DISASTER

Data Interoperability Solution At Stakeholders Emergency Reaction

285069

D5.22 Technical approach report – V2

Lead Author: Rubén Casado (TREE)
With contributions from: CTIC
Reviewer: Guillermo González-Moriyón (CTIC)

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	
Actual delivery date:	
Version:	1.2
Total number of pages:	31
Keywords:	Web services, standards, architecture implementation, data management

Abstract

This deliverable specifies the technologies, formats and protocols used to implement the DISASTER architecture as defined in previous deliverables. The Web Service architecture has been chosen as technical paradigm due to its loosely coupled, standard-based approach for building SOA solutions. This document describes which and how web services standards are employed to fulfil the DISASTER technical requirements. The implementation includes the management of the emergency related data resources.

Executive summary

The Web Service architecture has been chosen as technical paradigm to implement the DISASTER architecture due to its open, scalable and standard-based approach for building SOA solutions. The WS specifications employed in the construction of the architecture form the *Disaster Technical stack*. It is divided in six levels according to the nature of the included WS standards.

The bottom level is *Transport*, which refers to the message format and protocols used to exchange the information. This level includes HTTP, XML and SOAP specifications.

Description level includes the standards to describe both functional and non-functional characteristics of the services. WSDL and WS-Policy standards belong to this level.

Discovery level refers to the standards used to publish and organize the services included in the DISASTER solution. This level includes UDDI and WS-Discovery standards.

Messaging level refers to the mechanism provided to ensure suitable and correct message transmission. WS-Addressing, MTOM and WS-Eventing are part of this set of standards.

Quality of Service (QoS) level focuses on the reliability and security of the interactions. WS-Security and WS-ReliableMessaging are included in this level.

Finally, *Cooperation* level deals with the composition and coordination between multiple service operations when required. This level is composed by WS-BPEL and WS-Coordination standards.

In addition, DISASTER project provides a mechanism to allow interoperation between different EMS during the management of crisis scenarios. Such interoperability can be defined as a data exchange (e.g. geospatial information, map images, files, etc.) between software systems. Thus, a key issue in the DISASTER solution is the management of the data (o resources). The DISASTER architecture implementation takes into account not only the data management and policies (e.g. security and privacy) tasks, but also the most widely data formats and protocols used in the EMS such as WMS, WFS, GML, XML and JSON.

Document Information

IST Project Number	285069	**Acronym**	DISASTER
Full Title	Data Interoperability Solution At Stakeholders Emergency Reaction		
Project URL	http://www.disaster-fp7.eu/		
Document URL			
EU Project Officer	Jana Paskajova		

Deliverable	**Number**	D5.22	**Title**	Technical approach report – V2
Work Package	**Number**	WP5	**Title**	Integrated System Design

Date of Delivery	**Contractual**	M18	**Actual**	M18
Status	version 1.0		final	
Nature	prototype □ report **x** demonstrator □ other □			
Dissemination level	public **x**restricted □			

Authors (Partner)	Rubén Casado (TREE), Guillermo González-Moriyon (CTIC)			
Responsible Author	**Name**	Rubén Casado	**E-mail**	ruben.casado@treelogic.com
	Partner	TREE	**Phone**	

Abstract (for dissemination)	This deliverable specifies the technologies, formats and protocols used to implement the DISASTER solution defined in previous deliverables. The Web Service architecture has been chosen as technical paradigm due to its loosely coupled, standard-based approach for building SOA solutions. This document describes which and how web services standards are employed to fulfil the DISASTER technical requirements. The implementation includes the management of the emergency related data resources.
Keywords	Web services, standards, architecture implementation, data management

Version Log			
Issue Date	**Rev. No.**	**Author**	**Change**
February28th, 2013	0.1	Rubén Casado	Initial TOC.
June, 1st, 2013	0.2	Rubén Casado	V1 extension
June, 27th , 2013	0.3	Guillermo González-Moriyón	CTIC contribution
July, 1st , 2013	1.0	Rubén Casado	Version to review
July, 7th 2013	1.1	Guillermo González-Moriyón	Review
July, 23th 2013	1.2	Rubén Casado	Reviewed version

Table of Contents

List of Figures

Abbreviations

ACID: Atomicity, Consistency, Isolation, Durability

API: Application Programming Interface

DISMA: Disaster Management Software (German EMS)

EMS: Emergency Management System

ICT: Information and Communication Technologies

LCMS: *Landelijk Crisis Management Systeem* (National Crisis Management System)

MIME: Multipurpose Internet Mail Extensions

GeoJSON: JSON Geometry and Feature description

HTTP: Hypertext Transfer Protocol

JSON: Javascript Object Notation

OWL: Web Ontology Language

RDB: Relational Database

RDF: Resource Description Framework

SMTP: Simple Mail Transport Protocol

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

SKOS: Simple Knowledge Organization Sytem

URI: Uniform Resource Identifier

W3C: World Wide Web Consortium

WS: Web Service

WSDL: Web Service Definition Language

XML: Extensible Markup Language

Definitions

Base64: an encoding scheme to represent binary data in an ASCII string format.

Emergency Management System: A system that enables communication and collaboration amongst first responders and gives support to decision makers providing all the available information about an incident. An EMS frequently supports emergency stakeholders at operational, strategic and tactical level.

Encryption: the process of obscuring information to make it unreadable without special knowledge.

Endpoint: the entry point (operation) of a web service.

Mediation: Process that transforms data expressed in a given data scheme into equivalent data in other data schema.

Protocol: digital message formats and rules for exchanging those messages in or between computing systems and in telecommunications.

Policy: Non-functional requirements of a web service.

Service Oriented Architecture: a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functionalities that are built as software components that can be reused for different purposes.

Web Service: A software system designed to support interoperable machine-to-machine interaction over a network.

1 Introduction

The DISASTER project is oriented to achieve the data interchange between the Emergency Management Systems around Europe. Nowadays, the lack of a standard for these communications complicates this task. The architecture implementation (see [1] for further details about the architecture design) described in this document tries to improve these communications and data interchanges without modifying the current Emergency Management Systems. To achieve this objective, the implementation relies on well-known standards for data management and the software communications.

This second version of the deliverable is closely related to other deliverables of the project. First of all it is related to D2.52 [1], which designs the architecture from a theoretical point of view, defining components and describing their role in DISASTER. The deliverable D4.10 [2] studied the state of the art regarding mediation techniques which are the basis of the DISASTER mediation components to solve the detected communication problems. Finally this deliverable is related to D5.10 [3] where the state of the Emergency Management Systems (EMS) around Europe is compiled. It detects formats, transformations and other needs to accomplish DISASTER objectives.

There are two main sections in this deliverable: data management and architecture implementation. Data management refers to the formats, technologies and services required to acquire and produce the information expected by the emergency management systems. This task focuses on geospatial information using standard protocols such as WMS and WFS or formats such as GML. The selection of these protocols and formats relies on the conclusion achieved in D5.10 [3], where existing EMS were studied. We must highlight the LCMS[1], the Dutch national emergencies system, and the DISMA[2] German software that have been used to validate the current proof of concept [4]. Data management also includes general information, where protocols like the SPARQL endpoint or formats such as RDF, XML or JSON are used to transfer and store the information. In addition, DISASTER solution is service-oriented and the use of standards will facilitate its adoption. Working on this direction the architecture will be implemented using standards for transport, service description, and message exchange, quality of service and resource management. The standards referenced are widely accepted and will simplify the scalability of the project.

The document is structured as follows. Section 2 addresses the data management, where it is explained the protocols and formats included in the current version of the DISASTER architecture. Section 3 presents the web services-related technologies used to implement the architecture. The description of the standards specification will clarify their selection and use to improve the DISASTER solution and its extensibility. Conclusions of the work are presented in Section 4.

[1] *Landelijk Crisis Management Systeem*, http://lcms1.nl

[2] Disaster Managment, http://www.tuv.com/en/corporate/business_customers/plants_machinery_1/industrial_plants_2/disma_disaster_management_cw/disma_disaster_management.html

2 Data management

Based on the conclusions achieved in D5.10 [3], and due to the fact that there is no standard specifications to exchange information among EMS along Europe, DISASTER will manage different types of protocols and file/data formats. Some protocols are related to geospatial information (e.g. WMS[5] and WFS[6]) due to the importance of this kind of information in crisis management, and others to the data management (e.g. SPARQL endpoints [7] as a standard data query interface). Regarding data formats, DISASTER will have to deal with structured data, such as XML [8], JSON [9] or GeoJSON [10] in order to get or return the valuable information. The set of protocols and standards also must include Relational Databases (RDB) because it is the common storage system in legacy EMS. Furthermore, DISASTER will use RDF [11] to facilitate data integration and mediation.

One of the main objectives of DISASTER is to build a scalable solution capable of dealing with new formats and protocols so an extensible solution is required. In this document are described some of these protocols and formats that are aimed on this first stage of the project. Note that due to the extensibility of the architecture design, new formats and protocols will be easily included in future versions.

2.1 Protocols

2.1.1 WMS

Web Map Service (WMS) [5] is a protocol standardised by the Open Geospatial Consortium[3] in 1999. This service provides georeferenced images that can be overlaid on a map providing complementary information or can be a map itself. When the client asks for an image the map server returns a bitmap image (e.g. PNG, JPG) but it also can serve points, lines and polygons as vector graphics in SVG. Layers can be transparent allowing image compositions. WMS interface provides up to three different operations to request information:

- *GetCapabilities*: Returns information about the server and the layers available.
- *GetMap*: Returns a map image.
- *GetFeatureInfo*: Optional method only supported for *layers* with the attribute queryable=”1”. A layer is *queryable* if it can provide textual information about its content. Returns information about the feature included in the map picture.

Regarding WMS protocol, DISASTER will be focus on the queryable layers. DISASTER solution will be able to adapt the information extracted from the *GetFeatureInfo* method and adapt such data according to the destination feature style. Note that DISASTER is a non-intrusive solution so it cannot manage the image generation through the *GetMap* method. Further development could deal with image analysis by using pattern recognition techniques maps in order to automatically identify icons in the maps. In that way, DISASTER would be able to mediate the concepts behind such icons and generate a new image according to the destination needs.

[3]Open Geospatial Consortium http://www.opengeospatial.org/

2.1.2 WFS

Web Feature Service (WFS) [6] is a protocol to provide geopositioned information relative to the features which compose a map. The information provided by WFS is the background data used to generate a map picture, i.e. the WFS provides the raw data instead of returning an image map. Information is provided as structured data, mainly using GML [12] format, so the client can edit and/or analyse the geospatial information.

WFS is an standard approved by Open Geospatial Consortium and defines the interface to query, create, delete or modify features. The servers must implement the following operations:

- *GetCapabilities*: Returns the available options and layers provided by the server.
- *DescribeFeatureType*: Returns de XML-Schema to allow the client to parse the data.
- *GetFeature*: Executes the query itself and returns

DISASTER will mediate the communication between the WFS server and the client to adapt the information served according to the localization and language constraints required by the client. Using this approach the client will be able to manage the information in the same way it manages the local information, being a completely transparent process to the end-user. This solution will be achieved using the common ontology-based data model defined inside the DISASTER project [13]. RDF will be the intermediate language.

An example of use of the WFS protocol is the WFS to WMS mediation described in D4.20 [14] and shown in the current Proof of Concept [4]. By combining WFS and WMS, the DISASTER solution will be able to receive the geospatial data provided by WFS and to send a map image to the client using the icons, colours and styles expected instead of the ones used by the provider. This solution simplifies the information processing made in the client application without losing features. At the same time this solution provides protocol mediation for those clients who cannot manage WFS requests.

2.1.3 SPARQL endpoint

A SPARQL endpoint is a conformant SPARQL [7] protocol service which enables users, either humans or machines, to query a knowledge base using SPARQL language and receiving the results in machine-readable formats. The query is enunciated using SPARQL language, which is a standardised language to query RDF graphs. This feature makes the modification of the data model easier than other storage solutions like classical relational database, where the data model is strict. The presentation of the information obtained using the SPARQL endpoint is usually made using a software component that presents the results in a human-readable format, e.g. web page or PDF file.

DISASTER will make use of the SPARQL endpoint protocol to query the DISATER core knowledgebase where the common emergency knowledge will be stored as RDF triples. The results returned by the SPARQL endpoint will allow the system to convert the information between different localizations, e.g. representing a concept with different icons depending on the country which is visualising the information [4].

2.2 Formats

2.2.1 XML

XML (Extensible Mark-up Language) [8] describes the rules to format a document which is both human-readable and machine-readable. The XML standard is published by the W3C[4]. This format is widely used to return results in web services or to exchange data between applications. This format has been extended to create other formats including XHTML, RSS, RDF/XML and KML. From the point of view of DISASTER project the XML-based structure of GML and SOAP services makes this format very relevant, due to the importance of geographical information and the service-oriented architecture implemented.

DISASTER has to read the XML from one side of the communication, transform it to RDF, translate and enrich its content with the knowledge base and, finally, transform it again to the format expected by the other side of the communication. The use of XML will be linked to the communication methods used by the existing EMS, and DISASTER will be ready to deal with systems using XML to exchange information, systems requiring XML as input and systems providing their output as XML. An example of XML usage is the German EMS DISMA which is used in the Border Fire Proof of Concept [15]. This XML schema defines six different features: *Point*, *Polygon*, *Rectangle*, *Circle*, *Line* and *Text*.

2.2.2 RDF

RDF[11] is a model for data interchange on the web standardised by W3C. RDF structure is based on triples; each triple, called statement, is composed of subject, predicate and object. The RDF structure links the subjects with their properties, i.e. the objects, using the predicate. For example, in the statement “Fireman extinguishes the fire” the subject is “Fireman”, the object is “the fire” and the predicate is “extinguishes”. As mentioned before, RDF is oriented to the data interchange on the web, so the identification is made with URIs. This structure represents a directed graph where the links are the predicates and the nodes are the resources, i.e. subjects and objects.

One of the biggest advantages of RDF over XML is the ability of merging data with different schemas and the modification of the schema without the need of changing the data consumers. RDF is the basis for other technologies such as OW[5]L or SKOS [6] that would be useful during the development of the core knowledge base in DISASTER.

DISASTER will use RDF to facilitate the data mediation process storing the knowledge extracted from people involved in emergency management and in order to create a data model to describe the knowledge .But RDF works in conjunction with other technologies mentioned before as OWL and SKOS. Such data will be stored and accessed through an SPARQL endpoint.

2.2.3 JSON

JSON [9] is a text format for the serialization of structured data. It is derived from the object literals of JavaScript, as defined in the *ECMAScript Programming Language Standard*, standardized by ECMA International in the ECMA-262 specification [2] and ISO/IEC 16262 [3]. It is an open standard derived from JavaScript representation of data structures, but it is language-

[4] http://www.w3.org
[5] http://www.w3.org/TR/owl-features/
[6] http://www.w3.org/2004/02/skos/

independent and many languages have support to parse JSON data. It is mainly used to transmit data between the server and web application, but it is becoming an alternative to XML when transmitting data through a network connection and used as response in some web services.

As far as JSON and XML do the same, when talking about data interchange, DISASTER will have to deal with JSON in the same way it will do with XML. The basic flow is getting the data, transform it to a common format, i.e. RDF, and, after applying the corresponding transformations, send the data to the consumer using the most appropriate format.

2.2.4 GeoJSON

GeoJSON [10] is a dialect of the JSON format that, few years ago, the geographic community suggested using. The JSON format allows for a complex structure of fields, arrays, objects, and scalar types, which can be combined into an entire record. The idea of GeoJSON was to standardize the way of encoding spatial data structures. GeoJSON can be used to represent different types of features and each feature (geometry object) can contain additional properties. A complete GeoJSON data structure is always an object (in JSON terms).

According to [1], GeoJSON is a format for encoding a variety of geographic data structures. A GeoJSON object may represent a geometry, a feature, or a collection of features. GeoJSON supports the following geometry types: *Point*, *LineString*, *Polygon*, *MultiPoint*, *MultiLineString*, *MultiPolygon*, and *GeometryCollection*. Features in GeoJSON contain a geometry object and additional properties, and a feature collection represents a list of features.

Since GeoJSON is a standard geographic data format, DISASTER is capable of managing this format as both input and output stream.

2.2.5 RDB

In the last 30 years most information was stored in Relational Databases (RDB).This information is structured in tables related between them using keys that identify every row stored into the table. This kind of data storage reduces the information redundancy and allows transactions management (which avoid inconsistent data), and concurrence mechanisms (which allows simultaneous access to the data). A relational database provides four operations: create, read, update and delete. Such operations are commonly written in SQL. Relational databases are used as data store technology in the majority of software systems.

DISASTER will access the data served by the EMSs or other third-party services which could store the information in legacy RDBs or file formats containing small data storage. DISASTER should be able to read these databases directly and transform their content in order to serve it to other EMS.

2.3 Mediations

After the mediation architecture of DISASTER that has been established in D4.20, a number of efforts have been made under WP4 to initiate its implementation. Mediation distinguishes two top level families: transformation adapters and the mediation component. The former perform transformations at the format and protocol level while the latter carries out classification translation at the data level.

2.3.1 Transformation Adapters

The definition of transformation adapters on D4.20 [14] included the formats and protocols considered inside the scope of DISASTER project. The mission of transformation adapters is that of data lowering and lifting before and after mediation takes place, i.e., it provides methods to obtain data available in legacy formats, translate them at the RDF level and output them in new formats after translation.

At this stage, a subset of them has been already implemented and published as Open Source software projects. For a more detailed specification of each software component, including its license, usage and documentation to obtain it, please refer to D4.30 [16]. A brief summary of each adapter follows:

EMERGEL template CSV (CSV2SKOS): EMERGEL is primarily coded by domain experts using a spread sheet template, as defined in the ontology methodology (Deliverable D3.31 [13]). This transformation defines the formal mappings between this tabular template and an EMERGEL-compliant RDF representation following the SKOS vocabulary. This transformation is implemented as a program written in Tabels language, a Domain-Specific Language (DSL) to produce specific translations from spread sheets to RDF. Tabels[7] is a general purpose transformation tool that converts tabular data into RDF and it is available as an online service. Each transformation program leverages the powerful Tabels DSL to express the mappings between the cells of the input CSV template and the triples at the RDF output. Tabels executes the transformation program to produce EMERGEL vertical modules and mappings formulated in SKOS.

ESRI Shapefile (SHP2RDF): ESRI Shapefile format is the standard de factoto represent and exchange geographic information. A complete study of the transformation from this format to RDF has already been introduced in D4.20 [14]. The transformation involved several steps and some of them were implemented using existing, free of use software such as Tabels. However, certain steps were not available and thus have been implemented and published as part of DISASTER WP4 results. This is the case of the following 3 transformations.

ESRI Shapefile (SHP2KML): The transformation of ESRI Shapefile binary format to represent geometries into OGC KML, a geospatial standard supported by earth browsers and web maps is implemented as a Java library. This approach enables existing EMS or other emergency-related software with this transformation by means of using its defined API. This library can convert all the geometries into a single KML or generate one file per feature. It depends on GeoTools Java library[8] to perform two operations: the coordinate reference system (CRS) conversion and the geometry transformation.

OGC SLD (SLD2SVG): Style that is attached to geographic features in Shapefiles can be represented in multiple formats. The OGC standard for representing style is the Styled Layer Descriptor format (SLD), an XML-based markup language to specify styling on geospatial data. Being a more powerful language than other web-oriented formats such as GoogleMaps JSON or CartoCSS, the basic styling information attached to polygons or polylines is straight forward. However, the transformation of style information attached to markers, i.e., points, it is troublesome given the basic support for raster-based only icons on most web

[7] http://idi.fundacionctic.org/tabels/
[8] http://www.geotools.org/

map platforms. The SLD2SVG tries to bridge this shortcoming by interpreting basic marker style information in SLD and converting it to W3C SVG standard, which is supported for example in the Google Maps platform. It depends on GeoTools Java library to load the SLD file into an Abstract Syntax Tree (AST) that is traversed in order to translate it to SVG.

OGC SLD (SLD2JSON): This transformation is able to convert basic style information for polygons, polylines and markers described in SLD into Google Maps JSON. The output can be directly visualized in Google Maps. As explained in the previous transformer, while the translation of style for both polygons and polylines is straightforward, marker style needs to bypass limitations of current web maps platforms by leveraging the previous SLD2SVG transformation. It depends on GeoTools Java library to load the SLD file into an Abstract Syntax Tree (AST) that is traversed in order to translate it to JSON.

2.3.1.1 Technical Adapters

The above adapters are focused on semantic adaptations. But technical adapters are also needed to mediate between different data formats even if they are in the same semantic layer. These technical adapters include WMS and WFS adaption and EMS proprietary formats like DISMA XML. Description of technical adapters can be consulted in deliverable D2.52 [1].

2.3.2 Mediation Component

The mediation component mission is to translate RDF data expressed from a source classification to a target classification. While the state of the art on mediation techniques has been studied in D4.10 [2], the formal definition of this mediation component according to DISASTER requirements is described on D4.20 [14]. Its implementation is defined in 3 steps: (1) mapping definition, (2) mapping translation, and (3) mapping execution.

Mapping definition is a manual process carried out by consortium experts. The definition of vertical modules is under way with the first version of the work documented on D3.31 [13]. Along with the definition of vertical modules, experts pay special attention to identify mappings that can match relations between two concepts belonging to those vertical modules. At the moment of writing, the following partial mappings have been defined between:

- Some IATA codes and their equivalent in ICAO
- IATA handling codes for goods and GHS/ADR transport pictograms
- 3 country symbol sets (Denmark, Germany, UK)

All the previous vertical modules and mappings are manually defined on spreadsheets. This is the entry point for the mediation component implementation process, thus all spreadsheets follow a specific schema that enables further automatic processing. The schema is defined on D3.31.

Mapping translation takes the spread sheets with the definition of vertical modules and mappings created on the previous step and generates RDF triples to be included as part of EMERGEL. This output RDF is modelled following EMERGEL design guidelines. The software used to convert from tabular structures to RDF has already been introduced as CSV2RDF Transformation Adapter.

Mapping execution is the final task, which is the main activity of the mediation component. D4.30 defines its architecture and a reference implementation based on the Jena framework. The

mediation component is a software library that converts RDF expressed in a source classification into RDF expressed in a target classification. Both classifications are part of EMERGEL, which is also required by the mediation component as input. The last input of this transformer is a set of Jena rules that specify which transformation is required. The processing of these inputs generates a resulting RDF graph according to the target classification. Figure 1illustrates the DISASTER mediation component architecture.

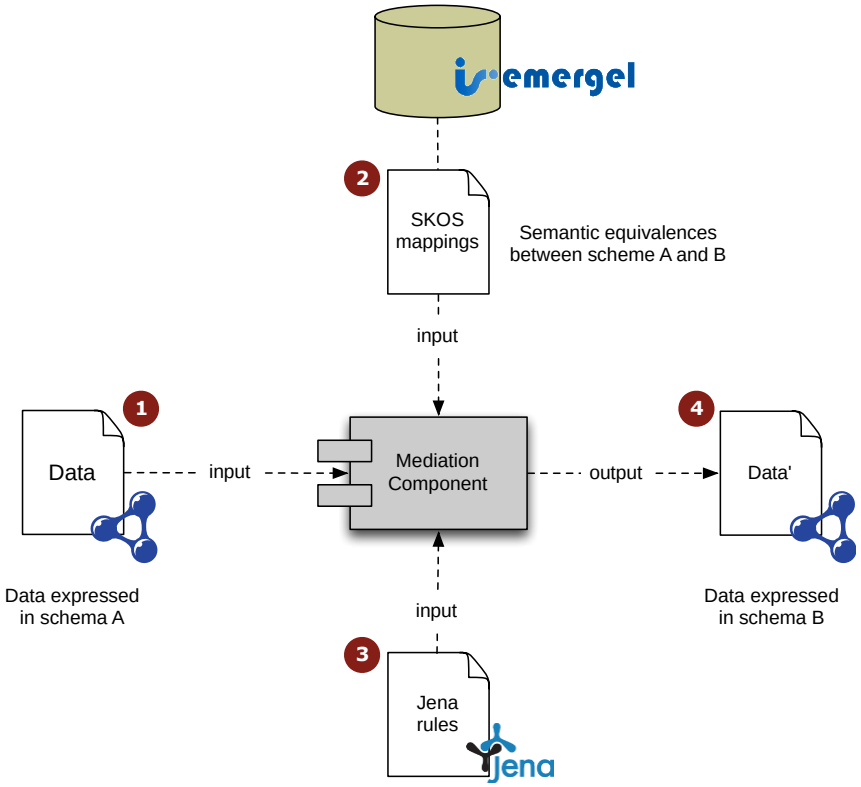

Figure 1. Architecture of mediation component

2.4 EMERGEL API

In order to allow third-party applications to access the ontology, an API has been defined and called “EMERGEL API”. The EMERGEL API is available as REST services following general DISASTER architecture approach. In addition, this API includes an SPARQL Endpoint interface to access the ontology directly. Technical documentation is detailed in D4.30 [16].

The reference implementation of EMERGEL API REST services has been developed using Play Framework 2.1.1 (The High Velocity Web Framework for Java and Scala). The web application that contains the services is deployed over an Apache Tomcat/7.0.26 using Java 1.6.0 24-b24.

3 Architecture implementation

This section presents the technologies used in order to implement the service oriented architecture defined in deliverable D2.52 [1]. The Web Service platform has been chosen as technical paradigm due to its loosely coupled, standard-based approach for building SOA solutions. Figure 2 summarizes the set of WS specifications that DISASTER will adopt in its implementation. This set of standards, called *Disaster Technical stack*, is not a random walk through a space of WS specifications but rather an organized, structured architecture with well-defined designs to fulfil the DISASTER project´s technical requirements.

The *Disaster Technical stack* is divided in six levels according to the nature of the included WS standards. The bottom level is *Transport* that refers to the message format and protocols used to exchange the information. *Description* level includes the standards to describe both functional and non-functional characteristics of the services. *Discovery* level refers to the standards used to publish and organize the services included in the DISASTER solution. *Messaging* level refers to the mechanism provided to ensure that messages are correctly delivered to the appropriate destination. *Quality of Service (QoS)* level focuses on the reliability and security of the interactions. Finally, *Cooperation* level deals with the composition and coordination between multiple service operations when required.

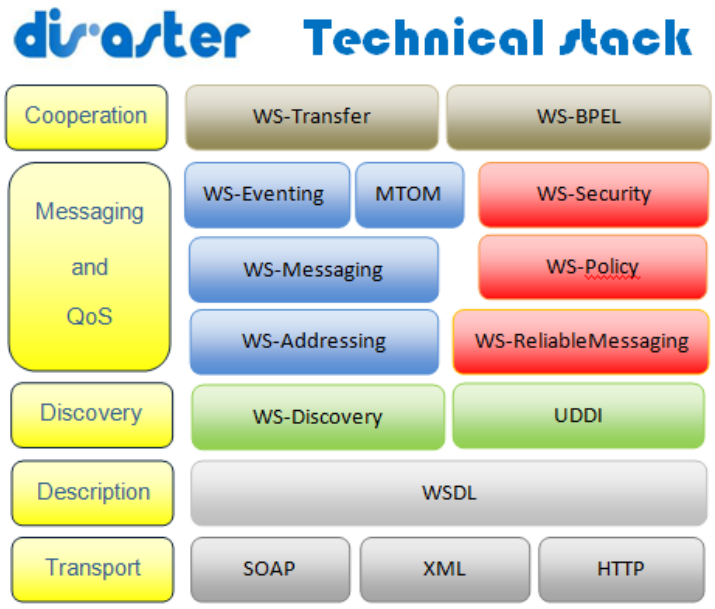

Figure 2. Disaster Technical stack

As defined in D2.52 [1], the DISASTER solution is a network of components (*Mediators*) and a central component (*Core*). The *Core* provides a set of functionality to the mediators making their implementation easier and uniform. That functionality includes data adaptation, data mediation and resource management. In terms of implementation, the *Core* exposes a WS interface where the functionality is split into concrete WS operations. Each *Mediator*is a gateway between a concrete EMS and the rest of existing resources. It allows consuming information from external sources but presenting such data adapted to the concrete EMS characteristics. The *Mediator* relies on the services provided by the *Core* to perform the majority of its activities. In terms of implementation, the *Mediator* is a WS client that interacts with the *Core*, but also it is WS itself providing an interface to the EMS to use the whole DISASTER solution.

Next sections describe the behaviour of the standards included in the *Disaster Technical stack* and how they are be used to fulfil the technical requirements.

3.1 SOA and Web Services

Following the trends within information technology, the technical solution proposed built for DISASTER is framed on the paradigm of service-oriented architecture and distributed computing. Thus, DISASTER retains the benefits of component-based development such as self-description, encapsulation, dynamic discovery and load, adding remotely invoking methods on objects, to passing messages between services [17].

The SOA is an architectural style for building software applications that uses services available in a network such as the Web and is a principle concept underlying beneath Web Services implementation [18].Using a service-oriented architecture provides the capability of using a single resource through its published service and not directly addressing the implementation. This loose coupling allows changes to the implementation by the service provider should not affect the service consumer [19]. This way, maintaining a consistent service interface in each DISASTER component, the service consumer could choose an alternative instance of the same service type without modifying their requesting application, apart from the address of the new instance. This behavior is in components such as, for example, Mediator component of DISASTER, which consume and provide different service interfaces regardless of their implementation.

The SOA approach to DISASTER development will produce a system that can be flexibly adapted to changing requirements and technologies, and, since it is designed based in several components, offers easier maintainable and more consistent systems of data and functionality. Web services are the key component in SOA and they are self-contained, self-describing, they can be published, located, and dynamically invoked. Furthermore, they provide access to sets of operations accessible through one or more standardized interfaces. According to the W3C, web services are XML software systems designed to support interoperable machine-to-machine interaction over a network. This interoperability is gained through a set of XML -based open standards detailed in this document. These standards are a series of protocols that support sophisticated communications between various nodes in a network and they are used to make web services interact with each other. Next sections present the standards used in the DISASTER solution.

3.2 Transport

This layer is placed at the bottom of the technical stack. It refers to the standard specifications used to transport the information between systems. Well-known and widely used technologies have been adopted as communication basis of the DISASTER architecture.

3.2.1 HTTP

The Hypertext Transfer Protocol (HTTP) [20]is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP functions as a request-response protocol in the client-server computing model improving or enabling communications between clients and servers within DISASTER architecture.

HTTP provides to DISASTER a standard set of rules that allow component to communicate with each other for the purpose of sharing and exchange information across the web. DISASTER

Resources, as HTTP resources, are identified and located on the network by Uniform Resource Identifiers (URIs) or, more specifically, Uniform Resource Locators (URLs) using the http or https URI schemes. Also, HTTP can reuse a connection multiple times to download images, data sets, etc after the page has been delivered.

3.2.2 XML

The Extensible Markup Language (XML) was already introduced in Section2.2.1as a data format. But DISASTER also uses XML for the interchange of data over the network. It is widely used in web services for the representation of arbitrary data structures, often to simplify data storage and sharing. One of the key goals of DISASTER is to exchange data between incompatible systems. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

3.2.3 SOAP

Simple Object Access Protocol (SOAP) [21] is the standard messaging protocol used by Web Services. As the protocol is platform-independent, flexible, and based on standard, ubiquitous technologies it is a good fit for DISASTER architecture in order to integrating software applications and sharing data.

SOAP is a network application protocol based on XML markup language used for doing one-way transmissions. This protocol is aimed to defining rules and encapsulating and encoding XML data for transmitting and receiving that data from a SOAP sender to a SOAP receiver. A SOAP XML document instance, known as a SOAP message, carries it payload of some other network protocol between a SOAP sender to a SOAP receiver. SOAP messages are exchanged between applications on a network and are not available for human consumption. The most elegant feature of SOAP is that the platforms and programming languages on either side of a SOAP conversation are independent of one another but are able to communicate as long as each side of the conversation does the following:

- Using HTTP or SMTP to send and receive data transmissions.
- Understanding how to construct and deconstruct binary attachments and MIME encoding rules.
- Using the enveloping and encoding rules established by SOAP to construct and deconstruct of XML documents.
- Performing actions indicated in the SOAP document.

Web services can be used for either *One-Way* messaging or *Request/Response* messaging. As shown in Figure 3, in One-Way messaging, SOAP messages only from the sender to the receiver. In Request/Response messaging, a SOAP message travels from the sender to the receiver, which is expected to send a reply back to the sender.

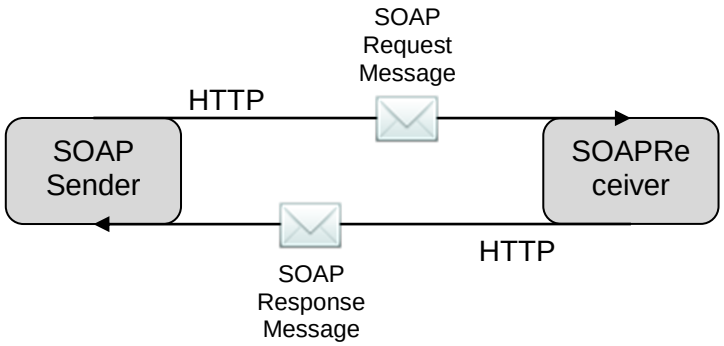

Figure 3. SOAP

In summary, SOAP defines how messages can be structured and processed by software in a way that is independent of any programming language or platform, and thus facilitates interoperability between applications written in different programming languages and running on different operating systems.

3.3 Description

This layer focuses on the description of functional and non-functional features of the services included in the DISASTER architecture. It includes standards for functionality description and policies management.

3.3.1 WSDL

The Web Services Description Language (WSDL) [22] is an XML-based interface description language that is used for describing the functionality offered by a web service. Using WSDL description within DISASTER Core or a Mediator Component, web services provide a machine-readable description of how the operations such as translation, filtering or resources management can be invoked (e.g. what parameters it expects, and what data structures it returns).

The WSDL describes DISASTER components services as collections of network endpoints, or ports. For this propose the specification provides an XML format where definitions of ports and messages are separated from their concrete use or instance, allowing the reuse of these definitions. In this way, WSDL describes the public interface to the web service. Any component involved in the DISASTER architecture (including new mediators) can read the WSDL file of a concrete service to determine what operations are available. In case of any operation uses special data types, these are embedded in the WSDL file in the form of XML Schema. So, a component can then use SOAP to actually call one of the operations listed in the WSDL file using XML or HTTP.

3.3.2 WS-Policy

The DISASTER solution allows interoperability between different emergency response teams by the adaptation of the shared resources. Therefore, the systems involved in DISASTER need to manage the information and resources under its control. As response teams depend more on technology, their ability to control access to resources enforce their administrative policies become central requirements. In a dynamic distributed environment like the architecture proposed by the DISASTER solution, this includes the need to manage and distribute these policies only to authorized entities. In the DISASTER architecture, services policies have a fundamental impact on interoperability. It is important to communicate to potential requesters any policies that a service

provider enforces when those policies impact the interaction either because they require requesters to follow a specific behaviour or a protocol or because they imply service-side behaviour that impacts requester requirements or expectations such as following a particular privacy policy [23]. The WSDL standard addressed the specification of functional capabilities of a service, but it does not support non-functional policies definition such privacy or security. To deal with this lack, the *Disaster Technical stack* employs the WS-Policy standard[24].

The WS-Policy specification defines an XML Schema that defines the structure of valid policy descriptions. Basic building blocks are so-called *assertions*. A single *assertion* may represent a capability, a requirement or a constraint and has a name and properties. WS-Policy introduces three *operators* used to group assertions and sets of assertions, respectively[25]:

- *ExactlyOne*: Exactly one of the child assertions must be satisfied
- *All*: All the child assertions must be satisfied.
- *Policy*: All the child assertions must be satisfied. Additionally, a name can be assigned to the assertion set

An example of policy is presented in Figure 4. The top-level *wsp:Policy* operator has one child assertion set, which is built up with the *wsp:ExactlyOne* operator. The overall policy is satisfied, if one of the two embedded *wsp:All* policy sets is satisfied. In other words, this policy specifies that the message must include either a timestamp and an encrypted signature or a username token.

```
<wsp:Policywsu:Id="DisasterPolicy">
<wsp:ExactlyOne>
<wsp:All>
<sp:IncludeTimestamp/>
<sp:EncryptSignature/>
</wsp:All>
<wsp:All>
<sp:UsernameToken>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Figure 4. WS-Policy

3.4 Discovery

Since DISASTER architecture is based on web services, it is necessary a mechanism to find the suitable service to execute the appropriate operation. The functionality of this layer includes services publication and discovery.

3.4.1 UDDI

DISASTER uses Universal Description, Discovery and Integration (UDDI)[26] as a mechanism to register and locate web service applications. UDDI is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory. Thus, it enables businesses to publish service listings and discover each other, and to define how the services or

software applications interact over the Internet. This business registration consists of three components:

- *White Pages*: address, contact, and known identifiers.
- *Yellow Pages*: industrial categorizations based on standard taxonomies.
- *Green Pages*: technical information about services exposed by the business.

In the DISASTER solution context, the UDDI plays the role of a metadata server of registered Web services. Some DISASTER elements such as mediator components can use a UDDI directory to find other required components. Figure 5depicts the UDDI behavior.

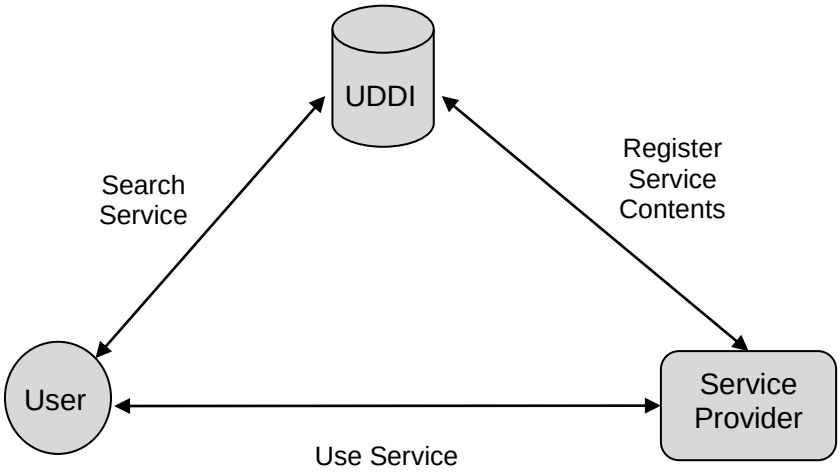

Figure 5. UDDI

3.4.2 WS-Discovery

Web Services Dynamic Discovery (WS-Discovery) defines a multicast discovery protocol that allows locating web services provided by the DISASTER on its network architecture. Usually, to locate a target service by name, a client sends a resolution request message to the multicast group, and again, the target service that matches sends a response directly to the client. WS - Discovery helps to minimize the need for polling, sending an announcement message to the same multicast group, when a target service joins a network. By listening to this multicast group, clients can detect newly-available target services without repeated probing.

Some components, within DISASTER, provide services that are used by other components. These components publish a set of several services in order to make them accessible to other potential consumers. To facilitate discovery, a provider can register a service with UDDI or publish additional documents. Using multicast techniques, such as WS-Discovery, reduces the need for centralized registers searching web services.

3.5 Messaging

This level includes WS standard solutions to provide an interoperable, transport-independent way of identifying message senders and receivers that are associated with message exchange. It is taken into account that non-textual information can be needed to be sent in the SOAP messages. Furthermore, notification messages for updated events must be considered.

3.5.1 WS-Addressing

DISASTER solution provides a set of functionality that is finally exposed as WS operations (endpoints). Thus, the way of using the architecture is by exchanging suitable messages between appropriate participants. First and foremost, mechanism must be provided to ensure that messages are correctly delivered to the right destination. Two problems must be addressed. The first is how to identify the WS endpoints provided by both the *Core* and *Mediators*. This issue is more subtle than in the Web-only perspective because of the rich and dynamic nature of the resources and services covered in the DISASTER project. The second problem is how to communicate these endpoints identifiers in the course of a DISASTER technical operation. To deal with such issues, DISASTER solution uses the WS-Addressing[27] standard. The goal of the WS-Addressing specification is to provide the means to identify a WS endpoints and a way to use such identifiers in SOAP messages for the delivery and exchange of messages between WS providers and WS requesters.

WS-Addressing defines two fundamental concepts: *endpoints references* and *message information headers*. An *endpoint reference* is a data structure to encapsulate all the information that is required to reach a service endpoint (i.e. any potential source of destination of messages). The endpoint reference contains runtime information and associated metadata. The runtime information includes the URI which the service endpoint can be reached. The associated metadata helps consuming applications configure their interaction with the endpoint.

The message information headers allow messages to be directed to service endpoints and to provide the information necessary to support a rich bidirectional and asynchronous interaction. Two message headers are required on every message: *To* and *Action*. *To* header contains the URI address of the target endpoint. *Action* header identifies the semantics of the message. These values include properties such as *source*, *reply* and *fault* endpoints or relationships (*relatedTo*) with previous messages. The previous features enable messaging systems to support message transmission through networks that include processing nodes such as the DISASTER *Mediators* and *Core*.

3.5.2 MTOM

The data managed by the DISASTER solution includes non-textual information (see Section 2). Such information can be large amounts of binary data that have to be sent in SOAP messages. A possible solution is to transform binary data into characters using the allowed Base64 encoding scheme. But this solution has two major problems: (i) encoding/decoding tasks take a long time, and (ii) the increase of the message size when the data is encoded as characters. Both problems are unacceptable for the DISASTER architecture due to the need of effective and efficient cooperation of the EMS in real disaster situations. A solution to these issues is provided by the SOAP Message Transmission Optimization Mechanism (MTOM)[28].

MTOM allows compression of binary data. Data that would otherwise have to be encoded in the SOAP message is instead transmitted as raw binary data in a separate MIME part. A large part of binary data takes up less space than its encoded representation, so MTOM can reduce transmission time. MTOM uses XML-binary Optimized Packaging (XOP)[29]for converting Base64 binary data to MIME data. Figure 6 depicts the send/receive process using MTOM.

Figure 6. MTOM

3.5.3 WS-Eventing

In the management of crisis scenarios it is crucial to have updated information about the situation. EMSs need to be notified when events occur in other services and applications. For example, the DISASTER solution implementation has to be able to notify to their current users if there is an updated version of the data they are using. A mechanism for registering interest is needed because the services interested in receiving such messages are often unknown in advance or will change over time. To achieve such objective, the *Disaster Technical stack* includes the WS-Eventing [30] specification.

WS-Eventing defines a protocol for one service (called a *subscriber*) to register interest (called a *subscription*) with another service (called an *event source*) in receiving messages about events (called *notifications*). The subscriber may manage the subscription by interacting with a service called the *subscription manager* designated by the event source. To improve robustness, a subscription may be leased by an event source to a subscriber, and the subscription expires over time. The subscription manager provides the ability for the subscriber to renew or cancel the subscription before it expires. Figure 7 depicts the main elements of the WS-Eventing standard.

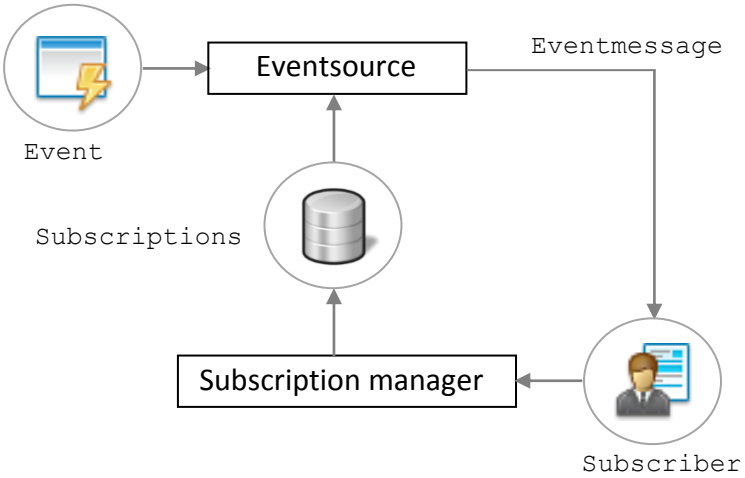

Figure 7. WS-Eventing

3.6 Quality of Service

The WS standards in this layer specify the requirements that are associated with the overall reliability of WS architectures. The specific issues involving this layer include security and reliability of message delivery.

3.6.1 WS-ReliableMessaging

The DISASTER architecture connects different EMS and resources to provide a context-aware mediation of the data. Such connexion is based on the Internet communication channels that are typically unreliable. Connexions break, messages fail to be delivered or in a different sequence to that in which they were sent could bring wrong information to the emergency response teams, and consequently, they could take erroneous decisions on the disaster management. WS-ReliableMessaging [31] addresses these issues and defines protocols that enable WS to ensure reliable, interoperable exchange of messages with specified delivery assurances.

WS-ReliableMessaging is a protocol that ensures the reliable delivery of messages in a distributed environment. It enables messages to be delivered reliably between distributed applications in the presence of software, system, or network failures. The basic value of this specification is the description of a foundation level support layer (*RM source* in the sender, *RM destination* in the receiver) for information replication as depicted in Figure 8. The *delivery assurance* is the concept that defines the quality of service for a message. Four delivery assurances are supported:

- *At least once*: Each message that is sent is delivered at least one time.
- *At most once*: No duplicate messages are delivered.
- *Exactly once*: Each message will be delivered exactly once. If a message cannot be delivered, an error is raised but no duplicate messages are delivered.
- *In-order*: The messages are delivered in the same order in which they were sent.

Messages for which the delivery assurance applies, contain a *sequence header block*. The protocol uses the concept of *sequence* to track and manage the reliable delivery of messages. Each sequence has a *unique identifier element*, and each message within a sequence, has a *message number element* that increments by 1 from an initial value of 1. These values are contained within a *sequence header block* accompanying each message being delivered in the context of a sequence. The receiving endpoint acknowledges receipt of the message within a *sequence* by indicating the range of messages it has received.

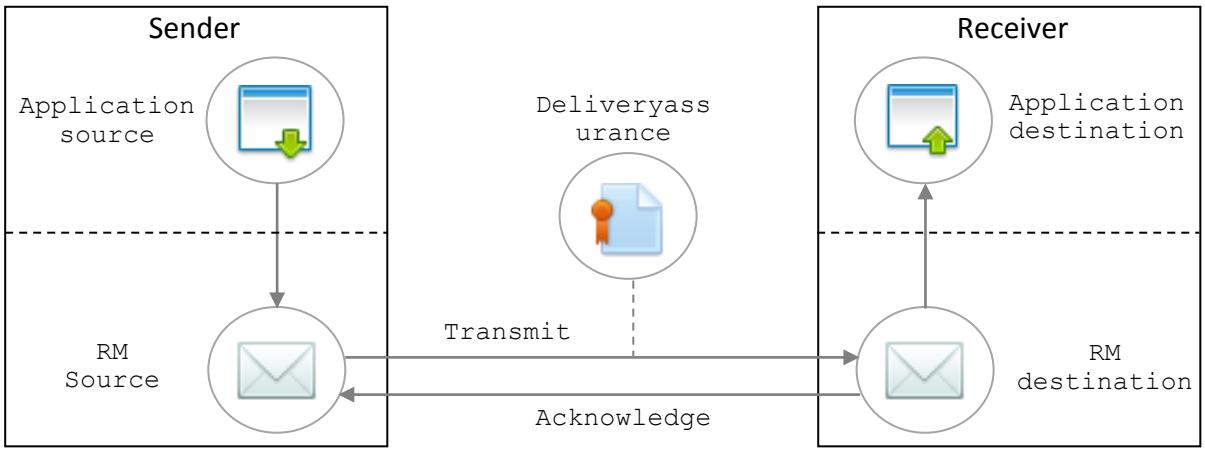

Figure 8. WS-ReliableMessaging

3.6.2 WS-Security

DISASTER solution introduces intermediaries (*Core* and mainly *Mediators*), which might need to modify (e.g. *adaptation* or *mediation*) some parts of passing messages. The architecture is designed with the goal of being open and scalable, so new EMS can easily join to the DISASTER solution by adding its own *Mediator*. These intermediaries are not always completely trusted and should not have access to sensitive data. Furthermore, DISASTER integrates resources from multiple systems with different security domains and technology, and thus the need for a mechanism to translate or exchange security information from a domain to another. WS-Security [32] is a family of specifications that addresses these concerns (Figure 9).

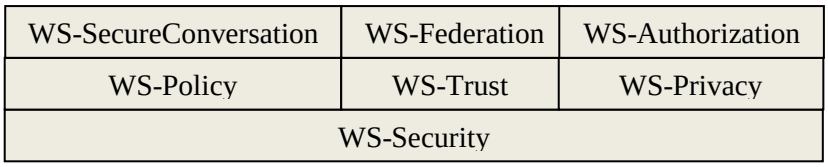

Figure 9. WS security specifications

There are three major problems involved in securing SOAP message exchanges, and WS-Security standards family provides answers for all of them. In fact, WS-Security specification only defines the way to let the receiver know how the message has been protected. To do the actual protecting, WS-Security references additional specifications. The first problem is to identify and authenticate the client. Because there are a lot of different ways to do create security tokens, WS-Security does not specify any particular means, but rather defines how different security tokens should be transferred within SOAP messages. In other words, it lets the receiver know how to extract security tokens from the message for processing. The second problem is ensuring integrity of the message. WS-Security uses digital signatures for that, employing the XML Signature [33] specification to provide a mechanism for digitally signing XML documents. The third problem is keeping the message safe from eavesdropping while it is in transit. To deal with this issue WS-Security employs the XML Encryption standard [34] , which provides a mechanism to encrypt XML documents [35].

In summary, WS-Security relies on the following key concepts:

- *Security tokens*: Authentication information included in the SOAP message
- *Signature elements*: All or part of a SOAP message can be digital signed
- *Encryption element*: All or part of a SOAP message can be encrypted.

WS-Security defines a *SOAP Security header* format that contains subelements for the above concepts. Some tokens might be issued by a trusted third party. In the WS-Security model, a trusted third party is called a *Security Token Service* because one of its tasks is to issue security tokens.

3.7 Cooperation

This level of specifications focuses on the need of composition and coordination of different services in order to achieve more complex processes inside the whole DISASTER architecture. The management of the resources included in DISASTER are also taken into account.

3.7.1 WS-BPEL

DISASTER solution provides its functionality as concrete services endpoints. Each endpoint executes a specific task such as adapting data from one format to other or publishing a new resource. But some processes are complex and involve different services. For example the process to achieve interoperability between an EMS WFS-provider and an EMS WMS-consumer, is composed of a set of tasks: *data format adaptation*, *data schema mediation* and *protocol transformation*. In this kind of scenarios, a mechanism for defining and managing services composition is needed. Such composition model must be sufficiently rich to express different scenarios that DISASTER services might exchange. WS-BPEL [36] was designed to natively address these requirements.

WS-BPEL, also known as BPEL4WS or BPEL, is a language that allows process to describe task dependencies that are connected via Web services. The main use of WS-BPEL is to define the logic to coordinate Web services in a process flow. WS-BPEL makes it possible to keep internal tasks separated from external processes. Hence, changing of internal tasks will not affect data exchange between services. WS-BPEL provides service-based applications with a standard for complex process orchestration and execution. It specifies how to send/receive messages, exchange message with partners, implement parallel or sequential process, manipulate data between partner interactions, define parallel/sequential execution, etc.

WS-BPEL features several basic activities which allow for interaction with the services being arranged in the workflow. These activities cover *invoke*, *receive* and *reply*. Furthermore, it is possible to wait for some time (*wait*), terminate the execution of the workflow instance (*terminate* activity), copy data from one message to another (*assign*), announce errors (*throw*), or just to do noting (*empty* activity). To allow the composition of complex operations, a variety of structured activities exists. *Sequence* offers the ability to define ordered sequences of steps, *flow* executes a collection of steps in parallel whereas the execution order is given by links between the activities. The *switch* activity allows branching, *pick* allows to execute one of several alternative paths and loops can be defined using the *while* activity. Furthermore, WS-BPEL includes the feature of *scoping* activities and specifying *fault handlers* and *compensation handlers* for scopes. *Faults handlers* get executed when exceptions occur, for instance, through the execution of the mentioned throw activity. *Compensation handlers* are activated when faults occur or when compensation activities that force compensation of a scope are executed.

3.7.2 WS-Coordination

As introduced in Section 3.7.1, DISASTER must coordinate different services in order to create bigger process. Some of these services compositions require transactional integrity. In other words, the composition represents a logical unit of work so if any included task fails, the whole composition has to be aborted. In order to ensure reliable execution of services compositions it is crucial that their activities are modelled as transactions such that they achieve a mutually agreed outcome. Web services transactions are defined as sequences of operations that are executed under certain constraints in order to maintain application correctness and data consistency. The fundamental principle of WS transactions is to provide web services applications with reliability and efficiency [37] . DISASTER solution uses WS-Coordination (WS-CO) [38], and the related standards WS-AtomicTransaction (WS-AT) [39]and WS-BusinessActivity (WS-BA) [40], to deal with the transactional requirements of their WS compositions.

WS-Coordination provides a generic coordination infrastructure for web services, making it possible to plug in specific coordination protocols (such as WS-AT and WS-BA.) which work between clients, services and participants. Usually one actor, the coordinator, spreads information to a set of participants to guarantee that all participants obtain a particular message. It defines a coordinator, which is an entity that provides services to applications that want to participate in a coordinated activity. WS-AT is focused on the existing transaction systems and protocols with strict ACID requirements. Existing transaction systems, that require an all or nothing outcome, form an important part of the companies' back-end infrastructure. WS-BA specification defines protocols that enable existing business process and workflow systems to interoperate. It allows managing long-lived transaction by using compensations.

3.7.3 WS-Transfer

The major goal of the DISASTER project is to provide a mechanism so that different EMS can interoperate during the management of crisis scenarios. Such interoperability can be defined as a data exchange (e.g. geospatial information, map images, files, etc) between software systems. Thus, a key issue is in the DISASTER solution is the management of the data (o resources). The DISASTER architectures must allow creating, consulting, modifying and deleting the resources. To manage the resources, DISASTER uses the WS-Transfer [41] specification.

WS-Transfer is a stateless protocol designed to create, read, update, and delete resources. Four single operations are included in the specification:

- *Create*: It creates a new resource instance based on initial data (or link where it is hosted) provided in the request
- *Get*: It gets a XML representation of a single resource
- *Put*: It modifies an existing resource
- *Delete*: It removes an existing resource

4 Conclusions

This deliverable contains the second version of DISASTER architecture-related technologies, defining which protocols and formats are relevant, at this stage, to manage data and information. Furthermore the architecture implementation is described.

A set of web services standards are tailored to implement the DISASTER service-oriented architecture. This stack will ensure the achievement of functional (e.g. specific data formats or communication protocols) and non-functional (e.g. security and policies) requirements. The network of mediators and the central component are the mean to allow DISASTER to be an extensible and scalable project. All this tiles defining the architecture fulfil the requirements specified in the Work Package 2 of the DISASTER project, especially in D2.40 [42] and D2.52 [1].

By using standards specifications, both in architecture implementation and data management side, the implementation will provide the desired interoperability. For example the definition of a common format as RDF simplifies the transformations, translations and enrichment of the data regardless of the initial or final format. Regarding the architecture, the use of web services standards as communication platform will facilitate the integration of new users, who will take advantage of every module implemented before.

References

[1] R. Casado, and E. Rubiera, "D2.52 Reference architecture & data model approach overview – V2," DISASTER project, 2013.

[2] G. González-Moriyón, "D4.10 Algorithms and technology for data mediation state of the art," DISASTER project, 2012.

[3] L. Currás *et al.*, "D5.10 EMS analysis & target selection: data models and data formats used," DISASTER project, 2012.

[4] R. Casado, "DISASTER Proof of Concept: Border Moor Fire Scenario," *The 8th International Conference on Geo-information for Disaster Management (Gi4DM)*, DISASTER project, 2012.

[5] OGC, "Web Map Service," 2006.

[6] OGC, "Web Feature Service," 2010.

[7] W3C, "SPARQL Query Language for RDF," 2008.

[8] W3C, "Extensible Markup Language," 1998.

[9] ECMA, "*JavaScript Object Notation*," 2002.

[10] GeoJSON, "JSON Geometry and Feature Description," 2008.

[11] W3C, "Resource Description Framework," 2004.

[12] OGC, "Geography Markup Language," 2007.

[13] E. Rubiera, and L. Polo, "D3.21 EMS Core Ontology - V1," DISASTER project, 2012.

[14] G. González-Moriyón, L. Polo, and R. Casado, "D4.20 Data mediation techniques - V1," DISASTER project, 2012.

[15] R. Casado, F. Shütte, and J. Groskopf, "D5.31 Proof of Concept compilation report & analysis -v1," DISASTER project, 2013.

[16] G. González-Moriyón *et al.*, "D4.30 Data mediation techniques V2," DISASTER project, 2013.

[17] Y. Su, Z. Jin, and J. Peng, "Building Service Oriented Sharing Platform for Emergency Management – An Earthquake Damage Assessment Example," *Advancing Computing, Communication, Control and Management*, Lecture Notes in Electrical Engineering Q. Luo, ed., pp. 247-255: Springer Berlin Heidelberg, 2010.

[18] Q. H. Mahmoud. "Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)," http://www.oracle.com/technetwork/articles/javase/soa-142870.html.

[19] K. Sahin, and M. U. Gumusay, "Service Oriented Architecture (SOA) based Web Services For Geographic Information Systems," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVII, no. 2, pp. 625-630, 2008.

[20] W3C, "HTTP - Hypertext Transfer Protocol," 1999.

[21] W3C, "http://www.w3.org/2000/xp/Group/."

[22] W3C, "http://www.w3.org/TR/wsdl."

[23] S. Weerawarana *et al.*, *Web Services Platform Architecture: Soap, Wsdl, Ws-Policy, Ws-Addressing, Ws-Bpel, Ws-Reliable Messaging and More*: Prentice Hall PTR, 2005.

[24] W3C. "Web Services Policy 1.5 - Framework," http://www.w3.org/TR/ws-policy/.

[25] H. Bernhard, "WS-Policy: On Conditional and Custom Assertions," in Proceedings of the 2009 IEEE International Conference on Web Services, 2009.

[26] OASIS, "http://uddi.org/pubs/uddiv3.htm."

[27] W3C. "Web Services Addressing (WS-Addressing)," http://www.w3.org/Submission/ws-addressing/.

[28] W3C. "SOAP Message Transmission Optimization Mechanism," http://www.w3.org/TR/soap12-mtom/.

[29] W3C. "XML-binary Optimized Packaging," http://www.w3.org/TR/xop10/.

[30] W3C. "Web Services Eventing (WS-Eventing)," http://www.w3.org/Submission/WS-Eventing/.

[31] OASIS. "Web Services Reliable Messaging (WS-ReliableMessaging)," http://docs.oasis-open.org/ws-rx/wsrm/200608/wsrm-1.1-spec-cd-04.html.

[32] OASIS. "Web Services Security "; https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[33] W3C, "XML Signature Syntax and Processing," 2008.

[34] W3C. "XML Encryption Syntax and Processing," http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[35] N. Chase. "Understanding Web Services Specifications, Parte 4: WS-Security," http://www.ibm.com/developerworks/ssa/webservices/tutorials/ws-understand-web-services4/index.html.
[36] OASIS. "Web Services Business Process Execution Language (WSBPEL)," http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[37] R. Casado, J. Tuya, and M. Younas, "Evaluating the effectiveness of the abstract transaction model in testing Web services transactions," *Concurrency and Computation: Practice and Experience*, pp. in press, 2012.
[38] OASIS, "Web Services Coordination,," *http://docs.oasis-open.org/ws-tx/wscoor/2006/06*.
[39] OASIS, "Web Services Atomic Transaction," http://docs.oasis-open.org/ws-tx/wsat/2006/06, [29 Nov 2011, 2009].
[40] OASIS, "Web Services Business Activity," *http://docs.oasis-open.org/ws-tx/wsba/2006/06*.
[41] W3C. "Web Services Transfer (WS-Transfer)," http://www.w3.org/Submission/WS-Transfer/.
[42] G. González-Moriyón, *D2.40 Technical implications compilation report* 2012.