



FAST AND ADVANCED STORYBOARD TOOLS

FP7-ICT-2007-1-216048

<http://fast.morfeo-project.eu>

Deliverable D5.1.2
User Manual of the FAST Catalogue

Ismael Rivera, NUIG

Date: 25/01/2010

FAST is partially funded by the E.C. (grant code: FP7-ICT-2007-1-216048).

Version History

Rev. No.	Date	Author (Partner)	Change description
1.0	25/01/2010	Ismael Rivera (NUIG)	First version of the deliverable

Executive Summary

The present deliverable is intended to be the developer's manual of the prototype of the semantic catalogue. That said, whoever developing a component which will interact with the catalogue will find in this document a description of its architecture, the functionality provided together with the API (Application Programming Interface), data interchange formats, code errors and exceptions as well as several examples of usage.

Document Summary

Code	FP7-ICT-2007-1-216048	Acronym	FAST
Full title	Fast and Advanced Storyboard Tools		
URL	http://fast.morfeo-project.eu		
Project officer	Annalisa Bogliolo		

Deliverable	Number	D5.1.2	Name	User Manual of the FAST Catalogue
Work package	Number	5	Name	Semantic catalogue of screen-flow resources and back-end Web Services

Delivery data	Due date	27/02/2010	Submitted	27/02/2010
Status	Status		final	
Dissemination Level	Public <input checked="" type="checkbox"/> / Consortium <input type="checkbox"/>			
Short description of contents	This deliverable is the technical documentation for the prototype developed as part of the WP5. It describes the catalogue's architecture, data interchange formats, APIs and examples of how to interact with it.			
Authors	Ismael Rivera, NUIG			
Deliverable Owner (Partner)	NUIG	email	ismael.rivera@deri.org	
		phone	+353 91 495338	
Keywords	FAST, semantic catalogue, gadget catalogue, RDF store			

Table of contents

1	Introduction	1
1.1	Goal and Scope	1
1.2	Structure of the document	1
1.3	Overall approach for WP5 and relation to other Work Packages and Deliverables	1
1.4	Changes from D5.2.1	2
2	Installation Guide	3
2.1	System Requirements	3
2.2	Obtaining the FAST catalogue	3
2.3	Installation Instructions	4
2.4	Configuration	5
2.5	Hello World!.....	6
3	Architecture Overview.....	7
4	RESTful Catalogue API	9
4.1	JSON Interchange Format.....	9
4.2	Content Negotiation	11
4.3	API Calls	12
4.4	Planning.....	27
4.5	API Error Codes	28
	Appendix A. Lists of Tables and Figures.....	31
	Appendix B. Generic Building Block JSON Structure.....	32
	Appendix C. Screenflow JSON Structure	33
	Appendix D. Screen JSON Structure	34
	Appendix E. Screen Component JSON Structure	36

1 Introduction

This section starts establishing the goal and scope of the present document, shows how it is structured and details the relation to others documents and work packages.

1.1 Goal and Scope

This is an introductory manual for developers who want to adopt and use the FAST semantic catalogue. It explains the main functionalities implemented so far, an overview of its architecture and a detailed *Application Programming Interface* or API of the complete set of the operations offered through a REST service.

1.2 Structure of the document

The deliverable presents both the external and internal architecture in Section 3, then in Section 4 it is detailed the Catalogue API, query formats, interchange formats, error codes and so on.

1.3 Overall approach for WP5 and relation to other Work Packages and Deliverables

The main objective of the Work Package 5, as specified in [DoW, 2007], includes the development of the FAST semantic catalogue, considered the back-end of the FAST platform, which will store and index any kind of resources such as screen-flows, screens, back-end Web services descriptions, locate appropriate gadgets, resources and Web services for users, storing and indexing user profiles, and resolve mediation problems between gadgets, back-end services and the different ontologies stored.

The deliverables stated in [DoW, 2007] for this WP are:

User manual of the FAST Catalogue This is the user manual of the catalogue. Since the se-

semantic catalogue will not provide a user graphical interface by itself: it relies on the GVS; this document will be oriented to developers, explaining technical aspects of the catalogue, its API and several examples to ease its use.

Prototype semantic catalogue for screen-flow gadgets and back-end services This deliverable is a software prototype of the semantic catalogue for experimentation purposes.

The semantic catalogue relies strongly on the FAST ontology defined in [Möller, 2009] since the ontology defines how every element within FAST will be internally stored by the catalogue, and takes into consideration specific requirements detailed in [Villoslada, 2009] where the functionalities of the complete FAST tool are defined.

1.4 Changes from D5.2.1

This should be the first version of this deliverable since in M12 the deliverable 5.1 was not scheduled. Nevertheless, last year an extra document was written as an appendix to the FAST Catalogue in order to detail and clarify technical decisions and to have a technical reference manual. Hence, the deliverable D5.2.1 will evolve to become the D5.1.2, so it will be no longer needed by itself.

There are several new sections and major changes which need to be mentioned:

- Section 2. Installation Guide describes a set of steps to be followed in order to install and execute the FAST Catalogue. Also some configuration instructions are specified.
- Section 4.2. Content Negotiation states how the FAST Catalogue has adopted several of the content negotiation principles.
- Section 4. RESTful Catalogue API has been intensively extended and modified in order to reflect all the new functionality and how every request or response is constructed.
- Several appendixes have been added with the JSON structure of every building block the Catalogue can deal with.

2 Installation Guide

This section provides instructions on how to manually install and configure the FAST Catalogue. The first part of this guide presents some requirements to be considered before the installation, then it gives broad general instructions, and the last part contains a more detailed installation notes for specific configurations.

2.1 System Requirements

In addition to the software itself, a standard FAST Catalogue installation has the following requirements:

- Java¹ (version 6 or later) is required to run the software.
- A Java Servlet Container that supports Java Servlet API 2.4 and Java Server Pages (JSP) 2.0, or newer. We recommend using a recent, stable version of Apache Tomcat². At the time of writing, this is either version 5.5.x or 6.x.

In addition, there are various optional dependencies which are required if you want to use certain advanced features (see Section 2.3).

2.2 Obtaining the FAST catalogue

It is recommended to have a Subversion client installed before you download the code (although you can theoretically download files without Subversion, this would mean tediously downloading each individual file manually). The recommended software is the official Subversion client, available from the Subversion project page³. Note that this client uses a command-line interface, which the instructions below use. Alternatively, you can get subversioning software with a graphical user interface such as TortoiseSVN.

To download from the latest release (recommended), enter the following command from the

¹<http://java.sun.com/>

²<http://tomcat.apache.org/>

³<http://subversion.tigris.org/>

command-line in the directory you wish to download to:

```
svn checkout https://svn.forge.morfeo-project.org/fast-fp7project/trunk/catalogue_service
```

2.3 Installation Instructions

Once you have got the source code, first step should be addressed is its compilation. You may manually compile the source yourself, however the FAST Catalogue comes along with a script to facilitate this task. This script has been made using a Java build tool called Ant ⁴. Hence, we recommended to get and install such a tool in order to follow the instructions below.

Now you can compile, package and run the application via:

```
ant clean
ant compile
ant dist
```

Or using a task which gather the previous three tasks:

```
ant main
```

After executing with no errors this task, a WAR file should have been created in the /dist directory. A WAR file, which stands for Web Application Archive, is just a JAR file used to distribute a collection of JavaServer Pages, servlets, Java classes, XML files, tag libraries and static Web pages (HTML and related files) that together constitute a Web application, in this case, the FAST Catalogue.

Then, last step is to deploy the WAR file into the servlet container. If the server chosen was Apache Tomcat, the procedure for deploying a Web application is:

1. Stop Tomcat.
2. Delete existing deployment. If you have previously deployed "foo.war" in TOMCAT_HOME/webapps, then it has been unpacked into webapps/foo/... You must delete this directory and all its contents. On Unix, this can be done with `rm -r $TOMCAT_HOME/webapps/foo`
3. Copy WAR file to TOMCAT_HOME/webapps/.

⁴<http://ant.apache.org/>

4. Start Tomcat.

Now you have successfully built the FAST catalogue, however we highly recommend you to read Section 2.4 and tune your FAST Catalogue instance.

2.4 Configuration

The following section covers the configuration options you may have to set up before start using the Catalogue. The FAST Catalogue relies on a Sesame repository for the persistent storage. Sesame repositories can be local or remote. Remote does not mean, it has to be installed in a different machine, but in a servlet container such as Apache Tomcat. The advantage of remote repositories is the possibility of accessing it via a SPARQL endpoint. These configuration aspect can be found in the file *repository.properties* in the root directory where the Catalogue has been deployed.

Independently the type of repository, the parameter *serverURL* needs to take the value of the URL where the FAST Catalogue instance has been deployed. If you are testing the application in your machine, it may be something like `http://localhost:8080/FASTCatalogue`.

Local repositories just need the parameter *storageDir* pointing at a local directory, which has write permission in order to allow Sesame to create files in that directory or to delete any file in the directory. For instance:

```
storageDir=D:\\FAST\\catalogue\\repository
```

Nevertheless, setting up a remote repository requires a few more steps. First the OpenRDF Sesame Server needs to be installed. The installation guide of this server is detailed in the Chapter 5 and 6 of the User Guide for Sesame 2.2 [B.V., 2008]. Once the Sesame server is deployed, and a repository has been created, the parameters *sesameServer* and *repositoryID* will take the values of the Sesame server URL and the repository identifier. For instance:

```
sesameServer=http://localhost:8880/openrdf-sesame  
repositoryID=c2
```

2.5 Hello World!

It is recommended to check if the FAST Catalogue has been successfully installed. To do that, a simple HTTP request, using a Web browser, can be sent to the URL the Catalogue is deployed, and specifying a type of resource such as Screen. The URL would be similar to `http://localhost:8080/FASTCatalogue/screens`. Depending on the variant chosen, the response may be different. For JSON, the response should be `[]`, but in HTML the message `No screens found.` should be shown.

3 Architecture Overview

This section presents a technical overview of the internal catalogue's architecture and which external components will interact with it. Basically three layers can be distinguished: Presentation, Business Logic and Persistence layer (see Figure 3).

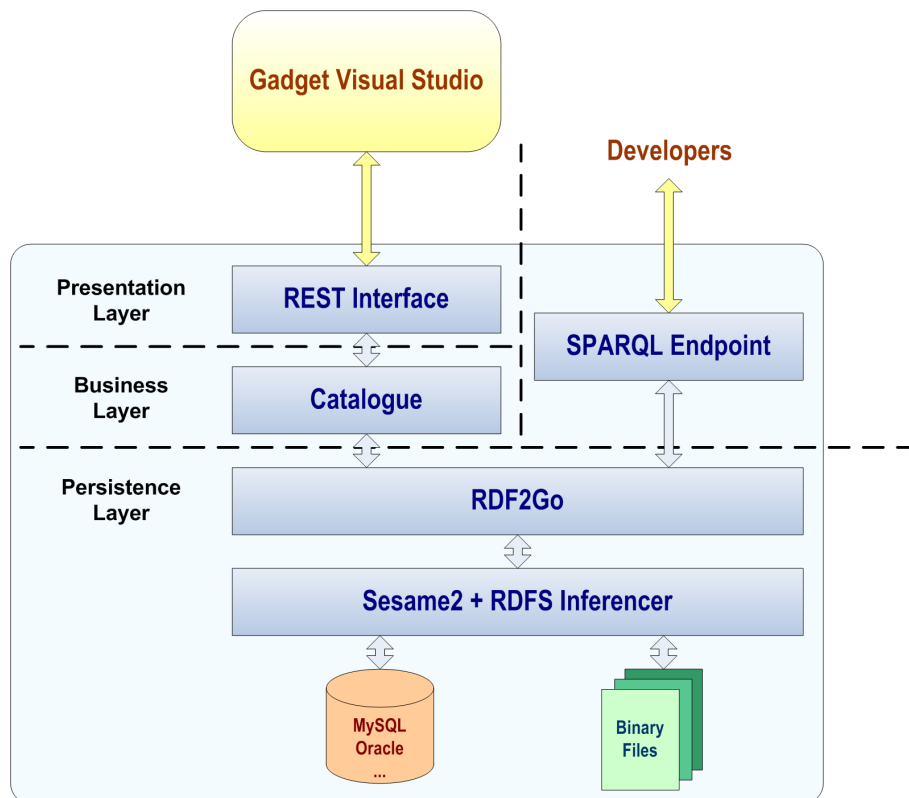


Figure 1: Catalogue's Architecture

The Presentation layer will be the public interface of the catalogue. The main purpose of this layer is to provide its functionality to other components within the Gadget Visual Studio. The service is offered in a REST (REpresentational State Transfer) style, making it easy to consume than other complex APIs. REST is an architectural style, not a toolkit or a standard. Even though, it makes use of standards like HTTP, URI, Resource Representations (XML, HTML, JSON, JPEG, etc.) or MIME types (text/xml, text/html, application/json, image/jpeg, etc.). Another characteristic of the REST style is its stateless assumption. The catalogue adopts this strategy; therefore every request needs a complete set of information in order to prepare the response.

As part of the Presentation layer, a SPARQL endpoint is offered using the SPARQL protocol service as defined in the SPROT [Grant et al., 2008] specification. The SPARQL endpoint is mostly

offered to enable other developers to query directly the catalogue knowledge base using SPARQL queries. This feature is supported by the Sesame RESTful HTTP interface for SPARQL Protocol for RDF.

The Business Logic layer contains all the FAST domain-specific processing. It provides functions to interact with all the elements of the domain model specified in [Möller, 2009]. It processes the input from the Presentation layer, creating specific objects modelling the business logic, and vice versa. In addition it interacts with the Persistence layer in order to persist the model.

The Persistence layer provides an API which allows the Business layer to work with a standard set of objects that read and save their state to the triple store. For this reason, on top of this layer it is used RDF2Go, an abstraction over triple (and quad) stores. The RDF2Go API allows interacting with the semantic representation of the model (triples) in a generic manner, and also brings the flexibility of choosing different triple stores to persist them. The selected implementation for the triple stored is Sesame 2. Moreover, this framework is completely extensible and configurable regarding to storage mechanisms, inferencers, RDF file formats, query result formats and query languages. For the actual catalogue prototype, the storage mechanism used is the native file storage system of Sesame, the inferencer used is a subset of the RDFS entailment rules [Hayes, 2004] following a forward-chaining policy.

4 RESTful Catalogue API

This section provides a high-level overview of the Catalogue API. It describes the calls or operations supported, specific parameters and responses for each operation, supported interchange formats and some examples to facilitate the understanding of the API.

4.1 JSON Interchange Format

The format supported by the Catalogue API for all the request and responses is JSON⁵. JSON is a lightweight data interchange format whose simplicity has resulted in widespread use among web developers, easy to read and write and able to using any programming language because its structures map directly to data structures used in most programming languages.

Every HTTP request and response is encoded using the MIME type `application/json` and the charset UTF-8.

4.1.1 Example of a screen

```
{
  "uri": null,
  "label": {"en-GB": "Screen - DERI"},
  "description": {"en-GB": "Returns people working in DERI"},
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "domainContext": {
    "tags": [],
    "user": null
  },
  "homepage": "http://www.deri.ie/",
  "icon": "http://www.deri.ie/icon.jpg",
```

⁵<http://www.json.org/>

```
"screenshot": "http://www.der1.ie/screenshot.jpg",
"rights": "http://creativecommons.org/",
"version": "1.0",
"preconditions": [],
"postconditions": ["?person rdf:type foaf:Person .
                    ?person foaf:workplaceHomepage http://www.der1.ie/"]
}
```

4.1.2 Internationalisation l18n

In order to offers an adaptable solution to various languages and regions without major engineering changes, internationalisation is considered from an early stage in the catalogue development. Underlying representation technologies used to develop the catalogue (RDF and JSON) allow to implement this feature. This feature is implemented by the addition of a language tag to every 'string' desired. The specification of this language tag is composed by the language code and the country code, following the ISO 639⁶ for languages and the ISO 3166⁷ for countries. The following example illustrates how to use it properly in both formats.

```
{
  "label": {
    "en-GB": "Simple Example",
    "es-ES": "Ejemplo Sencillo",
    "de-DE": "Einfaches Beispiel"
  },
  "description": {
    "en-GB": "This is a simple example of a screen",
    "es-ES": "Esto es un ejemplo sencillo de pantalla",
    "de-DE": "Dies ist ein einfaches Beispiel für einen Bildschirm"
  },
}
```

```
<rdf:Description rdf:about="http://www.morfeoproject.eu/fast/fco#ExampleScreen">
```

⁶<http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt>

⁷http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

```
<rdf:type rdf:resource="http://www.morfeoproject.eu/fast/fco#Screen"/>
<rdfs:label xml:lang="en-GB">Simple Example</rdfs:label>
<rdfs:label xml:lang="es-ES">Ejemplo Sencillo</rdfs:label>
<rdfs:label xml:lang="de-DE">Einfaches Beispiel</rdfs:label>
<description
  xmlns="http://purl.org/dc/terms/"
  xml:lang="en-GB">This is a simple example of a screen</description>
<description
  xmlns="http://purl.org/dc/terms/"
  xml:lang="es-ES">Esto es un ejemplo sencillo de pantalla</description>
<description
  xmlns="http://purl.org/dc/terms/"
  xml:lang="de-DE">Dies ist ein einfaches Beispiel für einen Bildschirm</description>
</rdf:Description>
```

By now it is allowed using internationalisation for the label and the description of a screen, but more attributes will be benefited by this feature.

4.2 Content Negotiation

Before any major details about how content negotiation is addressed by the catalogue, a few concepts need to be understood. First of all, say a URI is not a file name. It is common to see a URI as the location for a file, but a URI or Universal Resource Identifier is not a URL: it is not a file name or location, it is an identifier (or a reference) to a resource. Then by dereferencing a URI the actual resource can be retrieved. Let's see a request to retrieve the resource "Screen 123" is sent to the server. The resource is just a piece of information, however the server will answer an HTML document with a text describing the screen or an image representing it. On the Web, these equivalent representations of a resource are called variants, and content negotiation is the mechanism used to determine which of the representations of most appropriate for a given request.

In summary, the basic idea of content negotiation, as stated in the HTTP 1.1 specification [Fielding et al., 1999], is to serve the best variant for a resource, taking into account what variants are available, what

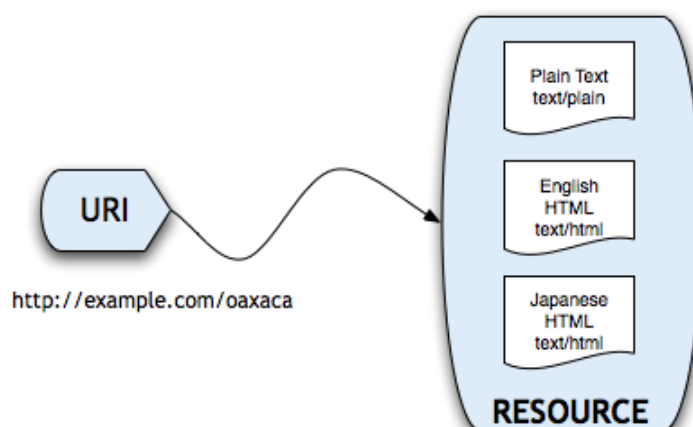


Figure 2: Content Negotiation

variants the server may prefer to serve, what the client can accept, and with which preferences: in HTTP, this is done by the client which may send, in its request, Accept headers (Accept, Accept-Language and Accept-Encoding), to communicate its capabilities and preferences in Format, Language and Encoding, respectively.

In fact, what the catalogue really does is "format negotiation" since the alternate representations are just based on the selection of the media type, through the Accept header, but does not consider different languages or encoding types. The formats supported are JSON, RDF+XML and HTML.

Lastly, content negotiation needs to identify which player is going to take the lead on it. There are two kinds of content negotiation which are possible in HTTP: server-driven and agent-driven negotiation. The approach followed by the catalogue is agent-driven negotiation, hence selecting a specific representation for a resource is responsibility of the user agent. If none is specified, by default, the server will choose the JSON representation.

4.3 API Calls

This section contains detailed descriptions of the interface provided by the catalogue via REST services, detailing request parameters, response elements, any special errors and examples of requests and responses. The URL format is also specified for each operation, where 'catalogueURL' has to be replaced for the real URL the service is installed (e.g. <http://demo.fast.morfeo->

project.org/catalogue).

4.3.1 CRUD operations for Screens

A screen is a resource which can be created, retrieved, modified or deleted since the catalogue provides CRUD support for screens. These features are provided following the REST philosophy, using the HTTP methods. Two concepts have to be defined: *collection*, as a set of screens which access URI is `http://catalogueURL/screens/` where *catalogueURL* is the specific URL where the catalogue server is installed, and *member* of the collection, in other words, the screen itself, which access URI is `http://catalogueURL/screens/<id>` where the `<id>` has to be replaced by the identifier of a screen. The details of the operations and which HTTP verb has to be used can be found in the Table 1.

Table 1: CRUD operations for Screens

Resource	HTTP method	HTTP body	Description
Collection URI	GET	N/A	List the members of the collection.
Collection URI	PUT	N/A	Not used.
Collection URI	POST	JSON representation of the screen	Create a new entry in the collection where the URI is assigned automatically by the collection. The URI created is returned by this operation.
Collection URI	DELETE	N/A	Not used.
Member URI	GET	N/A	Retrieve the addressed member of the collection.
Member URI	PUT	JSON representation of the screen	Update the addressed member of the collection or create it with a defined URI.
Member URI	POST	N/A	Not used.
Member URI	DELETE	N/A	Delete the addressed member of the collection.

Here are shown various examples of request and responses using the catalogue server. To create

a new screen, a POST request is send to the catalogue server, including the JSON representation of the screen as the body of the request:

```
{
  "uri": null,
  "label": {"en-GB": "Screen - DERI"},
  "description": {"en-GB": "Returns people working in DERI"},
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "domainContext": {
    "tags": [],
    "user": null
  },
  "homepage": "http://www.deri.ie/",
  "icon": "http://www.deri.ie/icon.jpg",
  "screenshot": "http://www.deri.ie/screenshot.jpg",
  "rights": "http://creativecommons.org/",
  "version": "1.0",
  "preconditions": [],
  "postconditions": ["?person rdf:type foaf:Person .
                    ?person foaf:workplaceHomepage http://www.deri.ie/"]
}
```

The URI will be not taken into account since the catalogue will generate a unique URI for each resource created. So, the response for this operation is:

```
{
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-GB": "Show information about a person working in DERI"},
  "domainContext": {
    "tags": [],
    "user": "null"
  },
  "homepage": "http://www.deri.ie/",
```

```
"icon": "http://www.deri.ie/icon.jpg",
"label": {"en-GB": "Screen - DERI"},
"postconditions": [],
"preconditions": ["?person rdf:type foaf:Person .
                  ?person foaf:workplaceHomepage http://www.deri.ie/"],
"rights": "http://creativecommons.org/",
"screenshot": "http://www.deri.ie/screenshot.jpg",
"uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552194269",
"version": "1.0"
}
```

To obtain all the screens stored in the catalogue a HTTP GET request is sent to the Collection URI and the response may be something like this:

```
[
{
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-gb": "Returns people working in DERI"},
  "domainContext": {
    "tags": [],
    "user": null
  },
  "homepage": "http://www.deri.ie/",
  "icon": "http://www.deri.ie/icon.jpg",
  "label": {"en-gb": "Screen - DERI"},
  "postconditions": ["?person rdf:type foaf:Person .
                    ?person foaf:workplaceHomepage http://www.deri.ie/"],
  "preconditions": [],
  "rights": "http://creativecommons.org/",
  "screenshot": "http://www.deri.ie/screenshot.jpg",
  "uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552164472",
  "version": "1.0"
},
```

```
{
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-gb": "Show information about a person working in DERI"},
  "domainContext": {
    "tags": [],
    "user": null
  },
  "homepage": "http://www.deri.ie/",
  "icon": "http://www.deri.ie/icon.jpg",
  "label": {"en-gb": "Screen - DERI"},
  "postconditions": [],
  "preconditions": ["?person rdf:type foaf:Person .",
    "?person foaf:workplaceHomepage http://www.deri.ie/"],
  "rights": "http://creativecommons.org/",
  "screenshot": "http://www.deri.ie/screenshot.jpg",
  "uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552194269",
  "version": "1.0"
},
{
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-gb": "Show information about a person"},
  "domainContext": {
    "tags": [],
    "user": null
  },
  "homepage": "http://www.deri.ie/",
  "icon": "http://www.deri.ie/icon.jpg",
  "label": {"en-gb": "Screen - DERI"},
  "postconditions": [],
  "preconditions": ["?person rdf:type foaf:Person"],
  "rights": "http://creativecommons.org/",
```

```

    "screenshot": "http://www.deri.ie/screenshot.jpg",
    "uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552373865",
    "version": "1.0"
  }
]

```

4.3.2 CRUD operations for Screen-Flows

A screenflow is a resource which can be created, retrieved, modified or deleted since the catalogue provides CRUD support for screenflows. These features are provided following the REST philosophy, using the HTTP methods. As in the previous section the concepts *collection* and *member* are used including their URLs. Table 2 details the operations offered for screen-flows.

Table 2: CRUD operations for Screen-Flows

Resource	HTTP method	HTTP body	Description
Collection URI	GET	N/A	List the members of the collection.
Collection URI	PUT	N/A	Not used.
Collection URI	POST	JSON representation of the screen-flow	Create a new entry in the collection where the URI is assigned automatically by the collection. The URI created is returned by this operation.
Collection URI	DELETE	N/A	Not used.
Member URI	GET	N/A	Retrieve the addressed member of the collection.
Member URI	PUT	JSON representation of the screen-flow	Update the addressed member of the collection or create it with a defined URI.
Member URI	POST	N/A	Not used.
Member URI	DELETE	N/A	Delete the addressed member of the collection.

The following is an example of the request and response while creating a screen-flow:

```
{
  "uri": null,
  "label": {"en-GB": "My First ScreenFlow"},
  "description": {"en-GB": "A example of screenflow"},
  "creationDate": "2009-02-05T13:11:41+0000",
  "creator": "http://www.ismaelrivera.es/",
  "domainContext": {
    "tags": ["people"],
    "user": null
  },
  "homepage": "http://www.screenflow.com/",
  "icon": "http://www.screenflow.com/icon.jpg",
  "screenshot": "http://www.screenflow.com/screenshot.jpg",
  "rights": "http://creativecommons.org/",
  "version": "1.0",
  "preconditions": [],
  "postconditions": [],
  "screens": [
    "http://www.morfeoproject.eu/fast/fco#Screen1233750841383",
    "http://www.morfeoproject.eu/fast/fco#Screen1233750828836",
    "http://www.morfeoproject.eu/fast/fco#Screen1233750820914"
  ]
}
```

```
{
  "creationDate": "2009-02-05T13:11:41+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-GB": "A example of screenflow"},
  "domainContext": {
    "tags": ["people"],
    "user": null
  },
  "homepage": "http://www.screenflow.com/",
  "icon": "http://www.screenflow.com/icon.jpg",
```

```
"label": {"en-GB": "My First ScreenFlow"},
"postconditions": [],
"preconditions": [],
"rights": "http://creativecommons.org/",
"screens": [
  "http://www.morfeoproject.eu/fast/fco#Screen1233750841383",
  "http://www.morfeoproject.eu/fast/fco#Screen1233750828836",
  "http://www.morfeoproject.eu/fast/fco#Screen1233750820914"
],
"screenshot": "http://www.screenflow.com/screenshot.jpg",
"uri": "http://www.morfeoproject.eu/fast/fco#ScreenFlow1233752391898",
"version": "1.0"
}
```

4.3.3 Find

The find operation search inside the catalogue for any screen which could be somehow related to the gadget the user is creating. It provides a recommended set of screens depending on the domain context, the canvas, etc. The specific URL to access this operation is <http://catalogueURL/find> using HTTP POST method. From now on, the method 'find' only considers the domain context, being a list of tags and the pre/postconditions of the screens. It will try to satisfy all the unsatisfied preconditions of a given list of screens also known as canvas. The results will be all the screens stored in the catalogue which fulfil some of these preconditions, and are tagged with the tags specified in the domain context. The request parameters are shown in Table 3 and the response parameters are shown in Table 4.

Following is shown an example of usage of this operation considering a canvas with the screen <http://www.morfeoproject.eu/fast/fco#Screen1234552373865>.

```
{
  canvas: [
    {
      uri: "http://www.morfeoproject.eu/fast/fco#Screen1234552373865"
```


Table 3: Find Request Parameters

Name	Description	Type
Canvas	The canvas is composed by a list of screens. Only the URI is needed.	Optional
Domain Context	The domain context contains a list of tags and a user.	Optional
Elements	It is a list of resources, previously stored in the catalogue. For example, the list of screens recommended last execution. Only accepts screens.	Optional

Table 4: Find Response Parameters

Name	Description	Type
N/A	A list of recommended screens is returned.	Required

```
    }  
  ],  
  domainContext: {  
    tags: [],  
    user: null  
  },  
  elements: []  
}
```

After the execution of the recommendation algorithm, the response given by the catalogue is:

```
[{  
  "creationDate": "2009-02-07T09:59:52+0000",  
  "creator": "http://www.ismaelrivera.es/",  
  "description": {"en-gb": "Returns people working in DERI"},  
  "domainContext": {  
    "tags": [],  
    "user": null  
  },  
  "homepage": "http://www.deri.ie/",  
}
```

```

"icon": "http://www.deri.ie/icon.jpg",
"label": {"en-gb": "Screen - DERI"},
"postconditions": ["?person rdf:type foaf:Person .
                  ?person foaf:workplaceHomepage http://www.deri.ie/"],
"preconditions": [],
"rights": "http://creativecommons.org/",
"screenshot": "http://www.deri.ie/screenshot.jpg",
"uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552164472",
"version": "1.0"
}]

```

4.3.4 Check

This operation verifies the state of certain list of screens depending on a specific criterion. The only criterion accepted in this stage of the prototype is 'reachability'. A screen will be reachable if it all its preconditions are satisfied by postconditions of the screens the canvas contain. The specific URL to access this operation is <http://catalogueURL/check> using HTTP POST method.

Table 5: Check Request Parameters

Name	Description	Type
Canvas	The canvas is composed by a list of screens. Only the URI is needed.	Optional
Domain Context	The domain context contains a list of tags and a user.	Optional
Elements	It is a list of resources, previously stored in the catalogue. For example, the list of screens recommended last execution. Only accepts screens.	Optional
Criterion	The criterion specifies what it has to be check. The only possible supported now is 'reachability' in order to check if the preconditions of a screen are satisfied.	Required

The scenario for the following example is composed by a screen in the canvas which precondition is a foaf:Person which foaf:workplaceHomepage is <http://www.deri.ie/>, and a screen in the elements list which precondition is a foaf:Person. The state to check in this case is the reachability

Table 6: Check Response Parameters

Name	Description	Type
Canvas	The canvas indicating which screens satisfy the criterion. More additional information may be included, for example, a satisfaction attribute for every precondition for reachability.	Optional
Domain Context	The domain context contains a list of tags and a user.	Optional
Elements	The elements indicating which satisfy the criterion. More additional information may be included as in the Canvas.	Optional

and satisfaction of the screens and preconditions.

```
{
  canvas: [
    {
      uri: "http://www.morfeoproject.eu/fast/fco#Screen1234552194269"
    }
  ],
  domainContext: [],
  elements: [
    {
      uri: "http://www.morfeoproject.eu/fast/fco#Screen1234552373865"
    }
  ],
  criterion: "reachability"
}
```

As you can see in the response, no screen is reachable at the moment, and no precondition is satisfied.

```
{
  "canvas": [{
    "preconditions": [{
      "expression": "?person rdf:type foaf:Person .
                    ?person foaf:workplaceHomepage http://www.deri.ie/",
```

```
    "satisfied": false
  }],
  "reachability": false,
  "uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552194269"
}],
"elements": [{
  "preconditions": [{
    "expression": "?person rdf:type foaf:Person",
    "satisfied": false
  }],
  "reachability": false,
  "uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552373865"
}]
}
```

4.3.5 GetMetadata

Although the information about a resource can be obtained by its specific retrieval operation supported by the CRUD interface, sometimes is needed to retrieve information which would imply a request per resource, hence this operation allow to get the metadata of a list of resources in only one request. This prototype supports this operation for screenflows and screens. The specific URL to access this operation is <http://catalogueURL/getmetadata> using HTTP POST method. Table 7 shows the parameters needed for the invocation and Table 8 details the different parameters may contain a response.

Table 7: GetMetadata Request Parameters

Name	Description	Type
N/A	A list of URIs.	Required

The following example send two URIs of screens and a URI of a screen-flow to the GetMetadata operation:

Table 8: GetMetadata Response Parameters

Name	Description	Type
Screenflows	A set of screenflows with all the metadata associated to them..	Optional
Screens	A set of screens with all the metadata associated to them.	Optional

```
[  
  "http://www.morfeoproject.eu/fast/fco#Screen1234552164472",  
  "http://www.morfeoproject.eu/fast/fco#Screen1234552194269",  
  "http://www.morfeoproject.eu/fast/fco#ScreenFlow1233752391898"  
]
```

The response obtained are two lists, one containing all the metadata regarding to the screen-flows and another one with the information about the screens:

```
{  
  "screenflows": [{  
    "creationDate": "2009-02-05T13:11:41+0000",  
    "creator": "http://www.ismaelrivera.es/",  
    "description": {"en-gb": "A example of screenflow"},  
    "domainContext": {  
      "tags": ["people"],  
      "user": null  
    },  
    "homepage": "http://www.screenflow.com/",  
    "icon": "http://www.screenflow.com/icon.jpg",  
    "label": {"en-gb": "My First ScreenFlow"},  
    "postconditions": [],  
    "preconditions": [],  
    "rights": "http://creativecommons.org/",  
    "screens": [  
      "http://www.morfeoproject.eu/fast/fco#Screen1233750820914",  
      "http://www.morfeoproject.eu/fast/fco#Screen1233750828836",  
      "http://www.morfeoproject.eu/fast/fco#Screen1233750841383"  
    ]  
  }  
]
```

```
],
"screenshot": "http://www.screenflow.com/screenshot.jpg",
"uri": "http://www.morfeoproject.eu/fast/fco#ScreenFlow1233752391898",
"version": "1.0"
}],
"screens": [
{
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-gb": "Returns people working in DERI"},
  "domainContext": {
    "tags": [],
    "user": null
  },
  "homepage": "http://www.deri.ie/",
  "icon": "http://www.deri.ie/icon.jpg",
  "label": {"en-gb": "Screen - DERI"},
  "postconditions": ["?person rdf:type foaf:Person .
                    ?person foaf:workplaceHomepage http://www.deri.ie/"],
  "preconditions": [],
  "rights": "http://creativecommons.org/",
  "screenshot": "http://www.deri.ie/screenshot.jpg",
  "uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552164472",
  "version": "1.0"
},
{
  "creationDate": "2009-02-07T09:59:52+0000",
  "creator": "http://www.ismaelrivera.es/",
  "description": {"en-gb": "Show information about a person working in DERI"},
  "domainContext": {
    "tags": [],
    "user": null
  },
}
```

```
"homepage": "http://www.deri.ie/",
"icon": "http://www.deri.ie/icon.jpg",
"label": {"en-gb": "Screen - DERI"},
"postconditions": [],
"preconditions": ["?person rdf:type foaf:Person .
                  ?person foaf:workplaceHomepage http://www.deri.ie/"],
"rights": "http://creativecommons.org/",
"screenshot": "http://www.deri.ie/screenshot.jpg",
"uri": "http://www.morfeoproject.eu/fast/fco#Screen1234552194269",
"version": "1.0"
}
]
}
```

4.3.6 Screen Find & Check

Request

```
{
  "canvas": [
    { "uri": "http://purl.oclc.org/fast/ontology/gadget#Screen1257860786707" }
  ],
  "elements": [
  ],
  "domainContext": {
    "tags": [],
    "user": null,
  },
  "criterion": "reachability"
}
```

Response

```
{
```

```
"canvas": [{
  "preconditions": [[{
    "id": "sc",
    "label": {"en-gb": "A search criteria"},
    "pattern": "?F http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://aws.amazon.com/AWSECom
    "satisfied": false,
    "positive": true
  }]],
  "reachability": false,
  "uri": "http://purl.oclc.org/fast/ontology/gadget#Screen1257860786707"
}],
"elements": [{
  "preconditions": [],
  "reachability": true,
  "uri": "http://purl.oclc.org/fast/ontology/gadget#Screen1257860530032"
}]
}
```

4.4 Planning

URL: <CATALOGUE_URL>/planner Request

```
{
  "goal": "http://purl.oclc.org/fast/ontology/gadget#Screen1257860720035",
  "canvas": []
}
```

Response

```
[
  [
    "http://purl.oclc.org/fast/ontology/gadget#Screen1257866875435",
    "http://purl.oclc.org/fast/ontology/gadget#Screen1257865645621"
  ],
]
```



```
[  
  "http://purl.oclc.org/fast/ontology/gadget#Screen1257860530032",  
  "http://purl.oclc.org/fast/ontology/gadget#Screen1257860786707",  
  "http://purl.oclc.org/fast/ontology/gadget#Screen1257860772723"  
]  
]
```

4.5 API Error Codes

There are two types of error codes, client and server.

- Client error codes are generally caused by the client and might be an invalid domain or an invalid request parameter. These errors are accompanied by a 4xx HTTP response code. For example: `ResourceNotFound`.
- Server error codes are generally caused by a server-side issue and should be reported. These errors are accompanied by a 5xx HTTP response code. For example: `ServerUnavailable`.

The following table lists all the error codes.

Table 9: API Error Codes

Error	Description	HTTP Status Code
ResourceNotFound	The resource <resourceURI> has not been found.	404 Not Found
AccessFailure	Access to the resource <resourceName> is denied.	403 Forbidden
InternalServerError	Request could not be executed due to an internal service error.	500 Internal Server Error
InvalidAction	The action <actionName> is not valid for this web service.	404 Bad Request
InvalidHttpRequest	The HTTP request is invalid. Reason: <reason>.	400 Bad Request
InvalidLiteral	Illegal literal in the filter expression.	400 Bad Request
InvalidParameterValue	The specified parameter value is not valid.	400 Bad Request
InvalidURI	The URI <requestURI> is not valid.	400 Bad Request
MissingAction	No action was supplied with this request.	400 Bad Request
MissingParameter	The request must contain the specified missing parameter.	400 Bad Request
NotYetImplemented	Feature <feature> is not yet available.	401 Unauthorized
ServiceUnavailable	The service is currently unavailable. Please try again later.	
UnsupportedHttpVerb	The requested HTTP verb is not supported: <verb>.	400 Bad Request
URITooLong	The URI exceeded the maximum limit of <maxLength>.	400 Bad Request

References

[DoW, 2007] (2007). Fast description of work.

[B.V., 2008] B.V., A. (2008). User guide for sesame 2.2.

[Fielding et al., 1999] Fielding, R. T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext transfer protocol – http/1.1.

[Grant et al., 2008] Grant, K., Feigenbaum, L., and Torres, E. (2008). Sparql protocol for rdf. Recommendation, World Wide Web Consortium (W3C).

[Hayes, 2004] Hayes, P. (2004). Rdf semantics. Recommendation, World Wide Web Consortium (W3C).

[Möller, 2009] Möller, K. (2009). Ontology and conceptual model for the semantic characterisation of complex gadgets. Deliverable 2.2.2, FAST Project (FP7-ICT-2007-1-216048).

[Villoslada, 2009] Villoslada, E. (2009). Fast requirements specification. Deliverable 2.3.2, FAST Project (FP7-ICT-2007-1-216048).

Appendix A. Lists of Tables and Figures

List of Tables

1	CRUD operations for Screens	13
2	CRUD operations for Screen-Flows	17
3	Find Request Parameters.....	20
4	Find Response Parameters.....	20
5	Check Request Parameters	21
6	Check Response Parameters	22
7	GetMetadata Request Parameters.....	23
8	GetMetadata Response Parameters.....	24
9	API Error Codes	29

List of Figures

1	Catalogue's Architecture	7
2	Content Negotiation	12

Appendix B. Generic Building Block JSON Structure

```
{
  "code": "http://url.com/.../code.js",
  "creationDate": "2009-04-20T17:00:00+0100",
  "creator": "http://www.fabio.es/",
  "description": {"en-gb": "This is a description of the building block"},
  "tags": {
    "tags": [
      "ebay",
      "amazon",
    ],
    "user": <String> // not defined yet
  },
  "homepage": "http://www.homepage.ie/",
  "icon": "http://url.com/images/icon.png",
  "id": "Id of the resource",
  "label": {"en-gb": "Label or title of the building block"},
  "name": "Amazon Item to Ebay Filter",
  "rights": "http://creativecommons.org/",
  "screenshot": "http://www.derri.ie/eBayList-screenshot.jpg",
  "type": "type of the building block", // screenflow, screen, form, operator y resource
  "uri": "http://localhost:8080/catalogue/screens/525",
  "version": "1.0"
}
```

Appendix C. Screenflow JSON Structure

```
{  
  <GENERIC BUILDING BLOCK JSON>,  
  "contains": [  
    "http://purl.oclc.org/fast/ontology/gadget#Screen1257860772723",  
    "http://purl.oclc.org/fast/ontology/gadget#Screen1257860720035",  
  ],  
}
```

Appendix D. Screen JSON Structure

```
{
  <GENERIC BUILDING BLOCK JSON>,
  "postconditions": [[{
    "id": "url",
    "label": {"en-gb": "Purchase URL"},
    "pattern": "?P
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      http://aws.amazon.com/AWSECommerceService#PurchaseURL",
    "positive": true
  }]],
  "preconditions": [[{
    "id": "cart",
    "label": {"en-gb": "A shopping cart"},
    "pattern": "?C
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
      http://aws.amazon.com/AWSECommerceService#ShoppingCart",
    "positive": true
  }]],
  "code": "URL of the code",
  "definition": {
    "buildingblocks": [
      { // Form
        "id": "form1", // unique for the container screen
        "uri": "http://purl.oclc.org/fast/ontology/gadget#Form670238"
      },
      { // Operators
        "id": "op1",
        "uri": "http://purl.oclc.org/fast/ontology/gadget#Operator213487"
      },
      { // Backend services
        "id": "bs1",
```

```
    "uri": "http://purl.oclc.org/fast/ontology/gadget#Backendservice38399"
  },
  {...}
]
"pipes": [
  {
    "from": {
      "buildingblock": "bs1",
      "condition": "cA"
    },
    "to": {
      "buildingblock": "op1",
      "condition": "cA", // can be the same id, unique for the building block
      "action": "filter"
    }
  },
  {
    "from": {
      "buildingblock": "form1",
      "name": "refresh"
    },
    "to": {
      "buildingblock": "bs1",
      "action": "list"
    }
  }
]
}
```


Appendix E. Screen Component JSON Structure

```
{
  <GENERIC BUILDING BLOCK JSON>,
  "actions": [{
    "name": "filter",
    "preconditions": [[{
      "id": "item",
      "label": {"en-gb": "Ebay List"},
      "pattern": "?Item
                  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                  http://aws.amazon.com/AWSECommerceService#Item",
      "positive": true
    }]],
    "uses": []
  ]],
  "libraries": [],
  "postconditions": [[{
    "id": "filterEbay",
    "label": {"en-gb": "Ebay List"},
    "pattern": "?eFilter
                  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                  http://developer.ebay.com/.../FindItemsAdvanced.html#Request",
    "positive": true
  }]],
  "triggers": ["itemAmazon"]
}
```