Appendix E – Entity-centric Service interface specification

The Entity-centric Services interface [$\mathbf{I_{ES}}$] of an Access Point is a set of HTTP calls with uniform HTTP-headers and a prescribed structure for the HTTP-body. The term "resources" is used for concepts and entities to be addressed. Concepts and entities must be described in a knowledge base which is an implementation of a specific version of a specific semantic model.

This specification describes the following topics:

- *Capabilities*, an overview of the functionality;
- *Information model*, required for addressing (REST) resources and describes how the resources and their status shall be modelled in a knowledge base;
- *Generic specification*, applicable for all API calls such as mandatory HTTP-header attributes and the path structure of the URL;
- *Access restrictions*, authorisation requirements related to the different types and roles identified for an actor;
- *Function specific specification*, a detailed description for each HTTP call of the path to a resource, how to use the header attributes and the expected result.

E.1 – Capabilities

The Entity-centric Services interface [$\mathbf{I_{ES}}$] supports the following entity-centric specific capabilities of an Access Point:

- To create and register an entity;
- To search for an entity and discover which Access Points can provide information;
- To provide information about a concept or entity;
- To perform an entity related action such as publish/subscribe;
- To change or update the attributes an entity including relations to other entities;
- To delete an entity.

Additionally, the following capabilities are supported as well:

- To support invitations for cooperation;
- To provide information about the Access Point itself;
- To route information to other software services as part of a workflow;
- To support life cycle management of concepts and entities and their states.

Section E.5 will elaborate how each capability is supported per API call.

E.2 – Information model

The purpose of the Entity-centric Services interface [I_{ES}] is to exchange information about entities such as cargo and vehicles. Entities are described as concepts in a semantic model.

Concepts

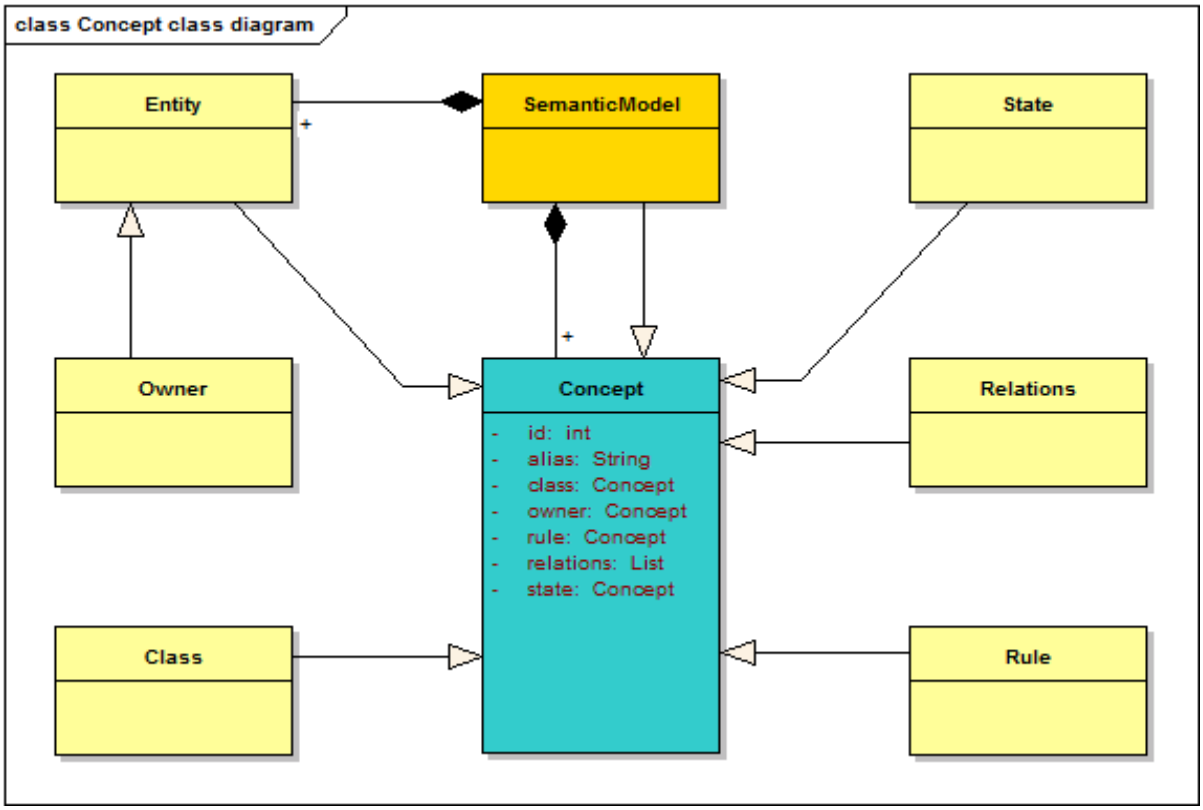

Figure 1 – Semantic concepts class diagram

Concepts and entities have six mandatory attributes which are described in table 20. In this table, the term "reference" is used as a shorthand for a unique pointer to a concept or entity defined in the semantic model being used by the Access Point.

IR-ES-01 The following attributes are mandatory for concepts and entities:

Attribute	*Description*
id	a unique reference (within the scope of an Access Point) of a concept or entity
alias	the business name, alphanumeric code or textual reference of a concept or entity
owner	a reference that represents the actor that controls the lifecycle of this concept or entity
class	A reference that specifies the type of entity as additional distinction in case multiple occurrence of the same name or alias
relations	a list of references that contains the relations with entities and concepts defined in a semantic model. For example, the range of operations for a business service description can be included here as a list of countries or continents
state	a reference that represents the state of the concept or entity. The following states are specified in [D2.1]: TEST, OPERATIONAL & OBSOLETE

Table 1 – Concept class attributes

IR-ES-02 The combination “class” and “alias” shall result in a unique reference.

Consequently, an “id” or “index” can directly be used to refer to a (REST) resource while “alias” needs a preceding classification. This results in two types of path structures:

/<index>

/<alias of a class>/<alias of the resource>

The shorthand “collection” is used in this document for <alias of a class> and the term “member” is used for <alias of the resource> which results in a common (REST) notation:

/<collection>/<member>

Facts

IR-ES-03 A “Fact” is an information model and used to exchange information about an entity. A Fact has “envelope” attributes which describe the context of the information that is included as “payload”. The envelope attributes refer to concepts defined in a semantic model.

Attribute	***Description***
Envelope	
subject[1]	a reference to an entity (the subject)
Source[2][3]	a reference to the originator of the information
Content[2][3]	a reference to the type of information and format being sent as payload
timestamp	the timestamp of occurrence or applicable for the information included in this Fact.
state	a reference to a business process state
place	a reference to a relative or absolute location
Payload	
value	contains the information to be exchanged which can be for example a single measurement, a specific information structure in JSON or a complete XML message.

Table 2 – Fact class attributes

IR-ES-04 The envelope-attribute content shall be used to specify the data structure of the payload. How to encode and decode the payload value should be described in the knowledge base of an Access Point.

IR-ES-05 The timestamp is the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC[2]), 1 January 1970, compliant to the Unix/POSIX timestamp (ISO 8601).

IR-ES-06 The value “0” shall be used when the timestamp is unknown and the value “-1” shall be used to indicated that the time of reception of the message shall be used as timestamp.

---

1 The combination subject-source-content can be used for routing purposes.

2 Coordinated Universal Time (UTC) is the primary time standard by which the world regulates clocks and time. It is one of several closely related successors to Greenwich Mean Time (GMT).

2 Coordinated Universal Time (UTC) is the primary time standard by which the world regulates clocks and time. It is one of several closely related successors to Greenwich Mean Time (GMT).

IR-ES-07 When the location information is not available, the place attribute shall refer to the concept that represents the value: “Unknown” in the semantic model.

Messages

IR-ES-08 The data structure of a message can be described in a semantic model with use of the following concepts:

- *Message*, with an identifying <message-name> that represents a data-structure;
- *Value (statement)*, with a name as identifier
- *Occurrence*, expressed by one of the following three keywords:
  - *required*, this value shall allows exist;
  - *optional*, this value might occur;
  - *repeated*, this value might exists zero or more times.
- *Value-type*, a standard type such as integer, string, Boolean or another message.

In a semantic model, these concepts inherit the alias-attribute from the Concept-class to store the name as a String.

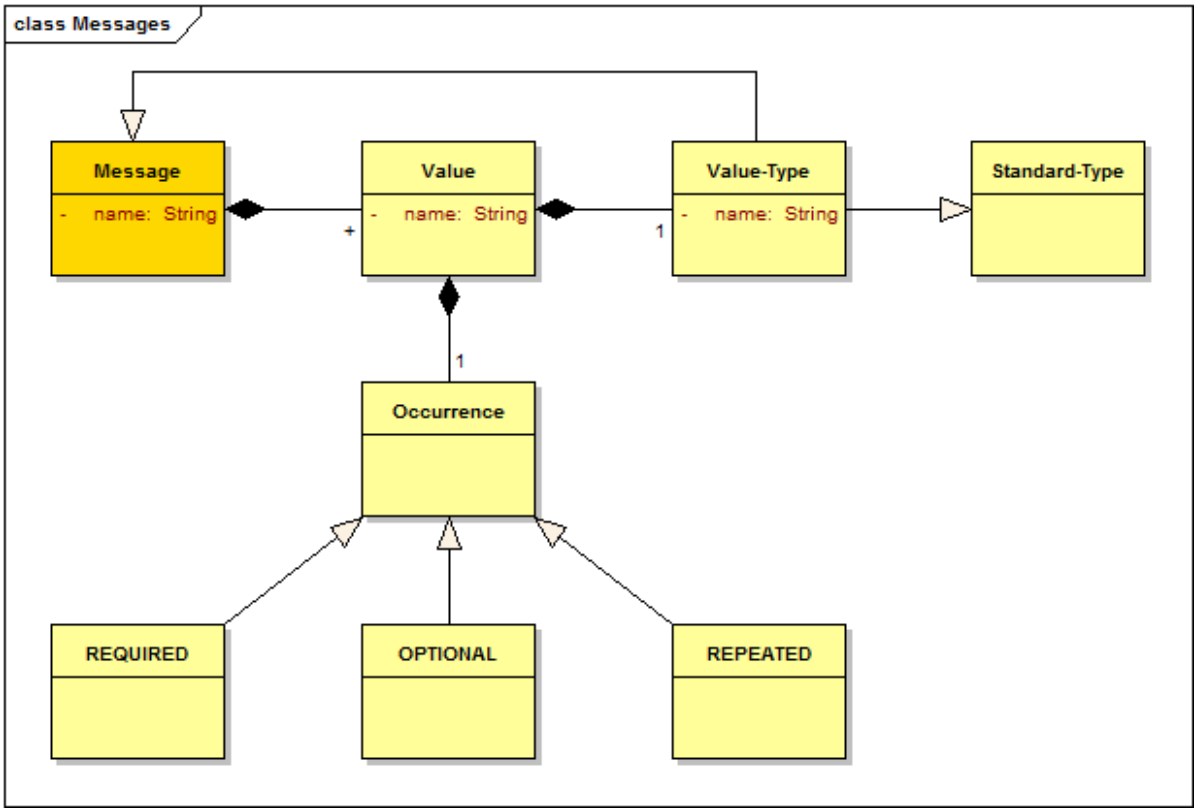

Figure 2 – Message class diagram

The following pseudo notation can be use to describe data-structures:

```
message <message-name> {
    // one or more value-statements are used to specify a sequence of the values
    <occurrence> <value-type> <value-name>
}
```

The sequence of value-statements also implies the sequence of the values in the data-structure. An example can be found in the next section. An overview of the proposed standard value-types is included in appendix F.

Note: The proposed description of data-structures is intended for encoding and decoding purposes to be specified in a semantic model. Compared to XML-Schema, the pseudo notation lacks of constrains and conditional statements but there are also no requirements identified for.

E.3 – Generic specification

This section describes requirements which are applicable for all REST API calls of the Entity-centric Services interface [$\mathbf{I_{ES}}$]. The common requirements specify the uniform use of the HTTP-headers and HTTP-bodies for both the request and the response.

HTTP Methods and required bodies

IR-ES-09 The following HTTP methods shall be supported:

- *GET*, to request information about an Access Point, concept or entity;
- *POST*, to create an new entity or to request an entity related action;
- *PUT*, update the status of an entity with use of Facts;
- *DELETE*, to remove entities from the operational process including their facts.

The POST and PUT methods require additional information to be provided in the HTTP-request-body.

HTTP – request header

IR-ES-10 The following attributes are mandatory for all API calls:

Attribute	***Description***
User	name (alias) of an entity of class “users” (that represents an organisation)
Application	name (alias) of an entity of class “applications”.
API-Key	signature of the user which should kept secret for other users to be provided by the system administrator of an Access Point where the actor (user/system) is registered.
Semantic-Model	name (alias) of the semantic model that is used for the request.
Content-Type	defines the type of the request body (e.g. text/plain, application/json, application/xml)

Table 3 – Request header attributes

IR-ES-11 The values of the User, Application, API-Key and Semantic-Model header attributes must be specified in the knowledge base of the Access Point. The knowledge base represents a specific version of a semantic model that is being used by the Access Point.

Important: The Entity-centric Services interface [$\mathbf{I_{ES}}$] doesn’t prescribe the data structure or encoding of the HTTP-body ! This should be described in the semantic model / knowledge base.

Message conversion(interpretation) and routing of the message content are separated by design in respectively a Semantic Gateway and an Access Point. Consequently, any format can be used including binary. It is an implementation decision to choose a message format that serves the owner of the Access Point in the best way.

The HTTP-request header-attribute: “Semantic-Model” is required to validate if the request and the included information can be understand by the knowledge base of an Access Point. If the requested Semantic-Model is not supported by the Access Point, a HTTP Status Code 403 plus error message shall be replied.

HTTP – response header

IR-ES-12 The following http-response header attributes shall be supported:

Attribute	***Description***
Status Code	the standard client codes as defined in RFC 2616 are applicable for all of the following requests.
Semantic-Model	name (alias) of the semantic model that is used for the request , the accompanying request body if applicable and the response.
Content-Type	defines the type of the response body and shall be equal to the requested Content-Type.

Table 4 – Response header attributes

The Status Code should be used by the API client to verify the outcome of the request. A text message will be returned in the response body to elaborate on the “Status Code”. (See table 14 – Text response). The Semantic-Model and the Content-Type are required to interpret the response properly.

Note: Be aware of Cross-origin resource sharing (CORS) restrictions in case of making XMLHttpRequests to another domain. This might require additional HTTP-header attributes to be implemented at the server. Use of JSONP to by-pass CORS restrictions is not recommended.

HTTP-bodies

IR-ES-13 The HTTP-request and response bodies can contains a set of one or more “Facts”. In case of an error or when the list of Facts is empty then a text message will be returned as HTTP response-body. (See for an overview of text-messages). A “Set“ is the analogy of a mailbag to enable a more efficient processing than sending each Fact as a separate request.

A Fact describes the context of the payload as described in table 2. Storing the context information as part of Facts, allows the requesting and receiving actors to handle multiple messages with different context in a single API call. The design decision to use a Set of Facts reduces the amount of http-calls/overhead and which consequently reduce the amount of the required processing capacity.

The specification of the data-structure shall be described in the semantic-model and the type of encoding (binary, CSV, JSON or XML) shall be indicated by the Content-Type. This means that the HTTP-body can contain any format such as images, PDF documents and even XML messages.

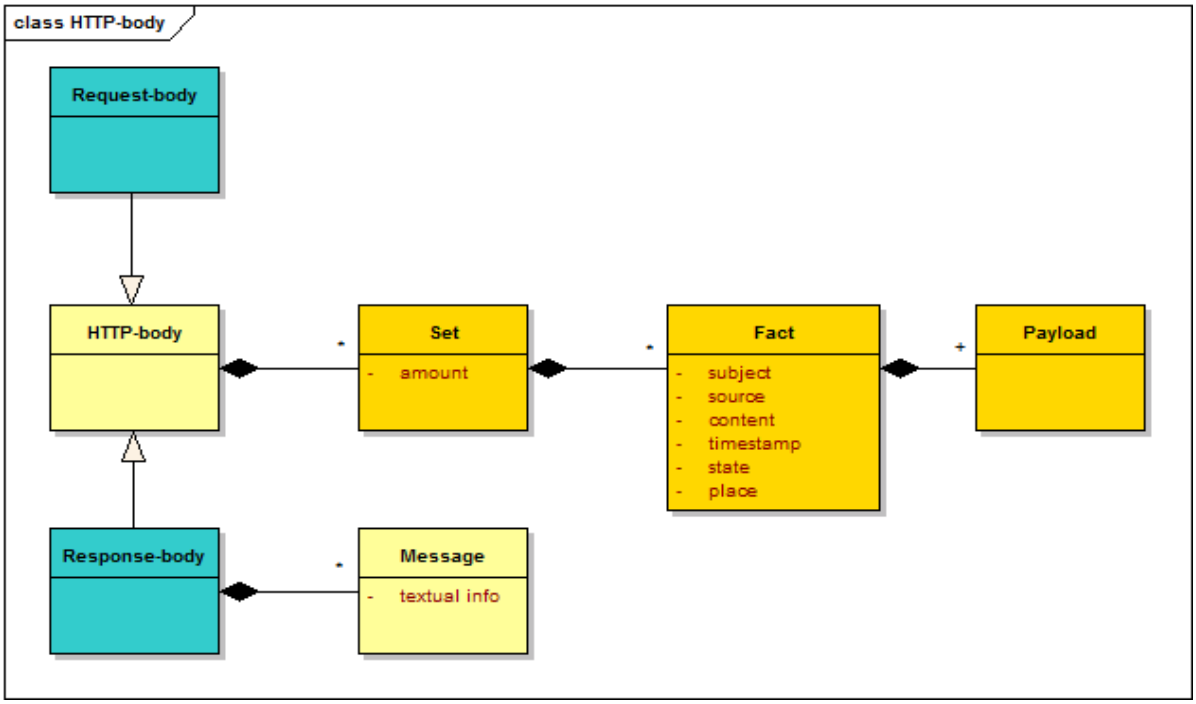

Figure 3 – HTTP-body information model

The following example describes the semantic description of a message (HTTP-body) that contains information about an Access Point.

This semantic description can be used to construct a binary, CSV, JSON or XML HTTP-body. In case of JSON and XML, the keywords and elements will have the same name as the value-name in the semantic description. If necessary, a semantic description can be made for each Content-Type, if different structures or names are required. However, this should be avoided to reduce the complexity of the semantic model and potential errors.

A description of the used value-types can be found in appendix F with exception of value-types: Set, Fact and Access-Point-Details which are included in the description on the right.

```
message Access-Point-Info {
        required Set set
}
message Set {
        required int32 number-of-facts
        repeated Fact facts
}
message Fact {
        required int32 subject
        required int32 source
        required int32 content
        required sint32 timestamp
        required int32 state
        required int32 place
        required Access-Point-Details details
}
message Access-Point-Details {
        required string organisation
        required string core-model
        required string application-version
}
```

Generic path structure

IR-ES-14 The Entity-centric Services interface [I_{ES}] has the following structure:

<HTTP method> https://<domain>/<collection>/[<group>]/<member>?<query>

Where:

- <*HTTP method*> is limited to GET, POST, PUT and DELETE;
- <*domain*> represents the URL of the Access Point, typically a combination of the host address and the path to the application;
- <*collection*> represents the type of functional groups. It is mandatory that collections are defined with a unique alias within a semantic model;
- <*group*> is an optional subset of a collection and are used to specify the class of the subject entity;
- <*member*> represents the identification a specific entity;
- [<*query*>] a string that represents optional filters, limiting and format keywords implemented as key-value pairs separated by an ampersand (&).

IR-ES-15 The <collection>, <group> and <member> parts shall defined as the alias of concepts in the semantic model.

IR-ES-16 The attributes of Concepts and Facts such as “alias”, “class” and “state” can be used as filtering keywords in the query string. See table 1 and 2 for a complete list and description of these attributes.

Using filter keywords as part of the URL is only practical for simple queries. More complex searches need to be specified in the request body of a POST method.

IR-ES-17 The following limiting keywords shall be supported in the query string:

Keyword	***Description***
limit	maximum number of output records (default = 99999999)
offset	The starting record to be send. In combination with the previous limit keyword, the output can be received in parts. (default = 0)

Table 5 – Limiting keywords

IR-ES-18 At least one of the following candidate formats shall be supported by the implementation of the Entity-centric Services interface [I_{ES}]. The HTTP-header request attribute Content-Type shall refer to the expected type of encoding (binary, CSV, JSON or XML).

Format	***Content-Type***	***Type of encoding***
bin	application/vnd.google.protobuf	Binary using Google’s protocol buffer encoding
csv	text/csv	Comma Separated Values
json	application/json	JSON format
xml	application/xml	XML format

Table 6 – Proposed formatting keywords

Note: The description of the data structure shall be included in the semantic model of an Access Point. The content-attribute of a Fact is a reference to that description. Therefore it is not necessary to put a reference to the data structure of the HTTP-body in the HTTP-header.

E.4 – Access restrictions

The following types and roles of actors are identified:

- *System administrator*, is a role of a person who takes care of the operational configuration and life-cycle management of software components;
- *Semantic model administrator*, is a role of a person specifies business goals, processes, activities and involved objects in business jargon (functional terminology)
- *Operator*, is a role of a person who uses information about business objects for operational purposes;
- *External application or web-service*, is software outside the trusted and secured execution environment and uses information about business objects for operational purposes;

Operators, external applications and web-services have the same authorisation rights and therefore only three different acting groups are applicable for authorisation requirements. Human actors require an application or web-service to gain access to the iCargo ecosystem because an Access Point will not have a User Interface.

IR-ES-19 Only certified applications and web-services can exchange information with an Access Point if they have a certificate and an authorization token (API-key). The distribution of an authorization token (API-key) is part of the Entity-centric service interface [$\mathbf{I_{ES}}$] as result of a successful invitation procedure. The certification of applications and web-services is not in the scope of the Entity-centric service interface.

IR-ES-20 The Entity-centric Services interface [$\mathbf{I_{ES}}$] shall rely on dedicated authorisation tokens to identify the software that support system- and semantic model administrators.

IR-ES-21 Only the members of the same group of the creator of a concept or entity can change the information of that concept or entity. To delete a concept or entity, this member have to change the state to “OBSOLETE”.

IR-ES-22 Only the system administrator can remove concepts/entities and related facts permanently in case the state of these concepts and entities is “OBSOLETE”.

IR-ES-23 A semantic model administrator can only create, change and update concepts as long as the concepts have a state different from “OPERATIONAL”. The collection of concepts with the state “OPERATIONAL” should not be changed or extended during operations, to preserve consistent and deterministic behaviour. The creation of new concepts or changes to existing concepts, should result in an update of the semantic model including its version number. After validation and compilation of the new semantic model, an Access Point specific implementation can be installed as the new knowledge base for that Access Point. Testing is supported by allowing changes to concepts with another state then “OPERATIONAL”.

Note: These limitations are not applicable for entities which can be created, changed and deleted at run-time during operations.

IR-ES-24 Data that is intended to be freely available for everyone (OpenData access) shall be provided indirectly via a separate software service. Therefore, the Entity-centric service interface [$\mathbf{I_{ES}}$] shall not support open (unrestricted) access.

The requirements in this section E.4 are summarized in the following table:

Type of request / ***Objective***	***User roles***				
					System administrator
					Semantic model administrator
					Operator / external software services
					Public
Inform					Notes:
Retrieve basic information about Access Point	•	•	•		
Retrieve operational semantic-model	•	•	•		
Retrieve					
Retrieve facts of an entity		•	•		
Retrieve all relations of a class		•	•		
Retrieve all concepts/entities of a class		•	•		
Retrieve information about concept or entity		•	•		
Create or perform action					
Add fact(s) to a specific entity directly		•	•		By group members only
Invoke action based on facts-rule(s)		•	•		Envelope driven tasks
Create a new concept or entity		•	•		
Invoke action based on entity-rule(s)		•	•		Entity driven tasks
Change					
Update properties of a concept/entity		•	•		By group members only
Relate concepts/entities to a group		•	•		By group members only
Add fact(s) for a specific concept/entity		•	•		By group members only
Delete					
Remove specific entity + all facts	•				only if state is “obsolete” !
Remove only the facts of a specific entity	•				only if state is “obsolete” !
Delete a specific concept or entity		•	•		change state to “obsolete”

Table 7 – Authorisation overview

Note: A group member represents a specific type and role of an actor such as “operator” and not a specific person or application.

E.4 – Function specific specification

The following table gives an overview of all available API-calls.

Objective	*Method – Path*
Inform	
Retrieve basic information about Access Point	GET /info
Retrieve operational semantic-model	GET /semantic-model
Retrieve	
Retrieve facts of an entity	GET /<index of entity>
Retrieve all relations of a class	GET /<collection>/<index of class>
Retrieve all concepts/entities of a class	GET /<collection>/<class>
Retrieve information about concept or entity	GET /<collection>/<class>/<entity>
Create or action	
Add fact(s) to a specific entity directly	POST /<index of entity>
Invoke action based on facts-rule(s)	POST /<collection>/<class>
Create a new concept	POST /<collection>/<class>/<new concept name>
Invoke action based on entity-rule(s)	POST /<collection>/<class>/<known entity name>
Change	
Update properties of a concept/entity	PUT /<index of entity>
Relate entities to a group	PUT /<collection>/<class>
Add fact(s) for a specific entity	PUT /<collection>/<class>/<entity>
Delete	
Remove specific entity + all facts	DELETE /<index of entity>
Remove only the facts of a specific entity	DELETE /<class>/<entity>
Remove a specific entity	DELETE /<collection>/<class>/<entity>

Table 8 – Overview of API calls

The shorthand <AP> will be used for https://<domain> in the following specifications.

Inform - request information

IR-ES-25 Basic information about the Access Point can be received with the following call:

GET <AP>/info

This will return at least the following mandatory information in a data-structure as described on page 89 [**IR-ES-13**].

Keyword	*Description*
Organisation	The name of the organisation that this represented by this Access Point.
Core-Model	URL of the (semantic) core model being used by this Access Point
Version	The software version number of this Access Point

Table 9 – Access Point information

IR-ES-26 The operational semantic model (= all concepts) can be received with the following call:

GET <AP>/semantic-model

The response body contains a set of Facts where all concepts with the same owner and state will be combined in one Fact with respectively the same source and state.

Fact attribute	*Value description*
subject	reference to the instance of the semantic model currently used by the Access Point
source	reference to the owner of the concepts included as payload. The owner is either the Semantic Model administrator or the user.
content	reference to the concept "Semantic-Model".
timestamp	the time of the request.
state	reference to the state of all concepts included in the payload at time of the request.
place	reference to the Access Point which should be included as entity.
value (=payload)	an array of Concepts described as an array of attribute values per Concept.

Table 10 – Semantic model information

The response data-structure data structure is very similar to the example on page 32 and now it contains a list of concepts with a value-type as defined below. This value-type has in principle the same attributes as defined in table 1 on page 2. However, the concept-attributes "owner" and "state" are omitted in the description on the left because their value is already represented respectively in the Fact-attributes: "source" and "state".

```
message Concept {
    required int32 id
    required string alias
    required int32 class
    required int32 rule
    repeated int32 relation
}
```

In JSON format, the result could be an array of attribute -values only. Since all attributes are mandatory, the sequence is fixed and keys can be omitted to reduce the message size:

```
[<number of facts>,                                                    // Set
    [<subject>, <source>, <content>, <timestamp>, <state>, <place>,    // Fact
        [<id>, "<alias>", <class>, <rule>, [<relation>,*]]]]           // Concept
```

Note: The response doesn't contain all the concepts defined at the Access Point but only the concepts created by the Semantic Model administrator and the concepts created by the users within the same group. Only the System Administrator of an Access Point can obtain a complete semantic model containing all concepts and entities.

The limiting and filtering attributes are applicable. For example, to retrieve only the concepts with status = OBSOLETE use:

GET <AP>/semantic-model?state=<reference value of OBSOLETE>

Retrieve

The following API calls are designed for retrieving information about resource (concepts and entities) stored at a specific Access Point. The address a specific resource, its local index can be used. Be aware that different behaviour is specified for using the local index or its alias of a resource.

Retrieve information about an entity

To get all facts of specific entity, use:

GET <AP>/<index of resource>

This will return a set of facts compliant to requirement [**IR-ES-13**]. Filtering and limiting keywords can be used to retrieve only relevant or the latest information instead of all facts.

Retrieve all relations of a class

To get all relations stored as concept attribute “relations” from a specific entity, use:

GET <AP>/<collection>/<index of class>

This will return a set of concepts and entities. If no relations are specified, an empty set with amount attribute equal to zero will be returned.

Retrieve all concepts/entities of a class

To get all entities of s specific class, use:

GET <AP>/<collection>/<class>

This will return a set of concepts and entities. If no relations are specified, an empty set with amount attribute equal to zero will be returned.

Retrieve information about concept or entity

To get the concept attributes about a concept or entity, use:

GET <AP>/<collection>/<class>/<entity>

This call uses the entity name or alias attribute while the first retrieve call uses the local index for that entity to get a list of facts.

Create or action

The HTTP POST method is used to invoke a resource specific action which needs to be implemented by the Access Point. The HTTP-request body is expected to contain information compliant to requirement [**IR-ES-13**] for the requested action such as the attribute values of an entity in case of creating an entity.

Add fact(s) to a specific entity directly

The following shortcut can be used to add facts directly to an entity:

POST <AP>/<index of entity>

If successful, a Status Code 201 will be return plus a text message as described in table 14.

Invoke action based on facts-rule(s)

The generic call to invoke an action based on the combination Subject-Source-Content is:

POST <AP>/<collection>/<class>

The action (method) to be invoked on the Access Point is defined as rules in the knowledge base as described in section 4 of this document.

There are three standard methods required for a minimum Access Point implementation:

- To create and register an entity;
- To resolve and identification code (alias) of an entity;
- To search for an entity and discover which Access Points can provide information.

These three calls will be described in more details, while other calls can be implemented too to support company specific actions.

Important: With exception of the three standard methods described above, the proposed API for Entity-centric Service interfaces is independent of any other method to be implemented! Because the relation between the combination Subject-Source-Content and the method to be invoked is configured in the knowledge base.

Register an entity

The following call can be used to add a new entity of specific class:

POST <AP>/<collection>/<class>

The (text)value of <collection> is the super-class of the <class> as defined in the semantic model. For example, the collection "entities" is the super-class of "business-services" in Semantic-Model v0.2 (2013-02-10).

Example: **POST** <AP>/entities/business-services

The following information need to be included in the http-request body:

Attribute	***Description***
Envelope	
subject	a reference to the class where the new entity should be part of
source	a reference to an actor who is creator/owner of the new entity.
content	a reference to NEW_ENTITIES
timestamp	the timestamp of occurrence or applicable for the information included in this fact
state	a reference to the state of the new entity (e.g. TEST, OPERATIONAL, OBSOLETE)
place	a reference to a relative or absolute location, use zero if unknown.
Payload	
value	The following attributes are required: • alias: a text value with the name of the new entity • relations: a list of references to related concepts and/or entities

Table 11 – Registration information

Resolve the identification code

To resolve the identification code or name of an entity and receive a list of uniform resource locators (URL's), the following request can be used:

POST <AP>/<collection>/<class>

The difference with the previous (register) capability for adding a new entity, is the request body. Based on the combination Source-Subject-Content, the appropriated action is being performed.

The following information need to be included in the http-request body:

Attribute	***Description***
Envelope	
subject	a reference to the class as part of the search criteria
source	a reference to an actor who requested the search
content	a reference to SEARCH_CRITERIA or SPARQL
timestamp	the timestamp of the request. Use -1 for current time and 0 for unknown or not applicable
state	a reference to the state of the entity (e.g. TEST, OPERATIONAL, OBSOLETE)
place	a reference to a relative or absolute location, use zero if unknown or not applicable
Payload	
value	<advanced search criteria: defined in SPARQL ?>

Table 12 – Search criteria for resolving

Note: This action/method is not implemented in the Access Point prototype but the interface call is valid for testing purposes The result will be a set of one fact which payload contains one or more URL's.

A more simple and quick search but limited to the scope of one Access Point is:

GET <AP>/<collection>/<class>?filtering keywords

This will return a list of all entities that matches the filtering keywords which are known at the Access Point that handles the request.

Discovery of entities based on a static classification of the entities;

To resolve the identification code or name of an entity and receive a list of uniform resource locators (URL's), the following request can be used:

POST <AP>/<collection>/<class>

The difference with the previous (register) capability for adding a new entity, is the request body. Based on the combination Source-Subject-Content, the appropriated action is being performed.

The following information need to be included in the http-request body:

Attribute	*Description*
Envelope	
subject	a reference to the class as part of the search criteria
source	a reference to an actor who requested the search
content	a reference to SEARCH_CRITERIA or SPARQL
timestamp	the timestamp of the request. Use -1 for current time and 0 for unknown or not applicable
state	a reference to the state of the entity (e.g. TEST, OPERATIONAL, OBSOLETE)
place	a reference to a relative or absolute location, use zero if unknown or not applicable
Payload	
value	<advanced search criteria: defined in SPARQL ?>

Table 13 – Search criteria for resolving

Note: This action/method is not implemented in the Access Point prototype but the interface call is valid for testing purposes. The result will be a set of one fact which payload contains one or more URL's.

The structure of the returned list shall be: <amount>,(<URL>)* where <amount> represents the number of hits that meets the selection

A more simple and quick search (but limited to the scope of one Access Point) is:

GET <AP>/<collection>/<class>?filtering keywords

This will return a list of all entities that matches the filtering keywords which are known at the Access Point that handles the request.

To get a specific business description, use:

GET <URL retrieved from discovery API call>

The URL provided by the discovery API call, as described on the previous page can be broken down in the following parts:

<AP>/<collection>/<class>/<entity-id>

Or use the identification code of a specific business service description:

GET <AP>/entities/business-services/descriptions/<entity-id>

This will return the business service description with identification <entity-id>.

Create a new resource

To create a new resource (concept or entity), the proposed alias must be unique within the range of the <class>. It is allowed to use the same concept name (alias) for multiple concepts and entities as long as they belong to different classes. For example, the concept of "size" can be applicable for pallets as well for containers with different meaning and dimensions.

If the proposed <new resource name> is unique within the range of a <class> than the following call can be used to add a new concept of specific class:

POST <AP>/<collection>/<class>/<new resource name>

If successful, a Status Code 201 will be return plus a text message as described in table 14.

Invoke an action based on resource-rule(s)

A rule can be defined and related to each individual resource and can be invoked with the following call:

POST <AP>/<collection>/<class>/<known resource name>

Depending on the action/method configured in the knowledge base, the response can either be a textual response according to table 14 or a set of facts with the expected values.

Note: This call will always invoke a method! If the resource name is unknown, the Access Point shall assume it is a request to create a new resource and invoke the create-method. An error will be returned if the HTTP-request body doesn't provide the expected content as describe for the call above.

It might look confusing to use to similar API calls for two different objectives but logically and technically both calls are handled in the same way, they both invoke a method with one specific <collection>/<class>/<resource name> as argument.

Change

All required changes shall be included as payload in a fact data-structure as specified in requirement [**IR-ES-13**]. If the request is successfully processed, a Status Code 200 will be return plus a text message as described in table 14.

Update properties of a concept/entity

The HTTP-request body shall contain per resource all the concept-attributes values as prescribed by requirement [**IR-ES-01**].

To update the attributes of a specific concept or entity, use:

PUT <AP>/<index of resource>

Relate entities to a group

This call is intended to support the linking process of relating one on or more resources to another resource such as cargo to a single pallet or container. The HTTP-request body shall contain a fact per grouping resource such as a container and as payload for that fact, a list of references of the related resources representing the cargo items.

To relate one or more entities to a specific concept, class or group, use:

PUT <AP>/<collection>/<resource>

Note: This call is not limited to entities only but can also be used for concepts.

Add facts to an entity

The HTTP-request body shall contain a set of all the facts to be added for the particular resource addressed in the path.

To add new information about an entity as facts, use:

PUT <AP>/<collection>/<class>/<entity>

Delete

Entities cannot be removed directly but the following three step procedure is required:

1. remove all facts of the entity first;
2. change the state of the entity to OBSOLETE;
3. finally a System Administrator can purge an entity from the database.

The second step also changes the status of the entity in the Entity Registry. The entity is being removed from the Entity Repository as well from the Entity Registry with step 3. If successful, a Status Code 200 will be return plus a text message as described in table 14 for the following three calls.

Remove specific entity + all facts

The following shortcut will remove all facts of a specific entity and changes the state of that entity to OBSOLETE:

DELETE <AP>/<index of entity>

This action can only be executed by the owner (the user that created the entity). When a System administrator uses this call, than the entity fill be purged.

Remove only the facts of a specific entity

To remove only the facts of a specific entity, use:

DELETE <AP>/<class>/<index of entity>

The facts will be removed permanently! This action can also only be performed by the owner.

Remove a specific entity

To remove a specific entity, use:

DELETE <AP>/<collection>/<class>/<index of entity>

This action can only be executed by the owner (the user that created the entity). The entity is not removed but it's status will be changed to "OBSOLETE". A system administrator of an Access Point can use this call to purge OBSOLETE entities from the system permanently, including all related facts.

Example: To remove a specific business service (including its description), use:

DELETE <AP>/entities/business-services/<id of business service>

E.5 - States – textual responses

IR-ES-xx Depending on the HTTP-"Status Code" and HTTP-method, the following textual responses shall be send in the HTTP-body in case of an error or when no data is returned.

Status Code	GET	POST	PUT	DELETE	Classification *Meaning of the status code* • Message (Content-Type: text/plain)
					Success
200	X	X	X	X	*The request was fulfilled.* • Info: *updated concept properties.* • Info: *deleted concept.* • Info: *removed concept.*
201		X			*Created.* • Info: *created new concept <alias>* • Info: *number of entities created: <N>*
202			X		*Accepted* Additional information: • Info: *no new relations to add.*
					Response related to a client side problem
400	X	X	X		*The request had bad syntax or was inherently impossible to be satisfied.* • Hint: *missing request body.* • Hint: *empty concept properties.* • Hint: *expected UPDATE_CONCEPTS as content.* • Hint: *exactly one fact is required.* • Hint: *illegal input: <content>* • Hint: *number of facts is <M> instead of <N>*
401	X	X	X	X	*Unauthorized request. Retry with a suitable Authorization header.* • Hint: *user is not the system administrator.* • Hint: *user is not the owner.*
403		X	X		*The request is forbidden.* • Hint: *semantic model <model> is not supported.* • Hint: *subject is different class as <group>.* • Hint: *illegal API address for subject <fact>*
404	X	X	X	X	*The server has not found anything matching the URI given.* • Hint: *unknown collection, group or alias.* • Hint: *index is not numeric.*
					Response related to a server side problem
500	X				*The server encountered an unexpected condition.*
501	X	X	X	X	*The server does not support the required functionality.* • Hint: *function is not implemented yet.*

Table 14 – Textual responses

Appendix F – Standard value-types

To enable re-use of existing and proven software libraries for Google's Protocol buffer, the same standard value-types are proposed for the description of data-structures.

Value type	*Notes*	*Java type*
double		double
float		float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	long
uint32	Uses variable-length encoding.	int
uint64	Uses variable-length encoding.	long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 228.	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 256.	long
sfixed32	Always four bytes.	int
sfixed64	Always eight bytes.	long
bool		boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	String
bytes	May contain any arbitrary sequence of bytes.	ByteString

Table 15 – Standard value-types