a graphical view of citizens' expectations;

Acquire Estimations

- Acquire an estimation of event evolution in the future
- Acquire an estimation of how social media signals align with macroeconomic trends;
- Acquire an estimation of the impact of the different economy scenarios

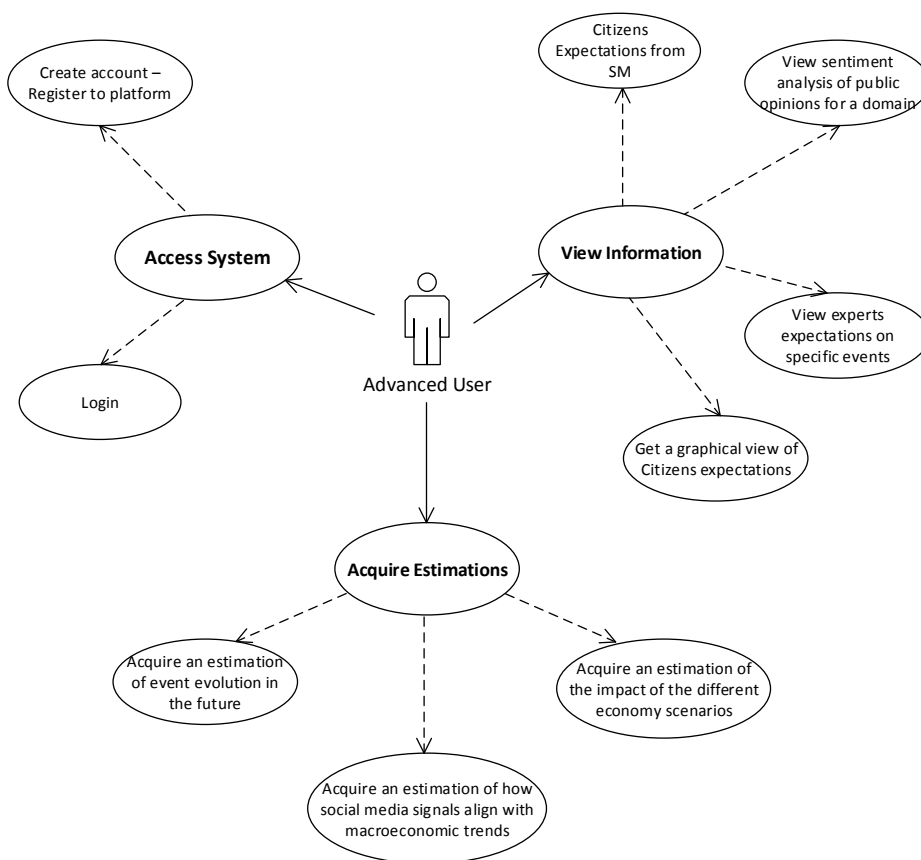


Figure 2: Use Cases for the Advanced User

2.3.2.1 Access Platform

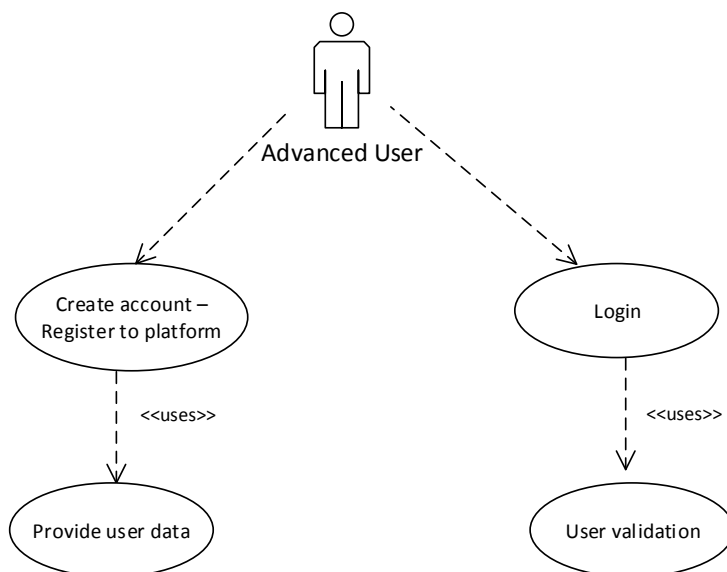


Figure 3: Access Platform use cases

- **Create account – register to platform**

The user initiates a request for registration to the system by providing his/her personal data;

- **Provide user data**

For registration the user provides the following data:

- first name,
- last name,
- job title,
- email address,
- preferred username and password

- **Login**

The users will have to login to the system by providing their username and password, in order to access the appropriate system's web pages and functionalities.

- **User Validation**

The system validates the end users' credentials;

2.3.2.2 View Information

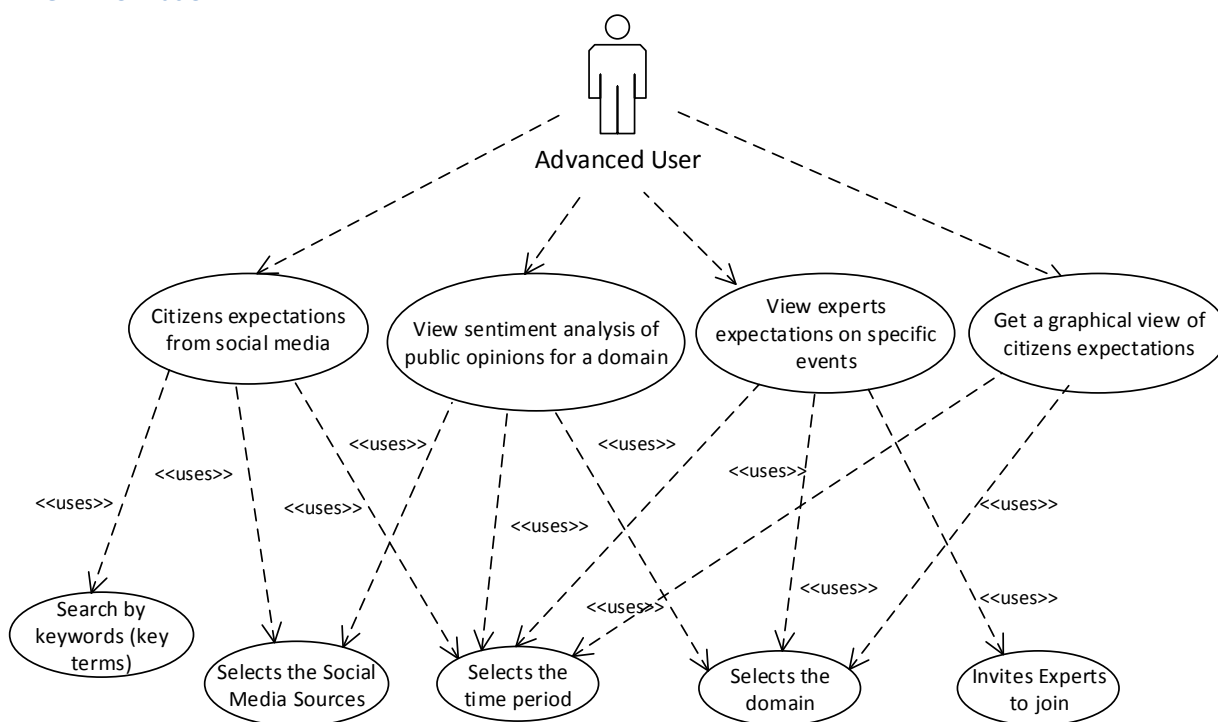


Figure 4: View Information use cases

- **UC5: Citizens' expectations from social media**

The user initiates a search in order to view Citizens' expectations from SM. In order to do that the user:

- Selects the time period,
- Selects the SM sources,

- Search by keywords (key terms)
- **UC6: View sentiment analysis of public opinions for a domain**
The user initiates a search in order to view the polarized public opinions about different aspects of a domain, in a specific time frame.
In order to do that the user:
 - Selects the domain
 - Selects the time period,
 - Selects the SM sources,
- **UC7: View experts expectations on specific events;**
The user initiates a new information market in order to view the experts' opinion on a specific event. The system will present this information as a graphical representation of the IM contracts / outcomes. In order to do that the user:
 - Selects the domain,
 - Selects the questions;
 - Selects the time period,
 - Invites experts to join;
- **UC8: Get a graphical view of Citizens Expectations;**
The user initiates a new 'game' scenario in order to initialise agents' expectations on the economy with values. The system will present this information as a graphical representation of the main outcomes of the virtual economy. In order to do that the user:
 - Selects the domain,
 - Selects the time period;

2.3.2.3 Acquire Estimations

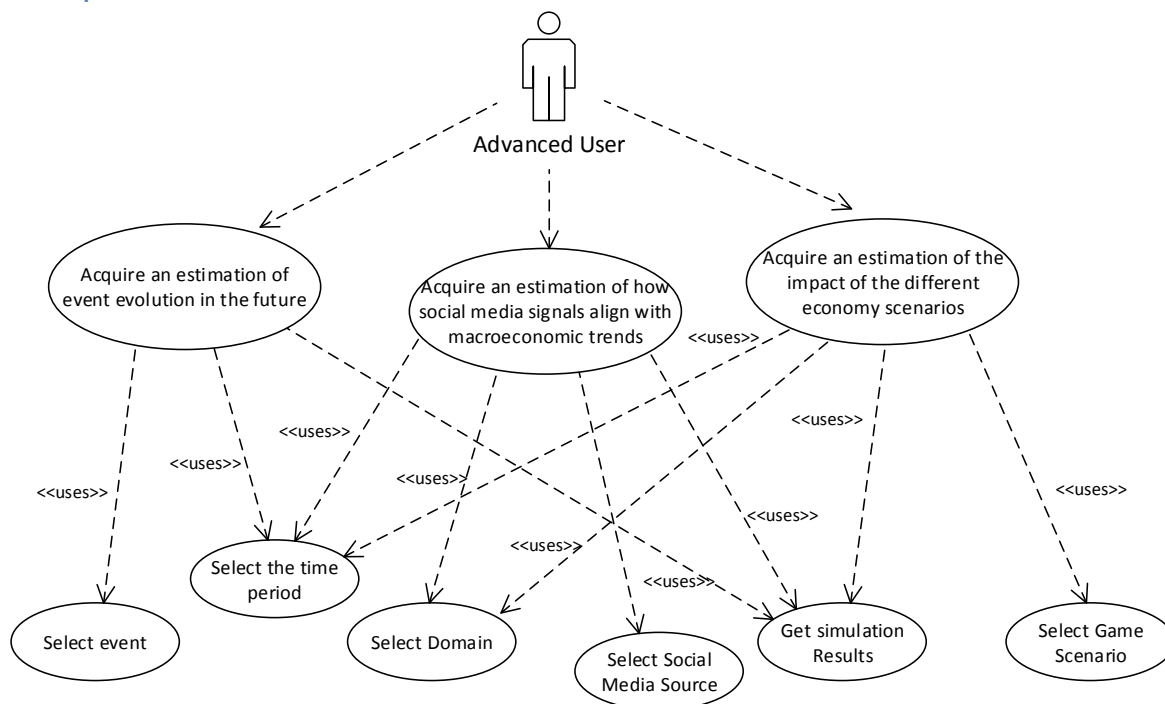


Figure 5: Acquire Estimation Use Cases

- **UC9: Acquire an estimation of event evolution in the future**

The user receives the results from the IM. The results will be presented as graphical representations of the IM contracts / outcomes.

In order to do that the user:

- Selects event;
- Selects the time period;
- Gets simulation Results;

- **UC10: Acquire an estimation of how social media signals align with macroeconomic trends**

The user initiates a search in order to have a graphical representation that depicts how the SM signals align with the macroeconomic aspects of a domain. In order to do that the user:

- Selects the time period;
- Selects the domain
- Selects the SM sources;
- Gets Simulation Results

- **UC11: Acquire an estimation of the impact of the different economy scenarios**

The user initiates a search in order to get a prediction on the public opinion for an economic scenario in a future time. In order to do that the user:

- Selects the time period;
- Selects the domain;
- Selects the game scenario;
- Gets simulation Results

2.3.3 Overview of the Use Cases for System Administrator

- Login
- Manage Users
- Manage Roles
- Monitor System

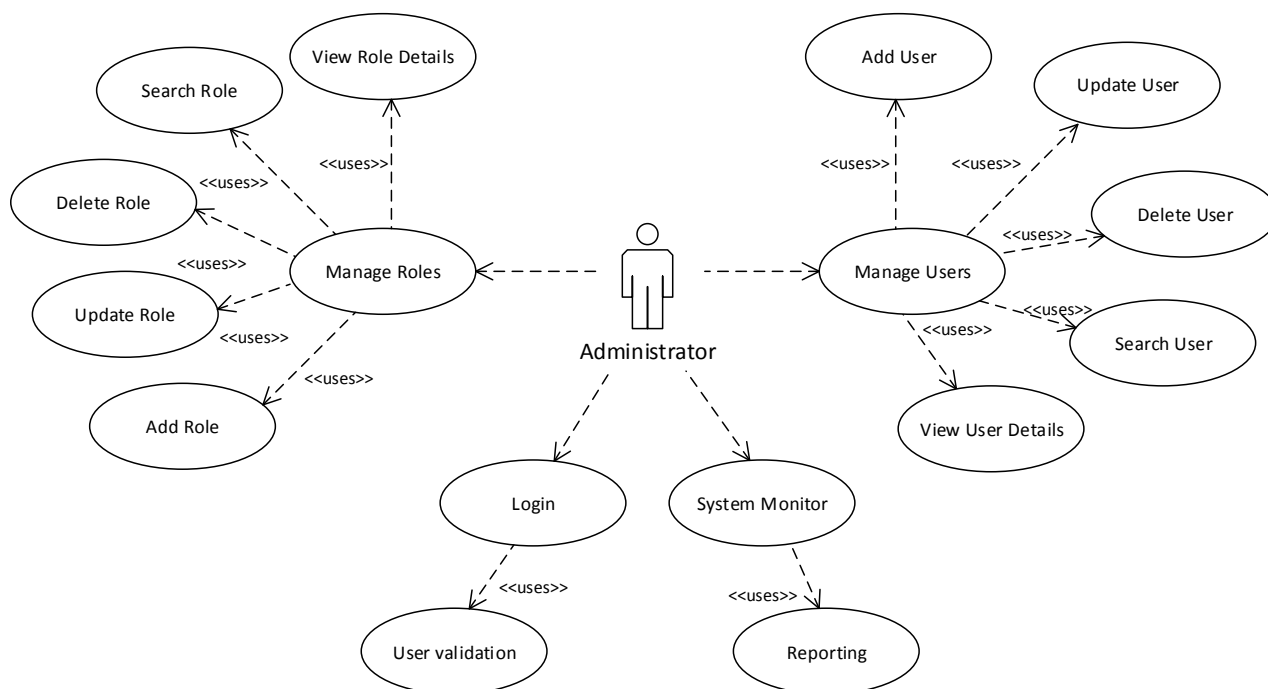


Figure 6: Administrator's use cases

- Login

Administrator as all other users in the system will have to login to the system by providing their username and password, in order to access the appropriate system's web pages and functionalities

- UC12: Manage roles

The system administrator has the following options in terms of role management:

- Add a new role
- Delete a role
- Search for roles based on criteria
- Update roles
- View a list of roles
- View selected role's details

- UC13: Manage users

The system administrator has the following options in terms of user management:

- Add a new user
- Delete a user
- Search for users based on criteria
- Update user's data
- Assign roles to a user
- View a list of users
- View selected user's data

- UC14: Monitor system

The administrator is also responsible for monitoring the well operation of the system. He will be able to view and examine logs as well as to have access to the system usage reports;

2.3.4 List of SYMPHONY Use Cases

The main use cases for the SYMPHONY platform, as derived from the Use Case Scenarios, are summarised in the following table.

Table 3: List of SYMPHONY Use Cases

UC CODE	USE CASES	ACTORS
UC1	Create account – register to platform	Policy Maker / Policy Advisor, Citizens / Industry / Civic society
UC2	Login	Policy Maker / Policy Advisor, Citizens / Industry / Civic society
UC3	Participate in the Information Market	Citizens / Industry / Civic society
UC4	Participate in the Game	Citizens / Industry / Civic society
UC5	Citizens Expectations from SM	Policy Maker / Policy Advisor
UC6	View sentiment analysis of public opinions for a domain	Policy Maker / Policy Advisor
UC7	View experts expectations on specific events	Policy Maker / Policy Advisor
UC8	Get a graphical view of Citizens' expectations	Policy Maker / Policy Advisor
UC9	Acquire an estimation of event evolution in the future	Policy Maker / Policy Advisor
UC10	Acquire an estimation of how social media signals align with macroeconomic trends	Policy Maker / Policy Advisor
UC11	Acquire an estimation of the impact of the different economy scenarios	Policy Maker / Policy Advisor
UC12	Manage roles	Administrator
UC13	Manage users	Administrator
UC14	Monitor system	Administrator

2.4 SYMPHONY Functional Requirement

This section elicits a set of functional requirements, based on the Use Cases identified in the previous sections and the elicited user needs as derived from D1.1. As the SYMPHONY platform design and implementation goes on, all these requirements will be refined and specified in more detail. In this direction, the SYMPHONY functional requirements can be clustered into the following categories:

- Social Media Mining

- Agent Based Model
- Gamification
- Information Market
- User Management

Based on these categories, the rest of this section is dedicated to the definition of the functional requirements of the SYMPHONY platform, which can be associated with the Use Cases. Thus, for every category of requirements, a table is provided, which contains the following:

- The incremental Requirement Identification Number (RIN);
- The respective requirement description;
- The association with the Use Cases of the SYMPHONY Platform and the Users' needs

It should be noted that in some cases, the association with scenarios use cases is not applicable, since the respective functional requirements may stem from the general Users needs survey, the applications scenarios or the objectives that the SYMPHONY platform should facilitate.

2.4.1 Social Media Mining

RIN	Description	Associated Use Cases
1.	Monitor citizens' expectations on key policy variables in near real time;	UC3 , UC4 , UC5 , UC6 , UC8
2.	Process information shared in social media;	UC3 , UC5 , UC6 , UC8 , UC10
3.	Analyze the text and sentiment of the social media information and derive beliefs and expectations regarding the economy;	UC3 , UC5 , UC6 , UC8 , UC10
4.	Search based on selecting different keywords;	UC5 , UC9
5.	See the trends related to key variables of the economy with respect to time (how the beliefs of social media users varied in time);	UC5 , UC6 , UC10
6.	Visualize social media data in different perspectives;	UC5 , UC6 , UC10

2.4.2 Information Market

RIN	Description	Associated Use Cases
1.	Place / initiate some questions for getting the citizens' expectations;	UC3 , UC7 , UC9

RIN	Description	Associated Use Cases
2.	Have a graphical representation of the IM contracts / outcomes;	UC7 , UC9

2.4.3 Agent Based Model

RIN	Description	Associated Use Cases
1.	Initialize agents expectations on the economy with values	UC8 , UC9 , UC10 , UC11

2.4.4 Gamification

RIN	Description	Associated Use Cases
1.	Be able to Invite participants to join a game session	UC11
2.	Get a graphical view of the European economy	UC4 , UC8 , UC11
3.	Be able to participate in the game and select a role;	UC4 , UC8 , UC11
4.	Chat room access	UC4 , UC8 , UC11

2.4.5 User Management

RIN	Description	Associated Use Cases
1.	The system should provide user/role based access	UC1 , UC2 , UC3 , UC4 , UC5 , UC6 , UC7 , UC8 , UC9 , UC10 , UC11 , UC12 , UC13 , UC14

2.5 SYMPHONY Non-Functional Requirements

This section analyses the non-functional requirements that should apply for the SYMPHONY platform components. The SYMPHONY platform should satisfy a set of non-functional requirements, which will ensure the normal operation of the system and the provision of a proper environment for the desired system level functionalities as analysed in the previous section. The non-functional requirements are depicted in the following table.

ID	Non-functional Requirement	Description
NF1.	Persistence	The system should provide back-up options for the contents of the services description and their associated profiles. All the repositories should be based on implementation able to tackle persistence problems, which ensure that data is maintained in case of a node failure.
NF2.	Performance	This requirement has to do with QoS characteristics, such as the response time of a SYMPHONY compliant service request or the needed resource utilisation schema to support the SYMPHONY functionalities
NF3.	Scalability/Expandability	The platform should be able to handle the increasing size of services and the simultaneous users accessing the supported functionalities, as well as being open to new ones.
NF4.	Availability	The platform should ensure that users have always access to data and associated assets 24/7 with 99.9% reliability. This requirement entails stability in the presence of localized failure.
NF5.	Usability	The platform should have an attractive and intuitive User Interface. All the UI level functionalities should be validated on their usability through user-centric assessment and observation.
NF6.	Interoperability/conformance with standards	The platform should conform to standards when developing the individual components and their interfaces, in order for the system to be maintainable and to be able to interface with existing solutions or be extended with future functionalities.
NF7.	Security	The platform should support security mechanisms, which cater for the resources identification and the trusted provision of services in a SOA-based paradigm

Table 4: SYMPHONY Non-functional Requirements

3 SYMPHONY System Architecture

3.1 High Level Overview

As is evident from the analysis of the SYMPHONY System Use Cases and the requirements of the overall system, the process of satisfying the various functionalities involves multiple and diverse components SYMPHONYs (namely the SYMPHONY dashboard, the Gamification engine, the Information Market, the Social Media Mining and the Agent Based Macroeconomic Engine). These software components should be able to exchange information in real time using the most efficient way, in order to satisfy the dependences and achieve project objectives. Moreover, the infrastructure hosting the software components might exist at different physical locations which might create communication issues. To accomplish these goals, the SYMPHONY System Architecture will satisfy the captured functional and non-functional requirements specifying the logical structure of the system, giving special focus on the programming interfaces that enables the interaction and communication among the individual components.

Not only because of the several distinctive particularities but because of the need to ensure the continuous and uninterrupted data flow between the software components, a new layer has been introduced named ESB / Orchestrator. The ESB / Orchestrator undertakes the role of an intermediate service being responsible for data conversion and communication handling. Moreover, to maintain resources and ensure a normal data flow, part of the ESB / Orchestrator's responsibility is to organize and prioritize connection requests and connection responses.

3.2 SYMPHONY Architecture Design

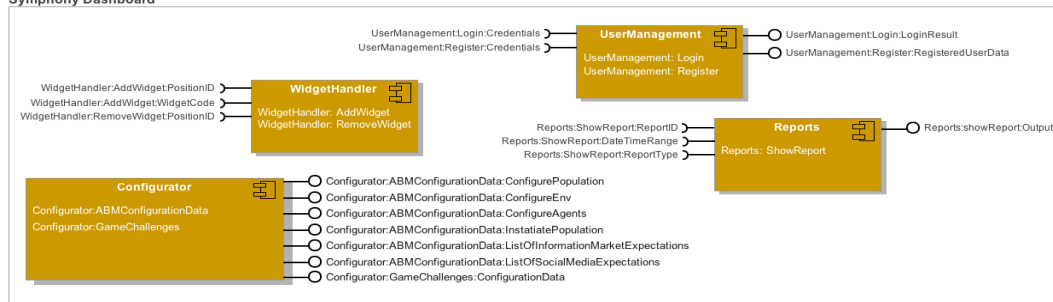
Based on the specifications, this section presents the logical view of the SYMPHONY platform architecture. Environment components along with the software components that compose the SYMPHONY platform are depicted in a layered architecture, which is an instantiation of a Service-Oriented Architecture (SOA)-based system reference architecture [1] [2] [3].

SOA is widely used as an architectural pattern for contemporary systems development and integration. Its main principle is the separation of functionality into different units which interoperate with each other over a network. These units are usually developed as Web services and are reused by the system in order to achieve the desired functionality. A 'Web service' is defined by the World Wide Web Consortium (W3C), as *"a software system designed to support interoperable machine-to-machine interaction over a network"* [4]. The Web services can interoperate by exchanging data with each other directly, or by being coordinated by a central mechanism of the system. In general, SOA focuses on loose coupling of services in terms of used technology and programming language. By using common standards, web services have little or no dependency on each other, a fact that offers flexibility and scalability to the system. SOA is considered to be the evolution of the distributed systems architecture approach.

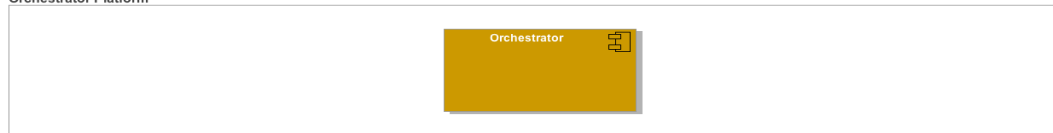
Following the SOA architectural pattern, the below figure (Figure 7) depicts the SYMPHONY's components [Social Media Mining, Information Market, Agent Based Macroeconomic Engine, Gamification Engine,

Orchestrator Platform and Symphony Dashboard] and their public data interfaces, functions, inputs and outputs of each component.

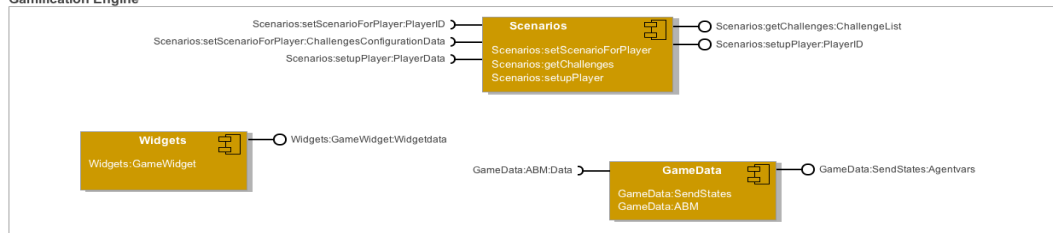
Symphony Dashboard



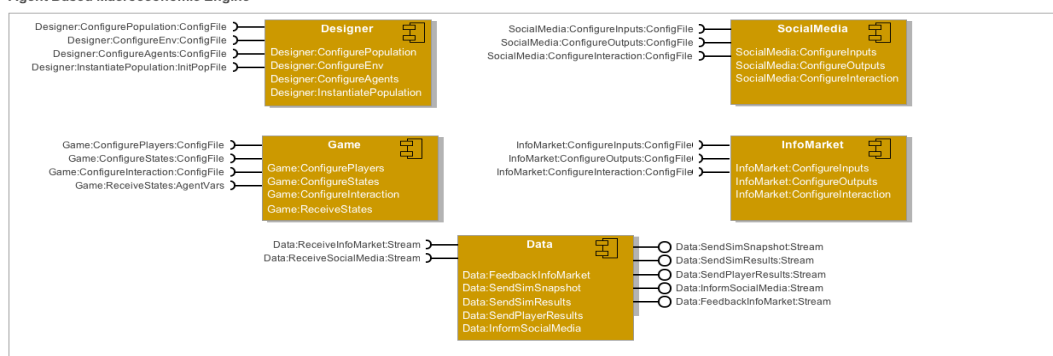
Orchestrator Platform



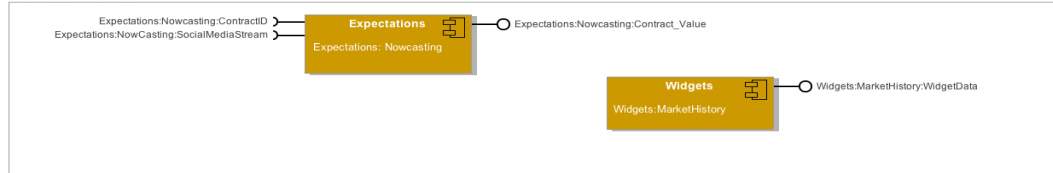
Gamification Engine



Agent Based Macroeconomic Engine



Information Market



Social Media Mining

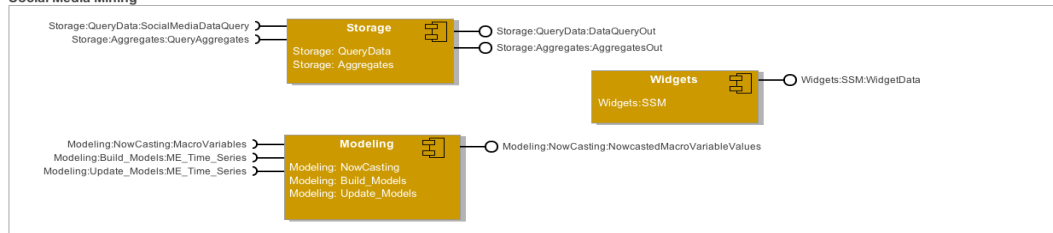


Figure 7: Components in the SYMPHONY platform with public data interfaces, functions, inputs and outputs

The figure above makes an overview of the components in SYMPHONY platform and the environment, as they have been identified in project's specifications. This view can be conceptually transformed into the layered-based representation of the SYMPHONY architectural design, which is shown in (Figure 8).

In this layered structure, the top layer consists of the Interaction Layer [5], which enables the communication with the target end users. The components placed there offer a graphical representation of the tools to enable the users to interact with the rest of the SYMPHONY platform and environment components. Furthermore, all configuration and reporting components, can be placed here as well.

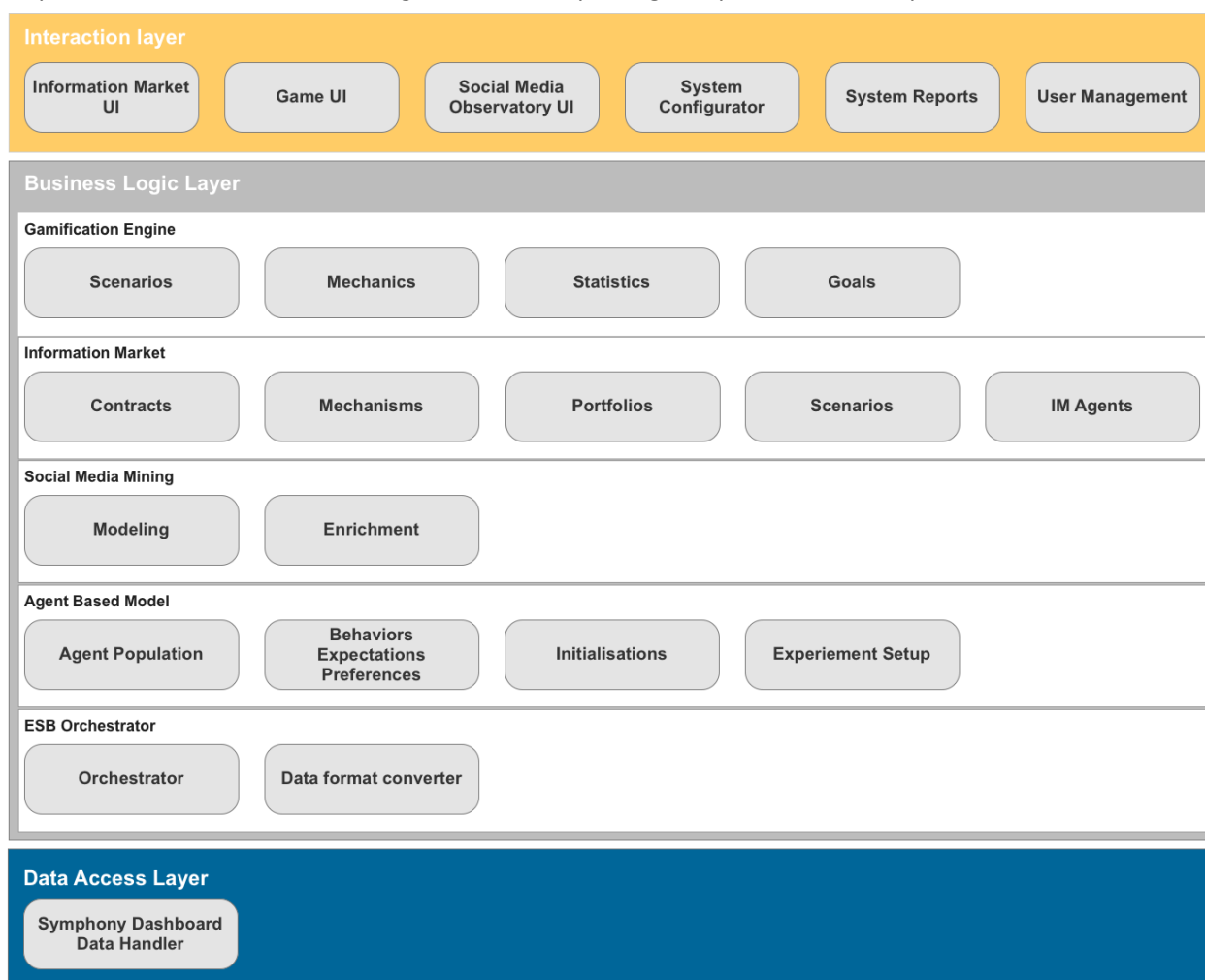


Figure 8: Layer-based conceptual representation of the SYMPHONY architectural design

At a second level, the Business Logic Layer [5] hosts the responsible components for the business logic of the platform. This is the part of the system that encodes the real-world business rules and determines how data can be created, displayed, stored, and changed. More precisely, this layer hosts parts of the software components that compose the Gamification Engine, Information Market, Social Media Mining, Agent Based Model and the ESB Orchestrator. Elements of business logic are maintained over entire SYMPHONY platform and environment components

At a third level, the Data Access Layer [5] contains all the components, which are responsible for providing data access functionalities to the upper layer components. These objects encapsulates complexity of the data source models for the SYMPHONY's Dashboard component.

It should be clarified that the internal structure of each component maintains software blocks, which may refer to more than one layer. For example, the *Gamification Engine - Statistics* component exposes an interaction layer module, which enables end user to access game statistics. On the other hand, the *User Management* component offers an interface for the administrators to manage Users and Roles, as well as it maintains a business logic behind that refers to the users' permissions with regards to the SYMPHONY's platform resources access.

The software components that are being analysed at the above paragraphs, seems independent but in fact they are not. Software components have the necessity of input data to function properly. The following figure (Figure 9) presents an overview of the environment and the software components of the SYMPHONY platform along with the communication lines that links the various and heterogeneous components.

These communication lines are used to exchange valuable information and receive feedback from the actors of the platform i.e. Citizens, Policy makers, Policy modelers, Administrators etc. The communication between the actors and the SYMPHONY dashboard is made through a web browser while the internal software component communication is handled using RESTful web services.

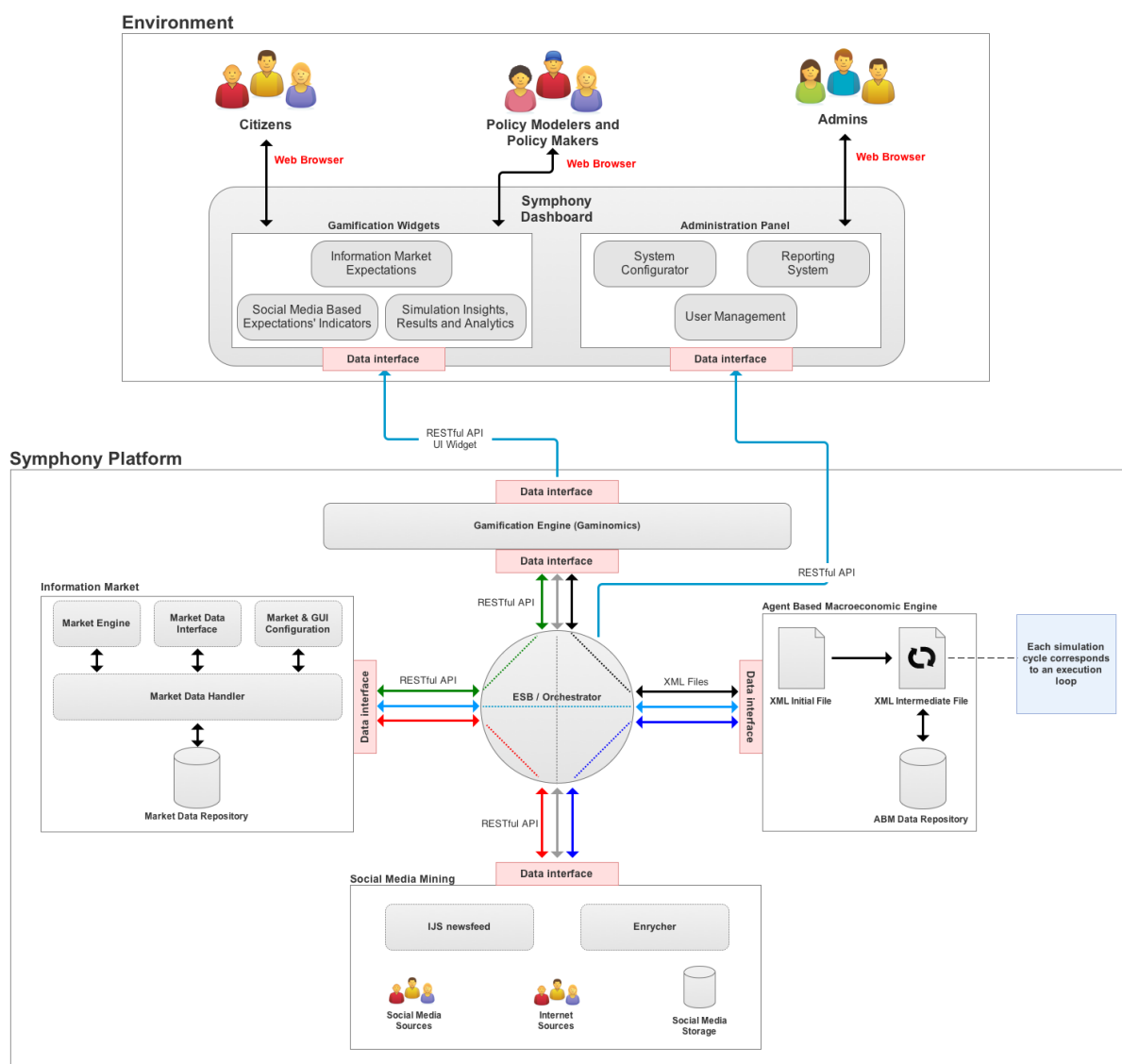


Figure 9: A logical view representation of the SYMPHONY architectural design

The chosen technology for SYMPHONY platform is the Representational State Transfer (REST) which has gained widespread acceptance across the Web as a simpler alternative to SOAP- and Web Services Description Language (WSDL)-based Web services. Key evidence of this shift in interface design is the adoption of REST by mainstream Web 2.0 service providers—including Yahoo, Google, and Facebook—who have deprecated or passed on SOAP and WSDL-based interfaces in favor of an easier-to-use, resource-oriented model to expose their services.

REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such

a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use.

3.3 SYMPHONY Components

3.3.1 SYMPHONY Front-End

3.3.1.1 Elements

The SYMPHONY Front-End is the SYMPHONY dashboard that contains the Gamification widgets and the administration panel. The SYMPHONY Dashboard constitutes the main access gateway for participants (Citizens, Policy modelers, Policy makers and Administrators) to the SYMPHONY platform. Using a web browser, Citizens, Policy modelers and Policy makers could interact with the Gamification Widgets as well as the Administrators are allowed to interact with the Administration Panel, set configuration parameters and access the reporting system. The following figure provides an overview of the SYMPHONY Front-End elements.

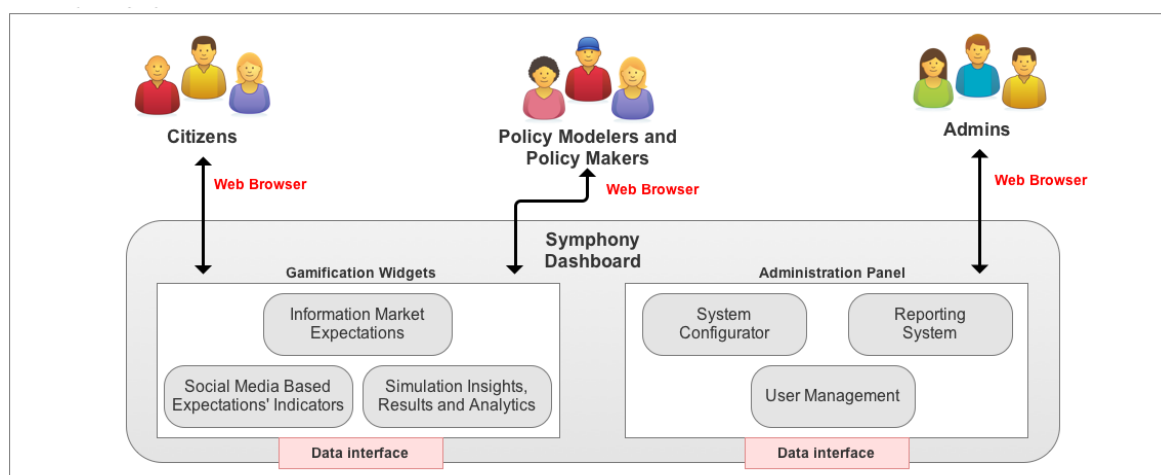


Figure 10: SYMPHONY Dashboard elements

Gamification widgets

Gamification widgets are the most prominent part of the SYMPHONY framework for the sole reason that they are the only gate to the end users.

The Gamification widgets are the access point to the Gamification engine and require very extensive user interaction. Widget elements will need to be tightly integrated into the SYMPHONY Dashboard. The Gamification widgets are built using HTML 5 and are enriched with live data using ajax calls to the Gamification Engine. The main idea is to provide gamification elements (recognitions and rewards) that will motivate the end users. Additionally, the Gamification provides game mechanics such as levels of difficulty to define and govern the game that relies on the Agent Based Model and Information Markets components.

In terms of data communication, Gamification Engine and the Gamification widgets are using RESTful services as it is depicted at Figure 9.

Administration Panel

The Administration panel consists of two sub areas (Figure 10), the System Configurator and the Reporting System. The access to the Administration panel is limited to users with Administrator role. Also the Administration panel is used to initiate component variables, configure system behavior as well as access usage and statistical reports.

3.3.1.2 Analysis

The SYMPHONY Dashboard aims to integrate information from multiple and heterogeneous components into a single and unified display.

It will be available as a web interface and thus all participants will be able to access it using any modern web browser. The look and feel will be unified for all participants but the interfaces and the data presented will vary, depending on the role each participant has in SYMPHONY platform. For example a Citizen won't be able to access neither the configurator nor the reporting system. To access these two restricted areas, a user must have an administrator role.

The above paragraphs describes the elements and features that will be implemented in the SYMPHONY Dashboard component. The below UML class diagram of the component describes its structure by showing the classes, the attributes and the operations (or methods).

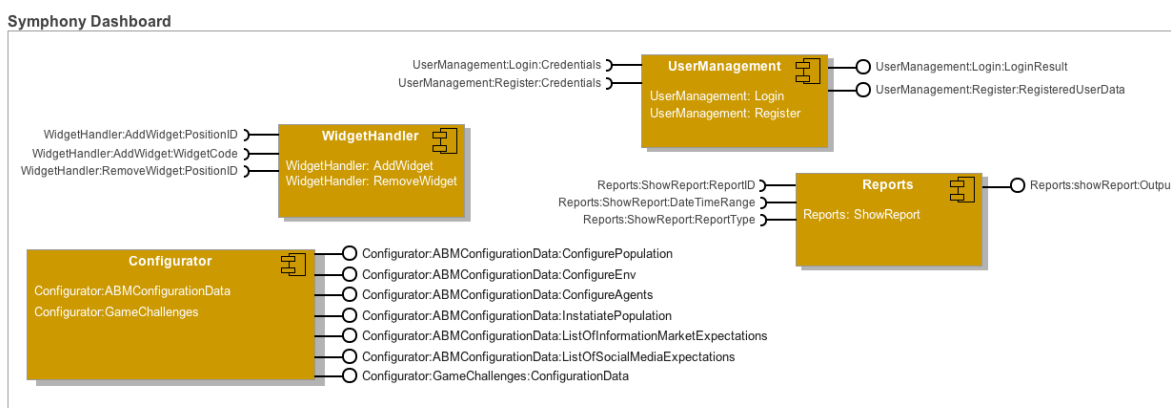


Figure 11: UML Class diagram of the SYMPHONY Dashboard

The table below presents the analysis of the Dashboard that states the name, the brief description and the Implementation priority of the Data Interfaces, the external methods that are available to other components, as well as the input and output parameters of the Dashboard.

Data Interfaces		
Name	Brief Description	Implementation priority (Must/Should)
WidgetHandler	Handles UI HTML widgets, provides	Must

	methods to add or remove a widget from the dashboard etc	
Configurator	Used to configure the SYMPHONY Platform, setting the initial variables and/or change the existing ones	Must
Reports	Reporting system used to inform the administrators about system usage statistics and simulation statistics	Must
UserManagement	Used to manage all SYMPHONY users from simple players to system admins	Must
Methods (External)		
InterfaceName:MethodName	Brief Description	Implementation priority (Must/Should)
WidgetHandler:AddWidget	Adds a new widget to the SYMPHONY Dashboard	Must
WidgetHandler:RemoveWidget	Removes an existing widget from the SYMPHONY Dashboard	Must
Configurator:ABMConfigurationData	Configure or set one or more system parameters regarding the ABM module	Must
Configurator:GameChallenges	Configure or set one or more system parameters regarding the Game module	Must
Reports:ShowReport	Construct and show the requested report to the user	Should
UserManagement:Login	Used to login to SYMPHONY dashboard using user credentials	Must
Usermanagement:Register	Used to register a new user providing user info	Must
Input		
InterfaceName:MethodName:InputParameterName	Brief Description	Implementation priority (Must/Should)
WidgetHandler:AddWidget:PositionID	ID of the position that the new widget will be placed	Must
WidgetHandler:AddWidget:WidgetCode	The code to initialise the widget	Must
WidgetHandler:RemoveWidget:PositionID	ID of the position to remove the widget from	Must
Reports:ShowReport:ReportID	ID of the report to be displayed	Should
Reports:ShowReport:DateTimeRange	Datetime range for the report's data	Should
Reports:ShowReport:ReportType	Report type	Should

UserManagement:Login:Credentials	Credentials provided to login at SYMPHONY platform	Should
UserManagement:Register:Credentials	User data provided to register at SYMPHONY platform	Should
Output		
InterfaceName:MethodName:OutputParameterName	Brief Description	Implementation priority (Must/Should)
Reports:ShowReport:Output	The output result of the requested report	Must
UserManagement:Login:LoginResult	Result providing information regarding the login attempt	Should
UserManagement:Register:RegisterUserData	Result providing information regarding the registration attempt	Must
Configurator:ABMConfiguratorData:ConfigurePopulation	Configuration data that are input to the ABM software component	Must
Configurator:ABMConfiguratorData:ConfigureEnv	Configuration data that are input to the ABM software component	Must
Configurator:ABMConfiguratorData:ConfigureAgents	Configuration data that are input to the ABM software component	Must
Configurator:ABMConfiguratorData:InstantiatePopulation	Configuration data that are input to the ABM software component	Must
Configurator:ABMConfiguratorData:ListOfInformationMarketExpectations	Configuration data that are input to the ABM software component	Must
Configurator:ABMConfiguratorData:ListOfSocialMediaExpectations	Configuration data that are input to the ABM software component	Must
Configurator:GameChallenges:ConfigurationData	Configuration data that are input to the Game software component	Must

Table 5: Description table of the SYMPHONY Dashboard

3.3.2 SYMPHONY Back-End

The SYMPHONY platform (Figure 12), consists of four main heterogeneous software components (Gamification Engine, Information Market, Social Media Mining and Agent based Macroeconomic Engine) and the Enterprise Service Bus / Orchestrator that is used to coordinate the communication between the main software components and to normalize the data interchange mechanism of the data interfaces that the heterogeneous software components provides and supports.

Symphony Platform

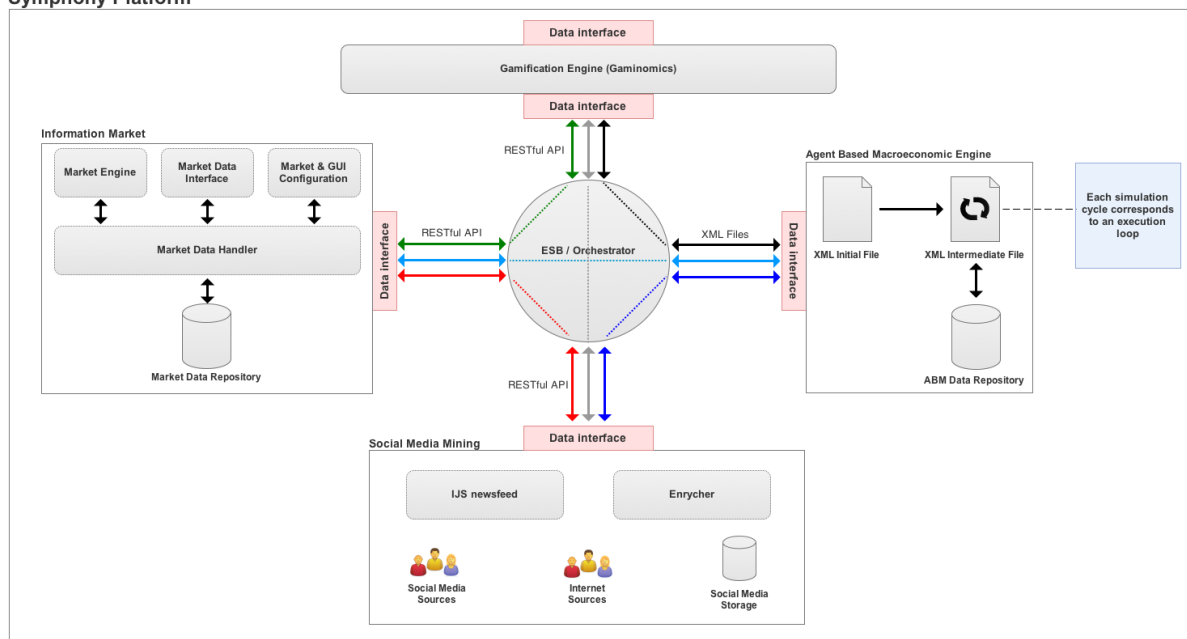


Figure 12: Software components that compose the SYMPHONY platform

The main software components that compose the SYMPHONY platform are analysed at the below subsections

3.3.2.1 Gamification Engine

The Gamification Engine aims to provide an enhanced user experience through a gamification layer that enables policy makers and citizens to engage with SYMPHONY's macro-economic engine in a meaningful and appealing way. To communicate with the other SYMPHONY components, the Gamification engine provides web services using RESTful API. Using the ESB / Orchestrator, the Gamification Engine communicates with the Information Market and the Agent Based Macroeconomic Engine components.

Through combination of the information markets (IM) and by reducing the complexity of engaging with the agent-based model (ABM) to a set of simple and intuitive graphical interfaces supported by gaming mechanisms, we will guide the user through different scenario sets. These scenarios are created by key stakeholders using SYMPHONY's Dashboard.

According to the above characteristics of the component, it has been designed the below component's UML diagram (Figure 13) that describes component's structure by showing the classes, the attributes and the operations (or methods) in order to fulfil the expected functionality and connectivity. More details will be analysed and presented in WP4.

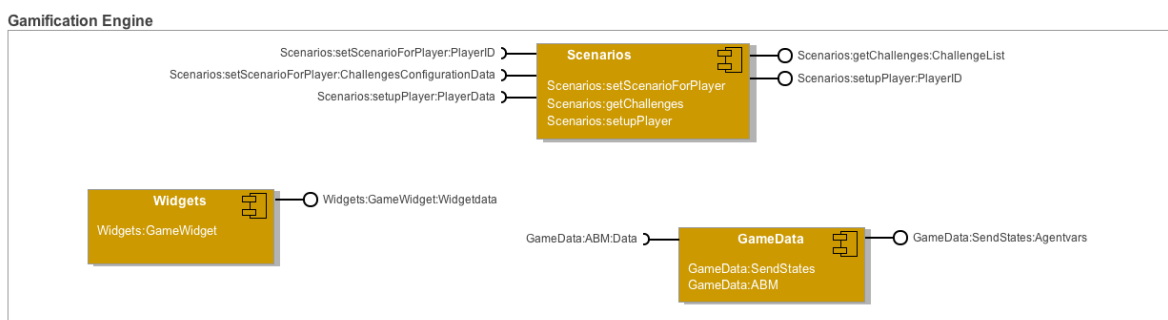


Figure 13 - UML Class diagram for the Gamification Engine

The Table 6 below presents the analysis of the component that states the name, the brief description and the Implementation priority of the Data Interfaces, the external methods that are available to other components, as well as the input and output parameters of the component.

Data Interfaces		
Name	Brief Description	Implementation priority (Must/Should)
Challenge	Pre-set up game rules – eg “keep firm afloat” or “keep country out of recession”	Must
GameData	Communication about data to and from the ABM	Must
Widgets	The various components of the game UI in widget form for dashboardage	Should
Methods (external)		
InterfaceName:MethodName	Brief Description	Implementation priority (Must/Should)
Challenge:SetChallengeForPlayer	Sets the rules for a player’s game session	Must
Challenge:getChallenges	Gets the available session rules	Optional
Challenge:setupPlayer	Create a player in the session	Must
GameData:GetInput	Returns the input provided by the game players this state	Must
GameData:ABMUpdate	Updates the gamification engine with the next game state	Must
Widgets:GameWidgets	Returns the HTML of the game’s widgets	Should

Input		
InterfaceName:MethodName:ParameterName	Brief Description	Implementation priority (Must/Should)
Challenge:setChallengeForPlayer:PlayerID	The ID of the player	Must
Challenge:setChallengeForPlayer:ChallengesConfigurationData	The challenge data	Must
Scenarios:setupPlayer:PlayerData	The data to associate with the player	Must
GameData:ABMUpdate:Data	Data from the ABM about the latest game state	Must
Output		
InterfaceName:MethodName:ParameterName	Brief Description	Implementation priority (Must/Should)
Challenge:getChallenges:ChallengeList	An array of challenge info	Should
Challenge:setupPlayer:PlayerID	The ID of the new player	Must
GameData:GetInput:Agentvars	The vars the players wish to change	Should
Widgets:GameWidget:Widgetdata	JSON containing widget information and HTML	Must

Table 6: Description table of the Gamification component

3.3.2.2 Information Market

Information Market (IMs) module manages speculative markets that serve to aggregate the beliefs of multiple traders in the price of contracts representing different outcomes of a future event. Contract prices provide a reasonable estimate of what the traders in aggregate believe to be the probability of the event and as such markets are able to generate forecasts. Individuals influence these prices by buying and selling contract shares based on their belief about the outcome. At the end of the dealing period, individuals are paid off based on the accuracy of their bids. It is planned to be delivered a fully functional IM where users with administration rights will be able to create new markets populated with contracts in the form of questions which will represent future events whereas traders will be able to provide information and their expectations with respect to the occurrence of the future events by buying or selling corresponding contracts.

The IM will integrate the SYMPHONY user model in order to identify user accounts as admins or traders.

To communicate with the other SYMPHONY components, the Information Market provides web services using RESTful API. Using the ESB / Orchestrator, the Information Market component communicates with

the Gamification engine, the Social Media Mining and the Agent Based Macroeconomic Engine components.

The IM will implement all the user interfaces required to create and manage markets as well as to trade in a market. Moreover we will consider to provide a set of restful interfaces which will allow third party applications to use the core functionalities, including creating and managing markets, trading in markets and retrieving historical information with respect to the price evolution of contracts in a market.

According to the above characteristics of the component, it has been designed the below component's UML diagram (Figure 14) that describes component's structure by showing the classes, the attributes and the operations (or methods) in order to fulfil the expected functionality and connectivity. More details will be analysed and presented in WP2.

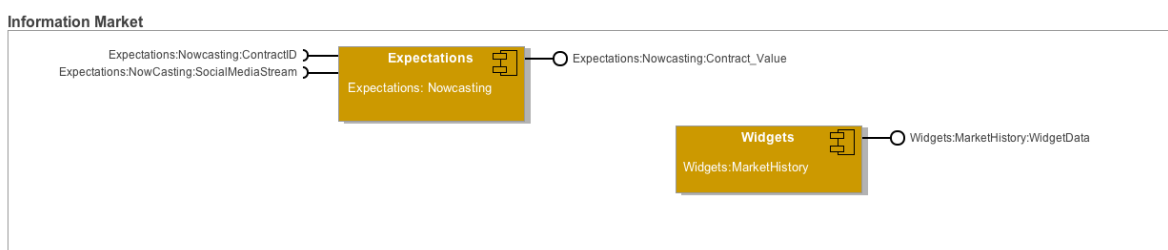


Figure 14: UML Class diagram for the Information Market component

The Table 7 below presents the analysis of the component that states the name, the brief description and the Implementation priority of the Data Interfaces, the external methods that are available to other components, as well as the input and output parameters of the component.

Interfaces		
Name	Brief Description	Implementation priority (Must/Should)
Expectations	Provides access to the current prices of contracts in IMs	Must
Widgets	Provides access to the IM functionalities through widget based interfaces and allows third party applications to implement IMs	Must
Methods (External)		
InterfaceName:MethodName	Brief Description	Implementation priority (Must/Should)
Expectations:Nowcasting	Nowcasts expectations on macroeconomic indices and energy prices by providing the current market	Must

	prices of corresponding IM contracts	
Widgets:MarketHistory	Provides graphs that display historical data which help users to analyze the aggregated expectations in the market	Must
Input		
InterfaceName:MethodName:InputParameterName	Brief Description	Implementation priority (Must/Should)
Expectations:Nowcasting: ContractID	Contract for nowcasting. The ContractID is an Integer value.	Must
Widgets: MarketHistory:MarketID	Provides the trading history and statistics for a specific MarketID. The MarketID is an Integer value.	Must
Output		
InterfaceName:MethodName:OutputParameterName	Brief Description	Implementation priority (Must/Should)
Expectations:Nowcasting:Contract_Value	Macro contract nowcasted value. The Contract_Value is of type Double.	Must
Widgets:MarketHistory:Graph	A graph showing the price fluctuation.	Must

Table 7: Description table of the Information Market component

3.3.2.3 Social Media Mining

Social media mining module refers to data mining of content streams produced by people through interaction via Internet based applications. To communicate with the other SYMPHONY components, the Social Media Mining provides web services using RESTful API. Using the ESB / Orchestrator, the Social Media Mining component communicates with the Information Market and the Agent Based Macroeconomic Engine components.

The Social Media Mining Component will be developed within SYMPHONY project and will perform a number of activities that will allow to the users to observe, enrich and store data from social media, as well as analyse how social media signals align with macroeconomic trends.

The core of the Social Media Mining Component will include the following functionalities:

- Observing subcomponent, which will receive data from social media.
- Enrichment subcomponent, which will perform enrichment of social media data.
- Storage subcomponent, which will handle storage and access to social media data.
- Modeling subcomponent, which will provide models for alignment of social media signals and macroeconomic data used in nowcasting.

User Interface module (for Social Media) will provide users the possibility to view social media data in an active way. According to the above characteristics of the component, it has been designed the below component's UML diagram (Figure 15) that describes component's structure by showing the classes, the attributes and the operations (or methods) in order to fulfil the expected functionality and connectivity. More details will be analysed and presented in WP2.

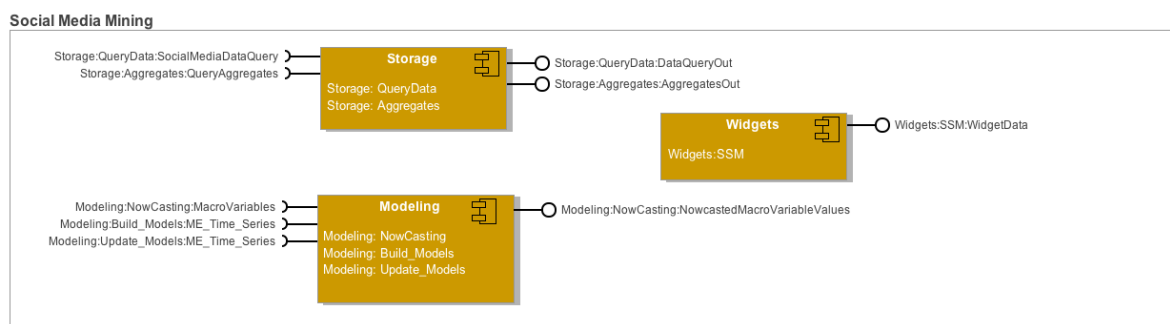


Figure 15: UML Class diagram for the Social Media Mining Component

The Table 8 below presents the analysis of the component that states the name, the brief description and the Implementation priority of the Data Interfaces, the external methods that are available to other components, as well as the input and output parameters of the component.

Interfaces		
Name	Brief Description	Implementation priority (Must/Should)
Storage	Stores data from Social media. Provides a list of query functionalities: search by hashtags, search by keywords, search by users, aggregations by different features, such as sentiment, frequency etc.	Must
Modeling	Learns models for nowcasting using SVM or linear regression to model time series from social media data.	Must
Widgets	Provides access to the SMM functionalities through widget based interfaces	Could
Methods (External)		
InterfaceName:Meth odName	Brief Description	Implementation priority (Must/Should)
Modeling:Nowcasting	Nowcasts macroeconomic values	Must

Storage:Query_Data	Query Data (by hashtags/userMentions/keywords)	Should
Storage:Aggregates	Compose Aggregates (sentiment aggregates, timeline aggregates etc.)	Should
Modeling:Build/Update_Models	Generates and later updates an appropriate model for nowcasting particular macroeconomic time series	Could
Widgets:SMM	Provides graphs that display historical data from Social Media	Could
Input		
InterfaceName:Meth odName:InputParam eterName	Brief Description	Implementation priority (Must/Should)
Modeling:Nowcasting:MacroVariable	Macro variable for nowcasting	Must
Storage:Query_Data:SocialMediaDataQuery	Query (hashtags, users, keywords, date, location etc.)	Should
Storage:Aggregates:QueryAggregates	Query for aggregate (keywords, hashtags, users etc.). Can also query by time, location, or name of aggregates (like sentiment, volume etc.)	Should
Modeling:Build/UpdateModelData	Data needed to build or (if already built) update models. Input data should be macroeconomic time series but also containing geographic location and timespan.	Could
Widgets:SMM:WidgetData	Provides the history and statistics for a specific widget data (MacroVariable parameters).	Could
Output		
InterfaceName:Meth odName:OutputPara meterName	Brief Description	Implementation priority (Must/Should)
Modeling:Nowcasting:NowcastedMacroVariableValue	Macro variable nowcasted value. Output data is in Json	Must
Storage:Query_Data:DataQueryOut	Query output - social media data with specific hashtags, keywords, dates, location etc. In Json.	Should
Storage:Aggregates:	Social media data aggregated by DateTime. In Json.	Should

AggregateOut		
Widgets:SMM:WidgetOutput	SMM widget output (for instance, a graph showing the MacroVariable fluctuation).	Could

Table 8: Description table of the Social Media Mining component

3.3.2.4 Agent Based Macroeconomic Engine

Agent Based Macroeconomic Engine is a software where the economy is represented as a system of interacting agents. The artificial economies allow one to adopt the scientific method by Galileo in the context of social science. The agent-based model of the economy incorporates different economic models characterized by the following features: they are composed of interacting agents, these agents may have behavioral rules and the interaction among the agents means that aggregate phenomena are intrinsically different from individual behavior. In this respect, the network that governs the interactions is crucial. With EURACE engine, large-scale agent-based simulations on high performance computer can be performed and computational experiments with different macroeconomic policy scenarios can be investigated.

The EURACE model represents a fully integrated macro-economy consisting of the real sector, the credit sector, the financial sector, the public sector, the foreign sector, the real estate sector and the environment in order to introduce sustainability aspects.

Serving as the simulation engine of SYMPHONY, it is required to outline its interactions with other major components within an integrated platform. The EURACE needs four major interface components.

In order ABM module to communicate with the other SYMPHONY components, the ABM provides as input/output an XML file. Using the ESB / Orchestrator, the Agent Based Macroeconomic Engine component communicates with the Gamification Engine, the Information Market and the Social Media Mining components.

According to the above characteristics of the component, it has been designed the below component's UML diagram (Figure 16) that describes component's structure by showing the classes, the attributes and the operations (or methods) in order to fulfil the expected functionality and connectivity. More details will be analysed and presented in WP2.

Agent Based Macroeconomic Engine

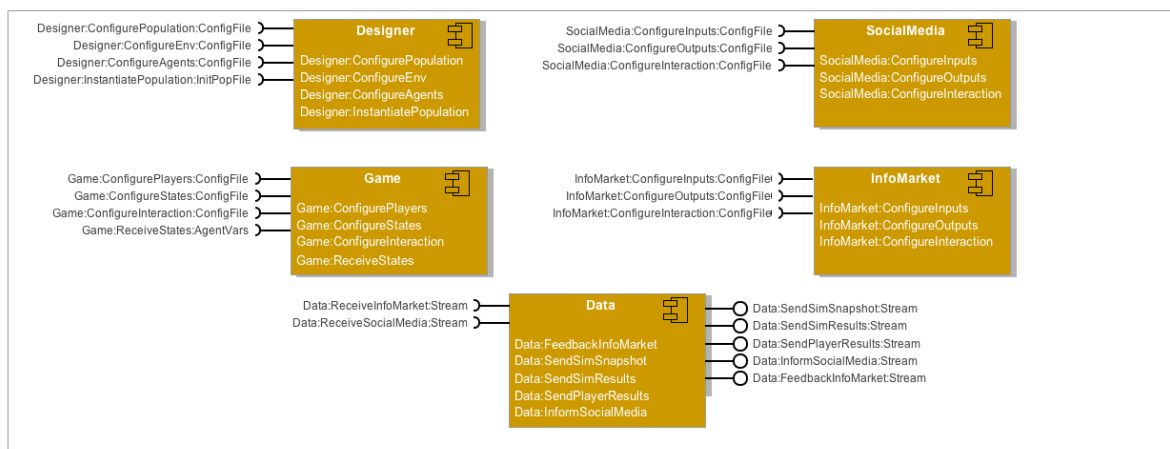


Figure 16: UML Class diagram for the Agent Based Model Component

The Table 9 below presents the analysis of the component that states the name, the brief description and the Implementation priority of the Data Interfaces, the external methods that are available to other components, as well as the input and output parameters of the component.

Data Interfaces		
Name	Brief Description	Implementation priority (Must/Should)
Designer	The component implements all of the user interfaces which are necessary to set up experiments and customize policies and agent types and population distributions as well as all of environmental variables. The interface can be through gamification module.	Must
Game	It describes and implements bi-directional interaction of real agents via the game platform and the simulation engine as a whole.	Must
SocialMedia	It implements the interactions of the engine with information to be fed from social media.	Must
InfoMarket	It implements the interactions of the engine with information to be fed from information market module	Must
Data	The component implements configuration of data transfer from the simulation environment to or from other modules. The orchestrator will provide and coordinate these data interfaces.	Must
Methods		
InterfaceName:MethodName	Brief Description	Implementation priority

		<i>(Must/Should)</i>
Designer:ConfigurePopulation	Configures population size, as of number of households, firms, banks, governments, etc.	Must
Designer:ConfigureEnv	Sets environmental variables	Must
Designer:ConfigureAgents	Initializes agents' balance sheets which refers to a detailed account of all monetary and real assets as well as monetary liabilities.	Must
Designer:InstantiatePopulation	Creates a fully configured population.	Must
Game:ConfigurePlayers	It creates number role and type of players.	Must
Game:ConfigureStates	It selects a set of agent variables, which may be updated by the players. These states will be provided to the simulation engine.	Must
Game:ConfigureInteraction	It sets periodicity and duration of concurrent interactions as well as any required conditional and asynchronous interactions.	Must
Game:ReceiveStates	It receives state variables of agents that need to be integrated within the engine. Hereby, all variables including any decision variables are treated as state variables.	Must
SocialMedia:ConfigureInputs	Configures information channels to social media.	Must
SocialMedia:ConfigureOutputs	Configures information channels for the feed-ins from the social media platform	Must
SocialMedia:ConfigureInteraction	It sets periodicity and duration of concurrent interactions as well as any required conditional and asynchronous interactions in between social media and simulation engine.	Must
InfoMarket:ConfigureInputs	Configures information channels to information markets.	Must
InfoMarket:ConfigureOutputs	Configures information channels regarding expert expectations from information market platform	Must
InfoMarket:ConfigureInteraction	It sets periodicity and duration of concurrent interactions as well as any required conditional and asynchronous interactions in between information market and simulation engine.	Must
Data:ReceiveInfoMarket:	It receives the set data which will be provided and set by information market module.	Must

Data:ReceiveSocialMedia	It receives data from the social media platform that need to be accommodated by the engine.	Must
Data:SendSimSnapshot	It transfers whole snapshot of the population for each designated time point. The data will be transferred to a remote database server. The data transfer process will convert the xml outputs to a designated lightweight streamable format.	Must
Data:SendSimResults	The module transfer any intermediary or final variables or their aggregates to a remote server at designated life time of a run.	Must
Data:SendPlayerResults	The module sends out player relevant variables at designated simulation time points.	Must
Data:InformSocialMedia	The module posts social media related information.	Should
Input		
InterfaceName:MethodName:ParameterName	Brief Description	Implementation priority (Must/Should)
Designer:ConfigurePopulation:ConfigFile ¹ Optional. The Configuration file.		Should
Designer:ConfigureEnv:ConfigFile	Optional. The configuration file.	Should
Designer:ConfigureAgents:ConfigFile	Optional. The configuration file.	Should
Designer:ConfigureAgents:ConfigFile	Optional. The configuration file.	Should
Designer:InstantiatePopulation:InitPopFile	Optional. A ready to use population file.	Should
Game:ConfigurePlayers:ConfigFile	Optional. The configuration file.	Should
Game:ConfigureStates:ConfigFile	Optional. The configuration file	Should
Game:ConfigureInteraction:ConfigFile	Optional. The configuration file	Should
Game:ReceiveStates:AgentVars	The methods will read any updated states of agent variables.	Must
SocialMedia:ConfigureInputs:ConfigFile	Optional. The configuration file.	Should

¹ All methods starting with 'Configure' need to have interactive user interface. Alternatively a file can be provided where it contains all configurations in a designated format.

SocialMedia:ConfigureOutputs:ConfigFile	Optional. The configuration file	Should
SocialMedia:ConfigureInteraction:ConfigFile	Optional. The configuration file	Should
InfoMarket:ConfigureInputs:ConfigFile	Optional. The configuration file.	Should
InfoMarket:ConfigureOutputs:ConfigFile	Optional. The configuration file	Should
InfoMarket:ConfigureInteraction:ConfigFile	Optional. The configuration file	Should
Data:ReceiveInfoMarket:Stream	It reads in information market data to be used by the engine in simulation run time	Must
Data:ReceiveSocialMedia:Stream	It reads in expectations driven by social media module. It will be used by the engine at simulation run time.	Must
Output		
InterfaceName:MethodName:ParameterName	Brief Description	Implementation priority (Must/Should)
Data:SendSimSnapshot:Stream	The data stream which holds the snapshot of a stream at designated time points.	Must
Data:SendSimResults:Stream	The data stream contains files which holds analyses results.	Must
Data:SendPlayerResults:Stream	The data stream which holds data relevant to each player.	Must
Data:InformSocialMedia:Stream	The data stream which may include information which may have impact on speculations at relevant social media sites.	Should
Data:FeedbackInfoMarket:Stream	The data stream which may include information which may have impact on information market speculations.	Should

Table 9: Description table of the Agent Based Macroeconomic Engine component

3.3.2.5 Platform Orchestrator /ESB

An ESB [6][7] is a software architecture construct, which provides fundamental services for complex architectures via an event-driven and standards-based messaging engine (the bus). Developers typically implement an ESB using technologies found in a category of middleware infrastructure products, usually based on recognized standards. An ESB takes the complexity out of integration, providing connectivity to a wide range of technologies and creating services that can be reused across an organization. An ESB does not itself implement SOA, but provides the features with which one may implement such.

Developers can exploit the features of an ESB in order to integrate applications and services without custom code. Developers can shield services, regardless of location, from message formats and transport protocols. Data can be transformed and exchanged across varying formats and protocols.

The below figure (Figure 17) depicts the architecture of a SOA system implemented with an ESB. The system integrates new and legacy applications that needs to communicate with each other and exchange information. A variety of technologies, protocols and message formats are used such as SOAP messages over HTTP, data transformations using XSLTs, etc.

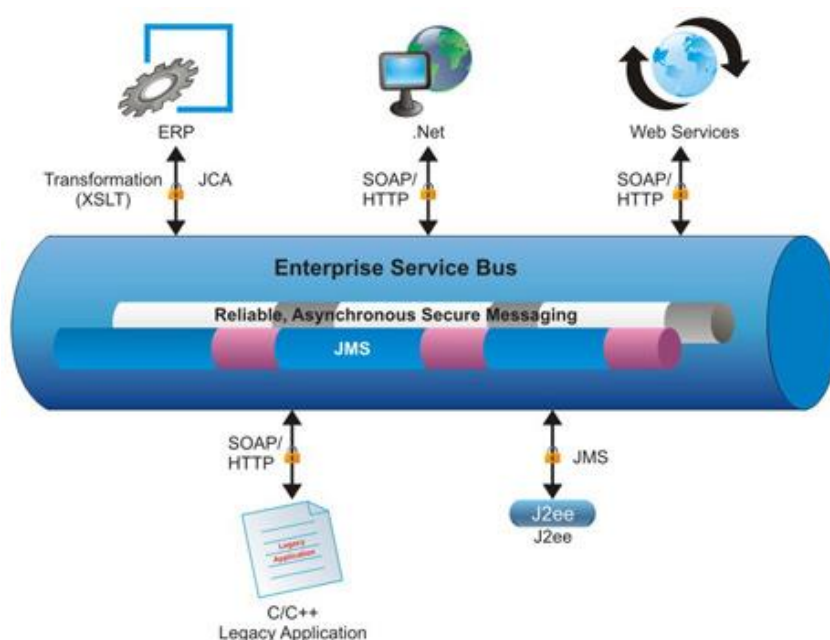


Figure 17: An example topology for ESB (please refer to [8])

The ESB hides the implementation details of one module from other components. All components send and receive messages to the ESB as it is responsible for delivering the messages to the recipients in the appropriate format.

In case of a momentarily unavailable component or a legacy component is being replaced by a new one, the system can run without affecting the functionality of the other components. The ESB will be implemented in a way that it would be able to wait for a component to be available again in order to receive/deliver a message.

Orchestrator Platform aims to provide application orchestration for the SYMPHONY project. Application orchestration is the process of integrating two or more applications and/or services together to automate a process or synchronize data in real-time. Often, point-to-point integration may be used as the path of least resistance. However, point-to-point integration always leads to a complex tangle of application dependencies (often referred to as "spaghetti code") that is very hard to manage, monitor and maintain. Application orchestration provides:

- an approach to integration that decouples applications from each other
- capabilities for message routing, security, transformation and reliability
- most importantly, a way to manage and monitor your integrations centrally.

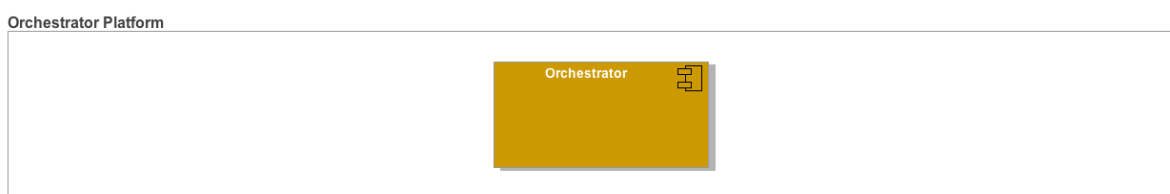


Figure 18: UML Class diagram for the Orchestrator Platform

The Orchestrator component is only used for internal use and do not expose any input or output for other software components to consume.

3.4 Data Communication

As depicted in the previous sections, all main software components are composed of sub-components that perform specific tasks and expose inputs and outputs. In order to function properly, the sub-components need data as inputs and exports processed data as outputs. At the figures mentioned below, the communication lines between the various sub-components are shown and analysed.

3.4.1 Dashboard's Communication

The below figure (**Errore. L'origine riferimento non è stata trovata.****Errore. L'origine riferimento non è stata trovata.**) depicts the required inputs for the “Dashboard software component” which consists of the WidgetHandler sub-component. The three exposed inputs of the WidgetHandler sub-component consume data from the following sub-components:

- **Game: Widgets** and specifically exposed output Widgets:GameWidget:WidgetData
- **IM: Widgets** and specifically exposed output Widgets:MakeHistory:WidgetData
- **SMM: Widgets** using exposed output Widgets:SMM:WidgetData

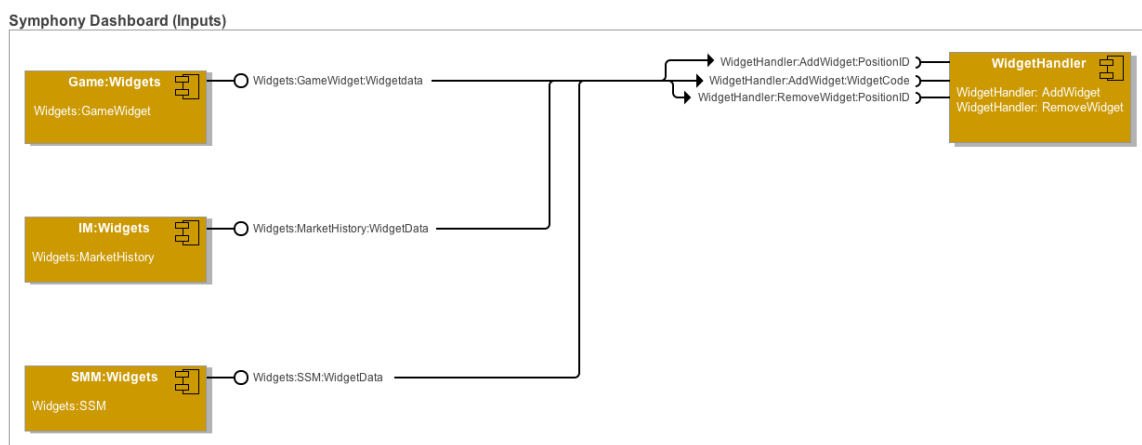


Figure 19: Required inputs for the “Dashboard software component”

For the exact data schema interchange the reader may refer at the Chapter 4 Data Description.

3.4.2 Gamification Engine’s Communication

The **Errore. L'origine riferimento non è stata trovata.** figure below presents the required inputs for the “Gamification Engine software component” which consists of the Scenarios and GameData sub-components. The exposed inputs consume data from the following sub-components:

- **Game: Widgets** and specifically exposed output Widgets:GameWidget:WidgetData
- **IM: Widgets** and specifically exposed output Widgets:MakeHistory:WidgetData
- **SMM: Widgets** using exposed output Widgets:SMM:WidgetData

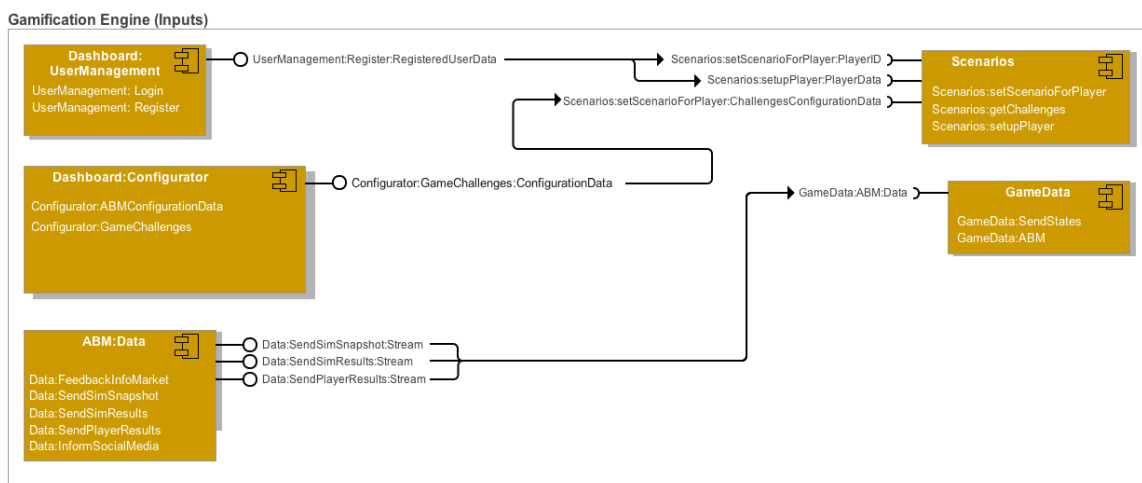


Figure 20: Required inputs for the “Gamification Engine software component”

The data schema could be found at chapter 4 of this document.

3.5 System Interactions & Information Flow

In this section, the architecture is described through components' data interactions and through block/sequence diagrams stressing the interactions between the different components that compose the Platform, such as the providing of input data using different software component or the consuming of output data using different/alternative software component. The "Components Interactions and Workflows" section shows the main interaction between components and interfaces defined in subchapter 3.2. Following the implementation that each component should provide in each layer of the SYMPHONY high level architecture, this section focuses on the interaction implementations, which enable the different SYMPHONY's components interfacing with each other and with the target stakeholders to implement the case studies where are described in section 2.3 and accomplish the needed functionalities.

3.5.1 Accessing the platform

As shown in Figure 21 - User Registration sequence diagram, the web guest user accesses the home page and through the registration module of the homepage submits his/her User Data (Username, password, Name, Surname, etc.) in order the platform to validate of the user data. As soon as the platform perform the validation, it notifies the user for the success of the registration process.

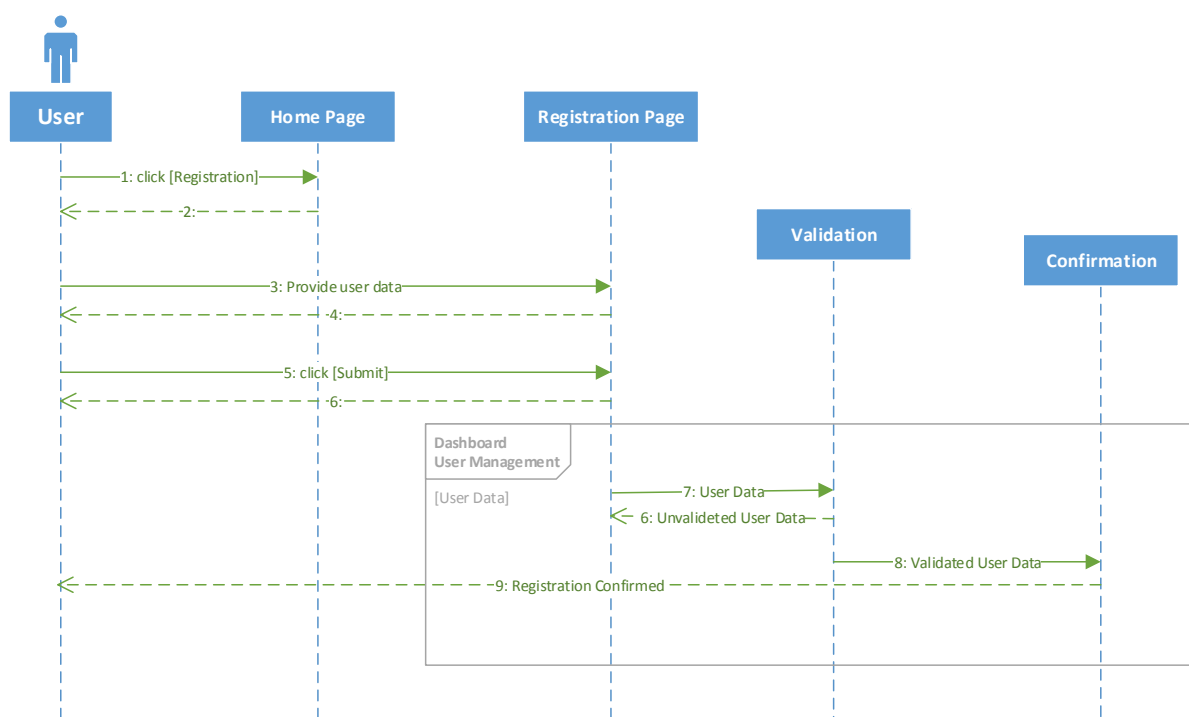


Figure 21 - User Registration sequence diagram

At the following sequence diagram (Figure 22 - User Login sequence diagram) is depicted the login to the SYMPHONY platform process of a user. As the user accesses the website submits his/her User credentials in order the platform to perform user's validation. As soon as the platform perform the validation, the user accesses the dashboard with the appropriate features according to his/her user role.

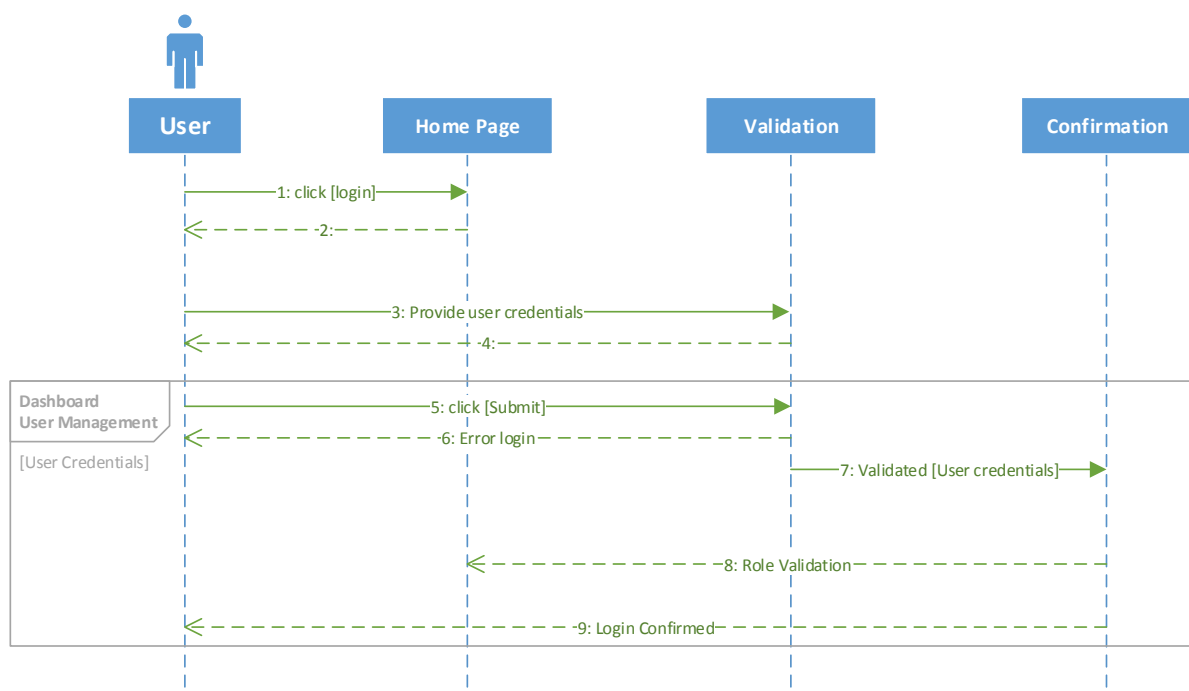


Figure 22 - User Login sequence diagram

3.5.2 Viewing Information

At Figure 23 is depicted the sequence diagram of the combination of Simple Users' expectations from social media & View sentiment analysis of public opinions for a domain Use Cases. This diagram starts with the login process which is presented at the earlier subsection. As the validated user logs in with the role of an Advanced User is able to Search for Citizens expectations that are expressed in social media by submitting through dashboard to SSM module the search parameters that are required. As the SSM module finishes the request processes, it returns a list of Simple Users' expectations. As soon as the list is available to the Advanced User's dashboard, the user can select a public opinion from the list, request and view the results of its sentiment analysis.

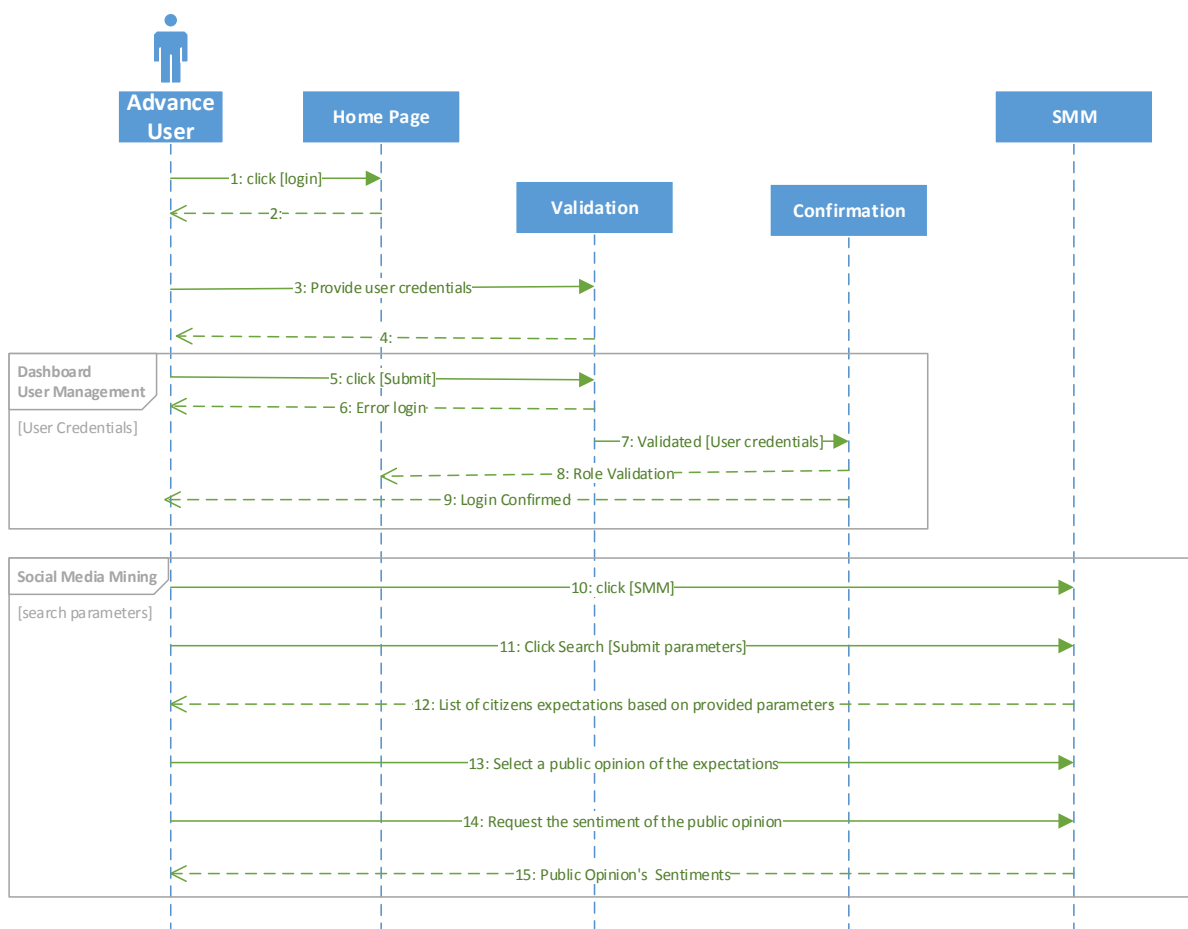


Figure 23 - Simple users' expectations from social media & View sentiment analysis of public opinions for a domain sequence diagram

At the following sequence diagram, which is depicted at Figure 24, is presented the process where an Advanced User has to follow in order to View experts expectations on specific events. In more details, as the user initiates a Market by submitting through the dashboard to the IM module the needed parameters, the IM notifies the end users (experts) that as the Market has been initiated they are able to buy or sell contracts for the initiated Market. After contracts' trading the policy maker can request to get a graphical representation of the IM contract/outcomes.

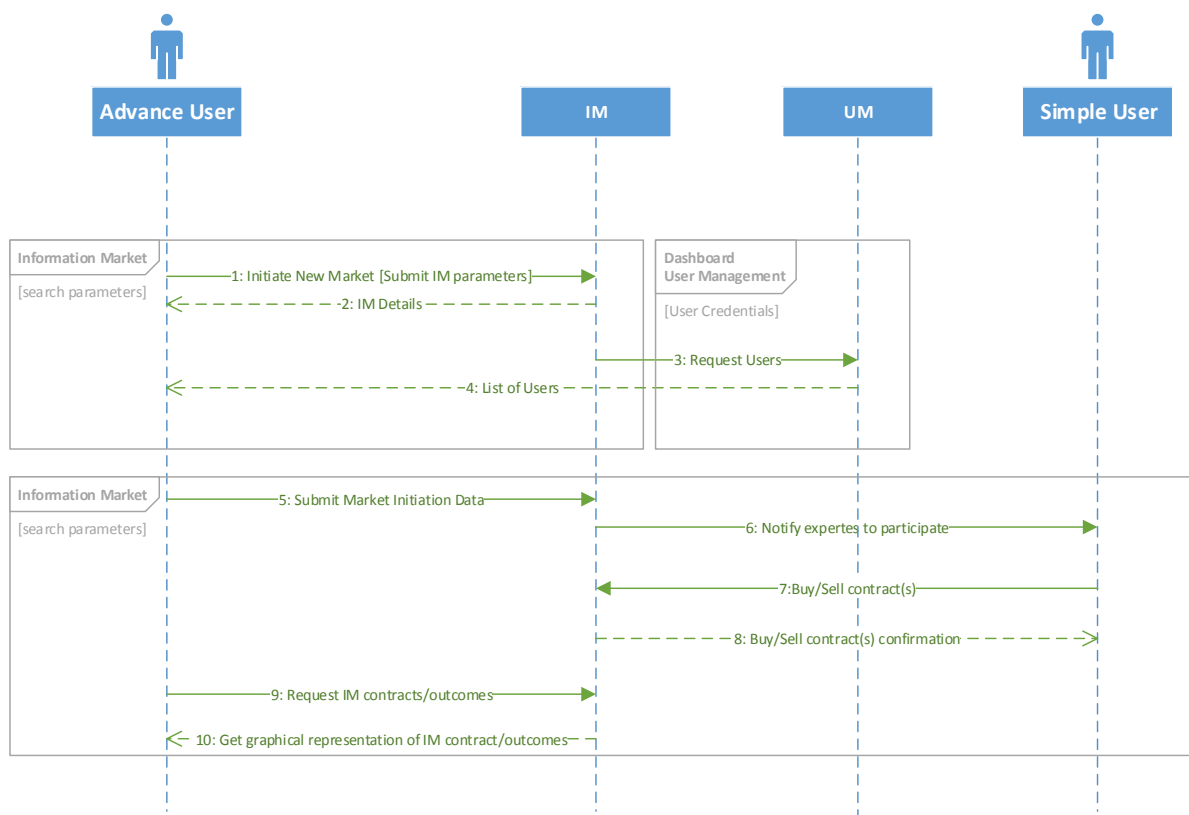


Figure 24 - View experts expectations on specific events sequence diagram

The following figure depicts the sequence diagram of the process to get a graphical view of Citizens Expectations. The process that an Advanced User has to follow, starts with the initiation of a new game by submitting the required configuration parameters/values to the Gamification module and to the ABM module through the dashboard. As the initiation values are submitted by the Advanced User to the aforementioned modules, the ABM module requests information (data) from IM and SMM modules in order to have all the needed information to run the model. The UM provides the list of the available end users in order to be invited to participate at the new game. As the end users accepts the invitation to play, they are able to start playing the game by changing the available game parameters. Based on the new parameters initiated by the end users the Gamification engine will provide a graphical representation of the

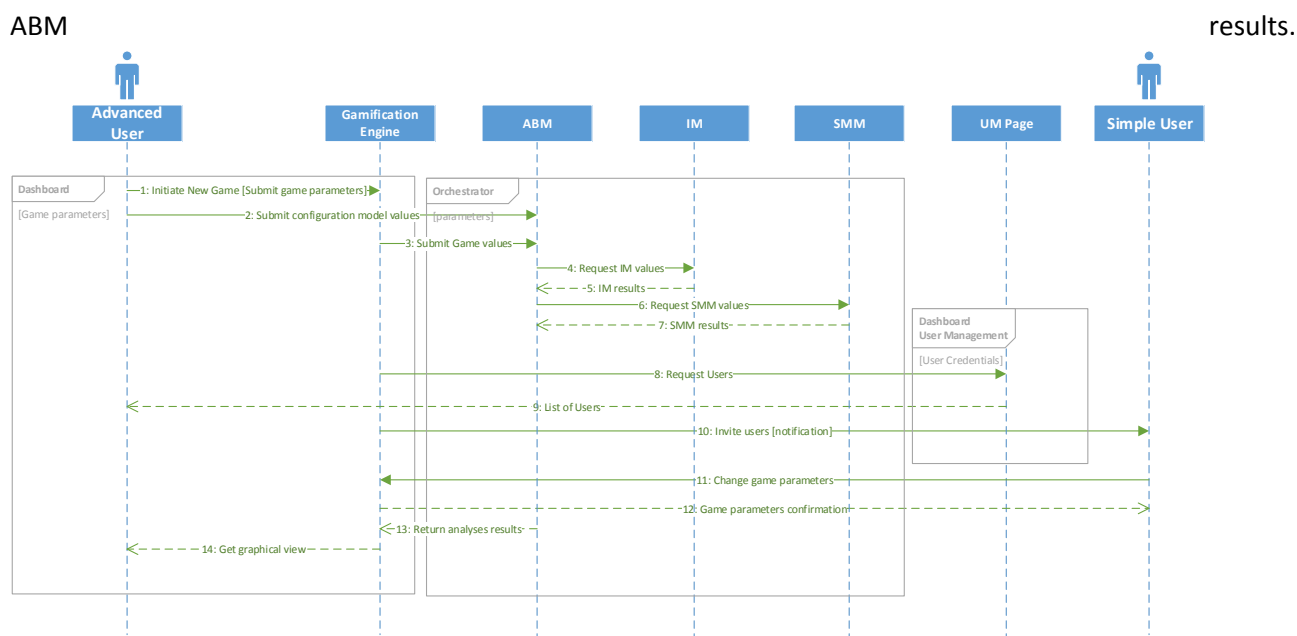


Figure 25 - Get a graphical view of Citizens Expectations sequence diagram

3.5.3 Acquiring Estimations

At the below sequence diagram, which is depicted at Figure 26, is presented the process where an Advanced User acquires an estimation of an event evolution in the future. As the Advanced User submits the search parameters, the dashboard requests from the IM module the available events in a list. The IM module returns the requested list and the Advanced User is able to request a graphical representation of the simulation results of a specific event for a specific period in the future. The graphical representation request is initiated from the dashboard to the IM module which continue by its turn in requesting models' data from the ABM simulation based on a specific period. As the IM module receives the simulation results, it will be able to reply to the advanced user's request with a graphical representation of event's evolution in the future.

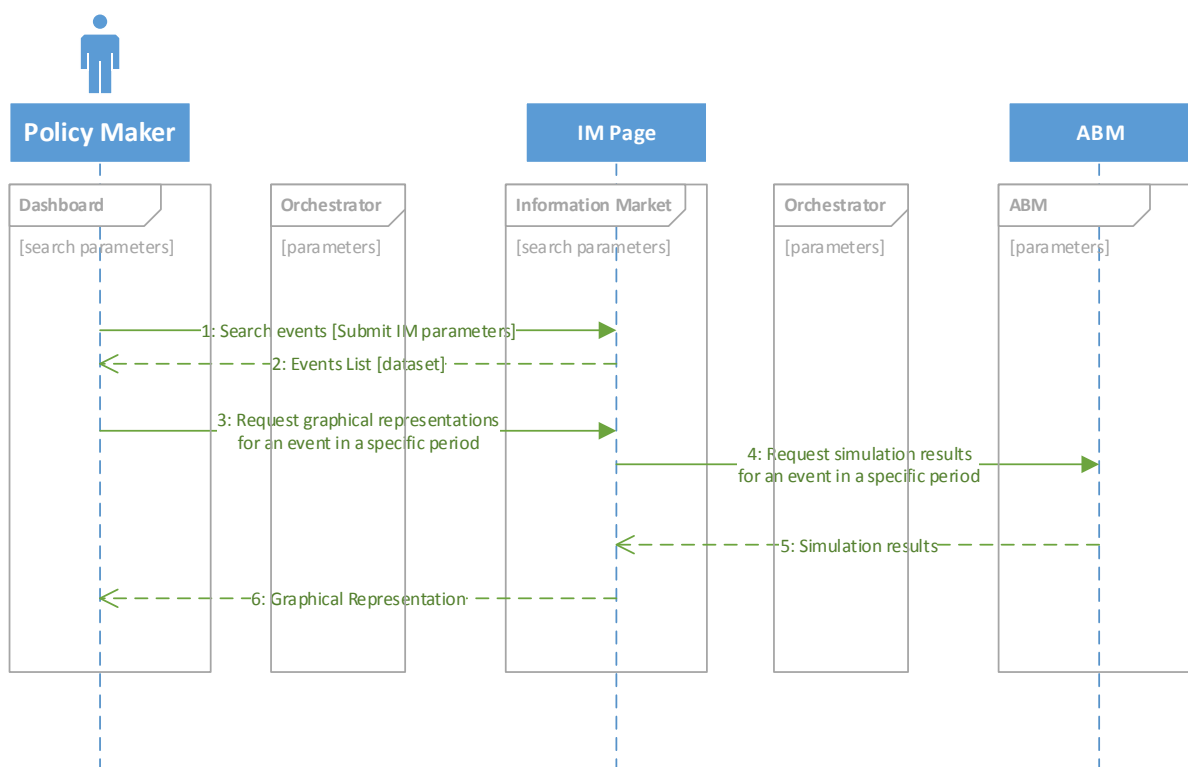


Figure 26 - Acquire an estimation of event evolution in the future sequence diagram

At Figure 27 is depicted the sequence diagram for acquiring an estimation of how social media signals aligns with macroeconomic trends process. The process starts when an Advanced User submits the search parameters to the SMM module requesting a list of the macroeconomic trends in social media. As soon as the advanced user receives the requested list with the trends, he/she selects a trend and requests from the SMM module for a graphical representation including simulation results of the trend for a specific period. These simulation results are being provided to the SMM by the ABM module in order to depict the ration of the alignment between social media and macroeconomic trends.

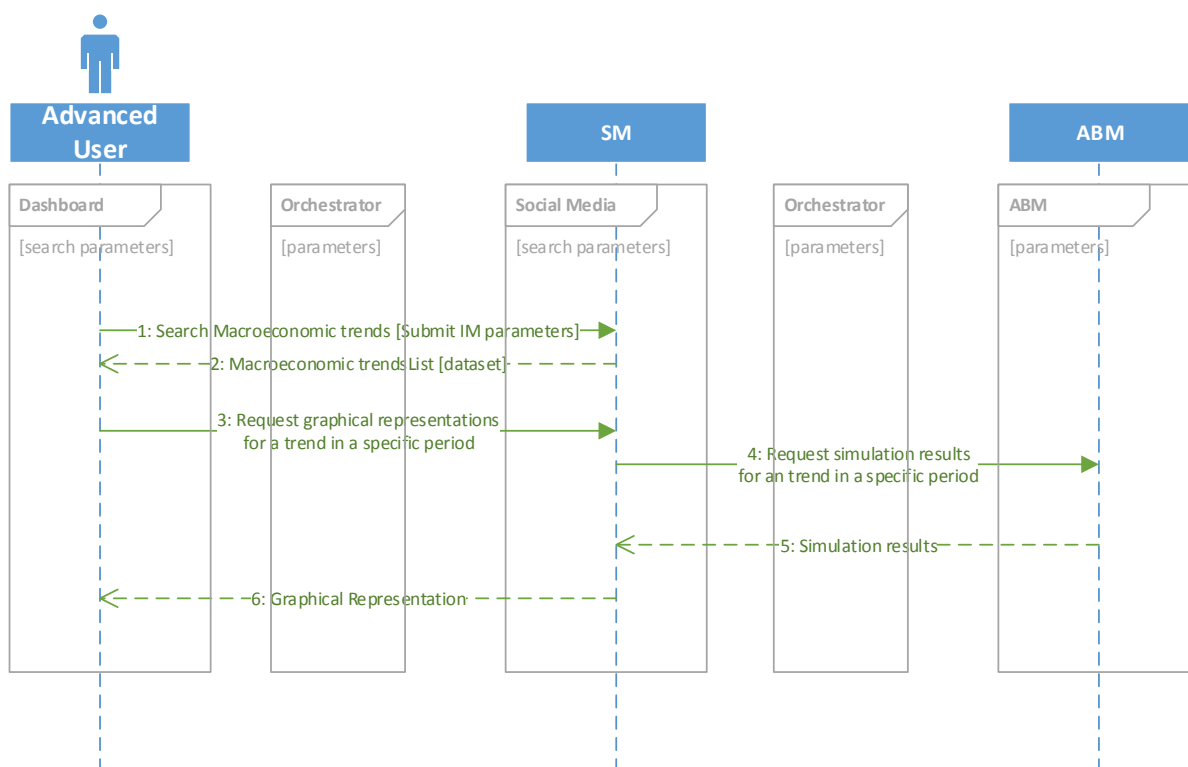


Figure 27 - Acquire an estimation of how social media signals align with macroeconomic trends sequence diagram

The following sequence diagram, which is depicted at Figure 28, presents the process when an Advanced User acquires an estimation of the impact of an economy game scenario. The Advanced User submits through the dashboard the search parameters to request a list of the available economy game scenarios in the platform. The dashboard sends a request to the Gamification engine in order to return the list with the available economy game scenarios. The Advanced User now is able to request from the Gamification Engine the impact of the economy scenario by selecting it and submitting it to the Gamification Engine which requests for simulation data by the ABM module in order to analyze and depict the impact of the selected economy game scenario.

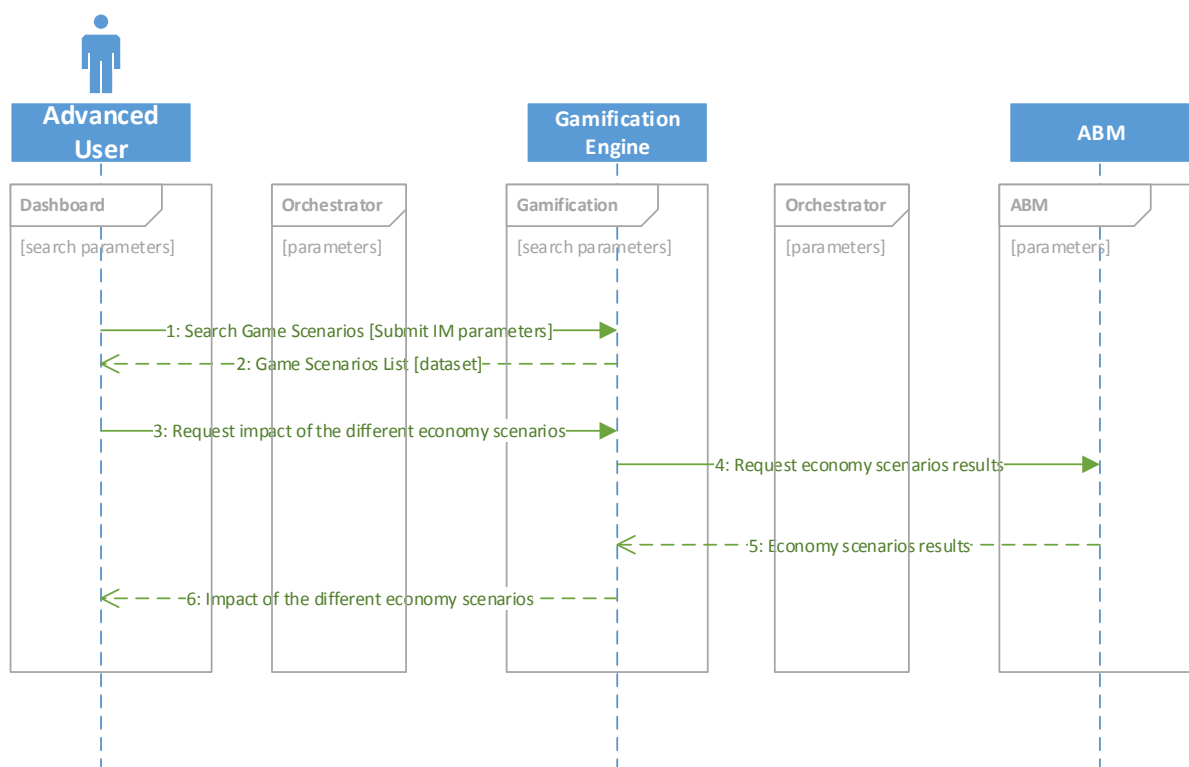


Figure 28 - Acquire an estimation of the impact of the different economy scenarios sequence diagram

4 Data Description

The Chapter (3.2 SYMPHONY Architecture) introduces in detail the SYMPHONY Architecture and presents the various and heterogeneous software components that compose the SYMPHONY platform. All software components defined in SYMPHONY Architecture consist of various sub-components that perform specific tasks and expose inputs and outputs that are used to exchange data between them.

Later in this Chapter the data communication schema is presented in detail. The inputs and outputs are analysed deeply exposing the hidden information they contain, composing the data exchange scene of the various sub-components.

Below the data schema is presented grouped by the main software components.

4.1 SYMPHONY Dashboard

Component Output	Field Name	Field Type (Optional)	Description
Configurator:ABMConfigurationData:ConfigurePopulation	ModelPop	File	A predefined ABM population file, which contains information regarding number of governments, banks, firms, households, players and their roles, etc.
Configurator:ABMConfigurationData:Conf	Experiment	File	An XML file which is used by ABM

gureEnv	XML		engine. The file contains information such as number of iterations to be run, a predefined runtime mode, experimental policy parameters and experimentation values, etc.
Configurator:ABMConfigurationData:ConfigureAgents	ModelXML	File	A validated XML file which contains variables and states of the EURACE Model to be run
Configurator:ABMConfigurationData:InstantiatePopulation	ModelInit	File	A predefined ABM initialization file, which holds initial values of each agent memory variable and model parameters.
Configurator:ABMConfigurationData:ListOfInformationMarketExpectations	InfoMarInXML	File	The XML file which contains list of variables to be read from information market component
Configurator:ABMConfigurationData:ListOfSocialMediaExpectations	SocMedInXML	File	The social media input vars. XML file which contains information regarding variables to be read from social media component
Configurator:GameChallenges:ConfigurationData	ID	CHAR(36)	GUID to identify the challenge
	Fields	Array of key=>value pairs	Config fields for this player's challenge
UserManagement:Login:LoginResult	LoginResult	Boolean	System answer to a user's login providing credentials
UserManagement:Register:RegisteredUserData	ID	String	Some method of identifying the user behind this player
	First Name	String	User's first name
	Last Name	String	User's last name
Reports:showReport:Output	Graph	Widget	A dashboard widget that graphically presents a graph as a report to system usage
Component Input	Field Name	Field Type (Optional)	Description
WidgetHandler:AddWidget:PositionID	PositionID	Predefined list	The position in dashboard where the widget will be placed. PositionID is an int of a predefined list containing all available positions in the dashboard

WidgetHandler:AddWidget:WidgetCode	WidgetCode	Text	Javascript code containing the initiator that is placed in the dashboard it will generate a widget
WidgetHandler:RemoveWidget:PositionID	PositionID	Predefined list	The position in the dashboard that should be cleared from existing widgets
UserManagement:Login:Credentials	Username	Text	User's username used to identify himself in the SYMPHONY platform
	Password	Hidden Text	User's password used to verify his identity in the SYMPHONY platform
UserManagement:Register:Credentials	Username	Text	User's username used to identify himself in the SYMPHONY platform
	Password	Hidden Text	User's password used to verify his identity in the SYMPHONY platform
	Email	Text	User's email used for system communication purposes
	First Name	Text	User's real first name
	Last Name	Text	User's real last name
	Role in the system	Predefined list	User's role in the SYMPHONY platform
Reports:ShowReport:ReportID	ReportID	Predefined list	The id of the report the user/admin asks to see / access
Reports:ShowReport:DateTimeRange	DateTimeRange	Date Time range	The date time range of the data to analyse
Reports:ShowReport:ReportType	ReportType	Predefined list	A predefined list of report types

4.2 Gamification Engine

Component Output	Field Name	Field Type (Optional)	Description
Scenarios:getChallenges:ChallengeList	ID	CHAR(36)	GUID to identify the challenge
	Name	Text	Name to display to the user

Component Output	Field Name	Field Type (Optional)	Description
	Description	Text	Description to display to user
	ConfigFields	TBD	Some method of telling the dashboard what can be configured for this challenge.
GameData:SendStates:Agentvars	TBD		To be agreed upon with the ABM team
Component Input	Field Name	Field Type (Optional)	Description
Scenarios:setScenarioForPlayer:ChallengesConfigurationData	ID	CHAR(36)	GUID to identify the challenge
	Fields	Array of key=>value pairs	Config fields for this player's challenge
Scenarios:setupPlayer:PlayerData	ID	String	Some method of identifying the user behind this player
	First Name	String	User's first name
	Last Name	String	User's last name

4.3 Agent Based Model

Component	Field Name	Field Type (Optional)	Description
INPUTS			
Designer:ConfigurePopulation:ConfigureFile	ModelPop	File	A predefined ABM population file, which contains information regarding number of governments, banks, firms, households, players and their roles, etc.
Designer:ConfigureEnv:ConfigureFile	Experiment XML	File	An XML file which is used by ABM engine. The file contains information such as number of iterations to be run, a predefined runtime mode, experimental policy parameters and experimentation values, etc.
Designer:ConfigureAgents:ConfigureFile	ModelXML	File	A validated XML file which contains variables and states of the EURACE Model to be run
Designer:InstantiatePopulation:InitPopFile	ModelInit	File	A predefined ABM initialization file, which holds initial values of each agent memory variable and model parameters.
SocialMedia:ConfigureInputs:ConfigureF	SocMedInX	File	The social media input vars. XML

Component	Field Name	Field Type (Optional)	Description
INPUTS			
ile	ML		file which contains information regarding variables to be read from social media component
SocialMedia:ConfigureOutputs:ConfigureFile	SocMedOutXML	File	The social media output vars. An XML file which contains information regarding variables to be send back to social media component
SocialMedia:ConfigureInteraction:ConfigureFile	SocMedIntXML	File	The social media interaction scheme. An XML file which contains information on interaction schema in between ABM and social media components. The schema will cover, for instance, time points of feedbacks retrieved from social media I and feedbacks to social media component regarding economical indicators.
Game:ConfigurePlayers:ConfigureFile	GamePlayerXML	File	The XML file that contains description and roles and number of players to be accommodated by ABM engine.
Game:ConfigureStates:ConfigureFile	GameStateXML	File	The XML file that contains set of variables and decision rules to be determined by game players.
Game:ConfigureInteractions:ConfigureFile	GameIntXML	File	The XML file which contains information regarding interaction of the game engine and the ABM engine
Game:ReceiveStates:Agentvars	GameData	File	The data stream file which contains values of variables set by the player
InfoMarkets:ConfigureInputs:ConfigureFile	InfoMarInXML	File	The XML file which contains list of variables to be read from information market component
InfoMarkets:ConfigureOutputs:ConfigureFile	InfoMarOutXML	File	The XML file which contains information regarding variables to be send back to information market component.
InfoMarkets:ConfigureInteractions:ConfigureFile	InfoMarIntXML	File	The XML file which contains information on interaction schema in between ABM and information market component.

Component	Field Name	Field Type (Optional)	Description
INPUTS			
Data:ReceiveInfoMarket:Stream	InfoData	File	The data stream file which contains values of variables to be read from information market component.
Data:ReceiveSocialMedia:Stream	SocMedData	File	The data stream file which contains values of variables to be read from social media component.
OUTPUTS			
Data:SendSimSnapshot:Stream	ABMDataStates	File	The data stream file contains memory states each single agent at designated time points of the simulation run time
Data:SendSimResults:Stream	ABMDataResults	File	The data stream file contains time series data for the designated economic variables
Data:SendPlayerResults:Stream	ABMDataPlayers	File	The data stream file contains time series data regarding the memory variables of each game player.
Data:InformSocialMedia:Stream	ABMDataSocMed	File	The data stream file contains values of economy variables that may specifically produced for social media component
Data:FeedbackInfoMarket:Stream	ABMDataInfoMar	File	The data stream file contains values of any economy variables that may specifically produced for information market component

4.4 Information Markets

Component Output	Field Name	Field Type (Optional)	Description
Expectations:NowCasting:Contract_Value	Current Value	Decimal	Contract's current value
Widgets:MarketWidget:WidgetData	PositionID	Integer	We provide only the widget's position id. It is a self-contained widget
Component Input	Field Name	Field Type (Optional)	Description
Expectations:NowCasting:ContractID	ContractID	Integer	Contract's id

4.5 Social Media Mining

Component Output	Field Name	Field Type	Description
------------------	------------	------------	-------------

Component Output	Field Name	Field Type	Description
Storage:QueryData:DataQueryOut	SocialMedia QueryResult	List/JSON array	Social media query data result represented as JSON array
	TimeWindo w (optional)	DateTime (or JSON array of DateTime-s)	Time window for particular query
	Location (optional)	String to define country/pla ce	Location for particular query
Storage:Aggregates:AggregatesOut	AggregateR esult	Float/String (depending on aggregate)	Aggregate value
	TimeWindo w (optional)	DateTime (or JSON array of DateTime-s)	Time window for aggregate query
	Location (optional)	String to define country/pla ce	Location for aggregate query
Modeling:NowCasting:NowCastedMacroVa riableValues	Nowcasted MacroVaria bleValue	Float Array	Macro variable values
	TimeWindo w (optional)	DateTime (or JSON array of DateTime-s)	Time window for nowcasting
	Location (optional)	String to define country/pla ce	Location for nowcasting
Widgets:SSM:WidgetData			
Component Input	Field Name	Field Type	Description
Storage:QueryData:SocialMediaDataQuery	Query	JSON String	Input query
	TimeWindo w (optional)	DateTime (or JSON array of DateTime-s)	Time window for particular query
	Location (optional)	String to define country/pla ce	Location for particular query
Storage:Aggregates:QueryAggregates	AggregateN ame	String	Aggregate name
	TimeWindo w (optional)	DateTime (or JSON	Time window for aggregate query

Component Output	Field Name	Field Type	Description
		array of DateTime-s)	
	Location (optional)	String to define country/place	Location for aggregate query
Modeling:NowCasting:MacroVariables	MacroVariableName	String	Macro variable name
	TimeWindow (optional)	DateTime (or JSON array of DateTime-s)	Time window for nowcasting
	Location (optional)	String to define country/place	Location for nowcasting
Modeling:Build_Models:ME_Time_Series	MacroVariableName	String	Macro variable name
	MacroTimeSeriesData	CSV/JSON array(what is available)	Macroeconomic time series data for particular macro variable
	TimeWindow (optional)	DateTime (or JSON array of DateTime-s)	Time window for building models
	Location (optional)	String to define country/place	Location for building models
Modeling:Update_Models:ME_Time_Series	MacroVariableName	String	Macro variable name
	MacroTimeSeriesData	CSV/JSON array(what is available)	Macroeconomic time series data for particular macro variable
	TimeWindow (optional)	DateTime (or JSON array of DateTime-s)	Time window for building models
	Location (optional)	String to define country/place	Location for building models

5 Implementation Plan

5.1 UI Design Principles

5.1.1 Human-Centered Design Methodology

The SYMPHONY platform front-end will be based on the Human-Centered Design. The Human-Centered Design is a methodology aimed at designing interactive systems taking into account, on one hand the technical problems of the System-Centered Design, and on the other hand the social issues involving the users, trying to understand more precisely the scope of the system.

ISO 13407 defines the Human-Centered design processes for interactive systems as:

The Human-Centered design is an approach for the development of interactive systems specifically oriented to the creation of usable systems. It is a multidisciplinary activity including techniques of human factors and ergonomic. [...] To apply the ergonomic to the system design requires to take into account the abilities, the limitations and the needs of the human beings. The Human-Centered systems support the users and motivate them to learn. The benefits include a higher productivity, an improvement in the working quality, a reduction of support and training costs and a higher satisfaction of the users².

Within this framework, a product can be considered mature if it functions correctly, provides all the functionalities required and if it is easy to use. At this stage of the design maturity process, the functionalities of the product should be adapted to the different typologies of users and to their needs and aims, in order to reach the different contexts of use. This methodology achieves a higher level of maturity when the product becomes “invisible” during its use, enabling the user to focus not on the tool but on the task that he is completing.

The traditional system design takes into account the system and its functional requirements, the User Centered Design through the definition of use cases identifies the tasks that the different typologies of users should implement through the system. This approach is based on the interactions between the user and the system, the first actor’s needs constitute the starting point for the System Design³.

More in detail, the User Centered Design (or Human-Centered Design) is divided in the following phases:

- Identification of the user needs
- Data Gathering
- Definition of the User Requirements
- Prototyping
- Usability test

²http://www.iso.org/iso/catalogue_detail.htm?csnumber=21197

³Roberto Polillo, Facile da usare, Apogeo, Milano, 2010, <http://www.scribd.com/doc/71062391/E-Cap-3-Usabilita>.

Identification of the user needs

Within the User-Centered approach, the first step in order to start designing the system, is to identify the typologies of users by specifying also their characteristics. The same relevance should be attributed to the context in which the system will be used and to the potential scenarios of usability. The factors that may influence the user during the use of the system should be considered.

Data Gathering

The identification of the requirements can be developed by using different tools, combined to obtain specific data.

More in detail the tools that can be used are:

- **Quantitative analysis**, such as interviews developed through questionnaires (by phone, online or paper survey). The questionnaires can be structured (closed questions), semi-structured (open questions with predefined answers) or non-structured (open questions).
- **Qualitative analysis**. Focus group technique allows to analyze more in detail the characteristics of the project to be developed within a specific group of people, similar to the target of users.
- **Field Survey**, to gather information on the context on which the user carries out his activities.
- **Analysis of the competitors and Best Practices Analysis**, allow to identify the best effective solutions, in order to highlight the strengths and weaknesses of the product to be developed, and to compare it to the products positioned in the same market.

Definition of the User Requirements

The requirements of the system are defined based on the user needs in order to clearly identify what are the functionalities and attributes of the final product. It will be also indicated how the system should act according to the user⁴.

The required functionalities are based on a set of interactions aimed to achieve an objective among one or more actors in the system. A use case is started by an actor for a specific aim and it ends when the goal is achieved. For each objective the user should perform several operations and sub-operations (the sequence required to achieve the scope). In order to identify these actions, usage scenarios have been described using a main course of events, each of them indicating a single action.

Prototyping

The prototype is the “representation of a product, of a system, or of a part of the system, that even if it is limited, can be used for its evaluation”⁵. It is not required that the model is completed, but can be also a model that simulates the functioning of the system (mock-up).

⁴ Catherine Courage & Kathy Baxter, “Understanding Your Users. A practical guide to user requirements. Methods, Tools, & Techniques.”, Morgan Kaufmann.

⁵ISO 13407

The prototyping phase allows to:

- show the decisions made in course of the project;
- identify several concepts for the final choice;
- take into account the feedback of the users from the beginning of the system design process;
- evaluate several alternatives of the project and different projects;
- improve the quality and completeness of the functionalities of the project.

The prototypes can be classified on the base of the objectives that the designer wants to achieve. For example, a prototype can be designed to evaluate the interface of the product intended as all the interaction typologies between the user and the product (look&feel prototype). We can also design prototypes that evaluates the role of the product within the life of the user (role prototype) and the technical aspects related to the implementation of the product (implementation prototype).

The prototypes can be classified by taking into account the current phase of the project. The low fidelity static prototype has a low cost of development and allows the evaluation of several design-concepts. The high fidelity interactive prototype is oriented to the final product and its scheme provides all the functionalities of the system.

The choice between the typologies of prototypes to implement depends on the typology of the system. Anyway, it is always preferable to first develop a low fidelity prototype to create several alternatives for the user and to execute fast modifications.

Usability test

The usability it is an indicator of the quality⁶ of the product, which shows if it is easy to use. An operational definition of the usability is provided by ISO 9241:

- *The usability of a product is the grade on which it can be used by specific users to effectively and efficiently achieve specific objectives in a predetermined user context⁷.*
- The usability can be defined on the base of three independent variables enabling the evaluation of a product.
- The efficacy is the accuracy and completeness through which the users achieve specific objectives⁸. It defines the grade of accuracy through which the system achieve the objectives identified by the users.
- The efficiency describes the amount of resources that the user spends to accurately complete a task. There are several typologies of resources.

⁶ Jakob Nielsen, Hoa Loranger, Web Usability 2.0, L'usabilità che conta, Apogeo, 2006, Milano, pag. XVIII.

⁷Traduzione italiana dal testo inglese "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (ISO 9241-11:1998), tratto da Roberto Polillo, Facile da usare, Apogeo, Milano, 2010.

⁸ Ibidem.

- The satisfaction is related to the comfort and to the acceptability of the system⁹.

The attributes required to develop a usable product depends on the typology of the user, from the task that he has to complete and on the context of usage.

Through the usability tests we can identify which are the potential usability issues. The test of the functionalities can be developed through a prototype, by evaluating different aspects and doing the modifications required before the implementation.

The different phases of a usability test:

- to develop an entry questionnaire to define the user profile, for instance the confidence of the user with several technologies;
- to develop an exit questionnaire to evaluate the interaction, by proposing to the user a set of questions related to the usability of the functionalities;
- to identify several tasks for all the functionalities of the system enabling the user to simulate a real user experience.

The number of users for the testing phase should be from 10 to 15: this number of users will show the higher percentage of issues of the system.

The evaluation of usability of a product can be replicated several times within the context of the iterative process. Tests can be developed also after the design of the system to test the real usability. Although usability can have many aspects, it is often associated with the following 5 attributes¹⁰:

- *Easy to learn*: The user can quickly go from not knowing the system to getting some work done with it.
- *Efficient to use*: Once the user has learned the system, a high level of productivity is possible.
- *Easy to remember*: The infrequent user is able to return to using the system after some period of not having used it, without having to learn everything all over.
- *Few errors*: Users do not make many errors during the use of the system, or if they do make errors they can easily recover from them. Also, no catastrophic errors should occur.
- *Pleasant to use*: Users are subjectively satisfied by using the system; they like it.

These will be the attributes where the user evaluation in WP6 will be focused.

5.2 Integration Plan

5.2.1 Agile Development Methodology

A research among the most dominant development methodologies¹¹ indicates that the most appropriate way of implementing integration mechanisms for the SYMPHONY platform would be 'Rapid Application

⁹ Roberto Polillo, Op. Cit.

¹⁰Nielsen, J. *Usability Engineering*. Academic Press, San Diego, CA, 1993

Development'¹². This implies that a system prototype is implemented, tested and evaluated in an iterative manner, using short cycles to add functionality to the prototype. This is more suitable for an R&D project aiming to deliver a system prototype, since it enables business oriented teams to continuously participate in the development of the integration mechanisms and guide the development towards their needs. In this manner, the processes of implementation and definition of the integration mechanisms will proceed in parallel until the end of the project by means of close collaboration between all the teams. One of the most popular types of Rapid Application Development is the 'Agile Methodology', which is associated with a list of terms and rules that have to be followed during development as described in the 'Agile Manifesto'¹³. Agile methodology implies and enforces collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. Some of the principles of the Agile Manifesto are:

- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Customer satisfaction by rapid delivery of useful software
- Close, daily co-operation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

The methodology workflow could be reflected in the following diagram¹⁴:

¹¹ http://en.wikipedia.org/wiki/Software_development_methodology

¹² http://en.wikipedia.org/wiki/Rapid_application_development

¹³ <http://agilemanifesto.org/>

¹⁴ <http://exelanz.com/why-cloud/development-methodology/>

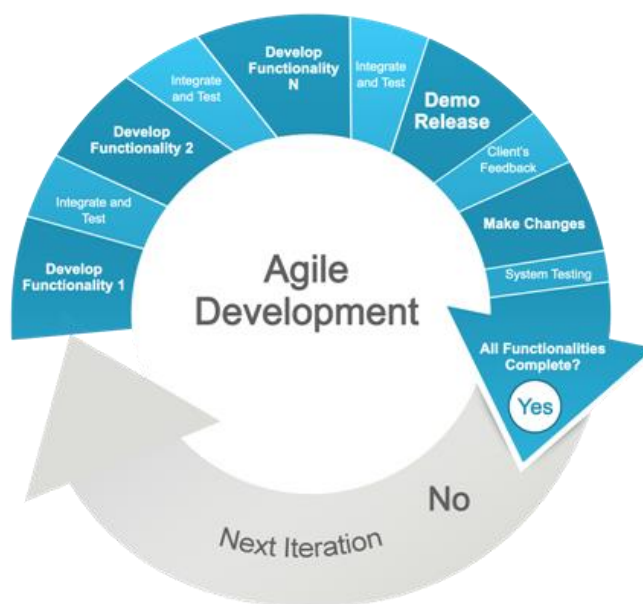


Figure 29: Agile methodology workflow

5.2.2 Agile Practices

These principles define the practices that are going to be followed for the implementation. Moreover, they imply the processes that are going to be adapted during the development cycle. The Agile Project Management Process Framework is presented in the diagram below¹⁵

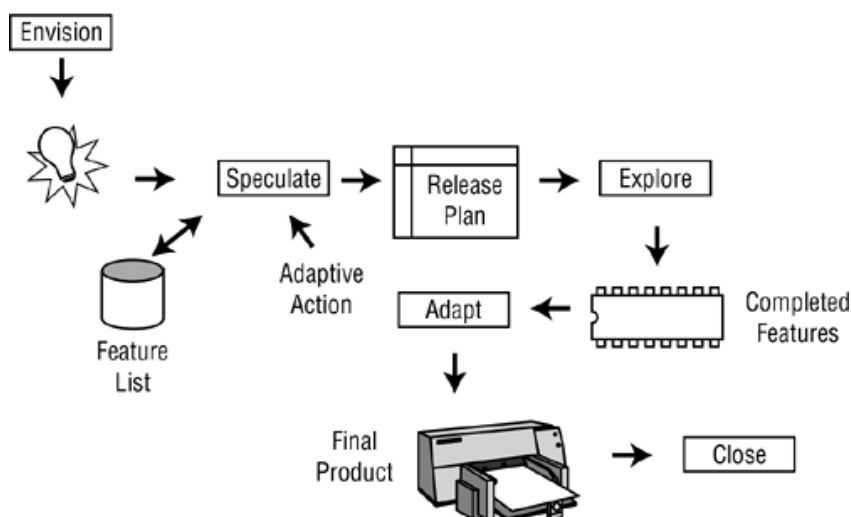


Figure 30: Agile Project Management Process Framework

To sum up, the five phases of agile project management are:

¹⁵ Agile Project Management: Creating Innovative Products. Jim Highsmith. 2009. Addison-Wesley Professional.

- Envision: determine the product vision and project scope, the project community, and how the team will work together
- Speculate: develop a feature-based release, milestone, and iteration plan to deliver on the vision
- Explore: deliver tested features in a short timeframe, constantly seeking to reduce the risk and uncertainty of the project
- Adapt: review the delivered results, the current situation, and the team's performance, and adapt as necessary
- Close: conclude the project, pass along key learnings.

5.2.3 Continuous Integration process

Continuous Integration process fits perfectly with the Agile Methodology. Continuous integration (CI) comprises practices such as daily builds and additional checks so as to prevent bugs. In order to enable automatic daily builds, Continuous Integration software gathers the whole source code in one place (with different revisions), automate the build process and testing, and provides the latest working executable to anyone involved in the project. The CI model comprises a set of activities for the process implementation: building the system, running tests, deployment activities, and finally reporting test and deployment results.

The practice of Continuous Integration assumes a high degree of tests, which are automated into the software: a facility that can be seen as “self-testing code”, often using a testing framework, such as JUnit4. Each automated build will perform the following steps:

- Compilation and creation of the build package
- Creation of a report on code metrics (such as package, dependency analysis and cyclomatic complexity)
- Creation of a report on how much source code follows declared Coding Standard
- Creation of a report on possible bugs by a static code analysis execution
- Automated deployment in testing environment
- Execution of automated test cases and creation of test report
- Publishing the build artifacts
- Notification to stakeholders and involved developers about build outcome by email and on central build dashboard

Continuous Integration is a software development practice where:

- Members of a team integrate their work frequently.
- Each integration is automatically compiled and built
- New artifacts are automatically deployed in testing environment
- Integration and Unit Testing is performed automatically
- Automated notifications to developers are sent
- Creation of reports about software quality code

The workflow is also depicted in the following diagram:

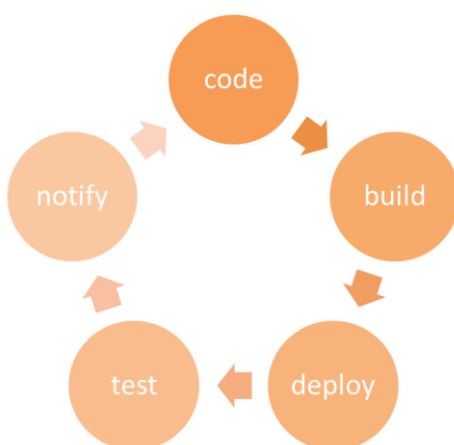


Figure 31: Continuous Integration Workflow

5.2.4 Development Infrastructure

Hudson CI or Jenkins CI (if applicable):

For the continuous integration workflow of the integration mechanisms for the SYMPHONY platform, we are going to use Hudson (or Jenkins) Continuous Integration server. Both systems:

- are open source tools to perform Continuous Integration.
- monitor a SCM (Source Control System) and if changes occur to start and monitor a build system (for example Apache Ant or Maven).
- will monitor the whole process and provide reporting functionality and notification functionality to report success or errors.

Both tools control the builds of the project through a graphical interface accessible by the administrator of the integration process. Thus, the integrator can easily change the maven goals to be executed as well as the execution mode and time of the scheduled maven jobs.

Apache Maven (if applicable): Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. It is important that all integration modules are compatible with maven rules, sharing the same .pom file. In this way, the integration will be easily controlled through maven and moreover, the project will be independent of an IDE; as a maven project can be opened in eclipse, netbeans, IDEA etc.

Trac or Redmine: An enhanced wiki and issue tracking system for software development projects.

Subversion or Git: A software versioning and a revision control system distributed under a free license alternatives: mercurial, ClearCase, Git

Sonar (if applicable): Sonar is an open source software quality platform. Sonar uses various static code analysis tools such as Checkstyle, PMD, FindBugs, Clover to extract software metrics, which then can be used to improve software quality.

Sonatype Nexus (if applicable): Sonatype Nexus sets the standard for repository management providing development teams with the ability to proxy remote repositories and share software artifacts. Download Nexus and gain control over open source consumption and internal collaboration.

In a more descriptive way, the workflow of the procedure is the following:

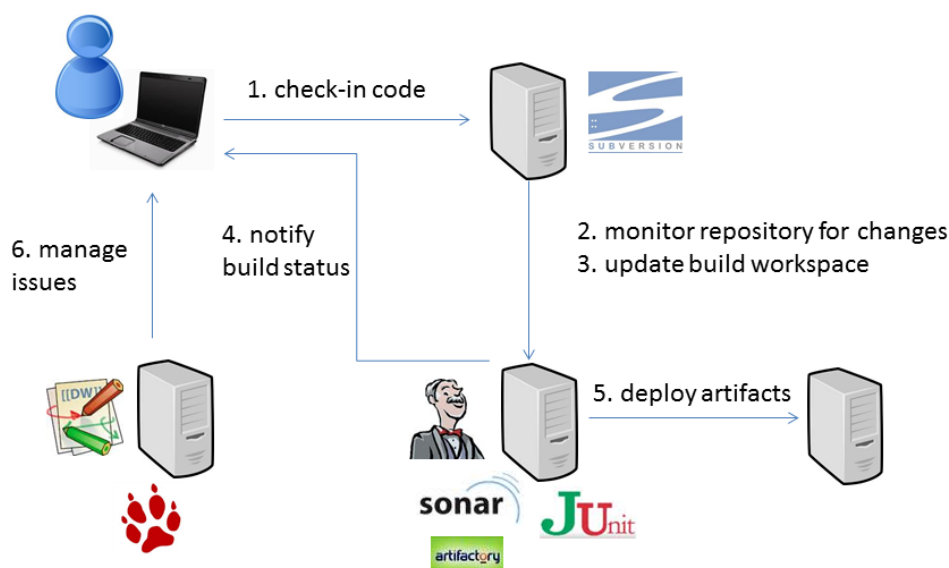


Figure 32: Continuous Integration Workflow (2)

5.2.5 Development Schedule

A high level of the time-plan for the development of each SYMPHONY component and their integration to the SYMPHONY's integrated platform is being presented below in the following sequence:

1. On **M12** of the project the Social media streams processing infrastructure
2. On **M18** of the project, the first prototypes of:
 - a. the early version of social media based policy indicators
 - b. the early version of the information market tool
 - c. the early version of the large-scale multi-country agent-based macroeconomic engine
 - d. the early version of the SYMPHONY Serious Game
3. On **M24** of the project the Initial SYMPHONY platform integrated prototype
4. On **M30** of the project, the final version of
 - a. the SYMPHONY Serious Game
 - b. the multi-country agent-based macroeconomic engine
 - c. the prototype of social media based policy indicators
 - d. the information market tool
5. On **M33** of the project the Final integrated SYMPHONY platform
6. On **M36** will be finalized the Stakeholders' evaluation of SYMPHONY platform with a report that will present the results of the use cases and stakeholders' assessment.

6 References

- [1] OASIS, "Reference Model for Service Oriented Architecture 1.0", August 2006
- [2] Waseem Roshen, "SOA-Based Enterprise Integration: A Step-by-Step Guide to Services-Based Application Integration", Mc Graw Hill, ISBN: 978-0-07-160553-3, May 2009
- [3] The Open Group, "SOA Reference Architecture", <http://www.opengroup.org/projects/soa-ref-arch/>, April 2009
- [4] <http://www.w3.org/TR/ws-gloss/>, last accessed 2011-07-20
- [5] http://en.wikipedia.org/wiki/Multilayered_architecture, last accessed 2014-02-27
- [6] David Chappell, "Enterprise Service Bus", O'Reilly: June 2004, ISBN 0-596-00675-6
- [7] Wikipedia link, Enterprise service bus, http://en.wikipedia.org/wiki/Enterprise_service_bus, last accesses 2011-07-20
- [8] Fiorano Enterprise Service Bus, <http://www.fiorano.com/products/ESB-enterprise-service-bus/ESB-enterprise-service-bus-architecture.php>, last accessed 2011-07-20



Appendix A. 1st Technical Questionnaire

The following questions are requested to be filled by SYMPHONY's technology partners who are going to develop the components in RTD WPs (WP2, WP3 & WP4) and that will be integrated in the SYMPHONY platform. Please provide your answers and you are kindly requested not to respond with a simple yes or no. Try to elaborate on your answers by giving an example of use, to the extent that this is possible.

QUESTIONS

1. Questions about your component

- 1.1. Please name your component and provide a short description of functionality you plan to deliver in the project and any foreseen interrelation with another symphony software component.

Component Name:

Component Description:
.....
.....
.....
.....
.....

- 1.2. Do you have an existing background tool that will be adopted in the project to support your component? If yes, please provide the following:

- i. Any past project that was used:

.....
.....
.....

- ii. The component architecture and technologies used in the development (Short description and if available architecture figure):

.....
.....
.....
.....

- 1.3. Did/Will you implement your component from scratch or is it an extension of a third party platform? If you have extended a third party platform please provide at least the following:

- i. Name:
ii. Version:

- 1.4. Does your component support / require user interaction? If yes, please provide a short description of the expected user input.

.....
.....
.....
.....

1.5. Which configuration your component need for initiation and runtime?

.....

.....

.....

.....

.....

.....

1.6. Have you ever performed a stress test on your component under heavy load (for the components that have already been developed)? If yes, please provide the test results.

.....

.....

.....

.....

.....

.....

1.7. Which person in your team is the designated component "owner" (for communication purposes)?

.....

.....

.....

.....

.....

.....

1.8. If known, please state the names of people actively involved in the development of the component (for communication purposes).

.....

.....

.....

.....

.....

.....

1.9. Will the component be provided as a service, binary or source code?

.....

.....

.....

.....

.....

.....

1.10. What will be the license of the delivered component?

i. Are there any restrictions in its use?

.....

.....

-
.....
.....
.....
ii. Will it be made available under an open source license?
.....
.....
.....
.....
.....
.....

2. Development Needs

2.1. In which programming languages does your component depend (i.e. java, C++, etc.)? Please provide at least the following:

- i. Name:
ii. Version:

2.2. Does your component depend on existing design tools (i.e. eclipse, process modeling tool, compilation process, etc.)? Please provide at least the following:

- i. Name:
ii. Version:

2.3. Which are the hardware requirements for your component to run (i.e. any restrictions in memory use, required CPU, etc.)?

.....
.....
.....
.....
.....

2.4. Which are the software requirements for your component to run (i.e. operating system, web servers like tomcat or apache, etc.)

.....
.....
.....
.....
.....

2.5. Does your component have any third party dependencies or use third party libraries? If yes, please provide at least the following:

- i. Name:
ii. Version:

2.6. Can you support automatic builds? Can you support Ant or Maven?

.....
.....
.....
.....
.....

2.7. Does your component have any database requirements? If yes please provide the following:

- i. required design (i.e. SQL-like, noSQL):
.....
- ii. database schema (i.e. ER diagram, JSON format):
.....

2.8. Is there a requirement on a specific Version Control System?

.....
.....
.....
.....
.....

3. Interoperability Requirements

3.1. Will your component offer an API to work as a service?

- i. **If yes**, please provide the full API Documentation for both input and output streams:
.....
- ii. **If no**, please describe the communication schema supported by your component:
.....

3.2. Can your component support asynchronous messaging (i.e. non real time response/communication between the components)?

.....
.....
.....
.....

3.3. Which are the foreseen dependencies / interactions with other symphony software components, if any? Please provide at least the following:

- i. List of components:
.....
- ii. Input and / or output requirements:
.....
.....

- iii. Type and aim of the information that need to be exchanged:
-
-
-
-

4. Security principles

4.1. Which are the security principles that should be taken into consideration to protect your component's data?

.....

.....

.....

.....

Appendix B. 2nd Technical Questionnaire

The following questionnaire provides a template for the description of inputs and outputs of the component that will comprise the SYMPHONY platform. It also includes a section to describe the interfaces of the described component to other components of the platform. A brief description of the interfaces, the methods and the parameters are also required in order to provide the guidelines for the implementation of the integrated system.

1. Please name your component and provide a description of functionality you plan to deliver in the project as part of it (2-3 paragraphs).

Please provide your answer here:

2. Please provide the UML class diagram of your component in terms of data interface (API) which is accompanied by the following description table.

Please provide your UML here and fill the following table

Description Table of the component:

Interfaces		
Name	Brief Description	Implementation priority (Must/Should)
Methods		
InterfaceName:MethodName	Brief Description	Implementation priority (Must/Should)
Input		
InterfaceName:MethodName:ParameterName	Brief Description	Implementation priority (Must/Should)
Output		

InterfaceName:MethodName:ParameterName	Brief Description	Implementation priority (Must/Should)

Example of a UML class diagram in terms of data interfaces (API)

