# eFreight SESA – Technical Report

## e-Freight SESA Architecture Design

### Overview

The SESA architecture provides facilities to register, discover and invoke Web services, automating some of the Web service usage tasks and dealing with some heterogeneities in the services (e.g., by taking care of data mediation).

gives an overview of e-Freight SESA architecture. The core of the SESA is the repository, which stores Web service descriptions and all related knowledge (notably, ontologies). Different components, exposed as Web services, exploit the knowledge in the repository to provide automated discovery and invocation of external Web services. The different components of the SESA architecture will be described in the next sections.



Figure : Overview of e-Freight SESA architecture

### Repository

The *repository* stores the metadata about services which are needed by the other components of the SESA architecture. The semantic descriptions of Web services registered in the platform are stored here and can be queried, so that the repository can be used in the SESA architecture also as a service registry.

The repository is an RDF triplestore, which can be queried via SPARQL. It is however not intended to be directly accessible from outside the SESA architecture[[1]](#footnote-1).

### Management Component

The *management* component allows administering the data stored in the repository, and in particular it allows storing and updating ontologies. These data are not used to describe a single Web service; therefore they are administered by the platform owner/administrator through this component, and not by single Web service owners through the registration component. The management component also allows administering permissions related to service registration for service owners.

The management component is not yet present in the first release.

### Registration Component

The *registration* component allows service owners to register their services in the platform, by storing the relevant information in the repository. Basically, service owners need to submit semantically annotated WSDL descriptions of their services; the registration component extracts relevant metadata and stores it in the repository as RDF triples.

In the future, the service registration component may be enriched with extra validation and checking functionalities with respect to some policy. Using the e-Freight ontology, for example, the platform administrator may specify that a service belonging to category A can or cannot request as input data of type X (where X is a concept in the e-Freight ontology, not a syntactic datatype). By checking the SAWSDL annotations (model references for service category and input/output data), the registration component can check compliance to such policy when a service is registered (or the description is updated).

### Discovery Component

The *discovery* component allows client applications to discover services that can fulfil their goals. In its simplest form, implemented in the first release, the discovery is based on functional categories (classifications) defined in the e-Freight ontologies. In the future, we will consider the possibility to extend the discovery component in order to take into account functional capabilities (condition/effect) and/or non-functional properties, depending on complexity and e-Freight applications requirements.

### Invocation Component (Invoker)

The *invoker* allows client applications to invoke external Web services without having to deal with their specific data schema. The client asks the invoker to invoke a certain Web service (which may have been previously identified through service discovery), passing to the invoker the input data in RDF format, according to the e-Freight ontology. The invoker retrieves from the repository the metadata which is needed to invoke that service, and in particular the ones about lifting and lowering transformations. The input data is lowered to the specific XML schema used by the requested Web service, which is then invoked. The data returned by the Web service is then lifted to RDF, and returned back to the client application.

In its simplest form, implemented in the first release, the invoker basically deals with data mediation, so that client applications can deal with just RDF data according to the e-Freight ontology, without having to worry about schema heterogeneities when invoking external Web services. Besides this, the invoker is transparent in the communication between client applications and external Web services.

In the first release, the invoker is able to invoke WSDL-based services. We will consider the possibility of adding other types of services, notably REST services, in a subsequent release. In the first release, only XSLT scripts are supported as lifting and lowering transformations.

## e-Freight SESA Interface

The following chapter describes the eFreight SESA interface. As an example, a fictitious service of the Valencia Port is used. The input and output are described with the help of a first ontologised version of the FAL Forms.

### Registration Interface

The registration component takes care of registering a WSDL service (annotated with SAWSDL), making it available for discovery and invocation. The SAWSDL annotations should include both schema mappings (sawsdl:liftingSchemaMapping, sawsdl:loweringSchemaMapping) and model references (sawsdl:modelReference).

serviceID **register**(WSDL)

|  |  |
| --- | --- |
| [IN] WSDL | The WSDL input is an URL pointing to the WSDL description. |
| [OUT] serviceID | The serviceID for the registered service or a fault will be returned. |

### Discovery Interface

The discovery component discovers services and corresponding operations based on their semantic annotations, given a list of categories.

outputData **discover**(categoryList)

|  |  |
| --- | --- |
| [IN] categoryList | The categoryList is a list of URIs. These URIs are concepts from the domain taxonomy. For example  [***http://example.org/crs/Taxonomy#Customs***](http://example.org/crs/Taxonomy#Customs)[***http://example.org/crs/Taxonomy#Goods***](http://example.org/crs/Taxonomy#Goods)  Other taxonomies, e.g. for non-functional properties, can be added. |
| [OUT] ouputData | The outputData are RDF triples containing the discovered serviceIDs and the corresponding operationNames. Using the serviceID and operationName one can call the invoker interface (see below). |

We also plan to add a “lookup” operation, which allows the client application to get the full description of a single service, given the respective serviceID.

### Invoker Interface

The invoker allows client applications to invoke external Web services without having to deal with their specific data schema. The client application asks the invoker to invoke a certain Web service (which may have been previously identified through service discovery), passing to the invoker the input data in RDF format, according to the e-Freight ontology. The input data is lowered to the specific XML schema used by the requested Web service, which is then invoked. The data returned by the Web service is then lifted to RDF, and returned back to the client application. The invoker basically deals with data mediation, so that client applications can deal with just RDF data according to the e-Freight ontology, without having to worry about schema heterogeneities when invoking external Web services. Besides this, the invoker is transparent in the communication between client applications and external Web services.

outputData **invoke(**serviceID, operationName, inputData**)**

|  |  |
| --- | --- |
| [IN] serviceID | The serviceID is returned by the service discovery process and consists of the concatenation of service namespace and service name. In our example service the serviceID is **http://www.efreightproject.eu/ValenciaPortWebService** (see *targetNamespace* in *wsdl:definitions* and *name* attributes in *wsdl:service* node) |
| [IN] operationName | The second parameter is the operation name; it is returned by the discovery process. In our example operationName is **submitFALForm** (see *name* attribute in *wsdl:operation*) |
| [IN] inputData | The last parameter is the input data described as RDF instances. The input data is described in section “Example Input Data – FAL Form” |
| [OUT] ouputData | The output data consists also of triples that are described in the domain ontology. |

On a lower level, the following SOAP Message has to be sent to the invoker endpoint.

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:see="http://see.sti2.at/">

<soapenv:Header/>

<soapenv:Body>

<see:invoke>

<serviceID> **http://www.efreightproject.eu/ValenciaPortWebService** </serviceID>

<operation> **submitFALForm** </operation>

<inputData>

<![CDATA[

<!-- Here comes the FAL Form representation as RDF -->

]]>

</inputData>

</see:invoke>

</soapenv:Body>

</soapenv:Envelope>

### Example Input Data – FAL Form

*<?xml version="1.0" encoding="utf-8"?>*

*<!DOCTYPE rdf:RDF [*

*<!ENTITY efreight "http://demo.sti2.at/efreight/ontologies/FalFormsOntology.rdf#" >*

*]>*

*<rdf:RDF*

*xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"*

*xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"*

*xmlns:geo="http://www.w3.org/2003/01/geo/wgs84\_pos#"*

*xmlns:dc="http://purl.org/dc/elements/1.1/"*

*xmlns:dct="http://purl.org/dc/terms/"*

*xmlns:efreight="&efreight;">*

*<efreight:GeneralDeclarationF1 rdf:about="#FAL\_Form1\_3f8b09d5-c430-4b6a-934b-f5e4ad25977b">*

*<efreight:hasArrivalOrDeparture rdf:resource="&efreight;Arrival"/>*

*<efreight:hasArrivalOrDepartureDateTime>2011-02-24T00:00:00</efreight:hasArrivalOrDepartureDateTime>*

*<efreight:hasPort>*

*<efreight:Port>*

*<efreight:hasPortCode>ESVLC</efreight:hasPortCode>*

*<efreight:hasPortName>Valencia</efreight:hasPortName>*

*</efreight:Port>*

*</efreight:hasPort>*

*<efreight:hasBriefDescriptionOfCargo/>*

*<efreight:hasBriefParticularsOfVoyage/>*

*<efreight:hasDepartureDateTime/>*

*<efreight:hasDeparturePort/>*

*<efreight:hasGeneralDeclarationId>354</efreight:hasGeneralDeclarationId>*

*<efreight:hasLastPortOfCall>*

*<efreight:Port>*

*<efreight:hasPortCode>AEAUH</efreight:hasPortCode>*

*<efreight:hasPortName>Abu Dhabi</efreight:hasPortName>*

*</efreight:Port>*

*</efreight:hasLastPortOfCall>*

*<efreight:hasNextPortOfCall/>*

*<efreight:hasNumberOfCrew>0</efreight:hasNumberOfCrew>*

*<efreight:hasNumberOfPassengers>0</efreight:hasNumberOfPassengers>>*

*<efreight:hasRemarks/>*

*<efreight:hasRequirementsInWasteResidueReception/>*

*<efreight:hasShip rdf:resource="#ship\_9HVG8">*

*</efreight:GeneralDeclarationF1>*

*<efreight:Ship rdf:about="#ship\_9HVG8">*

*<efreight:hasCallSign>9HVG8</efreight:hasCallSign>*

*<efreight:hasCoRDate/>*

*<efreight:hasCoRNumber/>*

*<efreight:hasFlagState>*

*<efreight:Country>*

*<efreight:hasCountryName>Malta</efreight:hasCountryName>*

*<efreight:hasCountryCode>MT</efreight:hasCountryCode>*

*</efreight:Country>*

*</efreight:hasFlagState>*

*<efreight:hasGrossTonnage>140000.0000</efreight:hasGrossTonnage>*

*<efreight:hasNetTonnage>40000.0000</efreight:hasNetTonnage>*

*<efreight:hasIMONumber>9191254</efreight:hasIMONumber>*

*<efreight:hasShipMasterName>Jan Tore Pedersen</efreight:hasShipMasterName>*

*<efreight:hasShipType>Cargo</efreight:hasShipType>*

*<efreight:hasShipName>BACCARA</efreight:hasShipName>*

*<efreight:hasShipPositionInPort/>*

*</efreight:Ship>*

*</rdf:RDF>*

### Example Service – Valencia Port

*<?xml version='1.0' encoding='UTF-8'?>*

*<wsdl:definitions name="ValenciaPortWebService"*

*targetNamespace="****http://www.efreightproject.eu/****"*

*xmlns:tns="http://www.efreightproject.eu/"*

*xmlns:ns1="http://schemas.xmlsoap.org/soap/http"*

*xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"*

*xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"*

*xmlns:xsd="http://www.w3.org/2001/XMLSchema"*

*xmlns:sawsdl="http://www.w3.org/ns/sawsdl">*

*<wsdl:types>*

*<xs:schema elementFormDefault="unqualified"*

*version="1.0"*

*targetNamespace="http://www.efreightproject.eu/"*

*xmlns:tns="http://www.efreightproject.eu/"*

*xmlns:xs="http://www.w3.org/2001/XMLSchema">*

*<xs:element name="submitFALForm"*

*type="tns:submitFALForm"*

*sawsdl:loweringSchemaMapping="http://valencia.example.org/falforms/falform\_lowering.xslt"/>*

*<xs:element name="submitFALFormResponse"*

*type="tns:submitFALFormResponse"*

*sawsdl:liftingSchemaMapping="http://valencia.example.org/falforms/falform\_lifting.xslt"/>*

*<xs:complexType name="submitFALForm">*

*<xs:sequence>*

*<xs:element minOccurs="0" name="query" type="xs:string"/>*

*</xs:sequence>*

*</xs:complexType>*

*<xs:complexType name="submitFALFormResponse">*

*<xs:sequence>*

*<xs:element name="return" type="xs:boolean"/>*

*</xs:sequence>*

*</xs:complexType>*

*</xs:schema>*

*</wsdl:types>*

*<wsdl:message name="submitFALForm">*

*<wsdl:part element="tns:submitFALForm" name="parameters"/>*

*</wsdl:message>*

*<wsdl:message name="submitFALFormResponse">*

*<wsdl:part element="tns:submitFALFormResponse" name="parameters"/>*

*</wsdl:message>*

*<wsdl:portType name="ValenciaPortWeb">*

*<wsdl:operation name="submitFALForm">*

*<wsdl:input message="tns:submitFALForm" name="submitFALForm"/>*

*<wsdl:output message="tns:submitFALFormResponse" name="submitFALFormResponse"/>*

*</wsdl:operation>*

*</wsdl:portType>*

*<wsdl:binding name="ValenciaPortWebServiceSoapBinding" type="tns:ValenciaPortWeb">*

*<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>*

*<wsdl:operation name="****submitFALForm****">*

*<soap:operation soapAction="" style="document"/>*

*<wsdl:input name="submitFALForm">*

*<soap:body use="literal"/>*

*</wsdl:input>*

*<wsdl:output name="submitFALFormResponse">*

*<soap:body use="literal"/>*

*</wsdl:output>*

*</wsdl:operation>*

*</wsdl:binding>*

*<wsdl:service name="****ValenciaPortWebService****" sawsdl:modelReference=”****http://example.org/crs/Taxonomy#Customs***

***http://example.org/crs/Taxonomy#Goods****”>*

*<wsdl:port binding="tns:ValenciaPortWebServiceSoapBinding" name="ValenciaPortWebPort">*

*<soap:address location="http://valencia.example.org/falforms/submit"/>*

*</wsdl:port>*

*</wsdl:service>*

*</wsdl:definitions>*

1. At least not for writing operations, which should go through the registration and management components. Other components of the e-Freight platform could be allowed to directly query the repository. [↑](#footnote-ref-1)