D4.2.2 Data Provisioning and Discovery (Prototype II)

Authors: ISOFT

Delivery Date:	Due Date:	Dissemination Level:
2013-10-01	2013-08-31	PU

This deliverable provides a description of the second prototype implementation of task T4.2. As stated in the Description of Work (DOW), this deliverable is a prototype (software) deliverable. As such, this document is reduced in length and its only purpose is to briefly describe the prototype functionality as well as to give installation instructions and usage clarifications. This document will be shipped together with the software itself.

SEVENTH FRAMEWORK PROGRAMME

Document History	
Draft Version	V0.1, ISOFT, September 23th, 2013 V0.2, ISOFT, September 26th, 2013 V0.3, ISOFT, September 27th, 2013 V1.0, ISOFT, September 30th, 2013 (Final Version)
Contributions	ISOFT - Georgi Pavlov - Irena Pavlova
Internal Review 1	Denise Bowen, TANet, September 27th 2013
Internal Review 2	Ahm Shamsuzzoha, UVA, September 28th 2013
Final Version	September 30th 2013

Table of Contents

List of Figures

List of Tables

Executive Summary

The prototype software (D4.2.2) described in this document is the second and last software deliverable of task T4.2 Data Provisioning and Discovery (DPD). The goal of this task is to perform the necessary research and development to deliver software prototypes, which:

- Introduce a customizable and extensible model for describing ADVENTURE members and their ADVENTURE-related assets (e.g. services) into a common, coherent information space, supporting the other ADVENTURE tools in their information retrieval needs.
- Provide tools for provisioning of new ADVENTURE members information into the system during their join phase.
- Provide tools for discovery of existing ADVENTURE members and their services, matching selection criteria during the play phase.

Prototype D4.2.2 extends the first implementation in terms both of functionality and technology, and provides the full set of features needed for effective ADVENTURE members' information management and partners/services finding and match making, both for human users and for programmatic integration scenarios.
This edition of the deliverable seeks to support end-to-end scenarios between Process Designer and DPD and to reach overall stability and feature completeness that is sufficient to start implementation of the project use cases.

During the implementation phase, a number of technology and technology provider assessments were performed. Many adaptations were developed in order to fit all components together in the technical environment and to overcome any limitations they may have had. The DPD project started delivering value to the open source software community even at this early stage, as a number of these improvements were contributed back in the form of issues reports, fix proposals and even development. Some OSS projects that benefited from that are: JayData, odata4j, Empire and Virtuoso Open Source edition.

This deliverable provides a description of the second prototype implementation of task T4.2. As stated in the Description of Work (DOW), this deliverable is a prototype (software) deliverable. As such, this document is reduced in length and its only purpose is to briefly describe the prototype functionality, as well as to provide installation instructions and usage clarifications. This document will be delivered together with the software itself.

1 Introduction

ADVENTURE – ADaptive Virtual ENTerprise manufacTURing Environment – is a project funded in the Seventh Framework Programme by the European Commission. ADVENTURE creates a framework that enhances the collaboration between suppliers, manufacturers and customers for industrial products and services.

1.1 ADVENTURE Project Aims

The framework proposed by ADVENTURE provides mechanisms and tools that facilitate the creation and operation of manufacturing processes in a modular way. ADVENTURE combines the power of individual factories to achieve complex manufacturing processes. It provides tools for partner-finding, process creation, process optimization, information exchange as well as real-time monitoring combined with the tracking of goods and linking them to Cloud services.

There have already been several research projects that address the combination of different independent manufacturers to so-called virtual factories. Most of these research projects focus primarily on the business-side in general and on aspects like partner-finding and factory-building processes in special. However no proven tools or technologies exist in the market that provide the creation of virtual factories applying end-to-end integrated Information and Communication Technology (ICT). ADVENTURE is aiming to provide such tools and processes that will help to facilitate information exchange between factories and move beyond the boundaries of the individual enterprises involved. The collaborative manufacturing process will be optimized by enabling the integration of factory selection, forecasting, monitoring, and collaboration during runtime.

ADVENTURE builds on concepts and methods of Service-oriented Computing and benefits from the advancements in this field. The monitoring and governance of the collaborative processes will be supported by technologies from the Internet of Things such as wireless sensors. Existing tools and services that can be integrated will be considered during the development of the platform for ADVENTURE.

The increased degree of flexibility provided through ADVENTURE will benefit SMEs especially as it helps them to react quickly to changes and to participate in larger, cross-organizational manufacturing processes. Furthermore, ADVENTURE will help manufacturers in assessing the environmental friendliness of actual manufacturing processes and resulting products and services. Other objectives of ADVENTURE include research in areas such as service-based manufacturing processes, adaptive process management, process compliance, and end-to-end-integration of ICT solutions.

1.2 Deliverable Purpose, Scope and Context

The purpose of this deliverable is to accompany the prototype implementation of task T4.2. As such, its main purpose is to briefly clarify the scope of the prototype and provide instructions for download, installation and use. As the main focus of the task is the development of the software itself, this accompanying document targets more on the supplementary information that is necessary to get familiar with its use in different aspects.

1.3 Document Status

This document is listed in the DOW as PU (Public). It presents the second prototype of the Data Provisioning and Discovery component that is targeted to the development team and to the user partners, but also to external audience as a final result of the Data Provisioning and Discovery task T4.2.

1.4 Target Audience

The first prototype of DPD (D4.2.1) was developed to verify and to validate the requirements, providing a limited set of basic features (even mock-ups), but no end-to-end complete functionality. That is why it was targeted mainly to the user partners and other ADVENTURE components developers. This second prototype of ADVENTURE DPD component has been developed by evolving the developments of the first one and also reflects the results of the evaluation of the initial technological choices and design decisions. Now DPD component comprises a set of fully functional software elements, presenting a common graphical design for ADVENTURE user interfaces, embedding the tools into the Dashboard framework, implementing the interactions between the implemented software components.

Embedding innovative technologies in an efficient way, the component not only supports all the relevant ADVENTURE use cases, but can also be exploited independently in different domains beyond project scope. In this context, though this second prototype is also aimed at project partners, its main target audience is external, including business users, developers and researchers. As a number of valuable contributions have been made to several Open source projects, open source developers' communities are also considered as an important target for dissemination of the final result of the Data Provisioning and Discovery task T4.2.

1.5 Abbreviations and General Terms

A definition of general, common terms and roles related to the realization of ADVENTURE as well as a list of abbreviations is available in the supplementary document "Supplement: Abbreviations and General Terms" which is provided in addition to this deliverable.
Further information can be found at: http://www.fp7-adventure.eu/glossary

1.6 Document Structure

This deliverable is broken down into the following sections:

- **Section 1** provides an introduction to the context, purpose, scope and structure of this document.
- **Section 2** clarifies the context and scope of the software prototype deliverable and its relationship with other architectural modules.
- **Section 3** outlines the requirements and prerequisites for using the software prototype.
- **Section 4** specifies the installation and configuration procedure for the software prototype.
- **Section 5** presents the new functionalities and main improvements implemented in this second prototype.

- **Section 6** details the implemented usage scenarios.
- **Section 7** lists the known limitations and defects and provides an outlook to further planned developments in next iterations.
- **Section 8** briefly summarizes the status of the prototype deliverable.

2 Scope and Relationship

The T4.2 software prototypes are positioned in the overall ADVENTURE system architecture as Data Provisioning and Discovery system component as shown in Figure 1

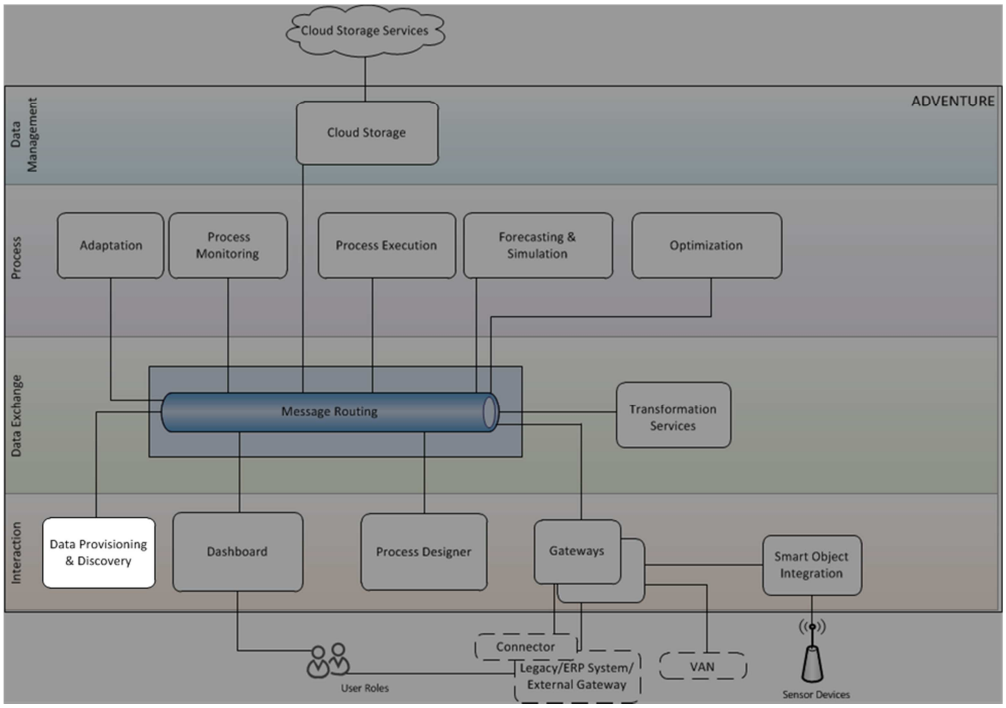

Figure 1: The D4.2.2 prototype into the ADVENTURE architecture

This second prototype complements the developments of the first one, and implements the critical use cases concerning Data Provisioning and Discovery (DPD) to a level that is sufficient to backup end-to-end scenarios necessary for the ADVENTURE components integration and use cases implementation.

The focus of this prototype is:

- The definition and realization of a coherent and comprehensive data model that can support the ADVENTURE scenarios decently, alongside the corresponding operations and system management routines.
- The provisioning of dual technology stack (OData/SPARQL) service interfaces for the core standardized, query operations on the data model supported by DPD as key integration strategy with the rest of the ADVENTURE components.
- The development of a decent user interface to support the direct interaction and manipulation of the DPD data model via standardized service interface.

Similar to the first prototype (D4.2.1), these development goals were developed in a planned, transparent and highly automated production process that included release and integration plan, product and development planning and process with regular feature milestone releases. This disciplined approach helped to distribute development efforts and plan and adapt them adequately with early communication features available to support planned integration process.

The architecture of this prototype edition adds to the planned set of components and refines the initial architecture, optimizing it for the needs of the project. The realized prototype top-level components are:

- Data Explorer

  The Data Explorer component is a coherent JavaScript/HTML/CSS framework and a set of UI components built on top of it. It is designed to realize both the standalone UI scenarios for DPD, as well as UI-level integration scenarios with external agents. Physically, it is realized in the *data-explorer* project.
- OData Service
  The OData Service component realizes a standard OData 2.0 service endpoint for DPD data model management and query for web clients. OData is a standardized protocol for creating and consuming data APIs. OData builds on core protocols like HTTP and commonly accepted methodologies like REST. The result is a uniform way to expose full-featured data APIs. The OData Service component is the primary service-level integration interface provided to DPD clients. Physically, it is realized in the *odata-service* project.
- Data Explorer Service
  The Data Explorer Service component is a REST API that presents a service interface for tailored operations[1] and uses JSON as data exchange format. The exposed operations in this prototype are used for classifications provisioning, used e.g. by the annotation explorer UI subcomponent, and for comprehensive search and matchmaking. The Data explorer Service complements the OData Service interface in the service-level integration scenarios provided to DPD clients. Physically, it is realized in the *data-explorer-service* project.
- Profile Manager
  The Profile Manager component is a flexible, extensible and configurable data mashup Java framework for DPD data model management. It hides the complexity of the integration with external data sources, communication, parallelism and synchronization behind its API. It's extensible by design for more data provisioning components via a defined *data providers* API, and allows multiple configurable data mashup processes. The Profile Manager component can be reused as is in various deployment scenarios as a Java component. In the DPD component its role is to support the Data Explorer Service component. Physically, it is realized in the *profiles-management* project.
- Profile Model
  The Profile Model realizes the Java API representation of the DPD data model and its persistency aspects via JPA. The component also implements a scenario-oriented Java interface that shields clients from persistency technology-specific bulk operations and delegates them to different realizations of this interface. The Profile Model component provides out-of-the-box JPA implementation of this interface. Physically, it is realized in the *dpd-model* project.
- DPD Data Model

[1] While these tailored operations could theoretically be implemented as part of the OData service as Service Operations, some limitations in the technical environment and technical components of choice prevent us from doing so in this release.

The DPD model is a conceptual data abstraction for the entities and their relationships that constitute an ADVENUTRE Member profile and support the operations, which realize the DPD scenarios. It is realized as a Relational Schema and technically managed in the OpenLink Virtuoso Open Source (VOS) server[2] RDBMS, where it is accessible via standard SQL protocol. In addition, the DPD data model contains mappings of the relational model to ontologies and classifications, which allows Virtuoso to expose the same data model managed in its RDBMS as an RDF graph to the web via a standard SPARQL 1.1 service endpoint. The SPARQL endpoint is used internally in DPD to perform partner matchmaking and recommendations via SPARQL. It can be potentially exposed to external clients to exploit more integration options and implement more usage scenarios. All this enables one of the key features in DPD – the dual service technology stack. Physically, the data model is realized in the *dataservices* project.

The DPD logical layering and inter-component dependencies, as well as external dependencies are described on Figure 2.

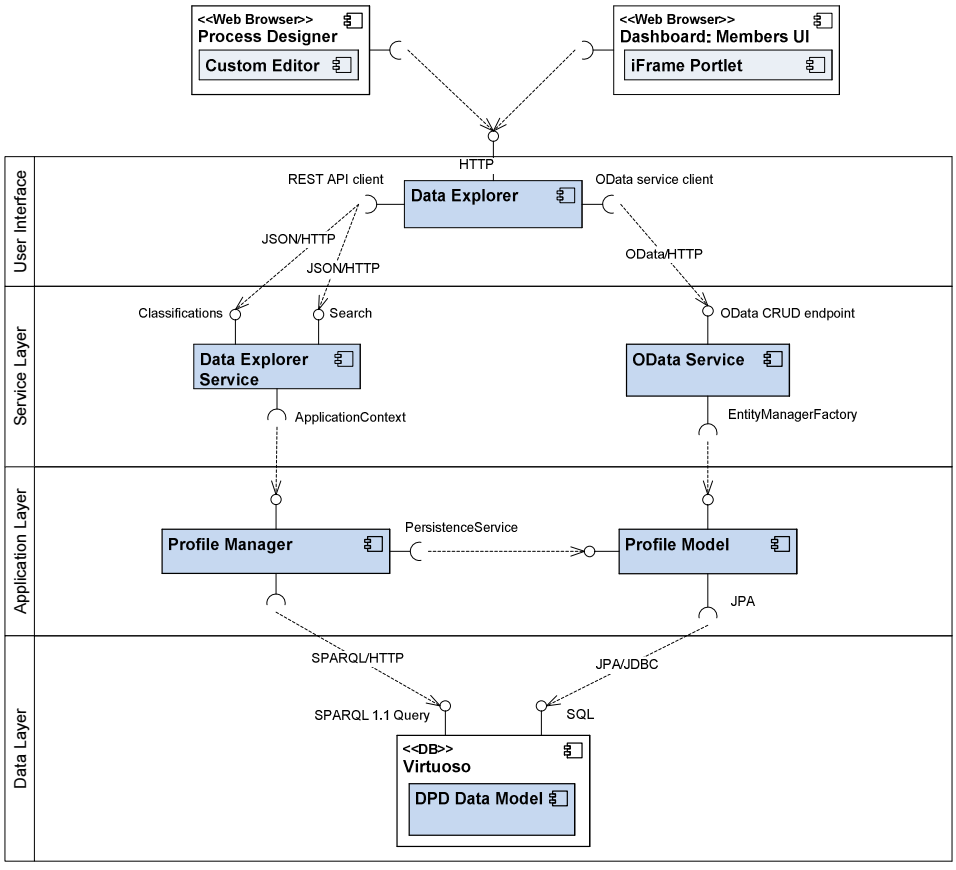

Figure 2: DPD Component architecture overview

[2] OpenLink Virtuoso can manage a coherent information space supported by various data modeling technologies and served by different data access protocols, among which are SPARQL and SQL.

This DPD prototype does not have dependencies on other ADVENTURE components. It is different in that from the previous release, which had dependency on the Cloud Storage and consequently on the Messaging components, settling the pattern for data consumption in DPD from the Cloud Storage. This change was driven by the change in the technologies for data modeling, which was necessary in order to realize the planned scenarios. Should this prove necessary and useful, future integration efforts will be performed to realize the data layer by means of the Cloud Storage component.

The ADVENTURE components that integrate and depend directly on DPD are:

- Process Designer (D5.1.2 Prototype II)

  Several custom editors in D5.1.2 use the UI components from the Data Explorer to query the DPD data model and feed it into the designed process models. These are: Company Services search (uses the Search sub-component), Process Model/Task Semantic Characterization Editor (uses the Annotation Editor sub-component) and Non-Functional properties editor (uses the NFPs Explorer sub-component). Those UI sub-components are consumed via their JavaScript API, which hides all the complexity around the DPD service façade consumption, and adapted by an integration code to the custom editors' framework in the Process Designer. The use of this DPD-based functionality is further specified in the D5.1.2 deliverable.

- ADVENTURE Dashboard (D6.3.1 Prototype I)
  The Dashboard has a dedicated space for exploring ADVENTURE Members. It integrates the whole standalone Data Explorer site that provides for browsing, searching and managing the members' profiles.
  In addition, it also integrates with DPD in the process of registering new users. The registration form is enriched with the minimal required data to create a new member profile and a request to the DPD OData Service is sent accordingly on form submission.
  Finally, the Dashboard's overview page (Dashboard) has widgets that consume data from DPD.

Another prototype that depends on D4.2.2 is the Optimization component. The Optimization component consumes the input of partner recommendations provided in the Process Designer's models by DPD (on earlier stage) along with non-functional requirements (constraints and preferences) and optimization objectives (e.g. minimize total cost). The dependency is thus on the data provided by DPD in the process model.
Similar to that, the Smart Process Execution Engine relies on Gateway endpoints, which were provided again by the DPD's company search functionality on earlier stage.

3 Requirements and Preparations

The diagram in Figure 3 illustrates the supporting infrastructure with the key application components and artifacts deployed, along with the protocols that realize the inter-component communication. It helps to get a better understanding of the target system and its environment.

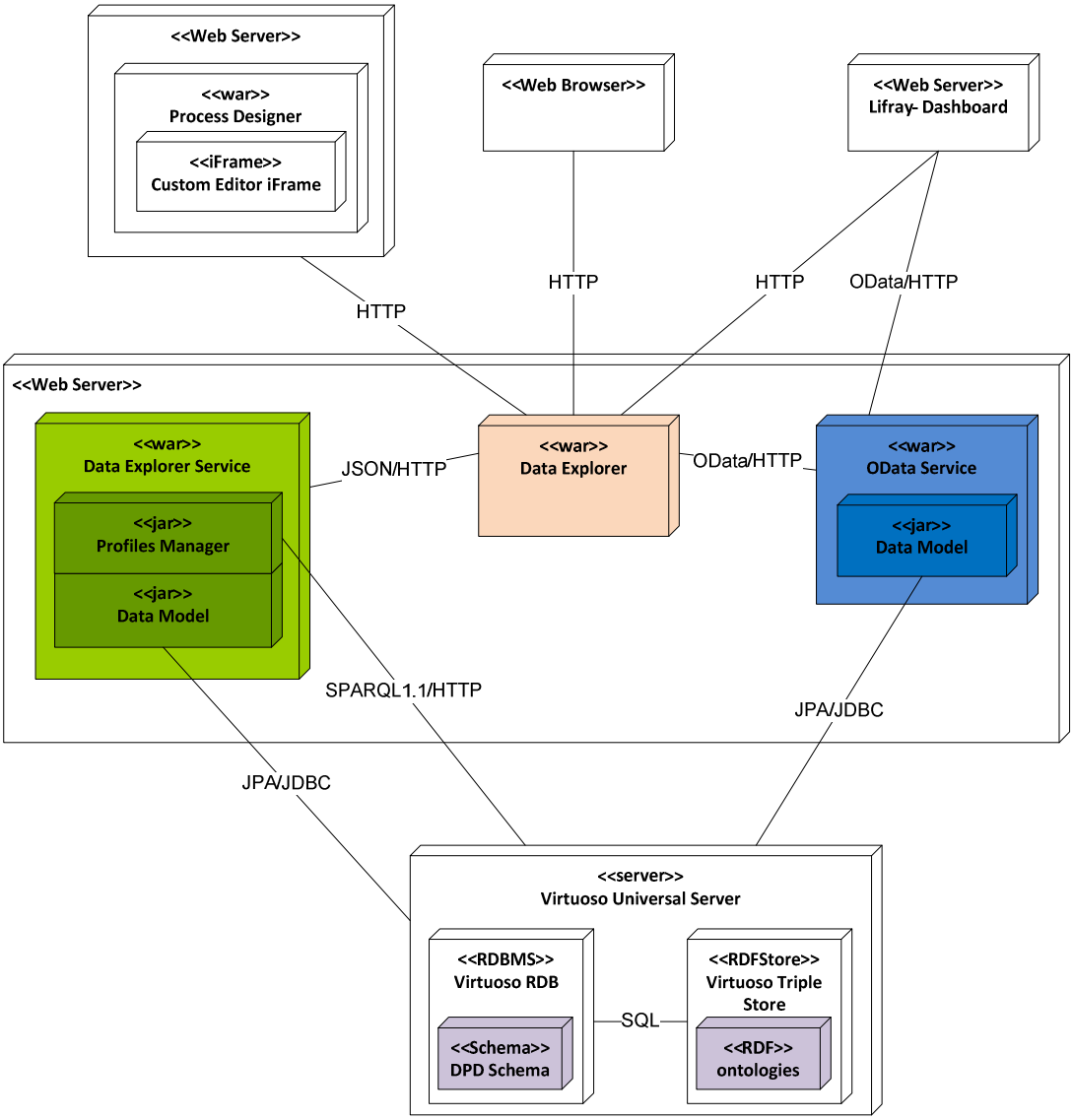

Figure 3: D4.2.2 deployment diagram

The DPD system prototype is deployed on a Java web server. The web server's role is to host and provide online access to the three key DPD components that interface clients – Data Explorer, OData Service and Data Explorer Service, all of which are provisioned as standard Java EE web applications. The communication protocols supported by each are based on HTTP. They could also be deployed as standalone Wweb application, each on its own Wweb server and will still work as a coherent system. However, in such deployment model one should take care for potential problems with the configu-

ration of the environment in such a distributed system (e.g. to avoid issues with cross-domain request).

As described on the diagram in Figure 3, the two services in DPD – the OData Service and the Data Explorer Service are deployed with their corresponding DPD component dependencies packaged with them.

Part of the whole system is also the data server. Its role is to provide a RDBMS and SPARQL (RDFStore) service .This role is realized by the OpenLink Virtuoso (Open Source edition) that provides both features. The RDBMS hosts the DPD data model schema and by means of mapping scripts and additional ontologies the RDFStore exposes this model as RDF graph via a standard SPARQL service endpoint. Consequently, the RDBMS is accessible to remote and local clients via Virtuoso's JDBC driver (packaged together with its using Java application – Data Model). And the SPARQL endpoint is accessible to any standard HTTP client that can send SPARQL queries as a part of the requests payload and according to the protocol. The Virtuoso data server can be deployed either together with the DPD system web server or on a dedicated host for high availability scenarios.

The traditional client to UI web applications is the web browser. The UI of the prototype has been developed primarily for, tested and confirmed to work with Google Chrome. The services can be consumed with any standard HTTP client that implements the corresponding protocol.

A more detailed view on the system requirements and preparations, distinguished by key stakeholder roles is presented in the next sections.

3.1 For Users

No particular technical skills or expertise are required from end users in order to work with the DPD user interface. What is required is a good understanding of the knowledge domain (classification, ontologies and taxonomies) used to describe the ADVENTURE members and their collaboration assets.

Concerning the technical environment for end users, it is required that they have a web browser with JavaScript enabled and Web access to the DPD application in order to use it. The prototype has been confirmed to work with Google Chrome. No further preparation is required in this edition.

3.2 For Developers

3.2.1 Contributing to DPD Prototype Development

Developers that are concerned with the DPD prototype development require the following development environment:

- Eclipse 3.6 (Indigo) or later (http://www.eclipse.org) with the following features installed:
    - Java and JavaScript development profiles. There are pre-packaged Eclipse downloads for either of these.
    - SVN Team provider (http://www.eclipse.org/subversive/) for using the remote source code management system. Configure the team provider to use the

ADVENTURE SVN repository for DPD development branch: http://fp7-adventure.eu/svn/repos/dpd/trunk/
    - m2eclipse (http://eclipse.org/m2e/) for integration with Maven (http://maven.org) build system.
- Java SE 7
- Virtuoso Open Source edition
  The new addition to the technical environment is the OpenLink Virtuoso server release 6.3 or newer from the 6th generation[3] (http://virtuoso.openlinksw.com).
- Tomcat 7

The project wiki (http://www.fp7-adventure.eu/wiki-neu/) contains a detailed and up-to-date description of the development environment and process.

3.2.2 Integrating the DPD Prototype

This edition of the prototype provides several different integration options. One can now choose tight coupling by means of the Profile Manager's Java API, or a loose, service-based coupling by means of the service façade of the DPD that comprises the OData Service and the Data Explorer Service.
The Data Explorer UI component was re-written as a client to the service façade and can be used as an example of how to consume the DPD services.
The Data Explorer Service uses the Java API (Profile Manager) and is an example how to consume and integrate with it.
The DPD projects use Maven for their dependency management. Other applications may use the same approach to automate the integration with different DPD components. Currently, the Maven artifacts are not published in any public repository. As the project matures, this will be considered as well.

3.3 For System Administrators

This section provides an overview from a system administration point of view, which concerns the activities around the required environment deployment and configuration.

Since the components interact with external agents via web interfaces it is required to have a Tomcat 7+ web server installed and configured for use on the system. Consider the deployment model in Figure 3 for options. The DPD system has been confirmed to work with Tomcat 7 and Jetty 6 or newer. In theory, any Java 6 EE compliant web server should fit the purpose.

DPD's Java components and some of its dependencies require Java 7 to be installed on the host system.

The data model managed by the module is stored in Virtuoso Open Source 6.3+. Use the default installation settings.

Use the setup instructions and configuration from the *datasources* project for Windows or Linux as appropriate. Once finished with the installation, execute the schema creation scripts via the Virtuoso SQL interface utilities (iSQL or the Conductor web interface) and then the sample data population script (if wanted). Finally, make sure to copy the required ontologies in a readable for Virtuoso directory (see the configuration file), and

[3] The 7th generation of Virtuoso is a major rewrite and is not yet test and confirmed to work with DPD.

execute the mapping script (iSQL or the Conductor web interface). Finally, test the set-up with the sample SQL and SPARQL queries. All these files and scripts are located in the *datasources* project.

There are no particular minimum hardware requirements. In practice, any decent server hardware should be fairly sufficient since the application resource consumption is modest. For high availability and scalability scenarios, consult with the Virtuoso and Tomcat documentation for instructions on how to size, setup and configure.

With regards to OS compatibility, the DPD system has been tested and confirmed to work both on Windows 7 and Linux (Ubuntu).

4 Installation (Deployment)

The release process for the DPD prototype is designed for automated, continuous integration of changes from the source code repository, through a test suite quality gate and builds, to the final live, running application. The automated infrastructure that supports this process consists of a Source Code Management (SCM) system, which is used by a Continuous Integration (CI) system to build and publish deployable artifacts and finally deploy on a hosting server. The access details for the different components of the automated system are listed in Table 1. For an up-to-date reference, use the project wiki: http://www.fp7-adventure.eu/wiki-neu/.

Table 1: Access details for the release infrastructure

Artifact	Location
SCM	http://fp7-adventure.eu/svn/repos/dpd/branches/releases/M24/trunk
CI	http://fp7-adventure.eu:8080/jenkins/job/M24-DPD/
Deploy artifacts	http://fp7-adventure.eu:8080/jenkins/job/M24-DPD/ws : • /data-explorer-service/target/data-explorer-service-2.0.0.war • /data-explorer/target/data-explorer-2.0.0.war • /odata-service/target/data-explorer-2.0.0.war
Hosted application	http://fp7-adventure.eu:8080/dpd-explorer/

These components support the development and release process for DPD prototypes. It is not necessary to duplicate this system to achieve this, especially if it's only required to build the source code and deploy the results locally. For such purposes, the following procedure is recommended.
Note: The procedure is designed to be comprehensive. If some step is not needed, it may be skipped. For example instead of starting from source code download and build one can simply download the deployable artifacts from the location specified above and directly proceed to deploying them locally.

4.1 Source Code Download

The source code of the D4.2.1 prototype is available online in a SVN SCM system repository at http://fp7-adventure.eu/svn/repos/dpd/branches/releases/M12/trunk for browsing. It can be checked out with the following SVN command:

```
svn checkout http://fp7-
adventure.eu/svn/repos/dpd/branches/releases/M24/trunk dpd-prototype .
```

Alternatively, any suitable IDE (e.g. Eclipse) or SVN client can be used to achieve that. A list of known SVN-supporting client tools is available at the project site:
The recommended approach is to use Eclipse (www.eclipse.org) 3.6 or later, with m2eclipse Maven integration (http://eclipse.org/m2e/) and SVN team provider (http://www.eclipse.org/subversive/) features installed and follow the instructions for the "Checkout source" step in the development process outlined in section 3.2.1. The expected result is that projects are setup and made available in the Eclipse workspace.

4.2 Building with Maven

The source code of the prototype is managed by Maven and developers can take full advantage of the dependency management, process automation and transparency that Maven offers. The system where the build will be performed requires a Maven installation. The build procedure has been confirmed to work with Maven 3.0+. The command to start a Maven build on the downloaded source from the project is: `mvn clean install` and it has to be invoked on the *dpd* project. That shall finally produce one .war file in each of the projects, which is a format specific to the Java EE standard for web applications, ready to be deployed on a Java EE compatible web server. Alternatively, using Eclipse with m2eclipse support, right click the *dpd* project and setup a Maven run configuration with the goals: `clean package`. Then run it. This shall generate /target folders in the projects with the .war files inside.

4.3 Deploying to Web Server

The files required for the deployment operation can be obtained from the previous step or downloaded directly from the project CI system (see Deploy Artifacts row in Table 1). Additionally each of the built maven projects can be deployed and run in a lightweight web container for test purposes using the `mvn jetty:run` (or `mvn tomcat:run`) command on it. A productive installation is usually however performed on an already existing Web server. Each Web server has its own deployment procedure. The prototype has been tested and confirmed to work with Tomcat 7 Web server (www.tomcat.org). For Tomcat, the deployable .war files have to be copied to the server's /webapps directory. Once the server is started it will unpack the archives into directories with the same name and the applications will be served under this application context also. A comprehensive documentation on the deployment operations for Tomcat can be found online: http://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html. Table 2 illustrates the mapping between DPD deployable components, deploy unit and access URLs path pattern:

Table 2: Components, deploy units and URL path mapping

Component name	Deploy unit	URL Path
DPD Explorer	dpd-explorer.war	/dpd-explorer
DPD OData service	dpd-odata-service.war	/dpd-odata-service
DPD Explorer Service	dpd-explorer-service.war	/dpd-explorer-service

5 What's New in This Prototype

This prototype has matured since its early development in the previous edition. After an elaborate and careful evaluation of a number of technologies and technology providers, as well as the initial architecture, it refined it to a design and selection of technology providers that fit together and is achievable with the provided resources and timeframes. During this process, a considerable set of new features and components were developed.

A positive effect of the development process was that not only did the DPD project consume open source contributions, but also contributed back to the open source community issues reports, fix proposals and even development. The projects that benefitted from that are:

- JayData (jaydata.org): a JavaScript framework that abstracts the application model and its persistence and lifecycle.
- odata4j (odata4j.org): a Java framework for OData service provisioning and clients
- Empire (https://github.com/mhgrove/Empire) : provides a standard JPA style interface to RDF databases using SPARQL
- Virtuoso: a middleware and database engine hybrid that combines the functionality of a traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system

The DPD project effectively provided a JPA support for the Virtuoso database engine to the EclipseLink (www.eclipse.org/eclipselink/) JPA Provider, making it possible to design a JPA Java model managed by the Virtuoso database. This is a brand new opportunity that was not offered freely to the date of its development.

In addition, we made the necessary integration with Maven for automatic provisioning of the Virtuoso database JDBC drivers as maven artifacts.

The third contribution to the Open Source edition of the Virtuoso server is the script for lifecycle management (start/stop/restart) of the Virtuoso service on Linux OS (Ubuntu distribution). It's an adapted version of an earlier release for the 5th generation that was no longer functional.

ISOFT is in discussion with the Virtuoso organization for the best way and terms to contribute these developments for the benefit of the OSS community.

5.1 Data Model

One of the key developments in this prototype is the comprehensive data model that supports the ADVENTURE use cases (and beyond) and supports the dual service technology stack in DPD. The data model was developed based on the feedback received until the M12 development and according to a defined methodology.

The methodology (Figure 4) was developed in order to ensure the delivery of a coherent data model across the different data modeling technologies that are employed – RDB and RDF in a controlled and predictable manner.

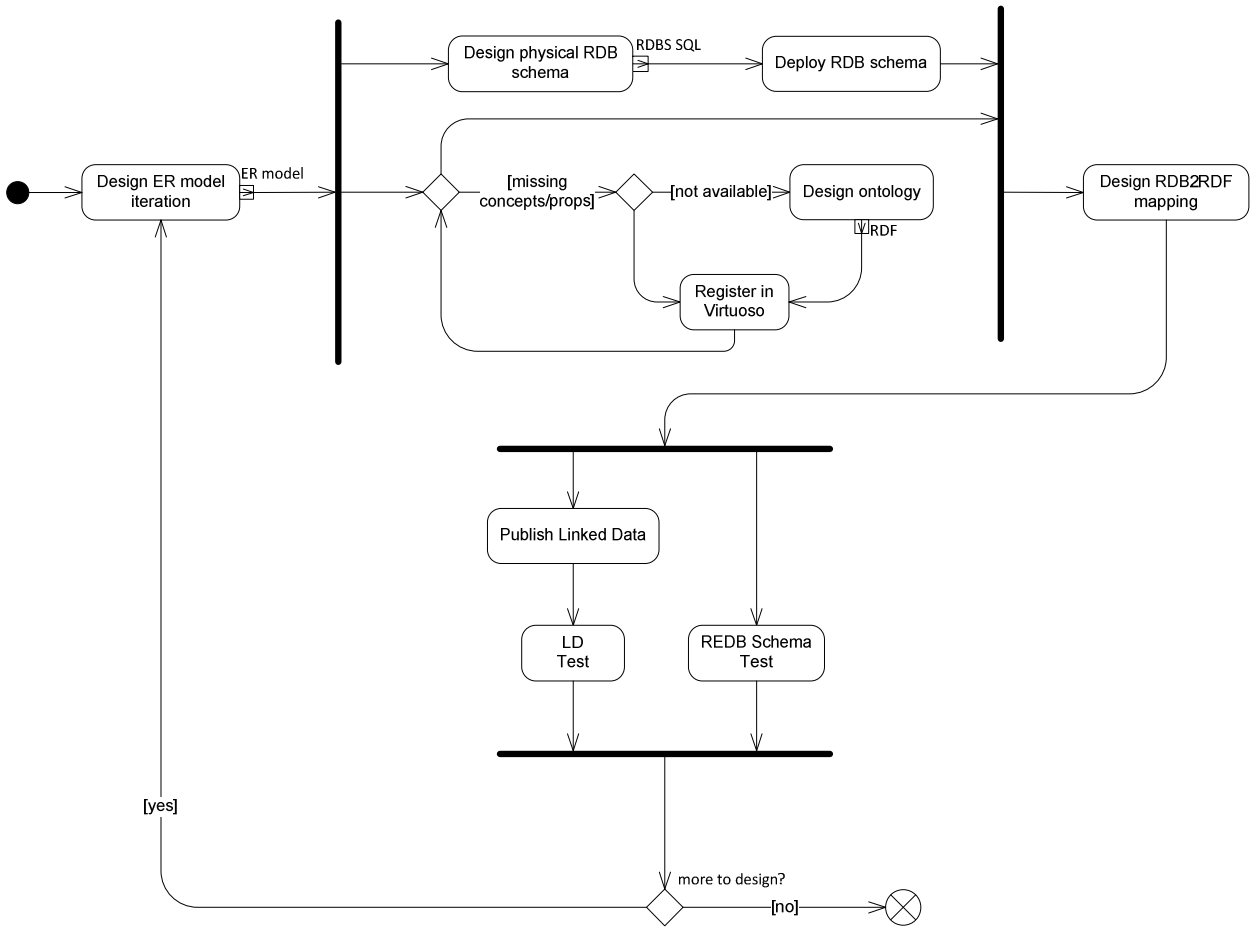

Figure 4: Data model development methodology

In summary, the main phases in the methodology are design conceptual model, physical design and deployment, mapping model schemas, publish and test.

The results of the development are an RDB schema created from the conceptual model (and all corresponding scripts to create, drop, populate with sample data, clear and test), the mapping script between the schema and the RDF view on it, alongside with the corresponding ontologies.

5.2 Profile Model: Java Binding to Data Model

The Profile Model component realizes the Java object model representation of the relational data model. The model is bound to the underlying database schema by means of the Java Persistency API. It also provides a Java interface that comprises common model operations and shields users from the persistency technology details.

5.3 Profile Manager: Java API for Profile Management

The Java API for Profile Management is realized by the profile-manager project. It uses the Java object model to realize application scenarios for handling the main DPD use cases and presents an API façade to shield from the technical details of persistency, and communication. Another major feature of the API is that it's designed to be highly extensible with regards to new data sources and highly flexible in terms of configuration.

5.4 OData Service

The OData Service is a brand new addition to the DPD system landscape and is the main part of the DPD service façade. It can be used by any standard OData v2.0 compliant client to browse, filter, group and perform some projections on the DPD data model as defined by the OData specification.

The technical implementation is built on top of odata4j (http://odata4j.org/), based on the Jersey JAX-RS REST server (https://jersey.java.net/), and uses the JPA Java object model defined in *the Profile Model* component to transform it to a OData data model and communicate with the backend (the Virtuoso server via its JDBC driver).

5.5 Data Explorer Service: API changes

The Data Explorer Service was initially introduced in the previous prototype and it was in the role of service provider for the CRUD operations for data provisioning and discovery of company profiles. Since these early days the prototype evolved significantly and the role of the service changed. Now, its purpose is to complement the OData Service with non-standard and utility operations, where the technology provider (odata4j) has limitations to support the specification. It was refactored to no longer support the proprietary CRUD REST operations with their JSON input/output format and the prototype shifted completely to the standardized protocol and data exchange format of OData.

The non-standard operations that the Data Explorer Service currently exposes are for contextual provisioning of classifications/ontologies and a tailored partner matchmaking operation. Should a necessity for more specific operations arise, they will be added to its interface.

5.6 Data Explorer Service: Classifications Endpoint

The Data Explorer Service exposes a REST API to provide contextual classifications or classification by identity. It is intended to be used by the Annotation Explorer from DPD but it's designed for general purpose and can be utilized by similar tools too.

The REST API for the Classifications service endpoint is defined in Table 3.

Table 3: Classifications endpoint in Data Explorer Service. REST API documentation

URL template	Description

/classifications	Lists all registered classifications, regardless of the context they are assigned to, formatted as a JSON array of JSON objects.
	Note: As classifications can have considerable size, this can be slow and enormous in data size. You are more likely to use one of the next operations for more specific (and faster/slimmer) results
	Example response:
	`[{`
	`"header": {`
	... metadata about the classification…
	`},`
	`"content": {`

	... JSON-LD formatted classification/ontology } }, ... more classifications in the same format…]

/classifications/{classification-id}	Returns a JSON formatted classification, identified by the {classification-id} template.
	Example response:
	{
	"header": {
	... metadata about the classification…
	},
	"content": {
	... JSON-LD formatted classification/ontology
	}
	}

/classifications/contexts/	Lists all registered contexts names as a JSON array of strings. The standard context names are process, profile, resource and service. Example response: ["process","profile","resource","service"]

/classifications/contexts/{context-name}	Returns the classifications associated with the context identified by the {classification-name} template as JSON array of JSON objects. The {classification-name} has to be a valid identifier and one of those listed by the /classifications/contexts/ operation.
	Example response:
	[{
	"header": {
	... metadata about the classification…
	},
	"content": {
	... JSON-LD formatted classification/ontology
	}
	},
	... more classifications in the same format…
	]

The classifications provided by the service are formatted as JSON objects. Generally the object layout is:

{
"header": {
... metadata about the classification…
},
"content": {
... JSON-LD formatted classification/ontology
}
}

The two main properties as listed above are:

- **header**
  The *header* property is a JSON object that provides metadata about the classification that is used by the consuming tools that want to show information about it. The properties that constitute this metadata described in Table 4.

Table 4: header object properties

Property name	Description
url	The original URL of the classification.
label	A user-friendly label.
version	The classification version.
description	An annotation describing the classification.
contentFormat	The content format of the classification provided in the *content* property (see below). Currently, the only one specified is “json-ld”.

- **content**
  The *content* property is JSON object that is actually the JSON-LD (http://www.w3.org/TR/json-ld/) standard formatted classification/ontology itself.

Some known limitations of the current implementation are:

- No configuration. Everything is pretty much hardcoded. That includes for example the mappings between contexts and classifications, the mappings between classifications and their identities, and the context names.
- The classifications are read into memory, which can cause problems for many and/or big classifications
- The service is read-only. No modifications are implemented via its interface.

5.7 Data Explorer Service: Profiles Endpoint Match Operation

The Data Explorer Service exposes a *match* operation now. Its purpose is to perform a matchmaking of partner profiles to a provided set of constraints and requirements. It is used in the Data Explorer UI and the Process Designer to search for partners.

Currently, it matches by service properties and classification.

The input specification to the operation is defined in Table 5.

Table 5: match operation in JSON input specification

```
[
{property-name}*: {literal | expression},
classification: [{literal}*]
]
```

Note that, if you provide an expression it has to be a valid SPRAQL expression (which in its majority means a valid XSD expression). Otherwise the alternative literal value is matched exactly.

The classification literals can be more than one and are matched exactly in this version.

All constraints specified by the input specification are considered non-optional.

The return value is a list of zero or more service ids that match the query criteria encoded in the input specification.

5.8 SPARQL Service Endpoint

The SPARQL service endpoint is another new addition to the services landscape in DPD. It's a feature provided by the Virtuoso server and complies with the standard SPARQL 1.1 Protocol specification. In this prototype edition, the partner profiles are exposed as RDF as a product of the mapping performed between the relational schema of the DPD model and standard, external and ADVENTURE-specific ontologies. The SPARQL service endpoint allows remote, web-based querying of the RDF view of the data model. Generally, it was designed to be deployed as a backend service, similar to the SQL interface to the data model in Virtuoso. As such, it is consumed by the match Data Explorer Service REST API operation that provides a tailored interface to its clients on its turn. However, it's possible to deploy DPD, so that the SPARQL service is exposed to external clients too for arbitrary queries and integration scenarios based on the SPARQL protocol. Further, it is possible to publish the RDF as Linked Data as well.

5.9 Data Explorer: UI Componentization and Framework

The Data Explorer underwent a serious refactoring and componentization in this edition. While on the surface, things changed slightly, under the hood there was a significant refactoring and new development. As a result now there are no mocked UI elements, and the UI features almost all of its planned features and UI. Following, is a list of some of the notable advances:

- Lists widgets UI component framework

  All lists and grids in the Data Explorer UI extend the same Lists UI component framework that ensures consistent behavior, look and feel, as well as significantly minimizes redundant and boilerplate code.
- Models, Presenters and Views
  The code splits the logic between Presenter and View objects that are tightly coupled in their roles and work on a common Model objects. Generally Presenters prepare models for consumption by Views and handle all events in the UI and the View lays out the UI and populates it using its API.
- Templates: A templating framework (Handlebars.js) was introduced to allow easier manipulation of the Views layout. It's mostly declarative now and can be customized significantly easier.
- Databinding: A databinding mechanism was developed in order to synchronize the Model with the View/Presenter state and consequently update the backend.
- JayData Entity model abstraction: The models were all implemented using JayData's abstraction, to take full advantage of their transparent support for various backend data services that manage the model (in this case – OData).

5.10 Data Explorer: Annotations Explorer

The Annotations explorer is a UI component that is employed both in the Data Explorer and the Process Designer. Its use case is to provide a simplified user interface that allows browsing ontologies and select concepts from them. It offers a JavaScript API that allows pre-selecting a concept or notifying for a concept selection to using programs. Using this API, it can be embedded in dialogs (provided out-of-the-box by the API) or other interactive UI to view or change an annotation.

Key features are:

- Type-ahead assistance
  A key design decision for the component was to avoid traditional exploration paradigms for ontologies and instead stick to common web 2.0 interaction patterns. One of the most popular patterns with regards to controlled input is the so called 'type-ahead input' (a.k.a. suggestion box). The Annotation Explorer combines several nuances of this and triggers a suggestion drop-down list populated with matching, concepts as the user types the first several letters of the concept that is desired. The assumption here is that similar to tagging, the user has some awareness for what it is trying to achieve. The tree view of the ontology is still available as last resort for free-style browsing.
- Tree view
  The tree view is opened on demand and is there as a second help to the users to realize the hierarchical path to the selected concept in the case of hierarchical ontologies (with parent-child relationships between concepts)
- Linked properties
  The properties of a selected concept are dynamically populated. Their names and values are provided as is from the ontology. The properties are rendered as links whenever possible. The user can navigate to the specified web location and learn more about the property.
- Pre-select
  The component features pre-selection. When provided with a valid concept identity it will search and pre-select it. This feature is used to view already defined annotations.
- Ready to use dialog mode
  The component features a simplified API that attaches a jQuery dialog to a specified document element. The open and close dialog events are readily bound to the pre-select and the select operations accordingly.
- Edit mode sensitive
  The component can be opened in read-only mode, meaning it will act as a viewer without ability to change the selection. Alternatively, as an editor it allows concept selection.

5.11 Data Explorer: Non-Functional Properties UI Component

The non-functional properties are designed as a reusable component extending the Entity lists framework. When in edit mode, the list features toolbar for change actions – create, update and delete. The create/update actions trigger a specialized editor that uses the ADVENTURE Non-Functional Properties ontology. Its key feature is the type-ahead assistance that allows users to select properties based on suggestions from a

drop-down list (populated from that ontology) as they type the property name (Figure 19).

5.12 Data Explorer: Search

The search view is a new addition to the user interface and use cases of Data Explorer. In this edition, it allows to specify non-functional properties constraints and classification requirements for a service (Figure 8). The search component works with the Data Explorer Service's match operation to provide a filtered companies list UI component.

It's designed as a reusable component and is employed in the Process Designer as well.

5.13 Data Explorer: New Backend Service

One of the biggest changes in the Data Explorer is that it switched from using the Data Explorer Service REST API to the OData Service API to realize the members' profiles browsing and management use cases.

The Data Explorer now uses JayData (jaydata.org) as a data model and service abstraction to communicate with the DPD OData Service.

6 Execution and Usage

This prototype provides various usage scenarios, depending on whether it is addressed as a service, a Java API or as hosted UI. The service and Java API usage scenarios obviously address programmatic usage targeting integration with external software agents, and the UI usage scenario targets human users as a normal web application would. The service usage scenarios also differ by the choice of service technology as DPD offers a dual service stack – OData and SPARQL. The UI usage scenario can also be divided into two options – standalone and integrated in the ADVENTURE Dash-board.
This section will list the implemented use cases, defined in the Functional Specification, and present the User Interface and Service APIs that do that. Considering the briefing nature of this document it is not a complete specification but rather an overview guide.

6.1 Data discovery

6.1.1 Use Cases

- DD-U02: View profile details
- DD-U03: Find ADVENTURE partners
- DD-U04: Find ADVENTURE services
- DD-U05: Find service for activity (implemented with Process Designer)
- DD-U06: Browse service details
- DD-U07: Find service static data
- DD-U09: Search by QoS criteria
- DD-U10: Find process models (implemented in Process Designer)
- DD-U11: Find process model templates (implemented in Process Designer)
- DD-U12: Browse process details (implemented in Process Designer)
- DP-U07: Load Profile

6.1.2 User Interface

The standalone user interface presented by the Data Explorer (Figure 5) consists of two main areas that are loosely coupled UI components integrated by an events-based mechanism. It is all organized around the Master/Detail pattern. The upper area is the ‘master’ list. The lower area is the ‘details’.

Figure 5: Data Explorer standalone UI

The master list view presents a selectable list of registered organizations with a few key attributes (Figure 6). The selection renders the profile view. The list supports paging, and navigation back and forth, as well as direct navigation to a page.

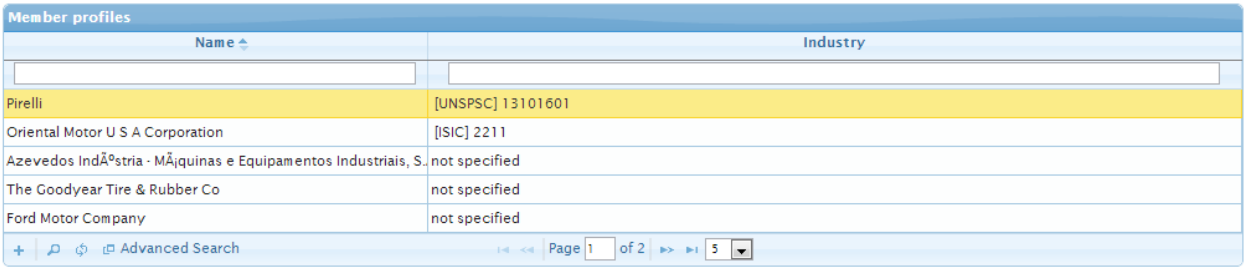

Figure 6: Member profiles interface

The list can be filtered, sorted and a local search on the list content can be performed (Figure 7), using the toolbar and control of the list UI widget.

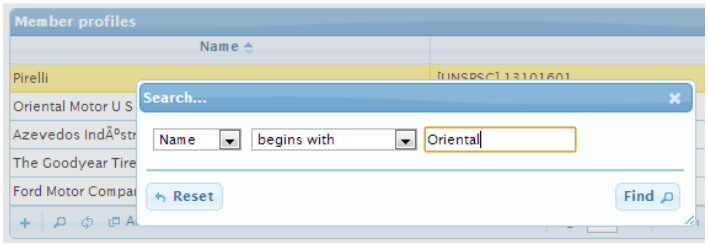

Figure 7: Search the list by property

All operations described so far are on the local data received from the service. They do not involve communication with the backend services. A new search request to the backend services can be triggered by means of the *Advanced Search* function (Figure 8) in the list widget toolbar. It will trigger a search for a partner that has a service matching the provided specification. The specification currently constitutes non-functional properties (specified as exact value or XSD expression) and classification.

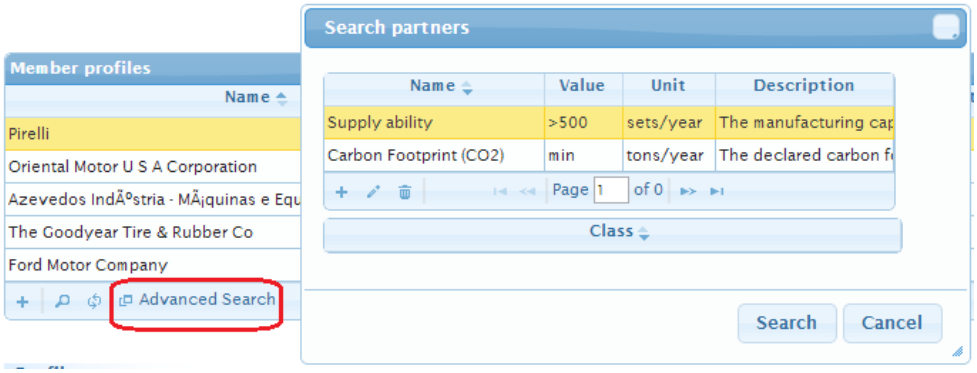

Figure 8: Search partner by service specification

A selection on a line from the profiles list broadcasts a browser event that triggers the load of the selected profile details in the details section.

The details section is split into several tabs. Each more complex view is governed by a template declaratively and can be relatively easily configured for a different layout.

The tab '*Organization'* lists the selected profile static, organizational data.

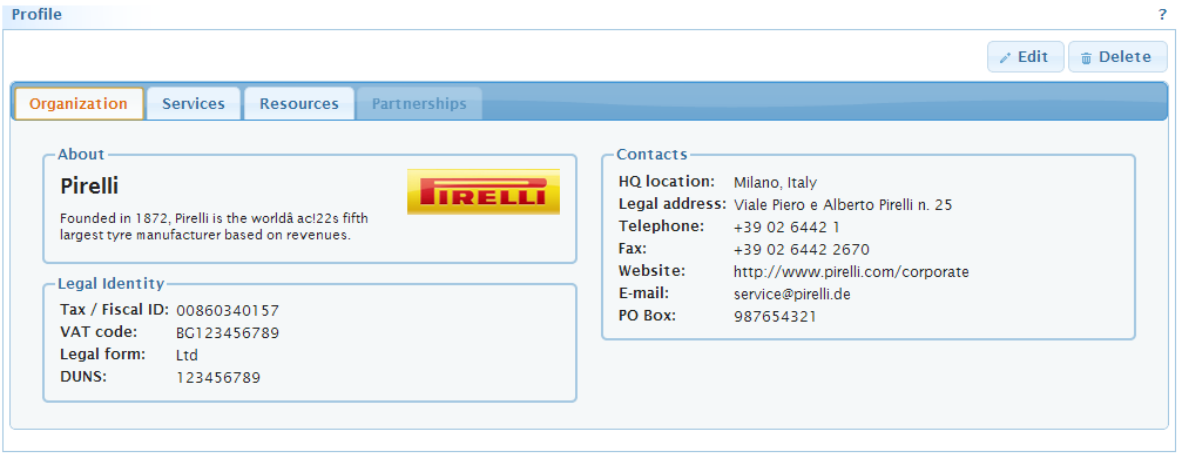

Figure 9: Member profile Organization details

The tab '*Services'* lists the services associated with this Member and their details again as per the Master/Detail pattern (Figure 10). The list widget supports navigation by paging, sorting by property and local search. Selecting a service entry in the list triggers the load of its details in the lower details area.

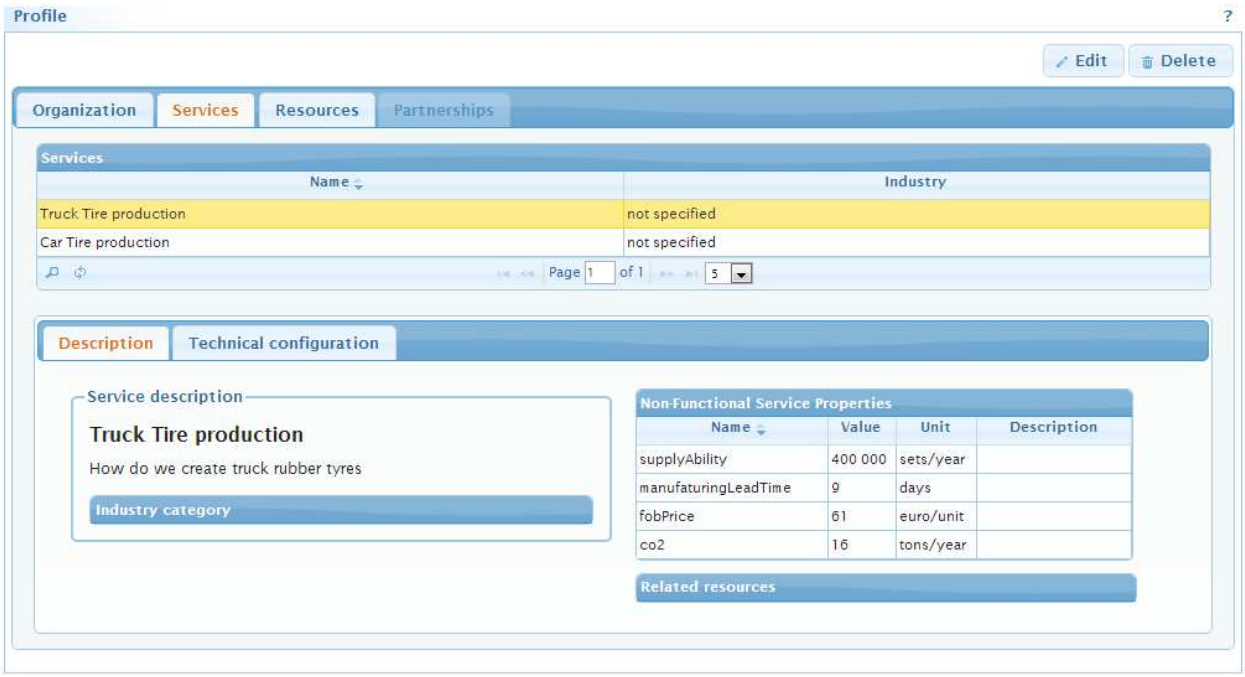

Figure 10: Member profile Service details

The Service details area is split into several tabs again (Figure 10).

The '*Description'* tab lays out business information about the service (Figure 10).

The '*Technical configuration*' tab lists the technical details about the Gateway associated with this service.

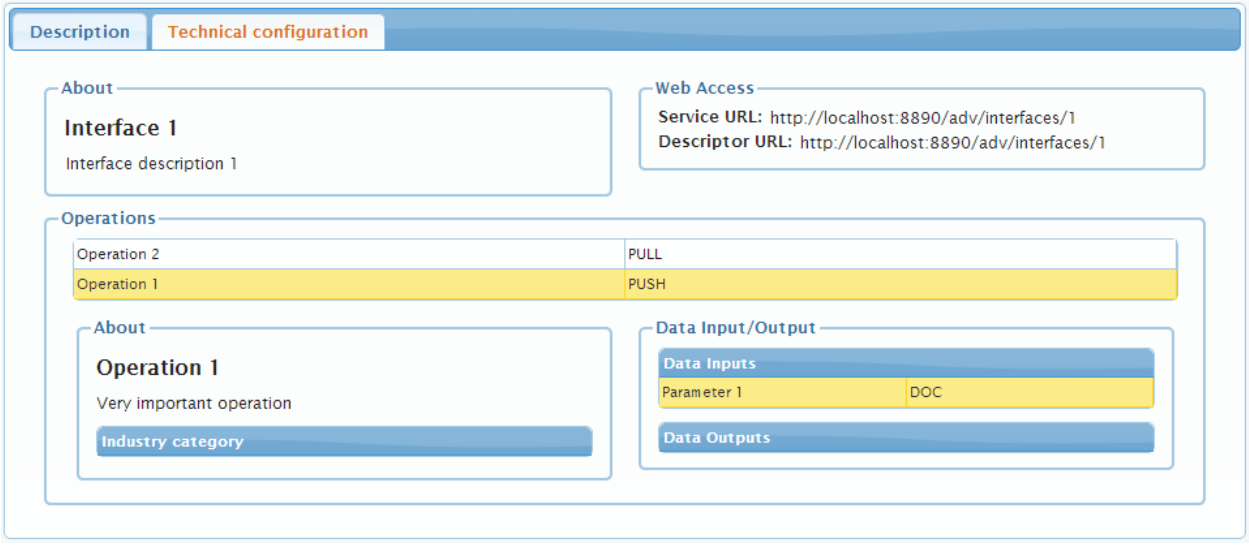

Figure 11: Service gateway technical configuration

The '*Resources*' tab in the Member profile details area lists any resources that have been associated with the services from this provider as produced or used.

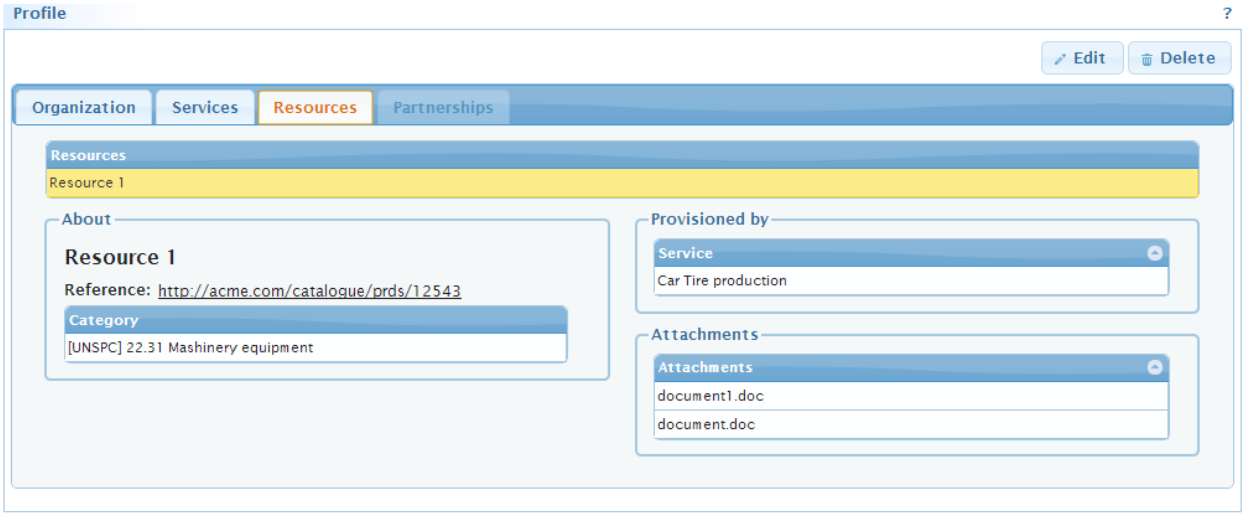

Figure 12: Member profile Resources details

6.1.3 Service API

Some of the OData requests performed to feed data in this UI are listed below.

- List all Member entities with their related Classifications inline

```
/dpd-odata-service/Adventure/DPD.svc/Member?$expand=classifications
```

- Load a Member entity by {id} literal

```
/dpd-odata-service/Adventure/DPD.svc/Member({id})
/dpd-odata-service/Adventure/DPD.svc/Member?$filter=(memberId eq {id})
```

- Find a Member and its services

```
/dpd-odata-service/Adventure/DPD.svc/Member({id})?$expand=services
```

- Find a Service and its non-functional properties

```
/dpd-odata-service/Adventure/DPD.svc/Service({id})?$expand=properties
```

- Find a Service and its classifications

```
/dpd-odata-service/Adventure/DPD.svc/Service({id})?$expand=classifications
```

The API specification of the Data Explorer Service is defined in Table 6.

Table 6: Data Explorer Service API specification

match	
URL Path	/dpd-explorer-service/profiles/match
Method	POST
Request headers	Content-Type: application/json Accept: application/json
Request entity	JSON formatted Example: { "Property 1": "xsd:decimal(?propertyValue)>9", "Property 2": "xsd:decimal(?propertyValue)>199", "classification": ["[UNSPSC] 13101601"]}

As for the SPARQL service, requests can be performed as defined by the SPARQL 1.1 protocol specification. In brief, Table 7 summarizes the specification for sending a query.

Table 7: SPARQL 1.1 service query API

URL Path	/sparql?query={SPARQL query string}
Method	GET
Request headers	Accept: application/sparql-results+xml (to get results in standardized XML)

A sample query returning the ids of the services that match the filter criteria for properties and classifications is presented in Table 8.

Table 8: Sample SPARQL query for mathcing partner services

```
select distinct ?member_id ?srvid from <http://localhost:8892/adv>
where{
  ?srv a adv:Service;
       adv:service_id ?srvid;
       adv:hasProvider ?member;
       adv:hasClassification ?classes;
       adv:hasProperty ?prop.
  ?prop gr:hasValue ?val;
        gr:name ?name.
  ?classes adv:classValue ?classname.
  ?member adv:member_id ?member_id.
  FILTER (
            (?classname='[UNSPSC] 13101601') ||
            (?name='Property 1' && xsd:decimal(?val)>9)  ||
            (?name='Property 2' && xsd:decimal(?val)>199)
         )
}
```

The literals in the FILTER section of the query are variable s is the number of the expressions inside.

6.2 Data Provisioning

6.2.1 Use Cases

- DP-U01: Create Profile
- DP-U02: Save Profile
- DP-U03: Classify Service Products
- DP-U05: Edit Profile
- DP-U06: Add Services
- DP-U08: Describe Service
- DP-U09: Delete Profile
- DP-U10: Remove Service
- DP-U11: Annotate

6.2.2 User Interface

Creating a new profile is as easy as clicking the plus button in the functions tool bar of the profiles master list (Figure 13). This triggers a dialog to input the mandatory property - organization name and then an empty form with the provided name is rendered in the details area to be filled in (Figure 14).

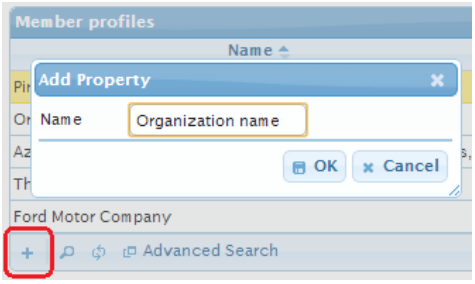

Figure 13: Create new profile

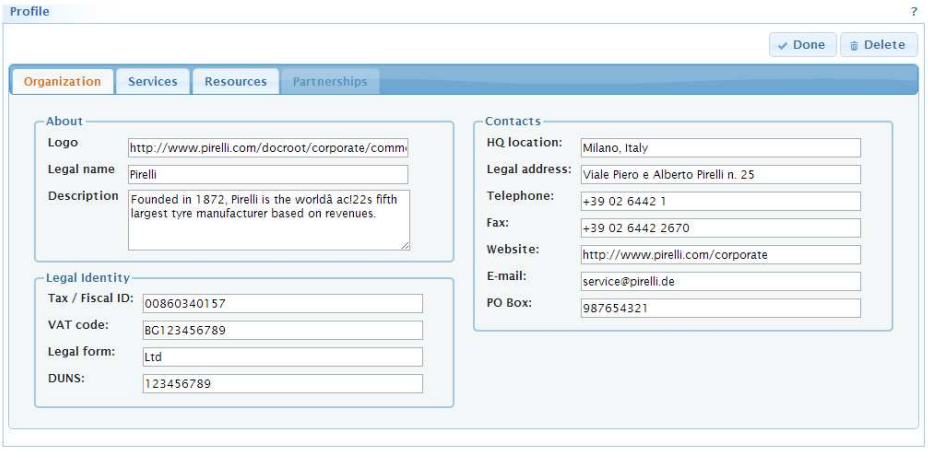

Figure 14: Organization details form

To delete an existing profile, click the ‘Delete’ button rendered in the toolbar of the profile (Figure 15). To edit an existing profile, click the ‘Edit’ button rendered in the toolbar of the profile (Figure 15).

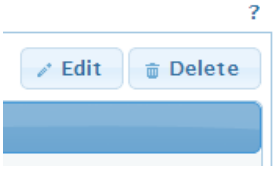

Figure 15: Edit or Delete profile - toolbar

To finalize changes, click the ‘Done’ button in the toolbar of the profile (Figure 16).

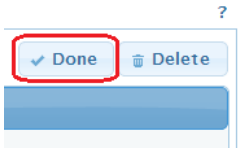

Figure 16: Commit changes

To add or delete services in a profile, use the services list toolbar functions - the buttons with plus and trash icons respectively (Figure 17).

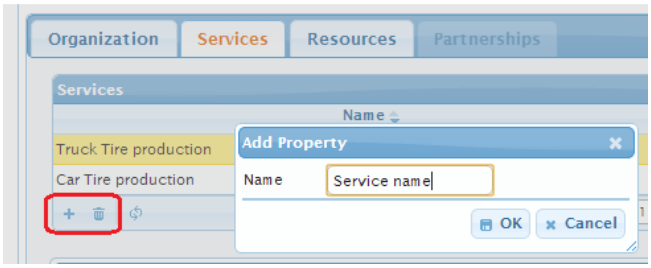

Figure 17: Add new or delete existing service

Adding a new service or rendering the ‘Services’ tab in edit mode, renders respectively an empty (with the provided service name only) or filled with current service details form in the ‘Description’ tab (Figure 18). It is ready to be filled in with non-operational details.

The ‘Technical Configuration’ tab will also render empty or filled with previous details, if any, upon new service or edit service triggered (Figure 18).

Similar to the settled pattern, one can use the plus and trash buttons on the Operations list widget to add or remove operations. Or select an existing operation and edit it in the form that renders accordingly.

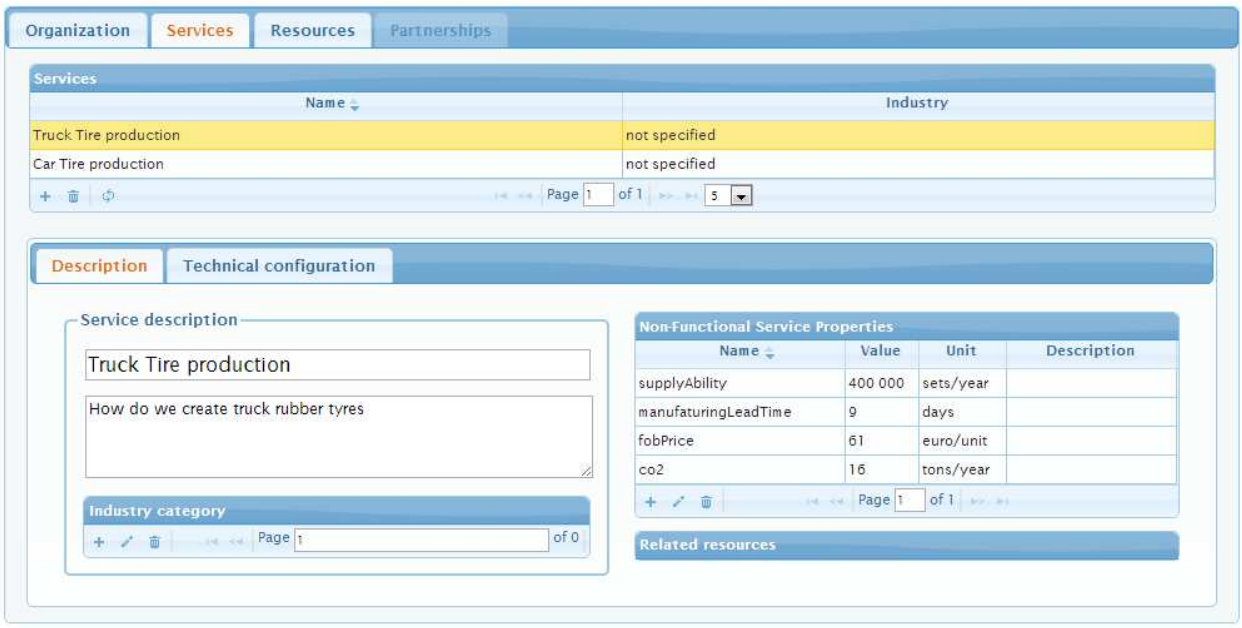

Figure 18: Edit service

The service non-functional properties are managed via the toolbar of the non-functional properties list widget. Clicking the plus or edit buttons brings up the Non-functional properties editor.

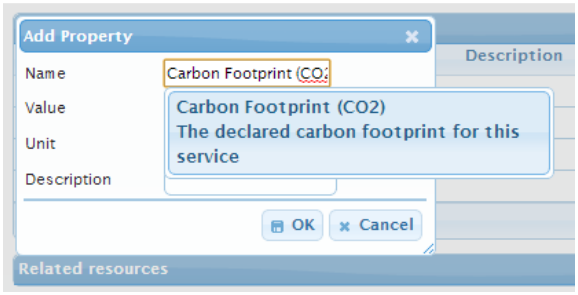

Figure 19: Add/Edit a non-functional property

The Annotation Explorer is used in several sections of the profile form. It is integrated with the classifications list widget, which is also reused considerably. Selecting the edit (pencil) or add (plus) button in its toolbar brings up the Annotation Explorer with a pre-selected concept or empty for new selections (Figure 20).

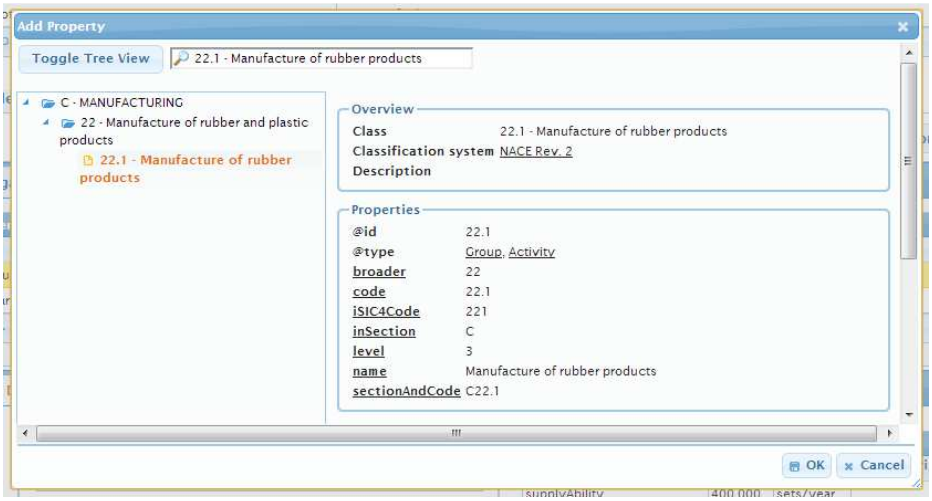

Figure 20: Annotation explorer

6.2.3 Service API

As per the OData specification, the change operations on OData entities are represented by PUT (update), DELETE (delete) and POST (create) HTTP methods. In addition, delta updates are also supported by means of the non-standard MERGE method. Refer to the OData documentation on details for the service API and protocol.

7 Limitations and Further Developments

The second prototype (D4.2.2) is a major advancement from the early stage of the first prototype (D4.2.1) in terms of features, completely new components and some of the old that were largely rewritten, and is a significantly more complex system. Despite the challenges that stem from that, all main scenarios have been finalized.

Some outstanding limitations are:

- No multiple classifications support
- No authentication and authorization support
- The UI-based search is limited to partner services finding, based on classification and non-functional properties only.

As this is the last prototype, future development will be performed as part of the integration and use case development efforts. This will be limited to the support for the use cases as necessity for such emerges. Possible future development topics are:

- Reasonable adaptations to support the use cases in the implementation of ADVENTURE
- Semantic content: The ontologies used for annotations will need to be introduced into Virtuoso. The ontologies needed by the use cases and other components need to be introduced for consumption. The Annotation Explorer needs to be adapted to use multiple ontologies and the Data Explorer Service needs to be adapted to provision them.

8 Summary

The D4.2.2 Prototype is the second and last edition of the DPD module prototypes, developed within task T4.2. Its main goal was to build on the lessons learned and feedback from the previous edition, and implement enough functionality to start the development of the project use cases. The prototype adds a whole new set of components:

- Data Model: a conceptual data abstraction of an ADVENTURE Member profile that support the operations, which realize the DPD scenarios. It's realized as a relational database schema and as an RDF graph model to support both SQL and SPARQL data access protocols.
- Profile Model: a component that realizes a Java language binding to the DPD data model and its persistency aspects via the standard Java Persistency Architecture technology. It also implements a scenario-oriented Java interface that makes transparent all the bulk operations that are specific to the persistency technology and its provider.
- OData Service: a standard OData service interface provider, based on the odata4j and jersey technology stack, to service in a standardized RESTful manner the DPD Data Model on the web.
- Profile Manager: a flexible, extensible and configurable data mashup Java framework for DPD data model management. It hides the complexity of the integration with external data sources, communication, parallelism and synchronization behind its API.
- SPARQL Service Endpoint: a standard service and protocol to query the RDF graph provisioned by the Data Model realization

This prototype edition also introduces a number of improvements and enhancements of the already introduced components:

- Data explorer: Annotations Explorer, Non-Functional Properties editor, a Model-View-Presenter pattern-based UI framework featuring data bindings and grid UI controls library, data entity model framework, new backend (OData) service and more.
- Data Explorer Service: added match operation to the profiles resource endpoint, added classifications resources endpoint along with the whole mechanism for contextual, service-based provisioning of classifications.

The prototype will be further stabilized and adapted as necessary to the needs of the use cases in the next project phases.