eCO-friendly urban Multi-modal route PlAnning Services for mobile uSers

FP7 - Information and Communication Technologies

Grant Agreement No: 288094
Collaborative Project
Project start: 1 November 2011, Duration: 36 months

D4.3 – Semantic content repository and web-based registration tool

Workpackage:	WP4 – Content Gateway Module
Due date of deliverable:	30 April 2013
Actual submission date:	26 April 2013
Responsible Partner:	CERTH
Contributing Partners:	CERTH, PTV

Nature: ☐ Report [✓ Prototype] ☐ Demonstrator ☐ Other

Dissemination Level:

[✓]	PU :	Public
☐	PP :	Restricted to other programme participants (including the Commission Services)
☐	RE :	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO :	Confidential, only for members of the consortium (including the Commission Services)

Keyword List: Service providers, Web Services (SOAP, REST), Automatic Web Service Categorisation, Service Invocation, Service Integration, Semantics.

The eCOMPASS project (www.ecompass-project.eu) is funded by the European Commission, DG CONNECT (Communications Networks, Content and Technology Directorate General), Unit H.5 - Smart Cities & Sustainability, under the FP7 Programme

The eCOMPASS Consortium

Computer Technology Institute & Press "Diophantus" (CTI) (coordinator), Greece

Centre for Research and Technology Hellas (CERTH), Greece

Eidgenössische Technische Hochschule Zürich (ETHZ), Switzerland

Karlsruher Institut fuer Technologie (KIT), Germany

TOMTOM INTERNATIONAL BV (TOMTOM), Netherlands

PTV PLANUNG TRANSPORT VERKEHR AG. (PTV), Germany

Document history

Version	Date	Status	Modifications made by
0.1	13.03.2013	First draft	D. Kehagias
0.2	21.03.2013	ToC finalized	D. Kehagias, F. Krietsch
0.3	15.04.2013	An example of use was added about SOAP services (6.1)	D. Kehagias,
0.4	23.04.2013	An example of use was added about REST services (6.2)	F. Krietsch
0.5	24.04.2013	First draft version of the document, integrating input from contributors	D. Kehagias, F. Krietsch
0.6	25.04.2013	Draft sent to reviewers	D. Kehagias
0.7	26.04.2013	Final version incorporating Reviewers' Comments	D. Kehagias
1.0	26.04.2013	Final version approved by the PCB	

Deliverable manager
- Dionisis Kehagias (CERTH)

List of Contributors
- Dionisis Kehagias (CERTH)
- Florian Krietsch (PTV)

List of Evaluators
- Spyros Kontogiannis (CTI)
- Julian Dibbelt (KIT)

Summary

This deliverable describes the Web-service registration tool as well as the underlying Semantic Content Repository, which comprise a significant part of the Content Gateway Module intended for potential service providers. The purpose of the web-service registration tool is to facilitate the integration into eCOMPASS of such web services provided by third party service providers. Examples of those services include POI services and Weather information services among others. Thus, this tool enables integration of the corresponding content that is necessary for the pilots, and is freely available by service providers outside the eCOMPASS consortium. The web–service registration tool provides a web interface for those service providers who are willing to register their services, making them accessible in the eCOMPASS context. In the case of SOAP, as well as RESTful web services for which a WADL description is available, the tool automatically identifies the domain that a service belongs to and matches its input and output parameters with the ones provided by the Content Gateway Module communication interface. This report outlines the implementation details of the Web-service registration tool and the Semantic Content Repository, and provides a few examples of use.

Table of Contents

1 Introduction

1.1 General

The *Content Gateway Module* (CGM) of eCOMPASS is a key component of the overall eCOMPASS architecture. The main role of CGM is to respond to each query with the appropriate content. On top of this, CGM is responsible for enabling interoperability with external services and information resources that are required by other integrated eCOMPASS components. For this purpose, a set of components and tools are developed within WP4 in order to facilitate the integration of external *web services* (WS) and data resources, provided by e.g. local providers. These include the Data Communication Interfaces (developed in Task 4.1), the Information Querying and Data Delivery Mechanisms (developed in Task 4.2) and the Semantic Content Repository and Registration Tool (developed in Task 4.3). Also, appropriate Data Trust and Security mechanisms are foreseen (Task 4.4) in order to safeguard the data exchange between the external data providers and the requesting parties through the CGM. This deliverable describes the Semantic Content Repository and Registration Tool whose purpose is to allow external Service Providers (SP), as well as data providers who make data available through WSs, to register their services in an open and interoperable way into the eCOMPASS context of use. More specific details on Task 4.3 are provided in what follows.

1.2 Task 4.3 Purpose and Scope

The purpose of Task 4.3 is to develop all the software components that are required for the implementation of the *Semantic Content Repository* (SCR) and the *Web Registration Tool* (WRT). The latter comprises a web interface that guides the interested SP to all appropriate steps for completing the registration process. The interface supports both REST and SOAP WS protocols, whereas it provides automatic categorization capabilities only for SOAP services, as well as for REST service, which are accompanied with a machine understandable document that describes the WS operations and their input/output parameters.

In addition to web interfaces represented by the WRT, the SCR provides the underpinning technical infrastructure for realizing the functionality of WS registration and in particular for meeting the technical needs for storing information about all registered SPs, as well as their services, and for facilitating their invocation. Such information is related to the technical characteristics of the WS, such as its physical location of the WS (i.e. its URI), but also includes semantic information (i.e., metadata) for facilitating the various “matches” between the newly registered services and the Common Data Interfaces that are supported by the CGM. The SCR provides all underpinning mechanisms and components for facilitating the process of WS registration thus enabling smooth service invocation and execution by the appropriate CGM mechanisms.

This deliverable describes the major outcome of Task 4.3., which is the development of the SCR and WRT. It serves both as a scientific reference for the technical details that describe the functionalities supported by the underlying SCR mechanisms, as well as a reference manual for using the functionalities offered by the WRT.

1.3 Structure of Deliverable

The rest of this deliverable is structured as follows.

- ***Section 2*** contains a description of the Semantic Content Repository architecture as part of the Content Gateway Module and describes the structure of the database that was designed for hosting information about the registered WSs.

- ***Section 3*** provides documentation about the functionalities supported by the Web Registration Tool user interface.
- ***Section 4*** describes the technical details of the automatic Web Service Categorisation mechanism and its key components and reveals its underlying technical details and the algorithmic approach that was adopted.
- ***Section 5*** presents some examples of use of the Web Registration Tool.
- ***Section 6*** summarises and concludes the deliverable.

2 Semantic Content Repository

This section describes the underpinning technical infrastructure of the WRT, namely the *Semantic Content Repository*, whose purpose is to host all information that is provided by any interested SP who are willing to register their services, so that it will become visible and thus callable within the eCOMPASS framework. This section begins by outlining the various components involved in the WS registration functionality, from an architectural point of view and then it presents the structure details of the underpinning database infrastructure.

2.1 Architecture

Figure 1 provides the architectural view of all components that are involved in the WS registration scenario.

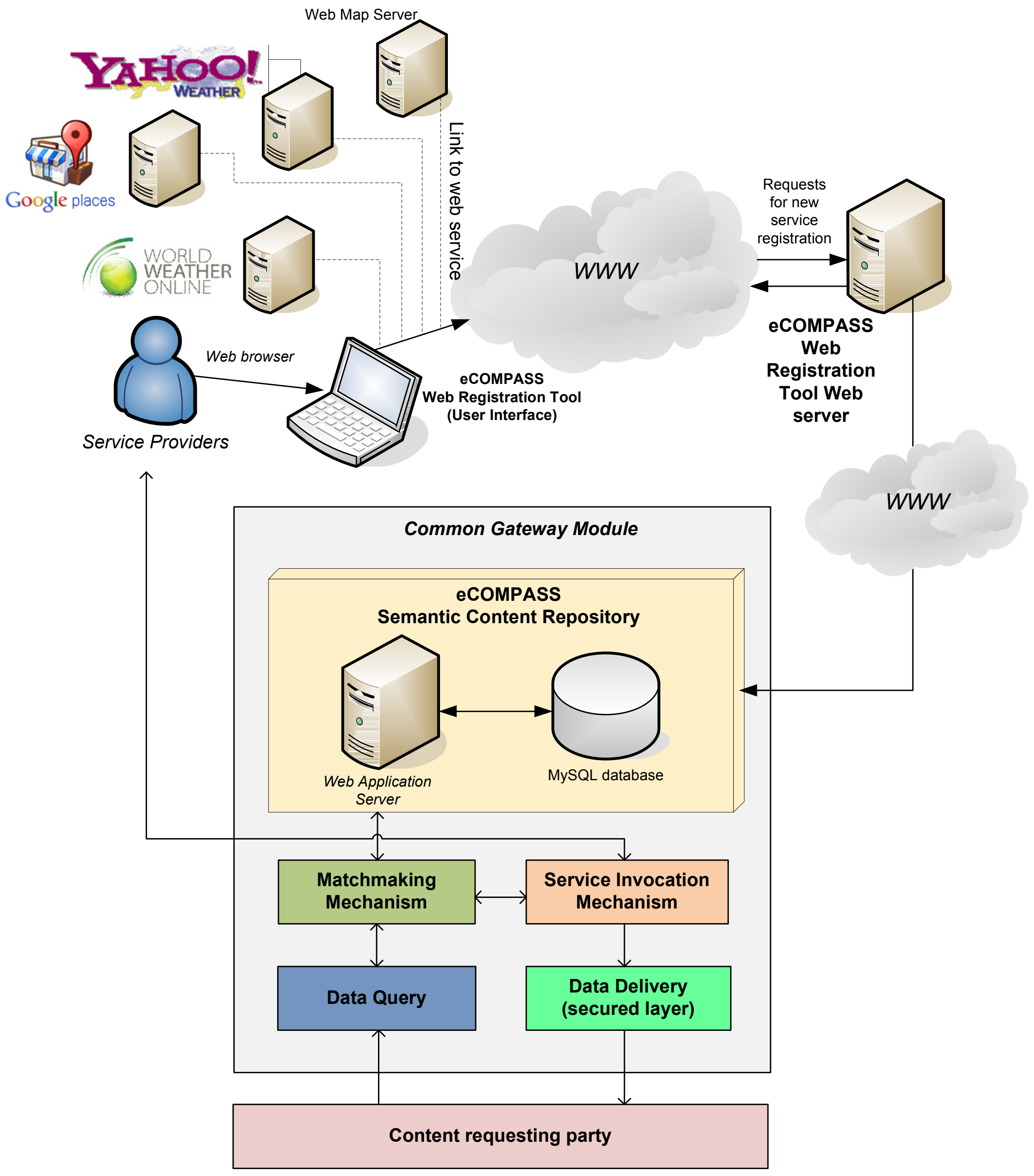

Web Map Server
Google places
Link to web service
WWW
Requests for new service registration
eCOMPASS Web Registration Tool Web server
Web browser
Service Providers
eCOMPASS Web Registration Tool (User Interface)
WWW
Common Gateway Module
eCOMPASS Semantic Content Repository
Web Application Server
MySQL database
Matchmaking Mechanism
Service Invocation Mechanism
Data Query
Data Delivery (secured layer)
Content requesting party

Figure 1: Semantic Content Repository Architecture and communication interfaces

In particular, the architecture schema involves the following entities:

- **Service providers (SPs)**, represent any "information service" providers who are willing to register their services into the eCOMPASS framework. We assume that all interested providers have developed and provide all necessary technical infrastructure for accessing their services in the form of WSs, i.e., they have developed the appropriate APIs and / or WS interfaces. Two WS protocols are supported: SOAP and REST.
- **The eCOMPASS Web Registration Tool (WRT)** is the web application and corresponding user interface that allows SPs for entering the required information in order to register their services, using any typical web browser. The information that is entered through the tool provides the essential links to the real WS, so that their invocation can be handled in an automatic way by the CGM's service invocation mechanism.
- **The eCOMPASS WRT Web server** is the necessary web server that is required for hosting the tool. It receives new requests for WS registration and sends them to the SCR.
- The **Semantic Content Repository** consists of two subsystems:
    - A web application server that hosts the automatic WS categorization functionality (described in Section 4).
    - A database, which stores all information concerning all registered SPs (its structure is described in detail in the next subsection).
- The entity denoted as **Content Requesting Party** represents any external software component or CGM client in general, which requests, through the **Data Query** component, particular content to be delivered through CGM's **Data Delivery** mechanism, according to the eCOMPASS Data Communication Interface protocol (see Deliverable D4.1.2).
- **Data Query** receives requests for content by any **Content Requesting Party** (see above).
- **Data Delivery** delivers the requested content to the corresponding **Content Requesting Party** (see above), in a data secure way.
- The **Matchmaking Mechanism** of CGM is responsible for selecting the most appropriate service provider that fulfils a particular request for content in an optimal way, according to a set of criteria, such as Quality of Service, user preferences, reliability of SP, etc.(for details on the Matchmaking mechanism, see Deliverable D4.2.1)
- **Service Invocation Mechanism**. As soon as the appropriate SP has been selected by the matchmaking mechanism, the Service Invocation mechanism is responsible for executing the corresponding WS wrapper and for calling the corresponding WS, based on the information entered into the Semantic Content Repository.

2.2 Underlying Database Structure

The semantic content repository is realized by a rational database, whose structure is illustrated in Figure 2.

A different table in the database has been created for each type of service, for the use of WRT. Each table contains the following default columns, which are used for storing the service characteristics along with some statistical data about the availability of the service and the Quality of Service.:

- "id", a unique identifier
- "Name", the name of the service provider

- "type", the WS protocol type (SOAP, or REST)
- "url", the URL that points to a WS description file (WSDL)
- "operationName", name of WS operation
- "output", name of WS operation's output parameter
- "email", the email contact address of the SP
- "implemented", a flag that indicates if a WS wrapper is implemented for this WS or not.
- "Ratio", service acceptance ration
- "Reputation", a degree of SP's reputation based on service availability
- "SelectedCounter", number of selections of this WS by the user
- "QoS", Quality of Service parameter for indicating service availability of this WS by its users
- "Availability", service availability for all times this service was requested
- "ResponseTime", time that takes for the WS to response upon request
- "RespTimeCounter", an auxiliary variable that counts how many times
- "Similarity", similarity score for selecting the service
- "userRank", user ranking of this WS

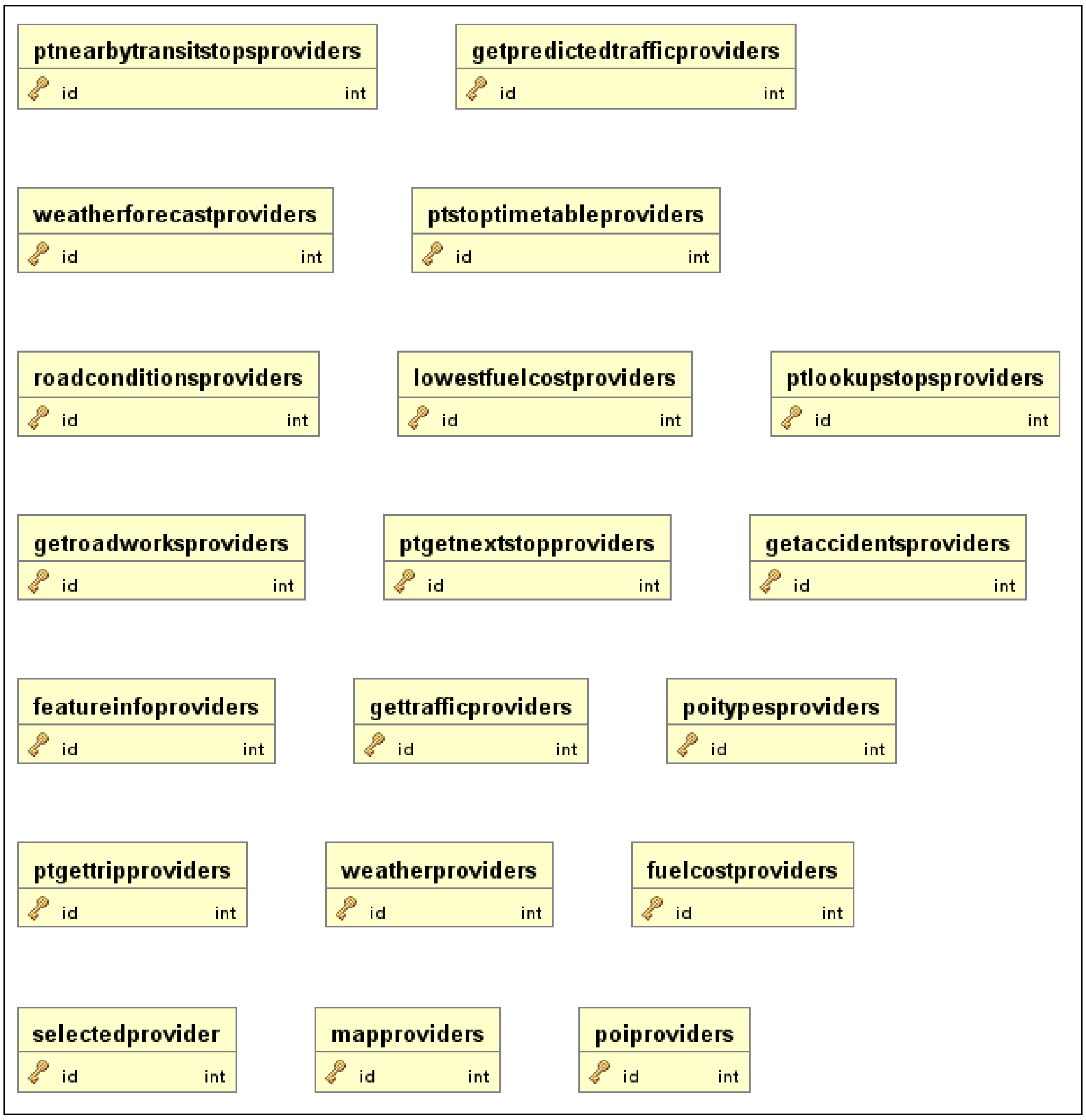


Figure 2: Database tables of the Semantic Content Repository

The rest of the columns of each database table are used for storing the mapped inputs and outputs of the operation that is to be registered in the repository. This information is entered automatically in the database in the case of SOAP WS, in which the WS categorization mechanism is activated.

The database contains the following tables:

- FeatureInfoProviders
- FuelCostProviders
- GetAccidentsProviders
- GetPredictedTrafficProviders
- GetRoadworksProviders
- GetTrafficProviders
- LowestFuelCostProviders
- MapProviders
- PoiProviders
- PoiTypesProviders
- PtGetNextStopProviders
- PtGetTripProviders
- PtLookupStopsProviders
- PtNearbyTransitStopsProviders
- PtStopTimetableProviders
- RoadConditionsProviders
- SelectedProvider
- WeatherForecastProviders
- WeatherProviders

The structure of these tables is presented in detail in what follows.

2.2.1 Table *FeautureInfoProviders*

The structure of the FeatureInfoProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
maxx_in	varchar(100)	Yes	NULL

Column	Type	Null	Default
maxy_in	varchar(100)	Yes	NULL
minx_in	varchar(100)	Yes	NULL
miny_in	varchar(100)	Yes	NULL
resx_in	varchar(100)	Yes	NULL
resy_in	varchar(100)	Yes	NULL
BboxSource_in	varchar(100)	Yes	NULL
heigh_in	varchar(100)	Yes	NULL
width_in	varchar(100)	Yes	NULL
exceptionalFormat_in	varchar(100)	Yes	NULL
featureCount_in	varchar(100)	Yes	NULL
infoFormat_in	varchar(100)	Yes	NULL
layers_in	varchar(100)	Yes	NULL
queryLayers_in	varchar(100)	Yes	NULL
requestName_in	varchar(100)	Yes	NULL
source_in	varchar(100)	Yes	NULL
styles_in	varchar(100)	Yes	NULL
version_in	varchar(100)	Yes	NULL
parameters_out	varchar(100)	Yes	NULL
subtype_out	varchar(100)	Yes	NULL
type_out	varchar(100)	Yes	NULL
featureInformation_out	varchar(100)	Yes	NULL

Table 1: The "**FeatureInfoProviders**" table structure

2.2.2 Table *FuelCostProviders*

The structure of the **FuelCostProviders** database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
city_in	varchar(100)	Yes	NULL

Column	Type	Null	Default
country_in	varchar(100)	Yes	NULL
currency_in	varchar(100)	Yes	NULL
type_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
radius_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
currency_out	varchar(100)	Yes	NULL
type_out	varchar(100)	Yes	NULL
price_out	varchar(100)	Yes	NULL

Table 2: The "FuelCostProviders" table structure

2.2.3 Table *GetAccidentsProviders*

The structure of the GetAccidentsProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
accidentCause_in	varchar(100)	Yes	NULL
accidentType_in	varchar(100)	Yes	NULL
generationTime_in	varchar(100)	Yes	NULL
observationTime_in	varchar(100)	Yes	NULL
severity_in	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_in	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_in	varchar(100)	Yes	NULL
startArrivLinkID_in	varchar(100)	Yes	NULL
startDepartLinkID_in	varchar(100)	Yes	NULL
startLatY_in	varchar(100)	Yes	NULL
startLongX_in	varchar(100)	Yes	NULL
startNodeID_in	varchar(100)	Yes	NULL
getaccidentsproviderscol	varchar(100)	Yes	NULL

Column	Type	Null	Default
endArrivLinkID_in	varchar(100)	Yes	NULL
endDepartLinkID_in	varchar(100)	Yes	NULL
endLatY_in	varchar(100)	Yes	NULL
endLongX_in	varchar(100)	Yes	NULL
maxSpeed_in	varchar(100)	Yes	NULL
roadID_in	varchar(100)	Yes	NULL
accidentCause_out	varchar(100)	Yes	NULL
accidentType_out	varchar(100)	Yes	NULL
generationTime_out	varchar(100)	Yes	NULL
observationTime_out	varchar(100)	Yes	NULL
severity_out	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_out	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_out	varchar(100)	Yes	NULL

Table 3: The "**GetAccidentsProviders**" table structure

2.2.4 Table *GetPredictedTrafficProviders*

The structure of the GetPredictedTrafficProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
accidentCause_in	varchar(100)	Yes	NULL
accidentType_in	varchar(100)	Yes	NULL
generationTime_in	varchar(100)	Yes	NULL
observationTime_in	varchar(100)	Yes	NULL
severity_in	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_in	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_in	varchar(100)	Yes	NULL
startArrivLinkID_in	varchar(100)	Yes	NULL
startDepartLinkID_in	varchar(100)	Yes	NULL

Column	Type	Null	Default
startLatY_in	varchar(100)	Yes	NULL
startLongX_in	varchar(100)	Yes	NULL
startNodeID_in	varchar(100)	Yes	NULL
endArrivLinkID_in	varchar(100)	Yes	NULL
endDepartLinkID_in	varchar(100)	Yes	NULL
endLatY_in	varchar(100)	Yes	NULL
endLongX_in	varchar(100)	Yes	NULL
endRoadID_in	varchar(100)	Yes	NULL
maxSpeed_in	varchar(100)	Yes	NULL
roadID_in	varchar(100)	Yes	NULL
time_in	varchar(100)	Yes	NULL
timeInMinutes_in	varchar(100)	Yes	NULL
generationTime_out	varchar(100)	Yes	NULL
observationTime_out	varchar(100)	Yes	NULL
roadworks_out	varchar(100)	Yes	NULL
severity_out	varchar(100)	Yes	NULL
speedValKpH_out	varchar(100)	Yes	NULL
speedValError_out	varchar(100)	Yes	NULL
speedValReasonError_out	varchar(100)	Yes	NULL
vehiclePercPercentage_out	varchar(100)	Yes	NULL
vehiclePercError_out	varchar(100)	Yes	NULL
vehiclePercReasonError_out	varchar(100)	Yes	NULL
avgVehSpeedKpH_out	varchar(100)	Yes	NULL
avgVehSpeedError_out	varchar(100)	Yes	NULL
avgVehSpeedReasonError_out	varchar(100)	Yes	NULL
roadID_out	varchar(100)	Yes	NULL
timestamp_out	varchar(100)	Yes	NULL
velocity_out	varchar(100)	Yes	NULL
statusTime_out	varchar(100)	Yes	NULL
statusPeriod_out	varchar(100)	Yes	NULL
statusError_out	varchar(100)	Yes	NULL
statusReasonError_out	varchar(100)	Yes	NULL
statusEnum_out	varchar(100)	Yes	NULL
trafficTrendType_out	varchar(100)	Yes	NULL

Table 4: The "**GetPredictedTrafficProviders**" table structure

2.2.5 Table *GetRoadworksProviders*

The structure of the GetRoadworksProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL

Column	Type	Null	Default
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
accidentCause_in	varchar(100)	Yes	NULL
accidentType_in	varchar(100)	Yes	NULL
generationTime_in	varchar(100)	Yes	NULL
observationTime_in	varchar(100)	Yes	NULL
severity_in	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_in	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_in	varchar(100)	Yes	NULL
startArrivLinkID_in	varchar(100)	Yes	NULL
startDepartLinkID_in	varchar(100)	Yes	NULL
startLatY_in	varchar(100)	Yes	NULL
startLongX_in	varchar(100)	Yes	NULL
startNodeID_in	varchar(100)	Yes	NULL
getaccidentsproviderscol	varchar(100)	Yes	NULL
endArrivLinkID_in	varchar(100)	Yes	NULL
endDepartLinkID_in	varchar(100)	Yes	NULL
endLatY_in	varchar(100)	Yes	NULL
endLongX_in	varchar(100)	Yes	NULL
maxSpeed_in	varchar(100)	Yes	NULL
roadID_in	varchar(100)	Yes	NULL
time_in	varchar(100)	Yes	NULL
generationTime_out	varchar(100)	Yes	NULL
observationTime_out	varchar(100)	Yes	NULL
roadworks_out	varchar(100)	Yes	NULL
severity_out	varchar(100)	Yes	NULL

Table 5: The "**GetRoadworksProviders**" table structure

2.2.6 Table *GetTrafficProviders*

The structure of the GetTrafficProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL

Column	Type	Null	Default
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
accidentCause_in	varchar(100)	Yes	NULL
accidentType_in	varchar(100)	Yes	NULL
generationTime_in	varchar(100)	Yes	NULL
observationTime_in	varchar(100)	Yes	NULL
severity_in	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_in	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_in	varchar(100)	Yes	NULL
startArrivLinkID_in	varchar(100)	Yes	NULL
startDepartLinkID_in	varchar(100)	Yes	NULL
startLatY_in	varchar(100)	Yes	NULL
startLongX_in	varchar(100)	Yes	NULL
startNodeID_in	varchar(100)	Yes	NULL
endArrivLinkID_in	varchar(100)	Yes	NULL
endDepartLinkID_in	varchar(100)	Yes	NULL
endLatY_in	varchar(100)	Yes	NULL
endLongX_in	varchar(100)	Yes	NULL
maxSpeed_in	varchar(100)	Yes	NULL
roadID_in	varchar(100)	Yes	NULL
generationTime_out	varchar(100)	Yes	NULL
observationTime_out	varchar(100)	Yes	NULL
roadworks_out	varchar(100)	Yes	NULL
severity_out	varchar(100)	Yes	NULL
speedValcKpH_out	varchar(100)	Yes	NULL
speedValcError_out	varchar(100)	Yes	NULL
speedValReasonError_out	varchar(100)	Yes	NULL
vehiclePercKpH_out	varchar(100)	Yes	NULL
vehiclePercError_out	varchar(100)	Yes	NULL
vehiclePercReasonError_out	varchar(100)	Yes	NULL
avgVehSpeedKpH_out	varchar(100)	Yes	NULL
avgVehSpeedError_out	varchar(100)	Yes	NULL
avgVehSpeedReasonError_out	varchar(100)	Yes	NULL
roadID_out	varchar(100)	Yes	NULL
timestamp_out	varchar(100)	Yes	NULL
velocity_out	varchar(100)	Yes	NULL

Column	Type	Null	Default
statusTime_out	varchar(100)	Yes	NULL
statusPeriod_out	varchar(100)	Yes	NULL
statusError_out	varchar(100)	Yes	NULL
statusReasonError_out	varchar(100)	Yes	NULL
statusEnum_out	varchar(100)	Yes	NULL
trafficTrendType_out	varchar(100)	Yes	NULL

Table 6: The "**GetTrafficProviders**" table structure

2.2.7 Table *LowestFuelCostProviders*

The structure of the LowestFuelCostProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
currency_in	varchar(100)	Yes	NULL
type_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
radius_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
currency_out	varchar(100)	Yes	NULL
type_out	varchar(100)	Yes	NULL
price_out	varchar(100)	Yes	NULL
address_out	varchar(100)	Yes	NULL
stationCurrency_out	varchar(100)	Yes	NULL
stationType_out	varchar(100)	Yes	NULL
stationPrice_out	varchar(100)	Yes	NULL
name_out	varchar(100)	Yes	NULL

Table 7: The "**LowestFuelCostProviders**" table structure

2.2.8 Table *MapProviders*

The structure of the MapProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
maxx_in	varchar(100)	Yes	NULL
maxy_in	varchar(100)	Yes	NULL
minx_in	varchar(100)	Yes	NULL
miny_in	varchar(100)	Yes	NULL
resx_in	varchar(100)	Yes	NULL
resy_in	varchar(100)	Yes	NULL
BboxSource_in	varchar(100)	Yes	NULL
heigh_in	varchar(100)	Yes	NULL
width_in	varchar(100)	Yes	NULL
exceptionalFormat_in	varchar(100)	Yes	NULL
requestName_in	varchar(100)	Yes	NULL
source_in	varchar(100)	Yes	NULL
layers_in	varchar(100)	Yes	NULL
styles_in	varchar(100)	Yes	NULL
transparent_in	varchar(100)	Yes	NULL
version_in	varchar(100)	Yes	NULL
parameters_out	varchar(100)	Yes	NULL
subtype_out	varchar(100)	Yes	NULL
type_out	varchar(100)	Yes	NULL
map_out	varchar(100)	Yes	NULL

Table 8: The "**MapProviders**" table structure

2.2.9 Table *PoiProviders*

The structure of the PoiProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
address_in	varchar(100)	Yes	NULL
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
language_in	varchar(100)	Yes	NULL
type_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
radius_in	varchar(100)	Yes	NULL
poiInformationDescription_out	varchar(100)	Yes	NULL
poiInformationName_out	varchar(100)	Yes	NULL
shortName_out	varchar(100)	Yes	NULL
POIid_out	varchar(100)	Yes	NULL
poiEntranceescription_out	varchar(100)	Yes	NULL
poiEntranceID	varchar(100)	Yes	NULL
typeOfPoi_out	varchar(100)	Yes	NULL
bearing_out	varchar(100)	Yes	NULL
latitude_out	varchar(100)	Yes	NULL
longitude_out	varchar(100)	Yes	NULL
serviceDays_out	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
radius_out	varchar(100)	Yes	NULL

Table 9: The "**PoiProviders**" table structure

2.2.10 Table *PoiTypesProviders*

The structure of the PoiTypesProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
address_in	varchar(100)	Yes	NULL
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
radius_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
typesOfPoi_out	varchar(100)	Yes	NULL

Table 10: The "**PoiTypesProviders**" table structure

2.2.11 Table *PtGetNextStopProviders*

The structure of the PtGetNextStopProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1

Column	Type	Null	Default
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
maxDist_in	varchar(100)	Yes	NULL
maxStops_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
time_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
distance_out	varchar(100)	Yes	NULL
startArrivLinkID_out	varchar(100)	Yes	NULL
startDepartLinkID_out	varchar(100)	Yes	NULL
startLatY_out	varchar(100)	Yes	NULL
startLongX_out	varchar(100)	Yes	NULL
startNodeID_out	varchar(100)	Yes	NULL
endArrivLinkID_out	varchar(100)	Yes	NULL
endDepartLinkID_out	varchar(100)	Yes	NULL
endLongX_out	varchar(100)	Yes	NULL
endLatY_out	varchar(100)	Yes	NULL
endNodeID_out	varchar(100)	Yes	NULL
linkID_out	varchar(100)	Yes	NULL
maxSpeed_out	varchar(100)	Yes	NULL
numOfLanes_out	varchar(100)	Yes	NULL
accidentCause_out	varchar(100)	Yes	NULL
accidentType_out	varchar(100)	Yes	NULL
generationTime_out	varchar(100)	Yes	NULL
observationTime_out	varchar(100)	Yes	NULL
severity_out	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_out	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_out	varchar(100)	Yes	NULL
accidStartDepartLinkID_out	varchar(100)	Yes	NULL
accidStartArrivLinkID_out	varchar(100)	Yes	NULL
accidStartLatY_out	varchar(100)	Yes	NULL
accidStartLongX_out	varchar(100)	Yes	NULL
accidStartNodeID_out	varchar(100)	Yes	NULL
accidEndDepartLinkID_out	varchar(100)	Yes	NULL
accidEndArrivLinkID_out	varchar(100)	Yes	NULL
accidEndLatY_out	varchar(100)	Yes	NULL
accidEndLongX_out	varchar(100)	Yes	NULL
accidEndNodeID_out	varchar(100)	Yes	NULL
accidMaxSpeed_out	varchar(100)	Yes	NULL
accidRoadID_out	varchar(100)	Yes	NULL

Table 11: The "**PtGetNextStopProviders**" table structure

2.2.12 Table *PtGetTripProviders*

The structure of the PtGetTripProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
date_in	varchar(100)	Yes	NULL
fromPublicCode_in	varchar(100)	Yes	NULL
fromShortName_in	varchar(100)	Yes	NULL
fromStopPointID_in	varchar(100)	Yes	NULL
fromStopPointName_in	varchar(100)	Yes	NULL
toPublicCode_in	varchar(100)	Yes	NULL
toShortName_in	varchar(100)	Yes	NULL
toStopPointID_in	varchar(100)	Yes	NULL
toStopPointName_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
summary_out	varchar(100)	Yes	NULL
publicCode_out	varchar(100)	Yes	NULL
shortName_out	varchar(100)	Yes	NULL
stopPointID_out	varchar(100)	Yes	NULL
stopPointName_out	varchar(100)	Yes	NULL

Table 12: The "**PtGetTripProviders**" table structure

2.2.13 Table *PtLookupStopsProviders*

The structure of the PtLookupStopsProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL

email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
modesOfTransport_in	varchar(100)	Yes	NULL
resultsCount_in	varchar(100)	Yes	NULL
term_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
publicCode_out	varchar(100)	Yes	NULL
shortName_out	varchar(100)	Yes	NULL
stopPointID_out	varchar(100)	Yes	NULL
stopPointName_out	varchar(100)	Yes	NULL

Table 13: The "**PtLookupStopsProviders**" table structure

2.2.14 Table *PtNearbyTransitStopsProviders*

The structure of the PtNearbyTransitStopsProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
maxDist_in	varchar(100)	Yes	NULL
maxStops_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL

Column	Type	Null	Default
time_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
distance_out	varchar(100)	Yes	NULL
startArrivLinkID_out	varchar(100)	Yes	NULL
startDepartLinkID_out	varchar(100)	Yes	NULL
startLatY_out	varchar(100)	Yes	NULL
startLongX_out	varchar(100)	Yes	NULL
startNodeID_out	varchar(100)	Yes	NULL
endArrivLinkID_out	varchar(100)	Yes	NULL
endDepartLinkID_out	varchar(100)	Yes	NULL
endLongX_out	varchar(100)	Yes	NULL
endLatY_out	varchar(100)	Yes	NULL
endNodeID_out	varchar(100)	Yes	NULL
linkID_out	varchar(100)	Yes	NULL
maxSpeed_out	varchar(100)	Yes	NULL
numOfLanes_out	varchar(100)	Yes	NULL
accidentCause_out	varchar(100)	Yes	NULL
accidentType_out	varchar(100)	Yes	NULL
generationTime_out	varchar(100)	Yes	NULL
observationTime_out	varchar(100)	Yes	NULL
severity_out	varchar(100)	Yes	NULL
totalNumberOfPeopleInvolved_out	varchar(100)	Yes	NULL
totalNumberOfVehiclesInvolved_out	varchar(100)	Yes	NULL
accidStartDepartLinkID_out	varchar(100)	Yes	NULL
accidStartArrivLinkID_out	varchar(100)	Yes	NULL
accidStartLatY_out	varchar(100)	Yes	NULL
accidStartLongX_out	varchar(100)	Yes	NULL
accidStartNodeID_out	varchar(100)	Yes	NULL
accidEndDepartLinkID_out	varchar(100)	Yes	NULL
accidEndArrivLinkID_out	varchar(100)	Yes	NULL
accidEndLatY_out	varchar(100)	Yes	NULL
accidEndLongX_out	varchar(100)	Yes	NULL
accidEndNodeID_out	varchar(100)	Yes	NULL
accidMaxSpeed_out	varchar(100)	Yes	NULL
accidRoadID_out	varchar(100)	Yes	NULL

Table 14: The "**PtNearbyTransitStopsProviders**" table structure

2.2.15 Table *PtStopTimetableProviders*

The structure of the PtStopTimetableProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL

output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
tripType_in	varchar(100)	Yes	NULL
isRealTime_in	varchar(100)	Yes	NULL
time_in	varchar(100)	Yes	NULL
erro_out	varchar(100)	Yes	NULL
publicCode_out	varchar(100)	Yes	NULL
shortName_out	varchar(100)	Yes	NULL
stopPointID_out	varchar(100)	Yes	NULL
stopPointName_out	varchar(100)	Yes	NULL

Table 15: The "**PtStopTimetableProviders**" table structure

2.2.16 Table *RoadConditionsProviders*

The structure of the RoadConditionsProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
date_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
pubReportTimeDate_out	varchar(100)	Yes	NULL

Column	Type	Null	Default
date_out	varchar(100)	Yes	NULL
langitude_out	varchar(100)	Yes	NULL
longitude_out	varchar(100)	Yes	NULL
weatherRelatedRoadConditionType_out	varchar(100)	Yes	NULL

Table 16: The "**RoadConditionsProviders**" table structure

2.2.17 Table *SelectedProvider*

The structure of the SelectedProvider database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
ServiceName	varchar(45)	No	
SelectedProviderID	int(11)	No	
SelectedCounter	int(11)	No	

Table 17: The "**SelectedProvider**" table structure

2.2.18 Table *WeatherForecastProviders*

The structure of the WeatherForecastProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
barPressureUnits_in	varchar(100)	Yes	NULL
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
date_in	varchar(100)	Yes	NULL
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL

Column	Type	Null	Default
tempUnits_in	varchar(100)	Yes	NULL
windSpeedUnits_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
pubReportTimeDate_out	varchar(100)	Yes	NULL
date_out	varchar(100)	Yes	NULL
dayOfTheWeek_out	varchar(100)	Yes	NULL
description_out	varchar(100)	Yes	NULL
high_out	varchar(100)	Yes	NULL
low_out	varchar(100)	Yes	NULL
tempUnits_out	varchar(100)	Yes	NULL
humidity_out	varchar(100)	Yes	NULL
pressure_out	varchar(100)	Yes	NULL
pressureUnits_out	varchar(100)	Yes	NULL
rising_out	varchar(100)	Yes	NULL
visiblity_out	varchar(100)	Yes	NULL
chill_out	varchar(100)	Yes	NULL
direction_out	varchar(100)	Yes	NULL
speed_out	varchar(100)	Yes	NULL
speedUnit_out	varchar(100)	Yes	NULL

Table 18: The "**WeatherForecastProviders**" table structure

2.2.19 Table *WeatherProviders*

The structure of the WeatherProviders database table is illustrated below.

Column	Type	Null	Default
id	int(11)	No	
Name	varchar(45)	No	
type	varchar(45)	No	
url	varchar(200)	No	
operationName	varchar(45)	Yes	NULL
output	varchar(45)	Yes	NULL
email	varchar(45)	Yes	NULL
implemented	tinyint(4)	Yes	0
Ratio	double	No	1
Reputation	double	No	1
SelectedCounter	int(11)	No	0
QoS	double	No	1
Availability	double	No	1
ResponseTime	double	No	0
RespTimeCounter	int(11)	No	0
Similarity	double	No	1
UserRank	double	No	3
barPressureUnits_In	varchar(100)	Yes	NULL
city_in	varchar(100)	Yes	NULL
country_in	varchar(100)	Yes	NULL
date_in	varchar(100)	Yes	NULL

Column	Type	Null	Default
latitude_in	varchar(100)	Yes	NULL
longitude_in	varchar(100)	Yes	NULL
tempUnits_in	varchar(100)	Yes	NULL
windSpeedUnits_in	varchar(100)	Yes	NULL
error_out	varchar(100)	Yes	NULL
pubReportTimeDate_out	varchar(100)	Yes	NULL
city_out	varchar(100)	Yes	NULL
country_out	varchar(100)	Yes	NULL
description_out	varchar(100)	Yes	NULL
latitude_out	varchar(100)	Yes	NULL
longitude_out	varchar(100)	Yes	NULL
publicationDate_out	varchar(100)	Yes	NULL
title_out	varchar(100)	Yes	NULL
weatherCondDescription_out	varchar(100)	Yes	NULL
tempUnits_out	varchar(100)	Yes	NULL
temperature_out	varchar(100)	Yes	NULL
timeDate_out	varchar(100)	Yes	NULL
chill_out	varchar(100)	Yes	NULL
direction_out	varchar(100)	Yes	NULL
speed_out	varchar(100)	Yes	NULL
speedUnit_out	varchar(100)	Yes	NULL

Table 19: The "**WeatherProviders**" table structure

2.3 Implementation Details

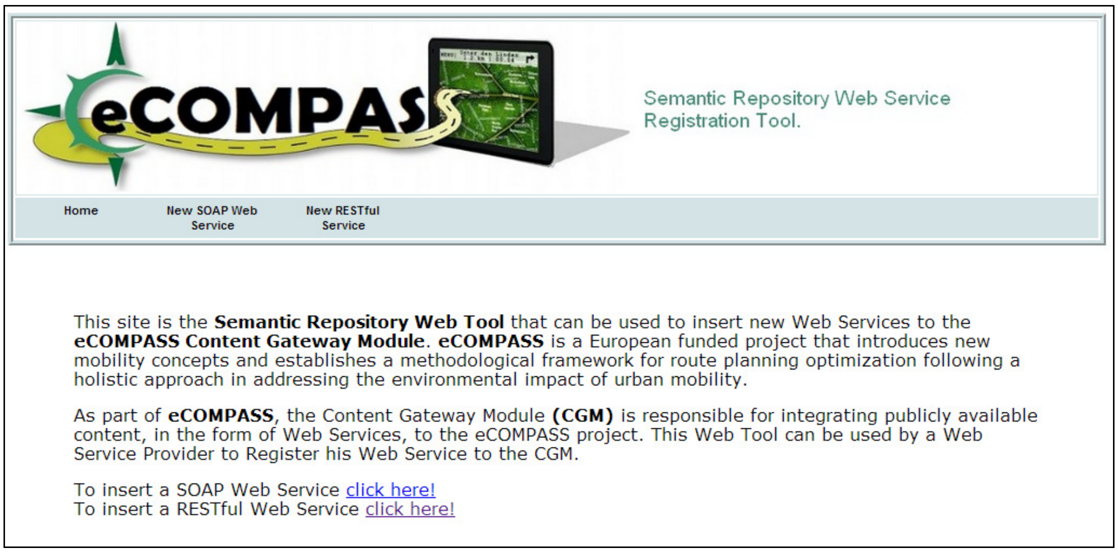
- **Semantic Content Repository** has been realized using MySQL v. 5.5.27 and Apache web server v. 2.4.3.
- **Web Registration Tool's** functionality has been programmed in PHP 4.0. The tool is hosted on an Apache web server v. 2.4.3

3 Web Registration Tool

In this section the appropriate documentation for using the WRT is provided in the form of illustrative examples of use for each step of the WS registration process. The WRT is available at the following URL:

http://160.40.50.57/ecompass/

The welcome page (Figure 3) contains some introductory instructions to the provider and how he/she should proceed.



This site is the **Semantic Repository Web Tool** that can be used to insert new Web Services to the **eCOMPASS Content Gateway Module**. **eCOMPASS** is a European funded project that introduces new mobility concepts and establishes a methodological framework for route planning optimization following a holistic approach in addressing the environmental impact of urban mobility.

As part of **eCOMPASS**, the Content Gateway Module **(CGM)** is responsible for integrating publicly available content, in the form of Web Services, to the eCOMPASS project. This Web Tool can be used by a Web Service Provider to Register his Web Service to the CGM.

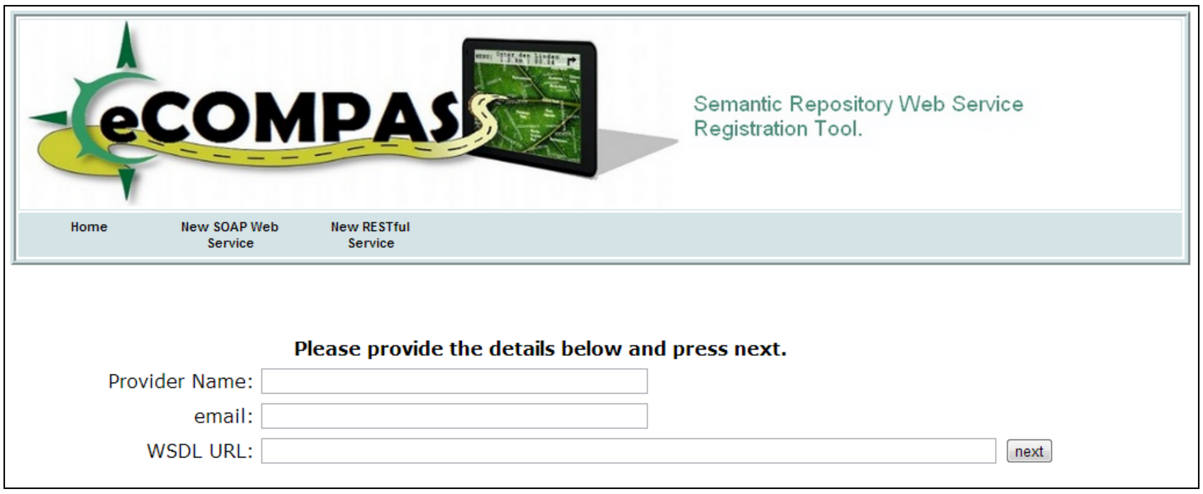
To insert a SOAP Web Service click here!
To insert a RESTful Web Service click here!

Figure 3: Welcome page of the Web Service Registration Tool

Through this page, the provider can register WSs that rely on the SOAP or REST protocols, through the available links. The procedure that is adopted is different for the two services, therefore we present them in two separate sections in what follows.

3.1 SOAP Services

Once the SOAP WS option is selected, the interested SP is redirected to the page that is shown in Figure 4.



Please provide the details below and press next.

Provider Name:	
email:	
WSDL URL:	

Figure 4: SOAP web service details input form

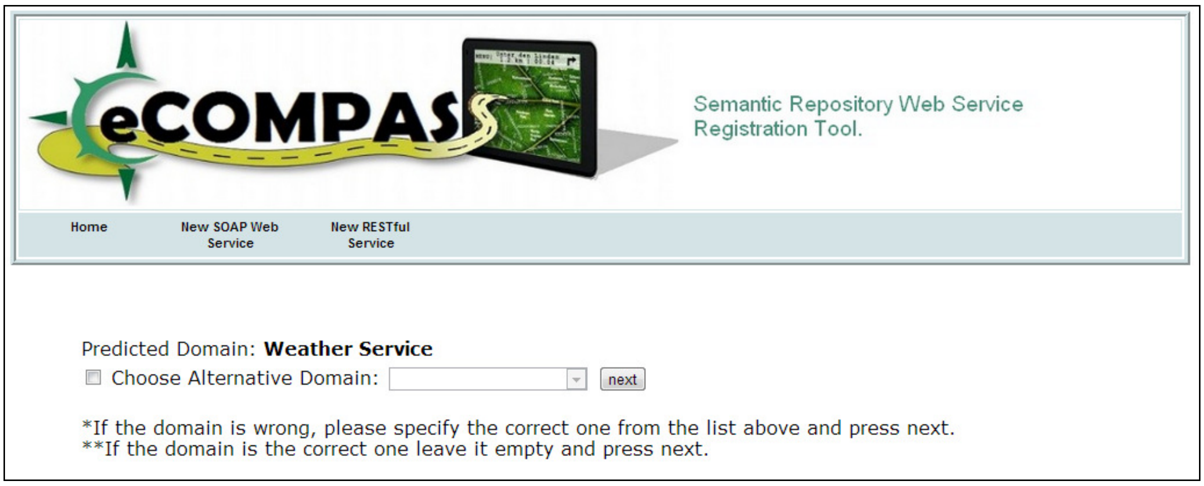
In this page the following information should be filled in: Provider's name, Provider's email and the online WSDL URL that describes the operations supported by the SOAP WS in question. It should be noted that both WSDL 1.1 and 2.0 versions are supported by the WRT.

When the "next" button is pressed, the backend mechanism tries to parse the online WSDL file containing the WS description and extracts all the required information containing in it. The purpose of this process is to extract the WS operation names, inputs and outputs names and the documentation that is available in the WSDL file. With this information, the framework can classify the WS that will be registered into the predefined domains of services shown in Table 20.

Service domain	Description
Fuel Services	Includes those web services that report the fuel prices at a specific region upon request.
Map Services	Web services that provide maps of a specific region, city or area (in the form of images), as well as GIS data (e.g. shape files), typically provided by a Web Map Server (WMS).
Points of Interest Services	These web services return a list of points of interest for a specific region
Public Transport Services	Services that provide static or dynamic information about specific means of public transport, e.g. bus timetables, estimated time to arrive at a specific stop, etc.
Traffic Data Services	Either historic or real time data about traffic conditions, usually provided by loop detectors or
Weather Services	Services about reporting current or forecasted weather conditions for a specific city or area.

Table 20: The available service domains of the web service registration tool

The WS registration process, based on the WSDL description, tries to predict the service domain to which the incoming service belongs and the result is presented in the "Predicted Domain" field.



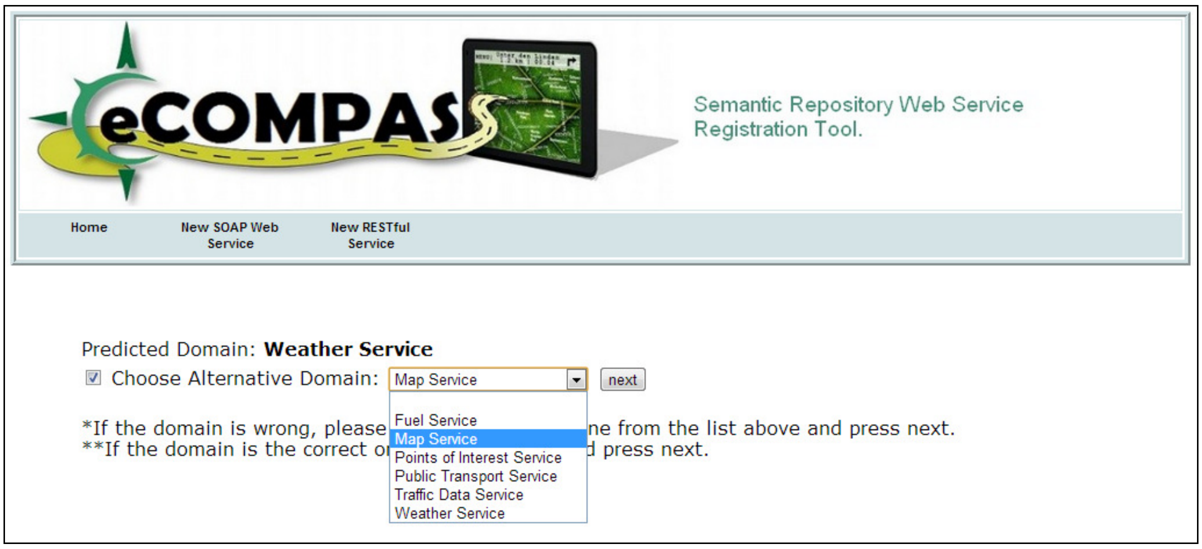
Predicted Domain: **Weather Service**

☐ Choose Alternative Domain:

*If the domain is wrong, please specify the correct one from the list above and press next.
**If the domain is the correct one leave it empty and press next.

Figure 5: Web service domain classification result

In case that the classification mechanism fails to predict the domain of the WS that is to be registered, the provider may select the domain of his preference at the dropdown box shown in Figure 6 and press "Next" in order to proceed to the next step.



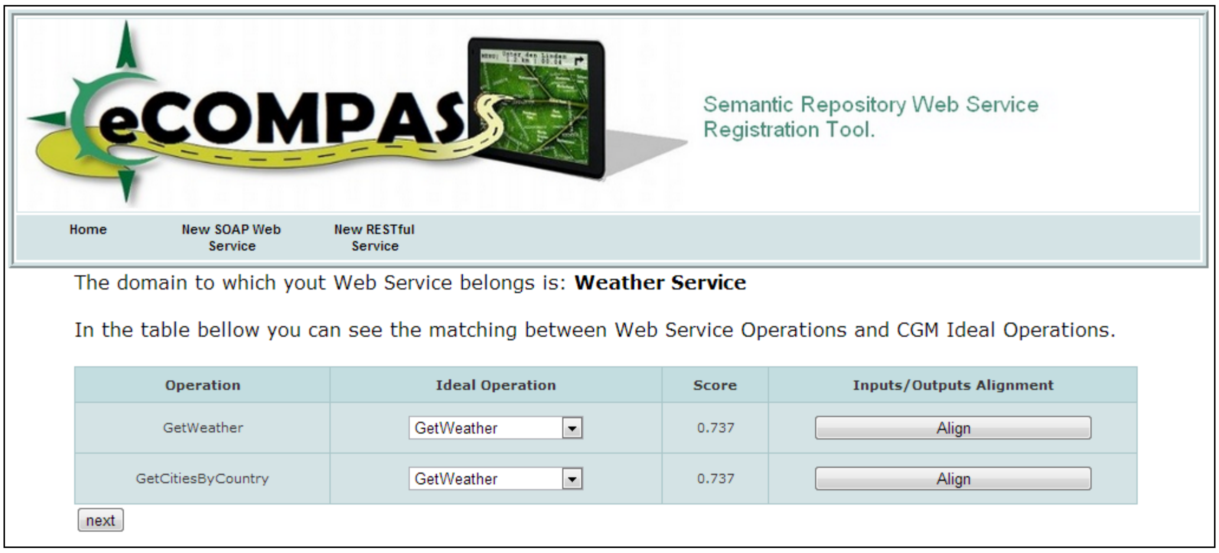
Predicted Domain: **Weather Service**

☑ Choose Alternative Domain: Map Service

*If the domain is wrong, please ne from the list above and press next.
**If the domain is the correct o d press next.

Figure 6: Web service domain selection

The next page (Figure 7) contains an overview of the operations of the WS.



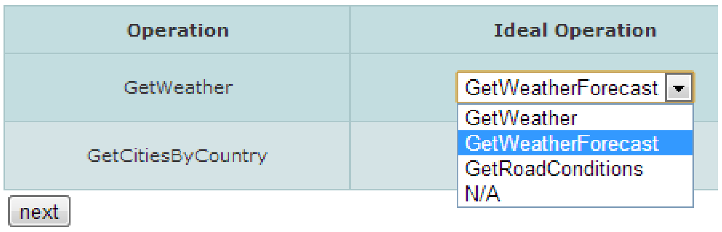
The domain to which yout Web Service belongs is: **Weather Service**

In the table bellow you can see the matching between Web Service Operations and CGM Ideal Operations.

Operation	Ideal Operation	Score	Inputs/Outputs Alignment
GetWeather	GetWeather	0.737	Align
GetCitiesByCountry	GetWeather	0.737	Align

Figure 7: Web service operation classification results

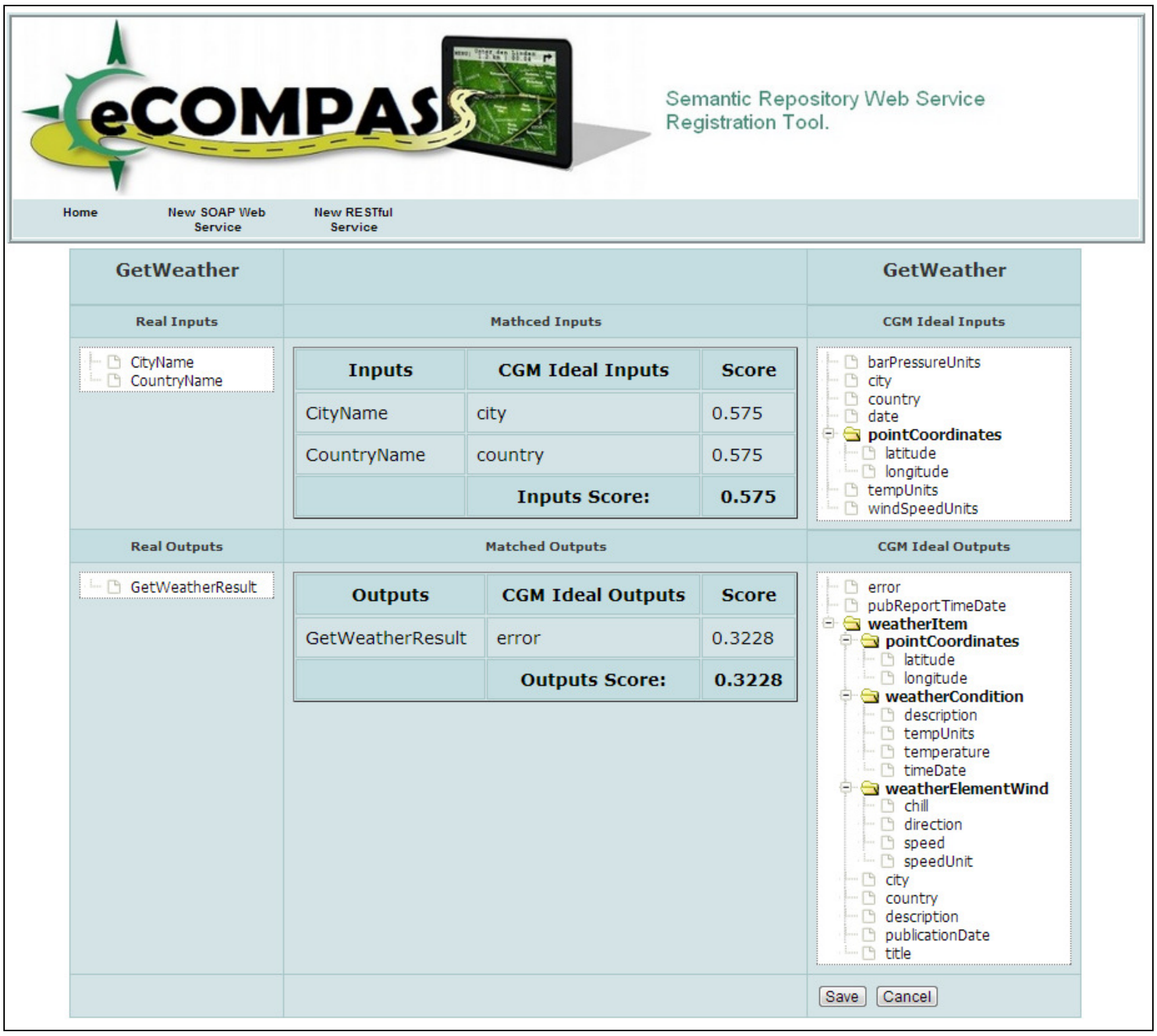
More specifically, Figure 8 displays for each operation of the WS the matched predefined "ideal" operation along with the matching score. The SP may change the matched ideal operation from the dropdown box and align the operation with the ideal of his/her preference. It should be noted that the alignment score is computed by the tool with the use of complex lexicographic algorithms, and will be recomputed in the case that the developer selects another ideal operation than the proposed one. The details of the WS categorization process are described in Section 4.



Operation	Ideal Operation
GetWeather	GetWeatherForecast
GetCitiesByCountry	

Figure 8: Provider may select another ideal operation

The alignment of each operation can be performed by pressing the "Align" button of the corresponding row, where the following page (Figure 9) will appear:



Inputs	CGM Ideal Inputs	Score
CityName	city	0.575
CountryName	country	0.575
	Inputs Score:	**0.575**

Outputs	CGM Ideal Outputs	Score
GetWeatherResult	error	0.3228
	Outputs Score:	**0.3228**

Figure 9: Input/Output classification results

In this page, the inputs and outputs of the WS operation are displayed in a tree-like structure in the left column, while the corresponding trees of the ideal operation are displayed in the right column. The table in the centre of the page provides the matching of the inputs/outputs between the WS operation and the ideal operation. Each row of the table contains the names of the i/o of the WS operation and the ideal operation, along with the matching score between these two concepts. The total input and output score is also presented.

In case that the matching algorithm fails to match an input or output parameter of a WS operation, the SP may correct the alignment between two concepts, just by dragging one concept from the left trees and drop it to a concept of the corresponding right tree (see Figure 10). By doing so, the new matching is displayed in the table and the new scores are computed.

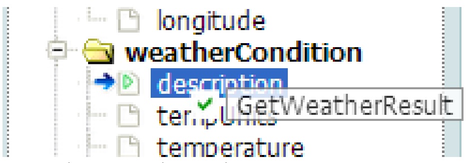
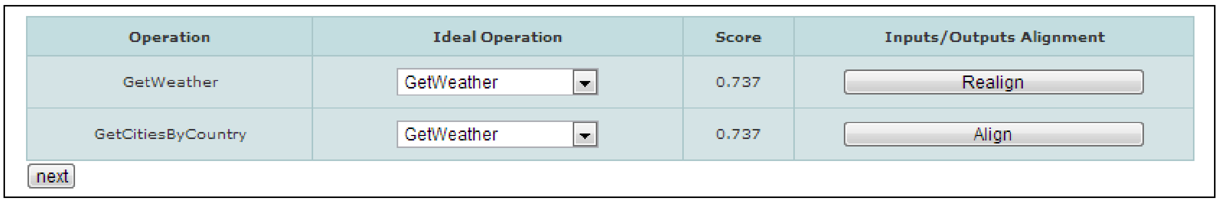


Figure 10: Changing the matching between two concepts (drag and drop)

It should be noted that alignment between inputs and outputs is not allowed. Additionally, alignment is a 1 to 1 process, i.e. each concept of each tree may have only one matching.

When a SP reviews the alignments and fine tunes the matches, he/she can finally register the WS operation with all this information to the SCR by pressing the "Save" button. In case that the alignment was wrong, the SP may realign the service with the appropriate button, as shown in Figure 11.

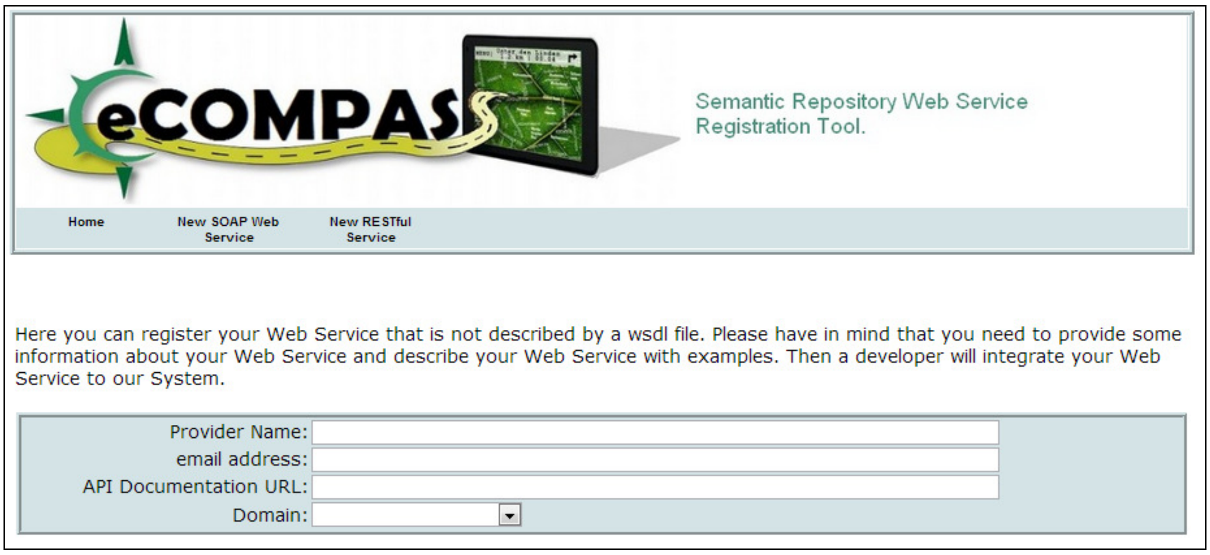


Operation	Ideal Operation	Score	Inputs/Outputs Alignment
GetWeather	GetWeather	0.737	Realign
GetCitiesByCountry	GetWeather	0.737	Align

Figure 11: Realignment of an operation is available only if the operation is already aligned

3.2 Restful Services

The process of registering a WS that relies to the REST protocol is quite straight forward. After the appropriate selection to the welcome menu, the provider is redirected to the page shown in Figure 12.



Here you can register your Web Service that is not described by a wsdl file. Please have in mind that you need to provide some information about your Web Service and describe your Web Service with examples. Then a developer will integrate your Web Service to our System.

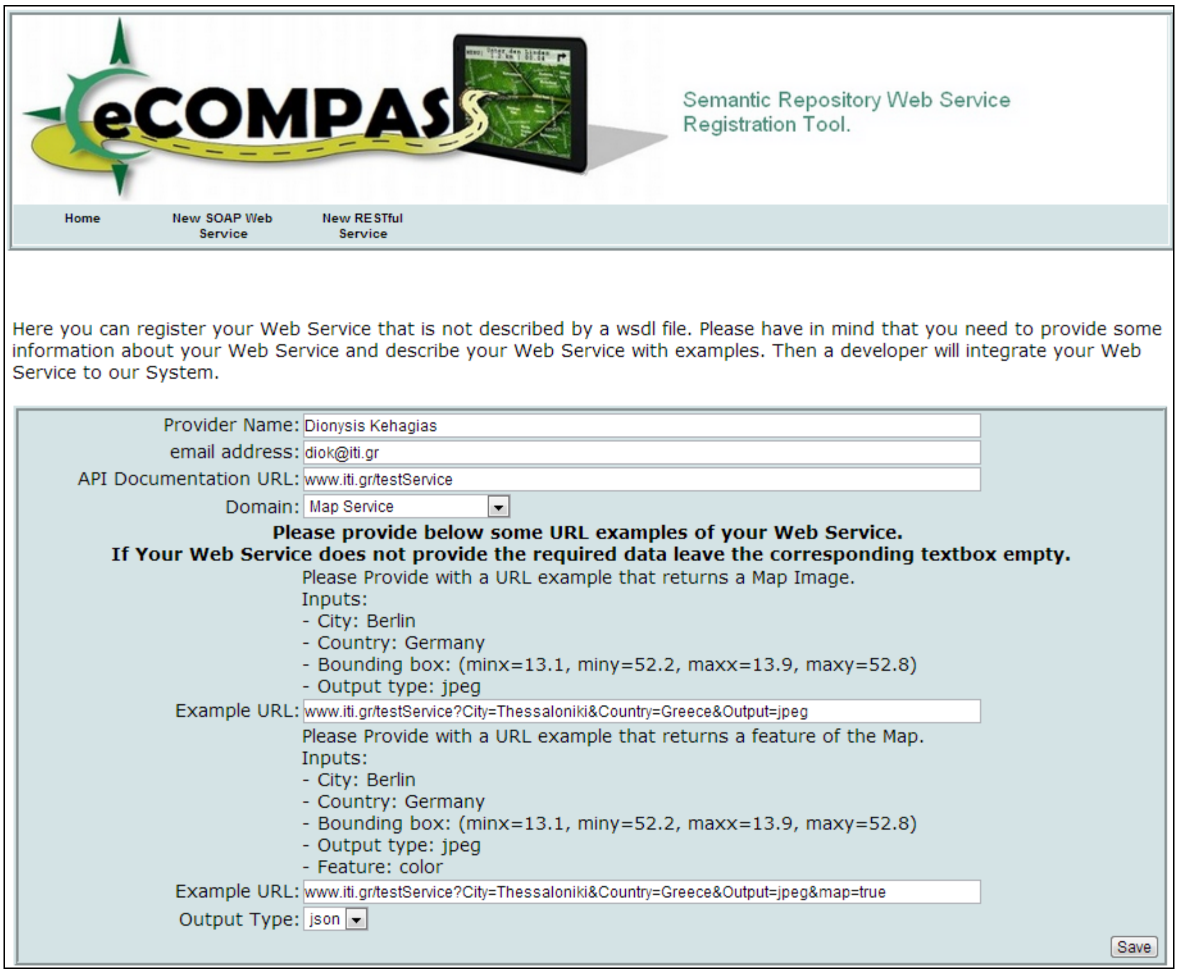
Provider Name:	
email address:	
API Documentation URL:	
Domain:	

Figure 12: REST service details input form

The following data should be inserted:

- The provider's name
- The provider's email
- The URL containing the documentation of the REST service.

Additionally, the provider should select the domain that the WS belongs to, from the list of predefined domains under the dropdown box. When the domain is selected, some additional information is needed about URL examples of the WS in order to extract the required parameters of the REST API.



Here you can register your Web Service that is not described by a wsdl file. Please have in mind that you need to provide some information about your Web Service and describe your Web Service with examples. Then a developer will integrate your Web Service to our System.

Provider Name: Dionysis Kehagias
email address: diok@iti.gr
API Documentation URL: www.iti.gr/testService
Domain: Map Service

Please provide below some URL examples of your Web Service.
If Your Web Service does not provide the required data leave the corresponding textbox empty.

Please Provide with a URL example that returns a Map Image.
Inputs:
- City: Berlin
- Country: Germany
- Bounding box: (minx=13.1, miny=52.2, maxx=13.9, maxy=52.8)
- Output type: jpeg

Example URL: www.iti.gr/testService?City=Thessaloniki&Country=Greece&Output=jpeg

Please Provide with a URL example that returns a feature of the Map.
Inputs:
- City: Berlin
- Country: Germany
- Bounding box: (minx=13.1, miny=52.2, maxx=13.9, maxy=52.8)
- Output type: jpeg
- Feature: color

Example URL: www.iti.gr/testService?City=Thessaloniki&Country=Greece&Output=jpeg&map=true
Output Type: json

Figure 13: REST service invocation examples

Finally, the provider should select the output type of the service between the two options "JSON[1]" and "XML[2]" formats. When the "Save" button is pressed, the service information is stored into the registry and the service will be ready for usage through the CGM.

It should be noted that the automatic categorization mechanism is not operational in the case of REST services. The reason for this is that automatic categorization relies entirely on information retrieval techniques, which are applied on machine interpretable descriptions of the WS operations. These descriptions are provided in the form of WSDL files in the case of SOAP WS. However, as opposed to SOAP, such files are not mandatory in the case of REST WS. On the contrary, the descriptions of REST WS are not provided in a standard structured way, but they are mostly available in the form of human-readable API documentation.

[1] http://www.json.org/
[2] http://www.w3.org/XML/

4 Web Service Categorization Mechanism

This section describes the prototype WS Categorization Mechanism of eCOMPASS Content Gateway Module that was developed in the scope of Task 4.3. The main purpose of this mechanism is to provide automatic or semi-automatic semantic characterization of all new WSs that are registered to the CGM in order to provide appropriate content, based on the categories that are supported (e.g. Weather services, POI info services, map services, etc.). The automatic mode is supported for SOAP WSs, as these provide textual descriptions in WSDL that can be parsed and be provided as an input to a text pre-processing mechanism in sequence to an information retrieval system. The output of such a system can be used for the efficient categorization of WS and its elements with respect to the communication interfaces that are defined in CGM.

In order to implement the capability of automatic semantic annotation of SOAP WSs, we present in this section a technique for automatic *categorization* of non-semantic WS, i.e., based on machine understandable descriptors that lack semantics. The WS categorization technique developed and deployed within CGM, aims to predict the application domain, to which the WS belongs, hence it provides semantic *categorization of* any WS to its potential domain. On top of this, the WRT also performs categorization of WS operations and input/output (i/o) parameters into existing descriptive classes. In what follows the different approaches for domain, operation and i/o parameter categorization are presented.

4.1 Domain Classification

The WSs that are supported by CGM and are relevant to eCOMPASS can be classified into the following domains:

- Traffic Prediction
- Weather services
- POI info services
- Map services
- Public Transport
- Fuel Services
- Logistics

For implementing automatic classification of SOAP WSs in one of the available domains, the eCOMPASS WRT, adopts the following approach.

4.1.1 Data Pre-processing and Preparation

The automatic WS categorization mechanism receives as input a collection of WS description documents *D*. Figure 14 describes the data pre-processing procedure that is applied to the initial dataset, whose purpose is to "clean" the data by removing words included in a stop-word list and by applying a stemming algorithm. The output of the pre-processing function is a set *V* of distinct words.

Initially, *D* is split into training and test sets by the application of function *split1* on dataset *D*. Splitting is performed in one of the two following ways: a) we select a specific number of random instances within the original dataset in order to form two concrete, non-overlapping test and training subsets, and b) we apply the leave-one-out cross validation technique [1] in order to create the test dataset. Based on the latter technique, a single instance from the original dataset is used as test data, and the remaining observations comprise the training dataset. This process is repeated for each instance that occurs in the original dataset.

```
Data-Pre-Processing($D$) : $V$

01. $D$: WS collection; $V$: feature vector
02. $V \leftarrow$ null;
03. $\{D_{tr}, D_{ts}\} \leftarrow$ split1($D$);
04. For each $w_i \in D_{tr}$
05.     $S \leftarrow$ parse($w_i$)
06.     $\{s_1, \ldots, s_i\} \leftarrow$ split2($S$)
07.     $S' \leftarrow$ stopwords($S$)
08.     $S'' \leftarrow$ stemming($S'$)
09. End for
10. For each $s_m \in S''$
11.   If $V \neq \varnothing$
12.      If $V \cap \{s_m\} = \varnothing$
13.         $V \leftarrow V \cup \{s_m\}$
14.      End if
15.   End if
16. End for
17. Return $V$
```

Figure 14: Data pre-processing procedure for the training dataset.

Set *S* is generated as a "bag of words" after all available WSDL documents w_i are parsed (line 05), and their features regarding operations, i/o parameters are extracted. From the various elements that comprise a WSDL document only the operations, along with their corresponding input and output parameters are taken into account, because these elements include all necessary information that is relevant to the operational characteristics of the corresponding WS (i.e., they describe the functionality of the WS). Also, additional information provided by the optional <documentation> tag that encompasses human-readable documentation inside any part of the WSDL document is also taken into account.

Function *split2* is then applied (Fig. 1, line 06) to the extracted bag of words for splitting them into distinct tokens. Splitting is performed in different ways for operation and i/o elements, respectively. In particular, when a word corresponds to an operation name the most appropriate naming convention among the ones utilized by WS developers is taken into account, whereas a different one is assumed to be adopted when it comes to i/o parameters. Generally, a valid operation name includes strings written in camel case or strings using the underscore character '_' to join separate words. The generated operation name usually describes the functionality of the operation in a developer-readable fashion. Taking this into account, we apply string manipulation functions, in order to extract those joint words from operation and i/o parameter names. For example, the following operation names "getCountryCodes", "get_company_profile" and "get_Weather_byZIP" result in lists containing the word tokens ["get", "Country", "Codes"], ["get", "company", "profile"] and ["get", "Weather", "by", "ZIP"], respectively. Operations are usually accompanied by <documentation> tag elements, which contain human readable comments that some developers usually provide. We also extract all distinct words from the <documentation> tags, which are separated by space characters.

In the next stage (Figure 14, lines 07-08), all the extracted tokens are filtered so that only unique elements remain. Firstly, all words are filtered by means of a stop-word list. The stop-word list contains articles, prepositions, WS-related words and generally words that appear frequently in WSDL documents, and therefore are not discriminated (e.g. the words *the*, *a*, *soap*, etc.). Furthermore, words that correspond to HTML tags or web links (which often are found in the documentation tags) are also removed. Generally, removing stop words is considered a necessary step for filtering non-relevant terms [2].

The next step involves removal of inflectional endings that results in reducing words to their stems by applying the Porter stemmer algorithm [3]. The unique features that remain after the pre-processing actions comprise the feature set (vocabulary) $V = \lfloor t_0, t_1, ..., t_{|V|} \rfloor$ that is used for representing each WS as a vector.

4.1.2 Reduction of the Feature Space Dimension

The feature set *V* that was generated as the output of the previous pre-processing procedure represents the training dataset. Specifically, the frequency *f* of occurrence of each feature is calculated for each WS, denoted as ws_i, and is used for forming the following Vector Space Model (VSM) $ws_i = [f_{i0}, f_{i1}, ..., f_{i|V|}]$ were the variable f_{ik} is equal to the frequency of occurrence of term t_k for the feature *k*. The same representation is also used for the operation categorization task.

Term weighting schemes such as *tf-idf* (term frequency-inverse document frequency) [5] are also valid and could be used instead of term frequency. However, although such techniques vectorize the data easily, the number of dimensions is equal to the number of features [6]. Since the WS descriptions can be very large, the number of the extracted features may typically range from a few hundreds to several thousands, thus causing an overfitting effect, having also a negative impact on the overall procedure performance.

In general the most typical task required for reducing the number of extracted features is to select only a subset of all features according to some feature selection criterion (e.g. based on the chi-squared statistic). Feature selection techniques use a scoring function in order to assign a score to each feature, so that only those features with the highest score are maintained among all features. Although feature selection techniques generally reduce the complexity of the commonest classification algorithms, thus increasing their accuracy, the effect of term removal, as a consequence of feature selection, is to increase the risk of removing potentially useful information [7].

In order to avoid this, we propose an alternative feature dimension reduction technique, which preserves all extracted terms that result after the application of the pre-processing procedure. The goal of our technique is to transform the training data to a lower dimension format, in order to reduce the number of feature dimension in question, by using the Bayes' theorem. This approach seems to improve the WS classification performance, in terms of required training time and classification accuracy, when used with a Support Vector Machine (SVM) classifier for text categorization [6]. Based on this, our technique adopts the use of the Bayesian theorem in conjunction with a Logistic Model Trees (LMT) classifier [8] in order to improve the accuracy of WS categorization.

Bayesian classifiers make strong assumptions about how the data are generated and posit a probabilistic model that embodies these assumptions. Then, they use the training data to estimate the parameters of the generative model. The classification of a new example is based on the Bayes' theorem by selecting the class that is most likely to have generated the example [9]. In our work, we apply the multinomial Naïve Bayes model [10] because it shows the best performance on conducting text classification. In the context of WS classification, the domain *c* to which a WS ws_i belongs, is determined as the one for which the probability $\mathrm{P}(c_j \mid ws_i)$ that ws_i belongs to the domain c_j, has its maximum value. $\mathrm{P}(c_j \mid ws_i)$ is calculated by the application of Bayes' theorem, as shown in Eq. (1). The term $\mathrm{P}(ws_i \mid c_j)$ is equal to the probability that for a given domain c_j, all features of ws_i occur in that domain. P(c_j) is the probability that ws_i belongs to the domain c_j, whereas P(ws_i) is the probability of occurrence of ws_i. For operation categorization, the same equations apply, but instead of WS instances (ws_i) we consider operation instances (op_i) and domains (c_j) are replaced by *ideal*, i.e., semantically described, operations (id_j).

$$\mathrm{P}(c_j \mid ws_i) = \frac{\mathrm{P}(ws_i \mid c_j)\mathrm{P}(c_j)}{\mathrm{P}(ws_i)} \tag{1}$$

where P(c_j) is calculated by Eq. (2) as the number $N_{t\in c_j}$ of features in the domain c_j, divided by the total number of features in the training dataset $N_{t\in D_{tr}}$. It is not necessary to calculate the value of P(ws_i) because it is always fixed. $\mathrm{P}(ws_i \mid c_j)$ is eventually calculated by Eq. (3) as the product of all probabilities of each feature t_k appearing in c_j.

$$\mathrm{P}(c_j) = \frac{N_{t\in c_j}}{N_{t\in D_{tr}}} \tag{2}$$

$$\mathrm{P}(ws_i \mid c_j) = \prod_{k=0}^{|V|} \mathrm{P}(t_k \mid c_j) \tag{3}$$

Each probability P(t_k | c_j) is calculated as the number $n_{t_k\in c_j}$ of occurrences of the feature t_k in the domain c_j divided by the total number of occurrences of all features in c_j.

$$\mathrm{P}(t_k \mid c_j) = \frac{n_{t_k\in c_j}}{\sum_{m=0}^{|V|} n_{t_m\in c_j}} \tag{4}$$

An alternative way of computing $p(t_k \mid c_j)$ is by deploying the Laplacian smoothing operation [11], as it is shown in Eq. (5). The basic idea is to add a constant term both to the numerator and denominator of Eq. (4), in order to smooth the estimation of P(t_k | c_j), in case a feature that does not occur in the training set occurs in the test set only.

$$\mathrm{P}(t_k \mid c_j) = \frac{1 + n_{t_k\in c_j}}{|V| + \sum_{m=0}^{|V|} n_{t_m\in c_j}} \tag{5}$$

4.1.3 Building the Classifier

The newly vector model representation of the training dataset is used as input for building a LMT classifier. LMT is a relatively new classification algorithm, which combines logistic regression and decision trees. We adopt LMT because it performs efficiently on small and/or noisy datasets. Another advantage of using logistic regression is that it produces as output explicit probability estimates for each class, rather than suggesting one output class.

LMT consists of a standard decision-tree structure with logistic regression functions at the leaves. It contains *N* inner nodes and a set of leaves (terminal nodes) *T*. Let *S* denote the whole instance space, spanned by all features that are present in the data. The tree structure gives a disjoint subdivision of *S* into regions S_t, and every region is represented by a leaf in the tree, as shown in Eq. (6).

$$S = \bigcup_{t \in T} S_t, \; S_t \cap S_{t'} = \varnothing, \; t \neq t' \tag{6}$$

Unlike ordinary decision trees, the leaves $t \in T$ have an associated logistic regression function F_t instead of just a class label. The regression function F_t takes into account a subset of all features present in the data. The *LogitBoost* algorithm [12] was adopted for building the logistic regression functions at the tree nodes, which uses the well-known *CART* algorithm [13] for tree pruning.

Given a set of k classes to be learnt, n different bi-partitions are formed for training n binary classifiers. As a result, a code word of length n is obtained for each class. The classes are encoded into code words and then a binary classifier is trained for each bit position. The n binary classifiers produce n binary predictions for a certain input instance (i.e., a WS when domain categorisation is concerned). For the selection of the dominant class the Hamming distance[3] is computed between the predicted code word and the code word of each class. The class with the minimum Hamming distance is selected as the dominant class.

The Hamming distance between two strings of equal length is the number of positions, for which the corresponding symbols are different. In other words, the Hamming distance is used for measuring the minimum number of substitutions that are required to change one code word into the other. If the minimum Hamming distance is d, then the code can correct at least (d-1)/2 1-bit errors [14]. Thus, if we make (d-1)/2 errors, the nearest code word will still be the correct codeword.

Figure 15 illustrates the various steps of the WS domain categorisation process. For the classification of a test WS, the output code word from the n classifiers is compared to the class code words, and the one with the minimum Hamming distance is selected to be the class label. Before the test WS is passed to the decision tree classifier, it has to be pre-processed in order to represent it according to the vocabulary V, which was determined at the training phase. After that the test WS is transformed according to Bayes formula (in the same way as for the training data) and is given as input to the binary LMT classifiers in order to predict the class to which the WS belongs. The leaves of each LMT correspond to logistic functions, which produce the class probabilities for the two binary classes "0" and "1".

4.2 Operation Classification

The purpose of operation categorization is to predict the semantics that characterize a randomly selected WS operation. To this end, we construct a classifier as described in the previous subsection for each WS domain, after applying the pre-processing procedure and the Bayes theorem, as discussed in sections 4.1.1-4.1.3.

In the case of operation categorization, the training dataset consists of operation semantics, described in appropriate ontologies, which are expressed in the Web Ontology Language (OWL). The various steps of the WS operation categorization process, are described as follows. Firstly, a pre-processing procedure is conducted to a test WS, for extracting the initial set of features. Then, we represent the WS according to the VSM that results from the application of the Bayes' theorem (as described in Section 4.1.2 for training data). The transformed WS operation is provided as an input to the n binary LMT classifiers, which produce a binary output.

[3] http://en.wikipedia.org/wiki/Hamming_distance

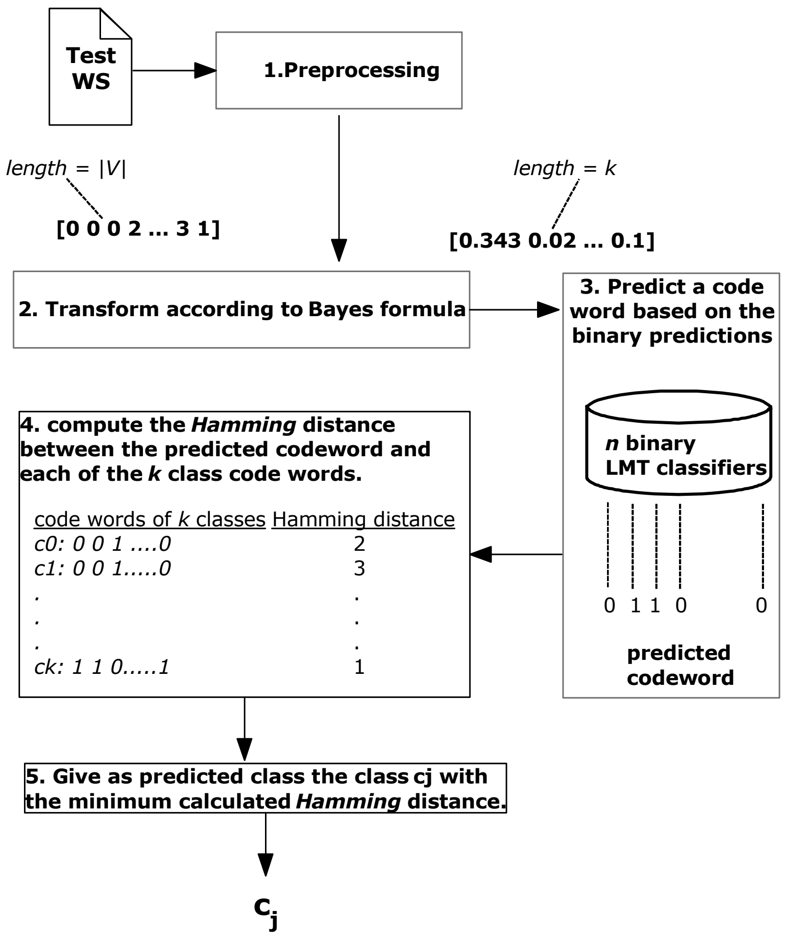

Test WS
1.Preprocessing
length = |V|
[0 0 0 2 … 3 1]
length = k
[0.343 0.02 … 0.1]
2. Transform according to Bayes formula
3. Predict a code word based on the binary predictions
n binary LMT classifiers
0 1 1 0 0
predicted codeword
4. compute the Hamming distance between the predicted codeword and each of the k class code words.
code words of k classes Hamming distance
c0: 0 0 1 ….0 2
c1: 0 0 1…..0 3
ck: 1 1 0…..1 1
5. Give as predicted class the class cj with the minimum calculated Hamming distance.
cj

Figure 15: Classification of an unknown web service.

In order to selecting the class label, we perform the following adaptation. We use *WordNet:Similarity* with *Jiang* metric, [15], for comparing lexicographically the name of the test WS operation with the names of the ideal operations of the domain ontology. Let *s* be the name of a randomly selected test WS operation and id_i the names of the ideal operations in the same domain, $i = 0 \ldots t$. For each ideal operation name id_i the *WordNet:Similarity* algorithm gives a matching score wn_i that reflects the matching degree of the two words. We then combine those scores with the probabilities produced by our classifier for each class in the following way. The classifier produces a probability for each one of the classes, computed by summing the probabilities that correspond to the binary predictions after normalizing the resulted sum. Thus, given the set of classes ID = $\{id_o, id_1, \ldots, id_t\}$ a set of corresponding probabilities Pr = $\{pr_o, pr_1, \ldots, pr_t\}$ are calculated. Then, for each id_i we calculate the final score as $ts_i = pr_i + wn_i$.

This additional lexicographic check improves the categorization results, tacking especially the case of rare instances, i.e., instances that correspond to classes with a few training data.

4.3 Input/Output Matchmaking

In the case of i/o categorization the previous categorization mechanism, adopted in both domain and operation categorization, cannot be applied, due to the lack of large manually annotated datasets. For this reason, an alternative algorithmic approach is proposed, which is based on the *Wordnet* lexical database.

The purpose of i/o categorization is to map one by one all i/o parameters from a WS operation to the appropriate i/o parameters of ideal operations defined in the domain ontology. The new proposed technique is based on a one-to-one similarity comparison between the i/o elements, with respect to their names, hierarchy and data type. For this purpose, Wordnet lexical database [15] has been utilized, for extracting words' Synsets (i.e., cognitive synonyms) and Hypernyms (i.e., semantic annotated synsets with more general meanings). The proposed algorithm for calculating the similarity scores between the real and ideal i/o parameters is described in Figure 16.

The goal of the *Input-Output-Similarity* procedure, which is described in pseudo-code in Figure 16, is to produce the similarity matrix ***S*** that contains the similarity scores for each pair of ideal-real i/o parameters. After calculating *S*, the Hungarian algorithm is applied in order to determine the best matchmaking between real-ideal i/o parameters. The algorithm takes as input an arbitrary WS, denoted as *ws*, whose operations we would like to semantically categorize and the corresponding WS ontology (denoted as ***O***) that describes in ontological form the class of WS to which ws belongs. As a first step, the algorithm parses the WSDL description of the particular WS operation we wish to categorize, using function *parseWSDL*, and puts the extracted i/o parameters along with their parent nodes into vector V_{io}. Then, we split each element $v_i \in V_{io}$, i=1,…, $|V_{io}|$ into tokens $T_i \doteq \{t_1, t_2, ...\}$ and for each $t_j \in T_i$, j=1,…, $|T_i|$ we compute its Synset $S_{i,j}$ and Hypernyms $H_{i,j}$ sets, and then their union $C_{i,j} \doteq S_{i,j} \cup H_{i,j}$. In a similar way, we parse the ideal operation using an OWL parser, which extracts the corresponding i/o parameters and puts them into vector V'_{io} along with their parent nodes in the ideal operation hierarchy. Again, each element $v'_k \in V'_{io}$, k=1,…, $|V'_{io}|$ is split into tokens $T'_k \doteq \{t'_1, t'_2, ...\}$. For each t'_l , l=1,…, $|T'_k|$ the Synsets $S'_{k,l}$ and Hypernyms $H'_{k,l}$ sets are calculated, and then their union set $C'_{k,l} \doteq S'_{k,l} \cup H'_{k,l}$. In Figure 17, the whole process of extracting the Synsets and Hypernyms of two operation inputs is presented.

The algorithm is directed to the comparison of sets of words. Sets $C_{i,j}$ and $C'_{k,l}$ are used for building a bipartite graph, where each entry corresponds to an i/o of the real (left) and the ideal operation (right). For each pair v_i, v'_k a separate (inner) bipartite graph is built where the left graph elements consist of the sets $C_{i,j}$ and the right ones of the sets $C'_{k,l}$. The scores between sets $C_{i,j}$ and $C'_{k,l}$ are calculated as the maximum score observed after conducting an one by one comparison of their element with the Wordnet similarity metrics [15].

Fig. 6 shows an example of how the score is calculated. If any of the words in a node consists of multiple tokens, then a new bipartite graph (second level inner graph) is formulated in order to compute the score between words independently. The comparison between two sets, as in Fig. 6, is performed by applying the Wordnet Similarity metrics, forming the bipartite graph that will lead to the optimum matching through the Hungarian algorithm.

The parameters v_i and v'_k are also compared in terms of their data type, forming the matrix $D_{i,k}$. For this purpose the *datatypeSimilarity* function is used, which returns 1.0 if the data type matches exactly, 0.5 if both parameters belong to the same complex data type and 0.0 in any other case. Parameters v_i and v'_k are finally compared in terms of their structure, i.e. if their parent nodes in the operations tree match with each other. The hierarchical tree parents names are stored in two vectors, where they are compared element-by-element by the same algorithm presented, ignoring the steps described in lines 31 and 32 of the algorithm shown in Figure 16, and returning the matrix $L_{i,k}$. This is implemented through the *structureSimilarity* function, which returns as an output the structure similarity matrix $R_{i,k}$. The final score matrix $M_{i,k}$ is calculated by the following Eq. (7).

$$M_{i,k} = 0.8L_{i,k} + 0.1D_{i,k} + 0.1R_{i,k} \qquad (7)$$

```
Input-Output-Similarity($ws$, $O$) : $M$

01. $ws$: a web service; $WS$: ontology
02. $M$: similarity matrix
03. $V_{io}$, $V'_{io}$ $\leftarrow$ null
04. $V_{io}$ $\leftarrow$ parseWSDL(ws)
05. For each $v_i \in V_{io}$
06.     $T_i$ $\leftarrow$ split($vi$)
07.     For each $t_j \in T_i$
08.             $S_{i,j}$ $\leftarrow$ synsets($tj$)
09.             $H_{i,j}$ $\leftarrow$ hypernyms($tj$)
10.             $C_{i,j} \leftarrow S_{i,j} \cup H_{i,j}$
11.     End for
12. End for
13. $V'_{io}$ $\leftarrow$ parseOWL($O$)
14. For each $v'_k \in V'_{io}$
15.     $T'_k$ $\leftarrow$ split($v'_k$)
16.     For each $t'_l \in T'_k$
17.             $S'_{k,l}$ $\leftarrow$ synsets($t'l$)
18.             $H'_{k,l}$ $\leftarrow$ hypernyms($t'l$)
19.             $C'_{k,l} \leftarrow S'_{k,l} \cup H'_{k,l}$
20.     End for
21. End for
22. For each $v_i \in V_{io}$
23.     For each $v'_k \in V'_{io}$
24.             $G_{i,k}$$\leftarrow$bipartite($C_{i,1}$, $C_{i,2}$,...|$C'_{k,1}$, $C'_{k,2}$,...)
25.             For each $c_{i,j} \in C_i$
26.                     For each $c'_{k,l} \in C'_k$
27.                         $s_{j,l}$$\leftarrow$WNSimilarity($c_{i,j}$, $c'_{k,l}$)
28.                     End for
29.             End for
30.             $L_{i,k}$ $\leftarrow$ max($s_{j,l}$)
31.             $D_{i,k}$$\leftarrow$DatatypeSimilarity($v_{i,}$, $v'_k$)
32.             $R_{i,k}$$\leftarrow$StructureSimilarity($v_{i,}$, $v'_k$)
33.             $M_{i,k}$$\leftarrow$ 0.8*$L_{i,k}$+0.1*$D_{i,k}$+0.1*$R_{i,k}$
34.     End for
35. End for
36. Return $M$
```

Figure 16: Algorithm for inputs – outputs classification.

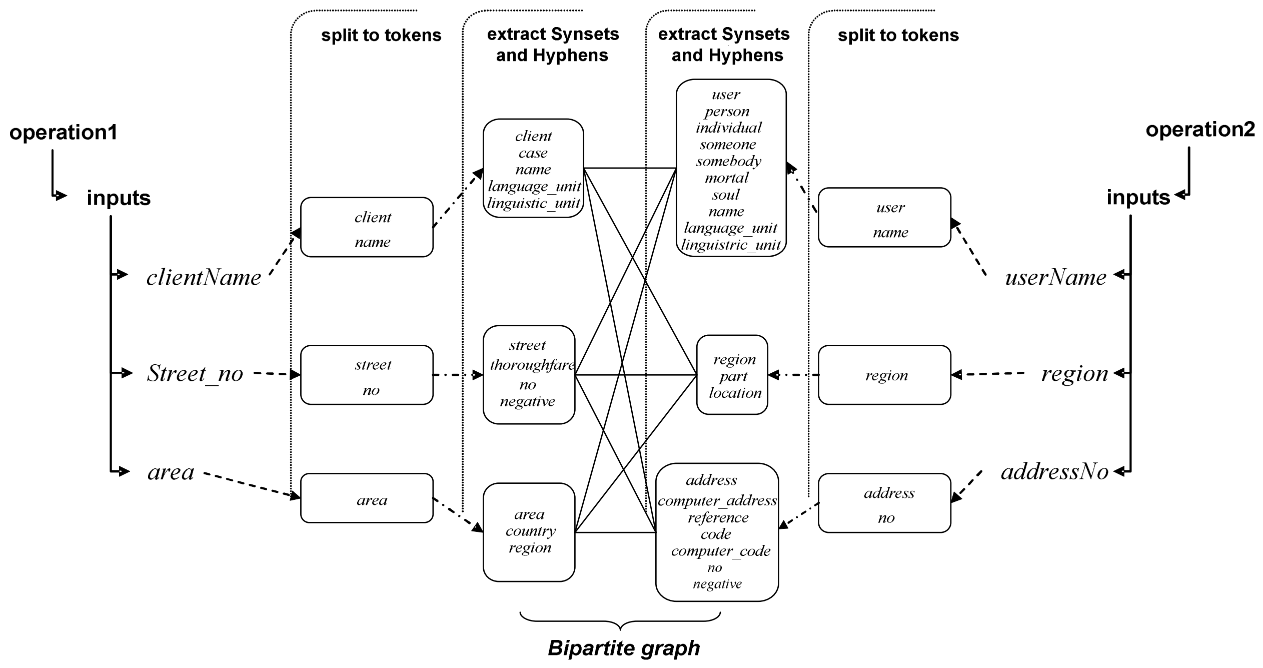

split to tokens
extract Synsets and Hyphens
extract Synsets and Hyphens
split to tokens
operation1
inputs
clientName
Street_no
area
client name
street no
area
client case name language_unit linguistic_unit
street thoroughfare no negative
area country region
user person individual someone somebody mortal soul name language_unit linguistric_unit
region part location
address computer_address reference code computer_code no negative
user name
region
address no
operation2
inputs
userName
region
addressNo
Bipartite graph

Figure 17: The pre-processing steps in order to form the bipartite graph for finding the best matches.

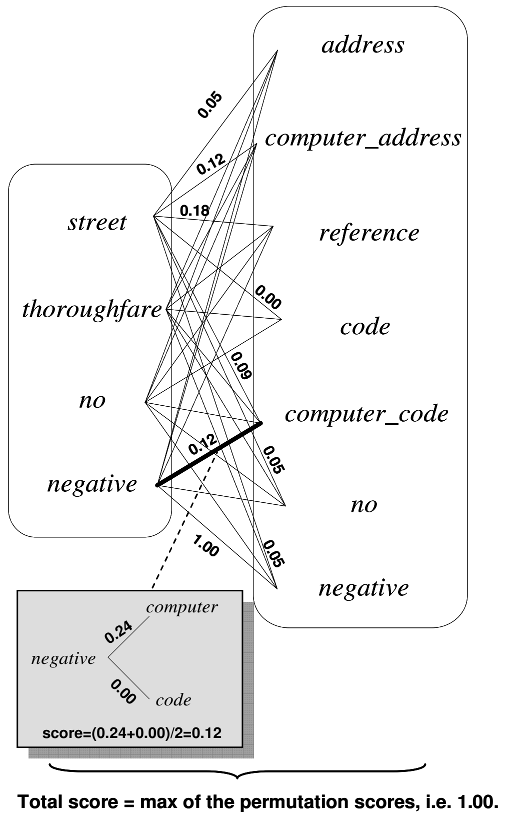


Figure 18: An example of how the score between two nodes in the bipartite graph is computed

It should be noted that the weights in Eq. (7) (as also shown in line 33 of the algorithm) were selected empirically, to be 0.8 for parameter names, 0.1 for their data type and 0.1 for their structure. This selection of weights gives more significance to the semantic notion they represent, but can be adjusted accordingly in order to express specific needs. Finally, the best scores from the final mapping between i/o of the real and the ideal operations are computed by solving the assignment problem by the Hungarian algorithm [16].

4.4 Preliminary Experimental Evaluation

In order to evaluate the accuracy of the proposed WS categorization mechanism, we have conducted a draft evaluation procedure for testing the cases of domain categorisation accuracy. The experimental setup and the preliminary results are summarized in what follows.

The original dataset that was used for the first evaluation procedure contains 249 SOAP WS that were collected from various open WS repositories, including www.webservicelist.com, www.xmethods.com, and www.programmableweb.com. All WS in our collection were manually annotated. Domain-level annotation includes the categorization of the WS into the following six application domains (the number of WSDL files used for each domain is shown in brackets):

- *Business and Money (96),*
- *Geographic (55),*
- *Communication (58),*
- *Tourism and Leisure (13),*
- *Transport (10),*
- *Weather (17).*

Moreover, different configurations of the training and test datasets are used within the evaluation procedure, in order to examine the impact of the data on the achieved accuracy. In particular, we use two different ways for selecting the training set: (a) leave-one-out cross validation, i.e., a single instance from the original dataset is used as test data, and the remaining ones as training data and (b) random selection of training and test data

In our experiment, we apply the leave-one-out cross validation technique to the original dataset for formulating the training set. Our goal is to evaluate the accuracy of the domain categorization mechanism, using the following metrics: Accuracy, Precision, Recall and F-measure. In this experiment we perform comparison of MWSAF (developed in [2]) to the developed mechanism (eCOMPASS Classifier), as well as to a number of its variations. We use the different variations of our classifier in order to assess the impact of each one of its specific components to the overall achieved accuracy. All benchmarked approaches are described as follows:

(a) eCOMPASS: The WS classification method that we developed for domain and operations categorisation described in the previous subsections,

(b) Bayes: a plain Naïve Bayes classifier implementation [10],

(c) LMT: a plain LMT classifier implementation [8],

(d) LMT-B, i.e. the application of the Bayesian theorem for feature space dimension reduction (see subsection 4.1.2) to an LMT classifier.

(e) MWSAF: The MWSAF tool [2].

Figure 19 shows the results of this comparison. We repeat the same experiment, by changing the test dataset. In particular, we split the original dataset into 10 different pairs of training and test datasets by randomly selecting 67% of the original WS as the training set and the rest 33% as the test set. Figure 20 presents the average values of accuracy, precision, recall and f-measure metrics all benchmarked classifiers for the 10 distinct datasets.

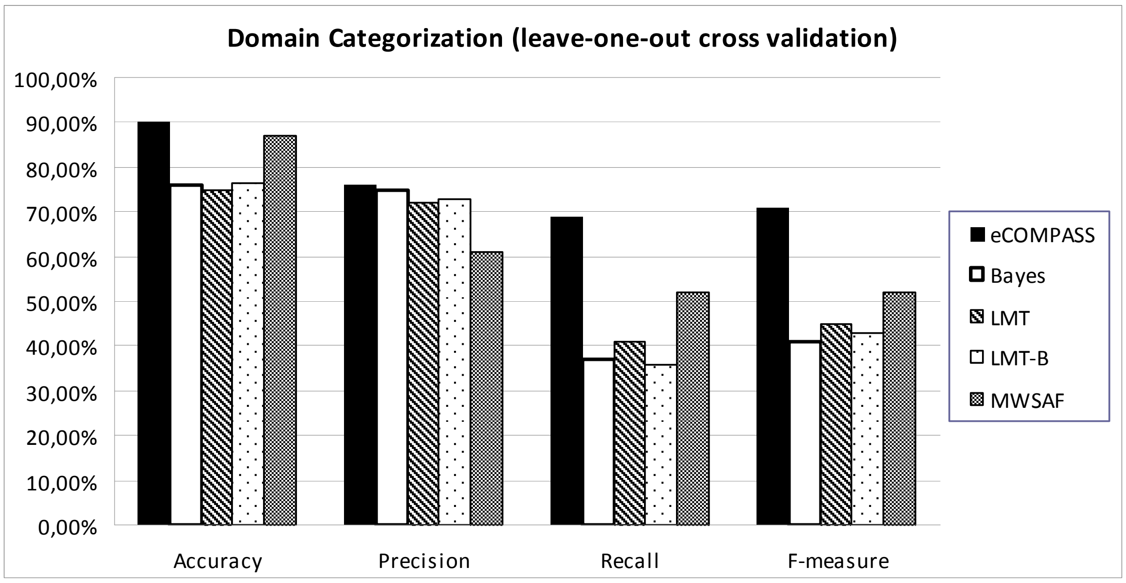


Figure 19: Domain classification accuracy, precision, recall and f-measure comparison for eCOMPASS, Bayes, LMT, LMT-B and MWSAF approaches with leave-one-out cross validation.

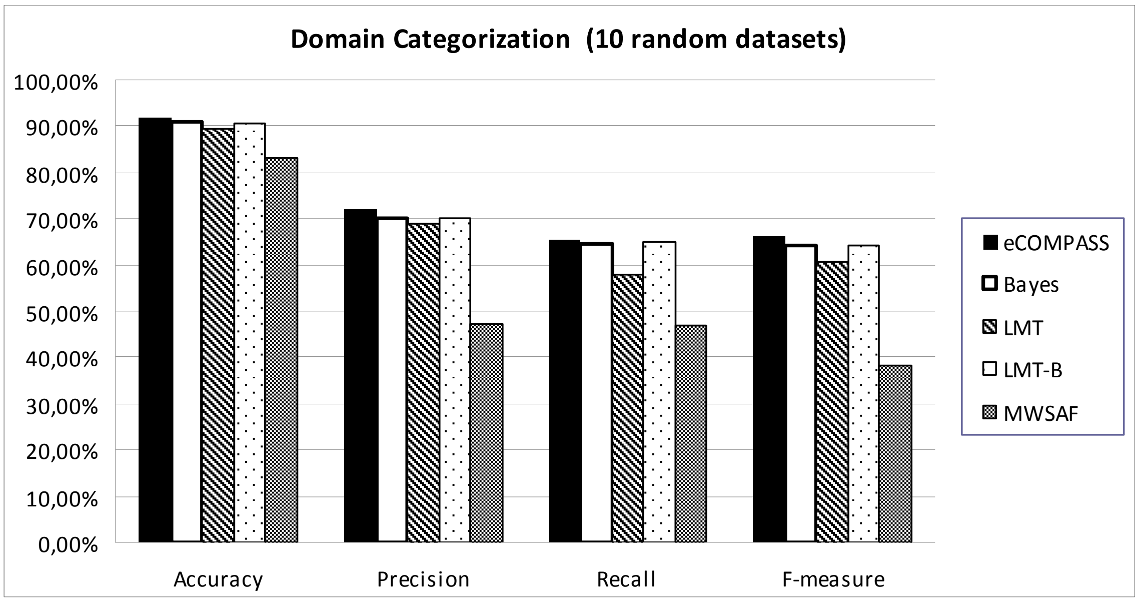


Figure 20: Domain classification accuracy, precision, recall and f-measure comparison for eCOMPASS, Bayes, LMT, LMT-B and MWSAF approaches with ten random training datasets.

As it is shown in the previous figures, our categorisation method outperforms MWSAF in all cases. Among the remaining methods, it also outperforms all of them as it shows the best value among all compared approaches for all used metrics. In this experiment, we have noticed that the maximum value of the achieved accuracy is 97.19% for our approach and 96.79% for MWSAF, with mean values 90.29% and 85.21% respectively.

After benchmarking the eCOMPASS WS categorisation mechanism against its variations, as well as the MWSAF tool, we proceeded to a statistics test in order to verify whether the achieved improvement is statistically significant with respect to the accuracy achieved by the benchmarked methods. To this end, we conducted a Wilcoxon signed-ranks test [17], for each benchmarked mechanism. The purpose of this test is to check whether the null hypothesis (i.e. the eCOMPASS and each one of its rivals perform the same) can be rejected. We tested the null hypothesis at a significance level of $a = 0.01$. The Wilcoxon signed-ranks test is a non-parametric test, which ranks the differences of the performance of each pair of classifiers for each dataset, ignoring signs, i.e., it performs comparison of the rankings of both positive and negative differences.

After performing the Wilcoxon signed-ranks statistical test, it turns out that the most statistically significant difference occurs between the performance of eCOMPASS and MWSAF WS categorization mechanisms. Hence for the eCOMPASS-MWSAF comparison pair we can safely conclude that the null-hypothesis is rejected, which means that the outperformance of eCOMPASS over MWSAF WS categorization mechanism is statistically significant, at a significance level of 0.99. Regarding the other benchmarked approaches, the performance of eCOMPASS has no significant difference to the one of LMT and LMT-B, whereas it is more statistically significant with respect to LMT, at a significance level of 0.97.

In a next set of evaluation setup a more detailed comparison is expected to be conducted also with respect to the additional characteristics and functionalities of the developed mechanism, i.e. operations and i/o parameters categorisation.

4.5 Future Extensions

Future research will focus on providing mechanisms which will be more generic and independent from training data. Furthermore the proposed WS categorization framework will be extended in order to support automatic categorization of REST services to the best possible extent.

5 Web Service Registration Tool: Examples of Use

In this section two examples of WS registration to the WRT are presented. The first example demonstrates how a SP may register a new SOAP WS in the domain of Weather.

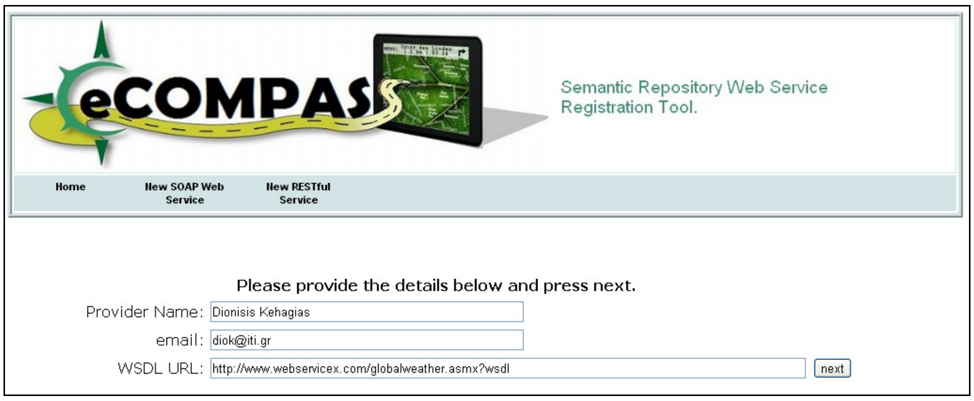
5.1 An Example of a SOAP Web Service about Weather Information

In this example, let us assume that the SOAP WS described by

http://www.webservicex.com/globalweather.asmx?wsdl

will be registered in the WRT.

The aforementioned WS relies on the SOAP protocol and provides operations for weather information. In order for the WS registration to take place, the SP should enter some personal information, such as the provider's name and an email, as well as the URL that points to the WSDL file, which describes the WS in question, as shown in the figure below.

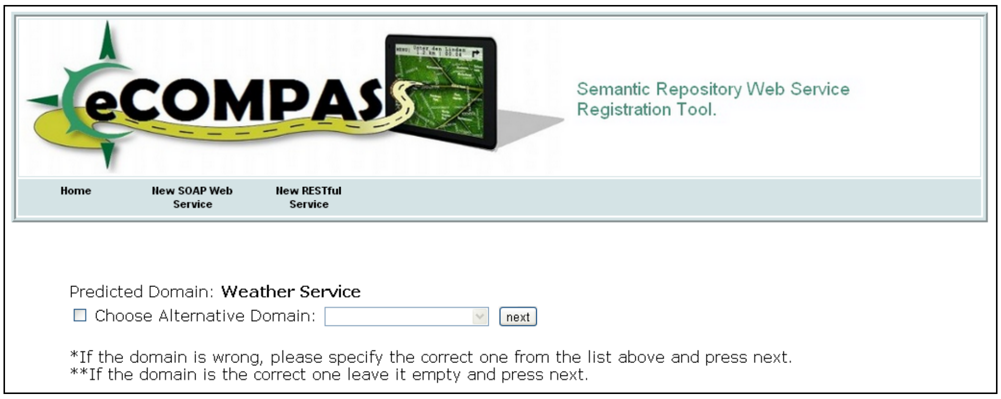


Please provide the details below and press next.

Provider Name:	Dionisis Kehagias
email:	diok@iti.gr
WSDL URL:	http://www.webservicex.com/globalweather.asmx?wsdl

Figure 21: Provider details and WSDL url form

The WRT parses the online WSDL file and extracts all the required information for classifying the WS to the available domains, as shown in the figure below.



Predicted Domain: Weather Service

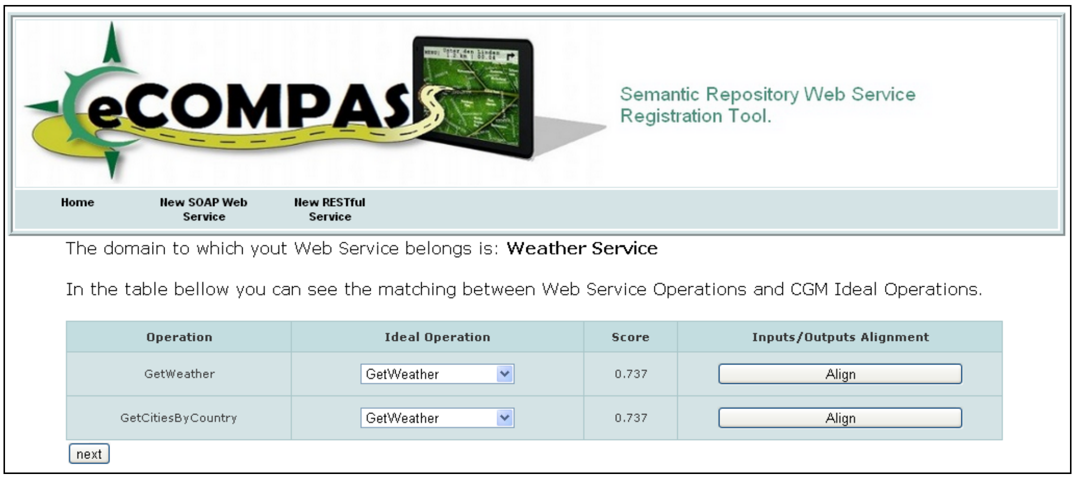
☐ Choose Alternative Domain:

*If the domain is wrong, please specify the correct one from the list above and press next.
**If the domain is the correct one leave it empty and press next.

Figure 22: Domain classification page

The WS is classified correctly in the "Weather Service" domain, thus the provider may continue by pressing the "Next" button.

The WS operations are displayed along with their matching prediction to domain's ideal operation and a score representing the matching score between each WS operation and the corresponding ideal operation (**Figure 23**).



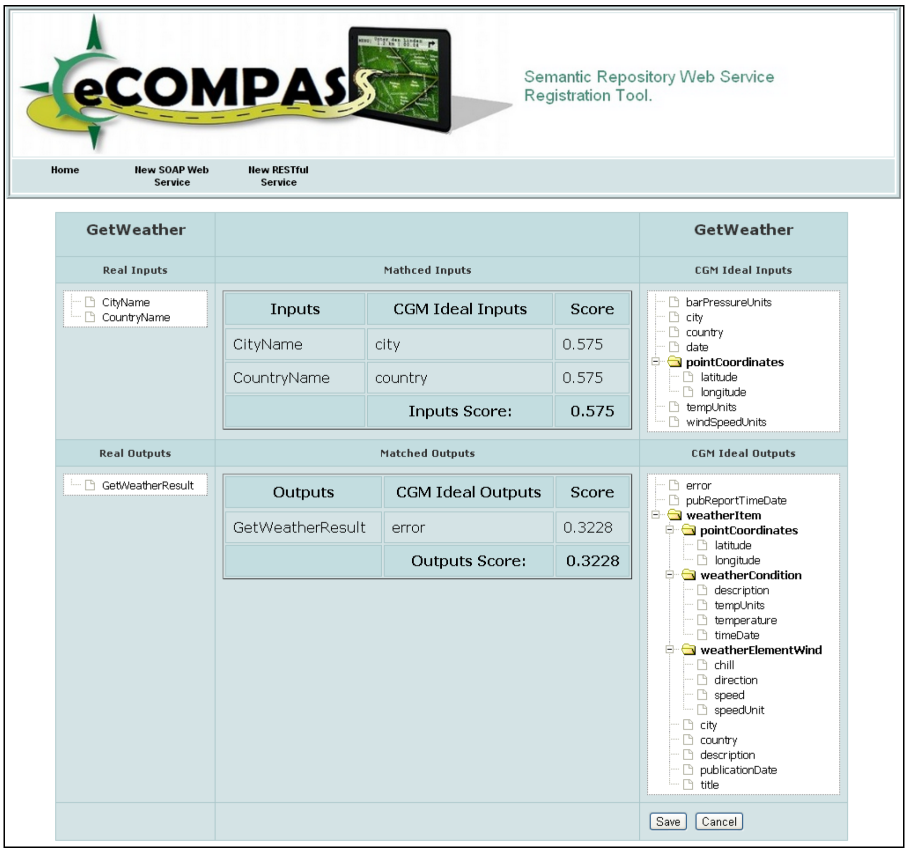
The domain to which yout Web Service belongs is: Weather Service

In the table bellow you can see the matching between Web Service Operations and CGM Ideal Operations.

Operation	Ideal Operation	Score	Inputs/Outputs Alignment
GetWeather	GetWeather	0.737	Align
GetCitiesByCountry	GetWeather	0.737	Align

Figure 23: Operation classification page

For aligning an operation, e.g. the "GetWeather" WS operation with the ideal operation "GetWeather", the provider should select "Align", and the page shown in **Figure 24** will appear.



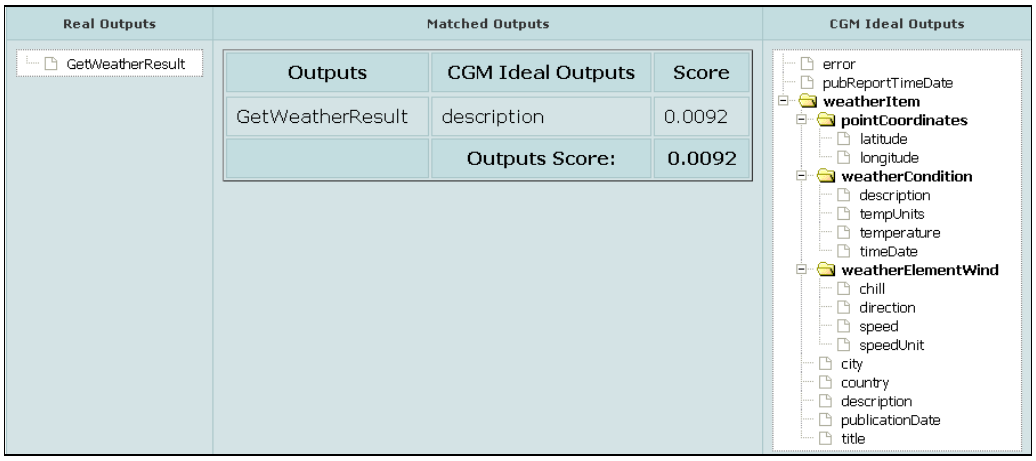
Inputs	CGM Ideal Inputs	Score
CityName	city	0.575
CountryName	country	0.575
	Inputs Score:	0.575

Outputs	CGM Ideal Outputs	Score
GetWeatherResult	error	0.3228
	Outputs Score:	0.3228

Figure 24: Input/output matching page

This page shows the inputs and outputs of the WS operation (on the left trees) and the inputs and outputs of the ideal operation (on the right trees). The matches between i/o are displayed in the two tables along with a matching score. It is obvious that the inputs of the WS are aligned correctly, while the matching mechanism has failed for the output of the WS operation. This can be adjusted by dragging the output from the left tree to an output (e.g.

"description") to the right tree. Then the new matching is displayed in the outputs table with a new score.



Outputs	CGM Ideal Outputs	Score
GetWeatherResult	description	0.0092
	Outputs Score:	0.0092

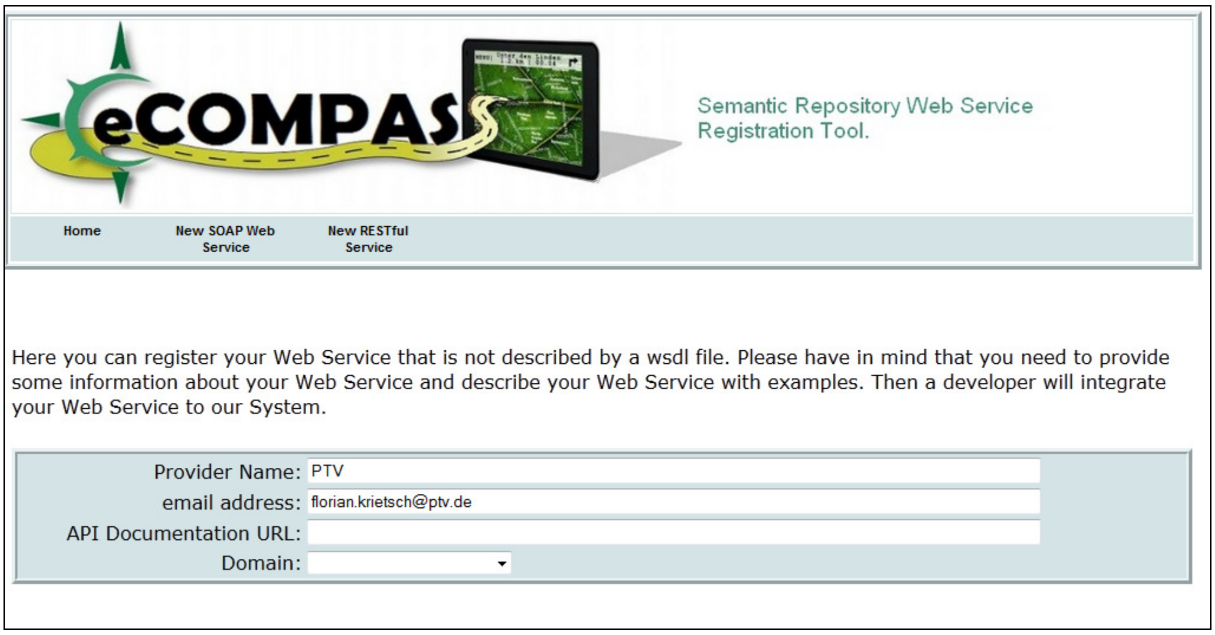
Figure 25: Manual alignment of an output

By selecting "Save", the WS operation registration is saved and the provider may proceed with the registration of the rest of his WS operations.

5.2 Registration of a Fleet Management Service

The second example demonstrates how a WS of a fleet management service can be registered. In this example a REST web service is applied to the gateway by using the Semantic Repository Web Tool.

The applied web-service visualizes a map. In order to register the web-service via the service registration, the service provider has to enter personal information, such as the provider name and an email, as well as the URL that points to the service.

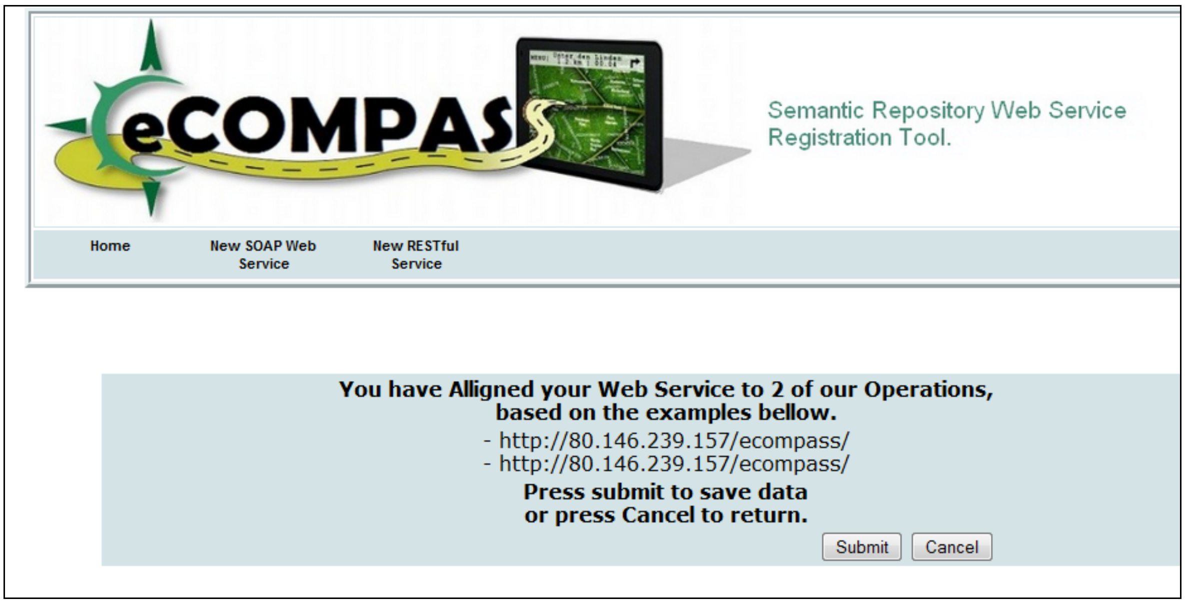


Here you can register your Web Service that is not described by a wsdl file. Please have in mind that you need to provide some information about your Web Service and describe your Web Service with examples. Then a developer will integrate your Web Service to our System.

Provider Name:	PTV
email address:	florian.krietsch@ptv.de
API Documentation URL:	
Domain:	

Figure 26: Provider details and URL of logistics service example

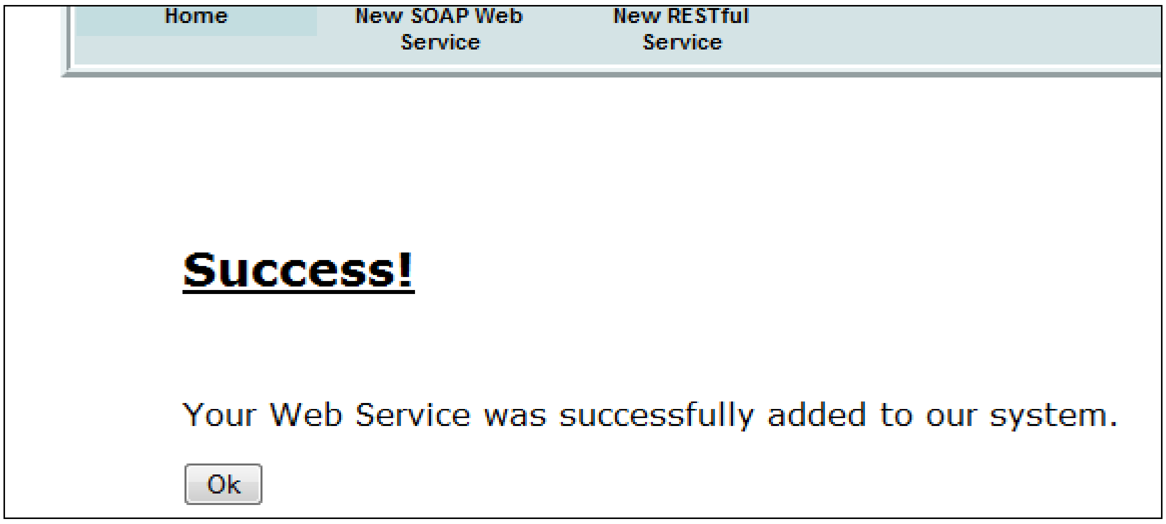
After filling out all mandatory fields the WRT can assign the service to its database.



You have Alligned your Web Service to 2 of our Operations, based on the examples bellow.
- http://80.146.239.157/ecompass/
- http://80.146.239.157/ecompass/
Press submit to save data or press Cancel to return.

Figure 27: Aligned logistics service web-service

The WRT then provides the user with a success notification dialog, as shown in **Figure 28**.



Success!

Your Web Service was successfully added to our system.

Figure 28: Successful registration of the logistics web-service

6 Summary and Conclusions

This deliverable is a report that accompanies the prototype implementation of the SCR and WRT, which have been developed in WP4 as part of the CGM and are publicly available online (http://160.40.50.57/ecompass). SCR and WRT play a complementary role, as the former provides the underpinning technical infrastructure for the functionalities that the latter provides to take place efficiently. The WRT is equipped with a web interface that allows any interested SPs who are willing to make their WSs visible in the context of eCOMPASS (or any other similar framework that relies on the CGM). In this way, WRT is available anywhere, anytime and theoretically any SP may register his/her services, no matter where he or she is established.

The tool allows the registration of both SOAP and REST-compliant services. The reason for this is that the majority of the available services today are based on the REST protocol. However, SOAP is still supported as the de facto industrial standard and for this reason we decided to include support for these also. Moreover, all services that adhere to the SOAP protocol can take advantage of the automatic WS categorization mechanism, whose purpose is to predict the domain, to which a WS belongs. On top of this, the automatic categorization mechanism is capable of recommending those operations, as well as the corresponding i/o parameters of the CGM Data Communication protocol, that provide the best match to the WS under registration. This is only possible for SOAP WS, because they are accompanied by WSDL files, and the WS categorization mechanism is based on information retrieval techniques that exploit information from the WSDL files. A WSDL file that accompanies any SOAP WS contains the implementation details of the WS, as well as additional documentation and a description of its internal elements, i.e. operations and i/o parameters. We implemented this automatic categorization mechanism in order to provide more user friendly functionality to the WRT tool, by adding a degree of automation, but also in order to facilitate the job of CGM developers who should implement specific WS wrappers capable of invoking the real WS for each WS that is registered into the SCR. In the case of REST services this task still needs to be performed in a manual way, thus increasing the required effort on the developers' side.

In this deliverable we have also described the SCM, which basically supports the functionality of the WRT. SCM does this in a two fold way: (a) by providing a web application server that implements the "business logic" of the automatic categorization mechanism, and (b) by hosting a database which stores all information entered by the SP at the registration process, in addition to other information that is updated externally (i.e. not by the SPs themselves), either in an automatic or manual way. For instance in order to provide updated values of the QoS parameter for a particular service, a special deamon (i.e. background process) runs that pokes the WS and based on its response rate, it updates the QoS value at real time. In this way, SCM can be considered as the back-end of the system, whereas can be seen as the user-oriented part of it, as it provides the main user interface.

This report-part of the deliverable also serves as a user manual for the SP who wants to use it for registering a new WS in eCOMPASS. Through the provided tool, the eCOMPASS framework in general and CGM in particular allow for external SPs to enter new WS in the transport data-related domains that are supported in this project. Based on this kind of interaction between SPs and the eCOMPASS framework in which users are also involved, it becomes clear that a set of new business models could be enabled through WRT and SCM. In particular, if seen from a business-oriented perspective, WRT and SCM provide the necessary infrastructure for enabling a business model according to which any SP can receive a fee for making their services available to a wide range of users who use the eCOMPASS services in a transparent way. In this business scenario the users pay for using the application and the money goes to the SP. Alternatively, the eCOMPASS framework could be seen as an intermediate procurement party, which demands a percentage of the user transactions based on the traffic they produce. Similar business scenarios could be also realized in this context.

Future work with respect to the WRT and SCM would involve all necessary modifications and/or additions in order to adopt the security mechanism that is developed in Task 4.4. In this way, the system will make sure that the right content will be provided to the right requesting party, based on an authentication procedure.

References

[1] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", Fourteenth International Joint Conference on Artificial Intelligence, pp. 1137–1143, 1995.

[2] A. A. Patil, A. Oundhakar, A. P. Sheth, and Verma, K., "METEOR-S Web service annotation framework", 13th international conference on WWW, ACM Press, 2004.

[3] M. F. Porter "An algorithm for suffix stripping". Program, vol. 14, no. 3, pp. 130-137, 1980.

[4] G. Salton, and C. Buckley, "Term-weighting approaches in automatic text retrieval", Information Processing and Management, pp. 513-523, 1988.

[5] G. Salton, and C. Buckley, "Term-weighting approaches in automatic text retrieval", Information Processing and Management, pp. 513-523, 1988.

[6] D. Isa, L. Lee, V. Kallimani, and R. RajKumar, "Text Document Pre-processing with the Bayes Formula for Classification Using the Support Vector Machine", IEEE Knowledge and Data Engineering, vol. 20, no. 9, pp. 1264 – 1272, 2008.

[7] F. Sebastiani, "Machine learning in automated text categorization", ACM Computing Surveys, vol. 34, no. 1, pp. 1-47, 2002.

[8] N. Landwehr, and M. E. Hall, "Logistic Model Trees", Machine Learning, vol. 59, no. 1, pp. 161-205, 2005.

[9] A. McCallum, and K. Nigam, K, "A Comparison of Event Models for Naive Bayes Text Classification". AAAI-98 Workshop on Learning for Text Categorization, 1998.

[10] S. Eyheramendy, D. D. Lewis, and D. Madigan, "On the naive Bayes model for text categorization", Artificial Intelligence & Statistics, pp. 332–339, 2003.

[11] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers", Machine Learning, vol. 29, pp. 131-163, 1997.

[12] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: aStatistical View of Boosting", The Annals of Statistic, vol. 38, no. 2, pp. 337–374, 2006.

[13] L. H. Breiman, J. Friedman, A. Olshen, and C. J. Stone, Classification and Regression Trees. Belmont, California: Wadsworth International Group, 1984.

[14] H. Witten, and E. Frank, Data mining: Practical Machine Learning Techniques and Tools 2nd ed. Morgan Kaufmann, 2005.

[15] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet::Similarity - Measuring the Relatedness of Concepts", Fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-04), pp. 38-41, 2004.

[16] H. K. Kuhn, "The Hungarian Method for the assignment problem", Naval Research Logistics Quarterly, vol. 2, pp. 83–97, 1955.

[17] F. Wilcoxon, "Individual comparisons by ranking methods", Biometrics Bulletin, vol. 1, no. 6, pp. 80-83, 1945.