284928

Collaborative Manufacturing Network
for Competitive Advantage

D4.5 – Linked Data Best Practises
(public)
<table>
<thead>
<tr>
<th>Grant Agreement No.</th>
<th>284928</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project acronym</td>
<td>ComVantage</td>
</tr>
<tr>
<td>Project title</td>
<td>Collaborative Manufacturing Network for Competitive Advantage</td>
</tr>
<tr>
<td>Deliverable number</td>
<td>D4.5</td>
</tr>
<tr>
<td>Deliverable name</td>
<td>Linked Data best practises</td>
</tr>
<tr>
<td>Version</td>
<td>V 1.0</td>
</tr>
<tr>
<td>Work package</td>
<td>WP 4 – Linked Data Integration</td>
</tr>
<tr>
<td>Lead beneficiary</td>
<td>TUD</td>
</tr>
<tr>
<td>Authors</td>
<td>Markus Graube (TUD), Jan Hladik (SAP), Frank Haferkorn (RST), Julianna Katona (K&amp;A), Tobias Münch (SAP), Davide Grosso (COMAU), Robert Schachner (RST)</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Walid Tfaili (EVIDIAN), Mohamed Hilia (EVIDIAN), Anat Goldstein (BGU)</td>
</tr>
<tr>
<td>Nature</td>
<td>R – Report</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU- Public</td>
</tr>
<tr>
<td>Delivery date</td>
<td>31/08/2014 (M36)</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable provides a description of the best practices for setting up and managing a Linked Data integration environment.

It is the result of an architectural alignment among the generic concepts developed in the ComVantage project and focuses on the 4 main outcomes of work package 4 (Linked Data Integration):

1. Ontology engineering methodology
2. Ontologies for manufacturing application areas
3. Linked Data adapters for data integration
4. Linked Data toolset to support the mapping, curating and visualization of data

It describes an integrated approach of Linked Data visions among all technical work packages. More detailed outlines to the Linked Data key concepts are described in related deliverables. Please find a list of all related deliverables in section 1.3.
# Table of Contents

## 1 OVERVIEW

1.1 INTRODUCTION ................................. 7
1.2 SCOPE OF THIS DOCUMENT .......................... 7
1.3 RELATED DOCUMENTS .............................. 7

## 2 CONCEPT OF LINKED DATA INTEGRATION

2.1 INTEGRATION INTO COMVANTAGE PROCESSES ........................................... 8
  2.1.1 Ontology Engineering ................................................................. 9
  2.1.2 Linked Data Provisioning ............................................................. 10
2.2 INTEGRATION INTO COMVANTAGE ARCHITECTURE ........................................ 10

## 3 ONTOLOGY ENGINEERING

3.1 ONTOLOGY ENGINEERING METHODOLOGY ...................................................... 11
3.2 TOOL SUPPORT ............................................................... 14
  3.2.1 OntoSketch .............................................................. 14
  3.2.2 Protégé ............................................................... 15
  3.2.3 Linked Open Vocabularies (LOV) .................................................. 15
  3.2.4 Maintenance of Ontologies ......................................................... 16
3.3 ONTOLOGIES IN COMVANTAGE ................................................................. 16
  3.3.1 Existing Ontologies ................................................................. 16
  3.3.2 Developed Ontologies (within ComVantage) .................................... 20

## 4 LINKED DATA PROVISIONING

4.1 BUSINESS AND ENGINEERING SOFTWARE INTEGRATION ................................ 21
  4.1.1 XIWrap .............................................................. 21
  4.1.2 D2RQ .............................................................. 22
  4.1.3 OData - LD Adapter .......................................................... 22
4.2 MIDDLEWARE ADAPTERS ................................................................. 26
  4.2.1 Data Harmonisation Middleware (DHM) Adapter JobController .................. 26
  4.2.2 Test Execution Environment (Tee) .................................................. 27
  4.2.3 Predictive Active Machine Maintenance Support (PAMMS) .................... 29
  4.2.4 The Extended Predictive Active Machine Maintenance Support (EPAMMS) .... 30
  4.2.5 DhmAdapterLiveLDP ......................................................... 30
  4.2.6 OPCUA2Gamma Adapter ......................................................... 31
  4.2.7 OPC UA Linked Data Adapter ..................................................... 32
  4.2.8 Silk .............................................................. 33
4.3 PUBLISHING GUIDELINES ................................................................. 34
  4.3.1 Introduction .............................................................. 34
  4.3.2 Documentation .............................................................. 34
  4.3.3 Functional Test and Standards Compliance ........................................ 35

© ComVantage Consortium – 2014
4.3.4 Human-readable Representations ................................................................. 36
4.3.5 Good-practice Use of RDF ........................................................................... 36
4.3.6 Vocabularies ................................................................................................. 36
4.3.7 Interlinking .................................................................................................... 37
4.3.8 Advertising the Dataset ................................................................................ 37

4.4 Tool Support ...................................................................................................... 37
4.4.1 Development .................................................................................................. 37
4.4.2 Visualisation ................................................................................................... 38
4.4.3 Validation and Verification ............................................................................. 43
4.4.4 Storage and Querying .................................................................................... 44

5 Conclusion ............................................................................................................ 44

6 References ............................................................................................................. 45

7 Appendix I: Glossary .............................................................................................. 47
List of Figures

Figure 1: Linked Data related processes (yellow) in ComVantage implementation process .......................... 9
Figure 2: ComVantage architecture with two collaboration partners sharing distributed data sources and business processes along a common end-to-end product value chain; D4.5 is mainly responsible for the lower parts beginning with parts of the Domain Access Server .......................................................... 11
Figure 3: OntoSketch User Interface ........................................................................................................... 14
Figure 4: Example of a rdf:List ..................................................................................................................... 17
Figure 5: UML class diagram of the GoodRelations ontology ........................................................................ 19
Figure 6: Overall architecture of the Domain Access Server leveraging the OData – LD Adapter ............... 23
Figure 7: Web Interface for N-Triples Query .............................................................................................. 25
Figure 8: The XML Output of N-Triples Query ........................................................................................... 26
Figure 9: The JSON Output of N-Triples Query .......................................................................................... 26
Figure 10: Workflow Diagram of the JobController .................................................................................. 27
Figure 11: Activity of Test Execution Environment component ................................................................. 28
Figure 12: Communication between PAMMS component and Linked Data store ........................................ 29
Figure 13: Relation between EPAMMS and PAMMS components .............................................................. 30
Figure 14: DHM server Deployment Architecture ....................................................................................... 32
Figure 15: Silk user interface showing the detection of possible replacements for shirt parts .................. 33
Figure 16: Sample Data which should be visualised for debugging purposes ............................................. 39
Figure 17: Raw RDF visualisation of exemplary dataset .............................................................................. 39
Figure 18: RDF visualisation in an UML object diagram style ................................................................. 40
Figure 19: RDF object diagram with two graphs containing 73 triples ..................................................... 41
Figure 20: Automatic RDF visualisation in an UML class diagram style of the message ontology .......... 42
Figure 21: Advanced visual graph interaction with the Graph Explorer prototype ................................. 43
1 OVERVIEW

1.1 Introduction
The ComVantage project aims at facilitating the collaboration of partners across organisational boundaries by applying novel collaboration and visualisation concepts for increased trust and latest data federation and security concepts in order to provide an appropriate data source integration as well as access control among different stakeholders. Within two main work streams the fundamental and generic concepts are developed and implemented within well-chosen application areas.

This deliverable is a result of task T4.5 “Linked Data best practises” and concludes the work of work package WP4 in the ComVantage project.

1.2 Scope of this Document
The purpose and scope of this document is the provision of a summary about best practices considering the Linked Data integration for an inter-organisational collaboration platform. It is based on prior work published in several ComVantage deliverables and comprehensive discussion between the generic concepts work packages (WP2 – WP5).

This deliverable summarises the results of work package 4 and puts an emphasis on the learnt best practises regarding Linked Data and information management inside ComVantage.

Main Activities have been:
- Summarise and conceptualise results
- Evolve guidelines

1.3 Related Documents
This deliverable provides a summary of the work package 4 results. Thus, it describes only the outcome whereas the details of the concepts and implementations can be read in the prior technical deliverables of work package 4 and the adaption deliverables in the application work packages:
- **D4.1.3 Data format specifications** - public
  The deliverable describes the Linked Data approach, the ontology engineering methodology and the ComVantage ontologies in detail.
- **D4.2.3 Middleware adapter set** - confidential
  The deliverable describes the technical concepts of Linked Data adapters for middleware technologies.
- **D4.3.3 Business and engineering software adapter** - confidential
  The deliverable describes the technical concepts of Linked Data adapters for business and engineering software.
- **D4.4.3 Linked Data support toolset** – confidential
  The deliverable describes a set of tools and the necessary processes for creating, linking, mapping, curating and managing multiple datasets.
- **D6.3.2 Adaption of Linked Data integration concept**
  The deliverable provides a description of necessary adaptations of the generic WP4 concepts in order to fulfil the requirements of the application area Plant Engineering and Commissioning.
- **D7.3.2 Adaption of Linked Data integration concept**
  The deliverable provides a description of necessary adaptations of the generic WP4 concepts in order to fulfil the requirements of the application area Customer-oriented Production.
- **D8.3.2 Adaption of Linked Data integration concept**
  The deliverable provides a description of necessary adaptations of the generic WP4 concepts in order to fulfil the requirements of the application area Mobile Maintenance.

Furthermore, since the work packages are heavily interweaved, the deliverable was aligned with the finalising best-practice deliverables in the other work package which provide further and more detailed information about the concepts and implementation from another view:

- **D2.3 ComVantage Architecture Best Practises** – public
  This deliverable provides a summary about best practices considering the integrated architecture of an inter-organisational collaboration platform.

- **D3.5.2 Guidelines for the secure collaboration model** - public
  This deliverable provides final guidelines with respect to the ComVantage modelling method and the secure collaboration infrastructure.

- **D5.4.2 UI guidelines for mobile collaboration** - public
  This deliverable describes final guidelines to improve mobile UIs for collaborative scenarios as well as best practices for the enabling foundation and system architecture.

- **D6.5 Plant Engineering and Commissioning - Final report on best practices** - public
  This deliverable describes best practices of applied generic concepts in the Plant Engineering and Commissioning application area.

- **D7.5 Customer-oriented Production - Final report on best practices** - public
  This deliverable describes best practices of applied generic concepts in the Customer-oriented Production application area.

- **D8.5 Mobile Maintenance - Final report on best practices** - public
  This deliverable describes best practices of applied generic concepts in the Mobile Maintenance application area.

- **D11.4 Final report on prototypical implementation of ComVantage** - confidential
  This deliverable reports about the final demo scenarios in each of the application areas and underlying technical set ups.

All deliverables of the ComVantage project that are declared as public can be accessed on the ComVantage webpage: [http://www.comvantage.eu/results-publications/public-deliverables/](http://www.comvantage.eu/results-publications/public-deliverables/)

### 2 CONCEPT OF LINKED DATA INTEGRATION

#### 2.1 Integration into ComVantage Processes

In order to set up a virtual factory based on the technical and organisational platform of ComVantage, all stakeholders have to perform some coordinated steps included in the overall implementation process (Ziegler, et al., 2014). This process is shown in Figure 1.

In general, implementation sub-processes are categorised into collaborative processes where all partners have to jointly create a common result, shared processes that are done by an internal or external provider for all partners, and company internal processes that have to be done by each partner individually. Collaborative processes require coordinated work and joint decisions by all partners. Shared processes are equal for all partners and thus have only to be done once. The results can be shared resources or services. Company internal processes do not require collaboration. The results feed into the Collaborative Manufacturing Networks (CMN). It is obvious that company internal processes can also be outsourced. However, the main difference to shared processes is that there are no synergy effects besides the joint use of (external) competencies.
The ComVantage overall implementation process consists of seven sub-processes, which are done sequentially in three phases. In the first phase, the common business processes (BPM) and roles (RM) are modelled. These are collaborative sub-processes, where joint decisions must be made. In the second phase, ontologies and apps are developed that are necessary to implement the BPM and RM. These are shared processes whose results are shared by all partners. In the third phase, each partner provides datasets for the developed ontologies and creates an Access Control Model (ACM) based on the RM to secure the provided data. In parallel, app ensembles are orchestrated based on the BPM, ontologies and apps. The app ensembles provide the datasets to authorised users according to the underlying BPM and are thus the final result of the entire process. All sub-processes are scalable to different sizes and qualities. Individual activities may be omitted and the tools may be selected according to the actual requirements.

As one can see, the Ontology Engineering sub-process is in the core of the setup of such a virtual enterprise. Without a proper definition of the used ontologies, there is no chance to semantically lift data, to secure them and to create app-ensembles which should run based on this information model.

The Linked Data Provisioning step relies on the ontologies created before in order to offer existing data as RDF to the partners. This step is crucial to have a common information model making it possible to get advantages from shared information.

Both processes with their tools are shortly described in the following lines in order to give a broad overview. The detailed best practises are described in section Error! Reference source not found. and section 4 regarding Ontology Engineering and Linked Data Provisioning respectively.

2.1.1 Ontology Engineering

In order to design the ontologies that serve as terminologies and data structures for the applications, several ontology engineering methodologies were considered, and the Enterprise methodology developed by (Uschold & King, 1995) was selected, since it provides sufficient guidance to provide the ontology engineers with a clear process without requiring disproportional overhead. This methodology suggests the following phases:

1. Identifying the purpose
2. Building the ontology, with steps
The first step can essentially be completed before the start of the CMN and laid out in the business plan. For step 2a, interviews with domain experts from the application partners have to be conducted to determine the essential terms of the corresponding domain and their relations. Step 2b can be performed with the help of the Protégé tool, generating RDFS or OWL ontologies based upon these terms. In order to integrate existing ontologies (step 2c), terms from popular ontologies are used where it is possible, e.g. the FOAF and vCard ontologies describing persons and their addresses or the GoodRelations ontology for prices and other costs. Links to DBpedia entities should also be included if appropriate. Step 3 involves using the developed ontologies within the respective application areas and evaluating them with respect to usability, clarity, and consistency. The final step is performed by documenting the developed ontologies in text documents and by using the rdfs:label and rdfs:comment properties within the ontologies themselves. In order to also enable non-experts to perform basic ontology operations, the OntoSketch tool was developed, which allows the refinement of existing ontologies by domain experts using a graphical editor.

2.1.2 Linked Data Provisioning

The main task of the Linked Data Provisioning phase is to generate, distribute and supply the CMN with the information necessary for the common business models. Sources of information can be, among others, Excel sheets, XML files, and proprietary engineering or control systems. The information has to be modelled with the approved ontologies in order to be useful for the CMN.

The Ontology Engineering phase delivers the meta-models of the information which is necessary for the shared business cases. Afterwards, the sources of this information (existing legacy systems or already Linked Data stores) can be identified. Linked Data adapters may be provided for this data if this information is not already present as Linked Data. The adapters use the approved ontologies from the Ontology Engineering phase to transfer the legacy data into Linked Data taking domain specific knowledge into account. If the update rate of the legacy data is rather low, simple automated model transformations can be applied which provide a RDF output for an information entity as input (e.g. a single Excel file). Various frameworks (e.g. Jena for RDF support) and application interfaces (e.g. Enterprise Server for Comos) may support this task. The RDF file is then stored in a triple store (e.g. Virtuoso) which offers access via SPARQL. If the update rate of the data is rather high (e.g. the provision of process values), a service may provide direct access to the necessary information as Linked Data. Some dynamic adapters provide a local SPARQL endpoint (e.g. XLWrap, D2RQ) whereas others only provide data via a REST web interface as Linked Data.

2.2 Integration into ComVantage Architecture

ComVantage follows a distributed and layered architecture which is described in deliverable D2.2 in detail. The technologies and best practises of Linked Data integration are one important basis for the overall ComVantage architecture, which is shown in Figure 2. All in all, there are five different layers which can be instantiated in parallel throughout different organisations and stakeholders. The application layer handles workflow modelling and application orchestration without big relations to Linked Data. The web layer is located at the top of the Domain Access Server and acts as a single point of access managing the access control. The data integration layer is the bottom part of the Domain Access server and makes heavy use of Linked Data technologies in order to integrate the legacy systems. These systems are managed in the data source layer which is the main area of the Linked Data integration technologies. Both tasks, ontology Engineering and Linked Data Provisioning, are interweaved with this part of the architecture. Furthermore, the best practises from this deliverable can thus also be used in other environments where a similar data source layer exists.
3 ONTOLOGY ENGINEERING

3.1 Ontology Engineering Methodology

As described in deliverables D6.3.1, D7.3.1, and D8.3.1, we considered three rather different methodologies for ontology engineering. They are briefly described in the following:

- The methodology developed by Grüninger and Fox (Grüninger & Fox, 1995), also called TOVE methodology, specifically requires first-order predicate logic (FOL) as ontology language, which is very expressive and thus has little in common with RDFS. The ontology development goes through the following stages:

  1. Capture scenarios: description of the scenarios the ontology will be used in natural language.
  2. Informal competency questions: formulation of questions the ontology should answer in natural language.
  3. Formal terminology specification: specify the terminology in FOL, i.e. choose names for constants, functions, and relations.
  4. Formal competency questions: formulate the questions from step 2 in FOL.
  5. Formal axioms and definitions: specify the constraints and the terms from step 3 in FOL.
6. Evaluation of competency and completeness: verify that the ontology developed in step 5 can answer the questions from step 4 correctly and completely.

- The methodology developed by Uschold and King (Uschold & King, 1995), also called Enterprise methodology, specifies the following phases:
  1. Identifying the purpose: determining why the ontology is being built, who will use it, and for which aim.
  2. Building the ontology:
     a) Capturing: identifying the key concepts, producing definitions for these concepts, and agreeing on names for these concepts. The authors recommend agreeing on the definition before deciding on the term to be used for the concept because people working in different areas tend to have different association with terms, which makes it difficult to reach an agreement if the term is chosen first. Moreover, unlike most other methodologies for software or ontology engineering, the Enterprise methodology suggests neither a top-down approach (going from the most general to more specific terms) nor a bottom-up one (in the opposite direction), but rather goes middle-out, i.e. it starts from the most frequently used concepts, which normally are located at the middle height in the ontology hierarchy.
     b) Coding: representing the conceptualisation produced in the previous stage formally in an ontology language.
     c) Integrating existing ontologies: finding usable terms from other ontologies and connecting them with the terms from the newly developed ontology.
  3. Evaluation: making a technical judgement of the ontology with respect to the requirements specification or the real world.
  4. Documentation: recording all important assumptions and decisions.

- Methontology, a methodology developed by Fernandez-Lopez et al. (Fernandez-Lopez, Gomez-Perez, & Juristo, 1997), is based on the experience acquired in developing an ontology in the domain of chemicals, therefore it was developed by a large team of experts and ontologies and is intended to be used by a huge application area. It uses the evolving prototypes paradigm from the software engineering world, which means that several phases of ontology development are identified, but the overall process is considered as circular, so that decisions made in an early phase can be changed if they turn out to be inappropriate in a later phase. This is not possible if the development process follows the waterfall model, where the decisions made in one phase are detected very late in other phases and thus impose heavy efforts for changing.

The phases identified by the Methontology developers are:
  2. Conceptualisation: structuring the domain knowledge by building a glossary of terms, consisting of concepts, entities or “things”, and verbs, which describe actions or processes. Moreover, rules describe the behaviour and relations between concepts and verbs.
  3. Integration: connection with existing (meta-) ontologies; integration of existing knowledge with the new ontology.
  4. Implementation: using an ontology development environment in order to formalise the concepts, verbs and rules described previously in an ontology language.
Besides these actions, the following tasks have to be performed constantly and in parallel with the development phases:

1. **Knowledge acquisition**: collecting information from all kinds of sources like books, figures, tables, or directly from experts using structured or unstructured interviews. This task is especially important in, but not limited to, the conceptualisation phase.

2. **Documentation**: since the developers of Methontology consider the lack of documentation guidelines for ontologies and the resulting lack of documentation as a central problem in the area of ontology development, each step and the corresponding decisions have to be documented thoroughly. Consequently, this methodology requires writing a requirements specification document, a knowledge acquisition document, a conceptual model document, a formalisation document, an integration document, an implementation document and finally an evaluation document.

3. **Evaluation**: carrying out a technical judgement of the ontologies and their documentation with respect to the requirements specification document. This comprises verification, i.e. testing the technical correctness of the ontology, as well as validation, i.e. testing if the developed ontology corresponds to the domain it is supposed to represent.

Our decision was in favour of the Enterprise methodology, for the following reasons:

- **TOVE** is the least appropriate one as it does not give a clear guidance and does not structure the ontology development process sufficiently: there is no strict differentiation between conceptualisation, formalisation and implementation; these phases are combined in steps 3-5. Moreover, it specifically demands the use of FOL, which is fundamentally different from RDF and thus also from the Linked Data principles. Consequently, the part regarding formalisation of axioms will play a much smaller role in **ComVantage** than in the TOVE project.

- **Methontology** represents the other extreme: it is highly formalised, requires a large number of different activities, some of which have to be performed sequentially while others have to be performed in parallel. Each of these activities has to be documented extensively, and it should ideally be performed by an independent team. This is inappropriate for the **ComVantage** project since there are less than 10 people involved in the development of a single ontology, and the person months allocated for this task are not sufficient for writing such extensive documentation, and it is also not required for information exchange.

- The Enterprise methodology appears to provide enough guidance to steer the ontology development process without introducing unnecessary overhead regarding the production of documents or running several tasks in parallel. Moreover, several features of the Enterprise methodology fit well with the intuitive approach that we used for the first version of the DC21 ontology: for example, we also started from the “middle” concepts, which are the different properties of shirts, and we also found it useful in the capturing phase to focus on the intended meaning of the concepts and decide about an appropriate term for this concept afterwards.

- Moreover, since the Enterprise methodology does not explicitly suggest a lifecycle model, we adopted the “evolving prototype” model from Methontology, since it fits well with the sequence of prototypes described in the **ComVantage** project plan.

After the completion of the final version of the **ComVantage** ontologies, we still consider this decision to be valid. The suggestions for structuring the interviews (definitions first, terms later) and the development of the ontologies (middle-out) have proved very helpful in practice. Following the evolving prototype paradigm allowed us to extend our data model with every prototype where needed. Together with the flexible data model of RDF (compared with the rigid data model of a relational database), this allowed us to react in an efficient manner to changing or added requirements. A more intensive use of competency question could have guided our way sometime a little better.
Moreover, since the ontologies are documented extensively in the deliverables (in particular in D4.1.3) additional documentation as suggested by Methontology would have introduced significant overhead with doubtful benefit. Similarly, the limited expressivity of RDFS regarding axioms would have rendered a large portion of the TOVE methodology useless.

### 3.2 Tool Support

#### 3.2.1 OntoSketch

Although our aim was to select an ontology engineering process with little overhead, the Enterprise methodology might still be too complex for a micro company that wishes to join an existing network, offering services that are slightly different from the existing ones. In order to simplify ontology engineering tasks in such a scenario, the ComVantage consortium has developed the OntoSketch editor (Brade, Schneider, Salmen, & Groh, 2013), which offers an intuitive touch and sketch based user interface and simplifies user interaction by restricting the ontology editing possibilities to the most important ones. Based upon the results of a study with experienced and inexperienced users, the user interface shown in Figure 3 was developed. It allows for different views (showing only entities that are relevant for a specific modelling perspective), creation of new classes, properties, and individuals (but not deletion of entities from a loaded ontology), and for input by handwriting.

This enables users to browse and visualise existing ontologies, extend them where required, and save the results so that they can be uploaded into a triple store. All in all, OntoSketch allows user without much experience on Linked Data to develop ontologies in a very easy way. It supports only a minimum of features and is really easy to learn and use in comparison to other tools like Protégé.

![OntoSketch User Interface](image)

*Figure 3: OntoSketch User Interface*

---

1 Triple store is explained in the glossary
3.2.2 Protégé

For the development of ontologies within the ComVantage project we used the Protégé\(^2\) editor, which to the best of our knowledge is the most popular ontology editor with almost 250,000 registered users.\(^3\) It has also reached a high level of maturity and reliability, and since it allows for plugins, it is highly customisable. Within the project, we frequently used the OntoGraf plugin to visualise the developed ontologies, e.g. in the deliverables for Task T4.1.

Since Protégé is an open source project, it can be used free of charge, which is important for small and middle-size companies. In comparison, the least expensive licence for the popular commercial ontology editor TopBraid Composer is available for $1850.\(^4\)

However, due to the peculiarities of ontology modelling, it turned out to be necessary to have project collaborators with experience in ontology modelling in general and the Protégé tool in particular. For example, renaming of entities is a complex process with significant side effects; it therefore cannot be performed with a function of the entity’s context menu, but only via the “Refactor” menu item, which prevented some users from finding the function. Moreover, the display of entities from included ontologies in the editing window and the OntoGraf visualisation requires specific configuration steps.

Another issue results from the differences between the paradigms of object-oriented programming and ontology engineering, which are obscured by the user interface and also by the terminology used in RDFS and OWL, where the term “class” has a different meaning from a class in Java or C++. For example, in an ontology it is not necessary, and often not desirable, to assign a domain and a range to an object property, whereas this is necessary for strongly typed OOP languages. However, this affects all ontology editors known to the authors of this document.

3.2.3 Linked Open Vocabularies (LOV)

The following four paragraphs are taken from deliverable D4.4.3:

LOV (Linked Open Vocabularies) is a platform (http://lov.okfn.org/) for offering various vocabularies for different areas. It is offering various ontologies and vocabularies for different industrial sectors. Moreover, it is intended “to provide easy access methods to this ecosystem of vocabularies, and in particular by making explicit the ways they link to each other and providing metrics on how they are used in the linked data cloud, help to improve their understanding, visibility and usability, and overall quality.”\(^5\)

“LOV targets both vocabulary users and vocabulary managers. Vocabulary users are provided with a global view of available vocabularies, complete with precious metadata enabling them to select the best available vocabularies for describing their data, and assess the reliability of their publishers and publication process. Vocabulary managers are provided with feedback on the usability of what they maintain and publish, common best practices their publication should stick to in order to keep being reliably usable in the long run.”

One goal of LOV is to provide “a technical platform for search and quality assessment among the vocabularies ecosystem, but it also aims at promoting a sustainable social management of this ecosystem. Beyond the LOV project, we have the vision of a future linked data Web supported by a living Vocabulary Alliance gathering as many as possible stakeholders in the long-term conservation of vocabularies.”

Additionally, LOV provides recommendations for metadata of Linked Data vocabularies (http://lov.okfn.org/dataset/lov/Recommendations_Vocabulary_Design.pdf). These make it possible to generate a nice documentation and human friendly representation of the vocabulary.

\(^2\)http://protege.stanford.edu

\(^3\)http://protege.stanford.edu/community.php


\(^5\)See http://lov.okfn.org/dataset/lov/about/ for the citations in this and the following two paragraphs.
There are activities to take the ontology “Machine Semantic” (MSem) and use the LOV platform to publish the results of the project ComVantage. There are some open issues related to necessary tagging of the MSem ontology before it can get part of LOV.

In the final state (M36) the MSem ontology will be to be part of the vocabulary of the LOV platform. Our experience with LOV (in end of M35) is that it takes some non-trivial extra efforts to make an ontology fit for a publication on this platform. There are some requirements about documentation of the ontology to be published. E.g. extra information like the rights and properties is requested and a detailed labelling of classes and properties. It is a good practice to use the LOV platform to publish a mature ontology.

3.2.4 Maintenance of Ontologies

Regarding the issue of maintaining or refactoring ontologies, we have found that it is important to choose the right term for an entity at an early stage. While it is easy to extend an ontology with an additional term, changing an existing term that is used in other ontologies requires a significant effort and strong coordination, even if the term is only used by a small community like the ComVantage consortium. This problem also has an impact on ontologies developed by other groups, e.g. FOAF. Here, it becomes obvious through the availability of four different terms for the concept of “family name”, two of which are labelled as “archaic”, whereas the other two are labelled “testing” (see Section 3.3.1.2 for detailed information).

Moreover, the URI prefix of FOAF (http://xmlns.com/foaf/0.1/) still carries the version number 0.1 although the current number is 0.99 and this version has significant differences from first one. The FOAF developers remark that “it long ago became impractical to update the namespace URI without causing huge disruption to both producers and consumers of FOAF data. We are left with the digits "0.1" in our URI. This stands as a warning to all those who might embed metadata in their vocabulary identifiers.”6 The ComVantage consortium decided to allow for version numbers in URI prefixes, e.g. http://www.comvantage.eu/ontologies/garment/0.5/, but to impose the restriction that the prefix without a version number, in this case http://www.comvantage.eu/ontologies/garment/, must always resolve and point to the current version.

We have also found that “hash URIs”, i.e. URIs that use the hash mark # as a separator between the ontology URI and the entity name sometimes cause problems with specific tools like D2RQ. We therefore decided to favour “slash URIs”, which use the slash sign / instead. Since hash URIs only have advantages for ontologies that are stored in a single text file rather than a triple store, this decision does not have drawbacks that are relevant for the ComVantage project.

3.3 Ontologies in ComVantage

3.3.1 Existing Ontologies

One of the main advantages of Linked Data is the possibility of reusing existing ontologies and thus avoiding the need to reinvent the wheel for every new application. We therefore aimed at using existing ontologies wherever possible. These ontologies and our experiences with their usage are described below.

3.3.1.1 RDFS

RDF Schema7 (RDFS) is the most basic ontology in the Linked Data world. Besides the entities defining the data model (like rdf:type, rdfs:domain and rdfs:range), we used the data properties rdfs:label and rdfs:description extensively in order to describe the meaning of the entities contained in our ontologies, thus enabling later reuse. The rdfs:label property was used for a terse (preferably only consisting of one word) name for the entity, whereas rdfs:description was used for a more detailed characterisation, which

6http://xmlns.com/foaf/spec/
7http://www.w3.org/TR/rdf-schema/
may consist of several sentences and also include peculiarities of our data model, e.g. domain and range restrictions or super classes.

RDFS provides also basic data structures like

- **Sequences** and
- **Lists**

In the ontology DataSeries (DS) these features have been used heavily. A list is of type rdf:List. Each list node has the property rdf:first linking to the actual content and rdf:rest for referencing the next node as shown in Figure 4.

```xml
:rootNode
  a rdf:List;
  rdf:first :node1.

:node1
  rdf:first :theNode1Data;
  rdf:rest :node2.

:node2
  rdf:first [ xsd:Integer 1 ];
  rdf:rest :nodeLast.

:nodeLast
  rdf:first :theNodeLastData;
  rdf:rest rdf:nil.
```

**Figure 4: Example of a rdf:List**

### 3.3.1.2 FOAF

Friend of a Friend\(^8\) is one of the most popular ontologies in the web of data. It is used to describe information about people, like names, email addresses, or relations. Due to the popularity of FOAF, the ComVantage consortium opted for this ontology to represent personal information. Due to its simplicity, FOAF offers little guidance, but it allows the user a high degree of freedom regarding the way he wants to use the entities. In this regard, FOAF is very different from sophisticated ontologies like GoodRelations (see Section 3.3.1.4).

Another advantage of FOAF is the fact that, unlike e.g. vCard (see Section 3.3.1.3), it originates from the internet community and thus fits very well into the social customer network environment envisioned for the Customer-Oriented Production area: it allows for an easy integration of customers’ homepages (foaf:page), email addresses (foaf:mbox), or user-generated designs (foaf:made).

However, FOAF also has drawbacks, where the lack of vocabulary elements for describing surface mail addresses is of particular relevance for the ComVantage application areas. In order to capture the shipping addresses occurring in the Dresscode21 database, it was therefore necessary to also include the vCard ontology (see below). Another, albeit minor, nuisance is the redundancy of terms with the same semantics. For example, FOAF offers four different terms to indicate a person’s last name, two of which (foaf:surname and foaf:family_name) are labelled as “archaic”, whereas the other two (foaf:lastName and foaf:familyName) are labelled as “testing”.

### 3.3.1.3 vCard

As mentioned before, FOAF lacks a vocabulary for describing surface mail addresses. Therefore, in order to capture a customer’s billing or shipping address, terms from a different ontology are needed. The

---

\(^8\) [http://xmlns.com/foaf/spec/](http://xmlns.com/foaf/spec/)
ComVantage consortium decided in favour of the vCard\(^9\) ontology since it is based on the well-known vCard terminology, has a high level of maturity and can satisfy all needs of the scenarios from the Customer-oriented Production application area. It has a sophisticated data model comprising a class `ns:Address`, which is the domain for the data properties `ns:locality`, `ns:postal-code` and `ns:street-address`.

Using entities from both the FOAF and the vCard ontologies did not cause any problems, thus the open character of Linked Open Data proved to be useful in practice. From our experiences, using terms from several ontologies is clearly preferable to developing a new ontology from scratch in cases where an ontology fitting the desired use case perfectly is not readily available.

### 3.3.1.4 GoodRelations

In comparison with FOAF or vCard, the GoodRelations\(^10\) (GR) ontology comes with a much more complex and sophisticated data model (see Figure 5): it provides a large amount of classes and properties, many of which have domain and range restrictions.

Since the desired use of the ontology within ComVantage (prices and fees for orders within the DC21 web shop) is not completely identical with the intended purpose of the ontology (description of product or service offerings made via the World Wide Web), this led to some difficulties.

For example, our intention was to use the `gr:PriceSpecification` class for the prices of instances of the `cvshop:Cart` and `cvshop:CartItem` classes. If we had used the `gr:hasPriceSpecification` property to make this connection, this would have implied that all instances of the Cart and CartItem classes also become instances of the `gr:Offering` class, which is counterintuitive since in the application area Customer-oriented Production (WP7) we are considering a webshop and not an offering made via the web, and thus most of the attributes of the Offering class do not apply to Carts and CartItems, e.g. availability times and eligibility regions.

We therefore extended the shop ontology with the object properties `hasTotalPrice`, `hasUnitPrice`, and `hasTax`, which are used to connect Carts and CartItems with the corresponding PriceSpecifications. In this case the strict definitions of the GoodRelations ontology made it more difficult to use within the project. Nevertheless, defining alternative object properties to include the GR classes required less effort than building our own price model.

\(^9\)http://www.w3.org/TR/vcard-rdf/

\(^10\)http://purl.org/goodrelations/v1
Figure 5: UML class diagram of the GoodRelations ontology

http://www.heppnetz.de/ontologies/goodrelations/goodrelations-UML.png

http://www.heppnetz.de/ontologies/goodrelations/goodrelations-UML.png
3.3.1.5 QUDT

QUDT is short for: “Quantities, Units, Dimensions and Data Types in OWL and XML”. It is an ontology under construction (2014 currently Release 2 is on its way). For more details have a look at: http://www.qudt.org/. The homepage www.qudt.org gives this overview on the project.

“The QUDT Ontologies, and derived XML Vocabularies, are being developed by TopQuadrant and NASA. Originally, they were developed for the NASA Exploration Initiatives Ontology Models (NExIOM) project, a Constellation Program initiative at the AMES Research Center (ARC). They now form the basis of the NASA QUDT Handbook to be published by NASA Headquarters.”

QUDT is an ontology that provides a wide range of physical units, dimensions and quantities which could be well integrated into our vocabularies. We decided to use a strategy having QUDT units as optional part of the Msem ontology. Describing physical properties of Sensors and Actuators the Msem ontology allows beside the use of the elaborated QUDT units and quantities also the use of plain strings. This allows using the Msem ontology both as standalone ontology and to be used in combination with QUDT.

3.3.2 Developed Ontologies (within ComVantage)

Based upon the ontology engineering principles sketched in this Section, we have developed the following ontologies in the different application areas Plant Engineering and Commissioning (PEC), Customer-oriented Production (COP) and Mobile Maintenance (MMA). They are described in detail in deliverable D4.1.3.

3.3.2.1 Jointly for PEC and MMA:

- Message Vocabulary (Msg) is useful for sending message information from one cooperation partner to another. Storing this information in the Linked Data Store makes it much easier to collect and query information in messages.
- Machine Semantic Ontology (MSem) describes the structure of a Virtual Enterprise broken down to ControlModules, Sensors and Actuators following partly ISA-88\(^\text{12}\).
- Data series Ontology (DS) is used to store and access huge amount of Data (so called Data Series). It is organised as combination of a linked-list (see rdf:seq) and an index-tree. It is optimised to access trailing Data elements of huge data series.

3.3.2.2 For PEC

- Users Ontology (User) is the description of detailed properties for each account. It includes information such as names, family name, company role, experience, e-mail, username and password, etc.
- Roles Ontology (Role) is designed to describe the role of the user that is accessing the system. It is used to allow or deny the data access providing detailed information for each account.
- Plants Ontology (Plants) is in charge of describing the structure of a customer plant. It collects information from the production lines receiving real values from the field that have to be compared with theoretical ones.

3.3.2.3 For COP

- Garment Ontology (garment) describing the different parts of a shirt, its style and production steps
- Shop Ontology (shop) for the ordering process, shipping and billing information
- Supplier Ontology (supplier) for relations with tailors, embroidery shops, and material suppliers
- Material Ontology (material) describing features of the individual shirt parts, e.g. zippers

\(^{12}\) For ISA-88 see http://de.wikipedia.org/wiki/ISA-88. It is also known as IEC 61512.
3.3.2.4 For MMA

- Repair, Preventive and Predictive Maintenance Ontology (RPPM) is a bridge between organisational aspects of Mobile Maintenance and technical issues. It covers aspects of Maintenance like roles of Human and Technical Actors and types of involved Companies.
- Test Execution Environment Ontology (Tee) is used to provide Tests via the Test Execution Environment, based on CETES. The vocabulary to store the results of these tests is also part of it.

4 LINKED DATA PROVISIONING

4.1 Business and Engineering Software Integration

One important aspect for an inter-organisational collaboration network is to share information in a semantic and integrated information space. This information is managed in special software designed for a specific purpose and usually stored as legacy data. However, parts of this business and engineering information have to be semantically lifted and provided in order to get common added value. For that purpose several adapters have been developed and used. The experiences gained are presented in the following subsections.

4.1.1 XlWrap

An important kind of data in companies are still spreadsheets. A lot of information is stored in Excel files since the initial engineering of these files is rather fast and requires low effort. The lack of flexibility coming from the fixed structure of rows and columns is countered by the creativity of the engineers. However, finally this information has to be merged in a common information space that is semantically described. The transformation of information from spreadsheets into Linked Data is done by XlWrap\textsuperscript{13}. This is a spreadsheet-to-RDF wrapper which is capable of transforming spreadsheets to arbitrary RDF graphs based on a mapping specification. It supports Microsoft Excel and OpenDocument spreadsheets such as comma-(and tab-) separated value (CSV) files and it can load local files or download remote files via HTTP. XlWrap is available as open source (Mozilla Public license). Its concept is described in deliverable D4.4.3 (Linked Data support toolsets) and its application inside ComVantage is described in deliverable D6.3.2 (Adaption of Linked Data integration concept).

XlWrap offers both a Jena API, which can be included in specific tools, and a SPARQL endpoint. The configuration is stored itself as an RDF model in a separate configuration file. It uses a specific vocabulary defined by XlWrap to define the mapping between cells in spreadsheets and RDF triples. Therefore, it uses a slight enhancement of RDF and introduces Named Graphs in its model. Hence, the model is usually stored as TriG\textsuperscript{14} file. It can also use various functions on cells before converting them to RDF. This includes for example simple arithmetic operations, sums, maxima and minima as well as date functions and string manipulations. Since it is open source these functions can be enhanced to specific needs.

In our use cases we decided to use a static conversion of the Excel sheet and store the resulting RDF dataset in a triple store. This allows a better integration with other data that is stored in the triple store since all information can be used in one query without additional retrieving of information from the XlWrap SPARQL endpoint. Of course, the use of the static conversion and transfer to an extern triple store is only possible because the information from the spreadsheets is static and do not change often. Other use cases with a more frequent update of the spreadsheets will need the use of the internal SPARQL endpoint embedded in XlWrap.

The configuration for this mapping is quite simple. After the declaration of the necessary prefixes, there is the main mapping configuration including the templates. The semantics of XlWrap mappings are quite easy.

\textsuperscript{13}http://xlwrap.sourceforge.net/
\textsuperscript{14}http://wifo5-03.informatik.uni-mannheim.de/bizer/trig/
to understand. However, a tool support for creating the mapping files would have a huge value especially for medium and small sized companies without much experience on RDF.

XlWrap has been proven to be a useful and reliable Linked Data adapter component, which can be quickly aligned to new spreadsheets. There are some limitations which really seldom meet actual needs. However, these could be easily overcome by forking the open source software and adding the necessary components. For a dozen of plugins these steps have already been taken. To the best of our knowledge, there is no more powerful spreadsheet to Linked Data adapter.

4.1.2 D2RQ

The D2RQ\(^{15}\) database adapter is used for providing reading access to the existing customer database within the Customer-oriented Production application area. It essentially works by assigning a URI to every row of every table and linking the content of each column to the corresponding URI by an object or data property. Details about the software itself are described in D4.4.3, and the use within the Customer-oriented Production application area is described in D4.3.3 and D7.3.2.

D2RQ has turned out to be reliable and the functionality proved to be sufficient for the requirements of the Customer-oriented Production application area. However, since D2RQ does not offer a GUI, but is based on a configuration file that has to be edited manually, the usage is error-prone and collaborative development is difficult. Moreover, some features of the adapter were discovered only at a rather late stage of the development due to the lack of a UI support and the limited documentation. This refers to, for example, the possibility of defining inverse roles (where the property in the Linked Data model goes in the opposite direction of the reference in the database) or the possibility to define several mappings from a single database table.

A feature that would be desirable for ComVantage but is not offered by D2RQ is the provision of different access points by a single server. This would allow for different endpoints depending on user roles and thus support our access and security model. Since this feature is not provided, access control has to be performed within the DAS server.

In spite of these shortcomings, D2RQ has turned out to be a useful and reliable Linked Data adapter component, whose limitations do not justify the development of a new adapter from scratch. To the best of our knowledge, there does not exist a more powerful database to Linked Data adapter.

4.1.3 OData - LD Adapter

The OData – LD Adapter makes use of the OData standard\(^{16}\) to extract data from enterprise systems. The OData Standard was developed by Microsoft and is applied for many enterprise systems. Within SAP, the standard is of key relevance to expose business data of SAP solutions to the web. Therefore, a broad coverage of OData interface for various SAP solutions is ensured and simplifies their consumption for web-based applications, as for instance used in the ComVantage project.

The OData – LD Adapter has been developed as a RESTful service application in order to allow Web-based and standalone applications to easily access it and to get the query results in different formats, i.e. JSON\(^{17}\) (application/sparql-results+json) and XML\(^{18}\) (application/sparql-results+xml).

The main feature of the adapter is to transform a SPARQL query to an OData request in order to retrieve data from enterprise backend systems. The results are mapped to the used meta model and are formatted as RDF (keeping the Linked Data design principles). The prototype provides an interface for web application as well as a GUI for testing and demonstrations.

\(^{15}\) [http://d2rq.org](http://d2rq.org)

\(^{16}\) [http://www.odata.org/documentation/odata-version-4-0/](http://www.odata.org/documentation/odata-version-4-0/)

\(^{17}\) [http://www.w3.org/TR/sparql11-results-json/](http://www.w3.org/TR/sparql11-results-json/)

\(^{18}\) [http://www.w3.org/TR/rdf-sparql-XMLres/](http://www.w3.org/TR/rdf-sparql-XMLres/)
4.1.3.1 Application interface

The OData – LD Adapter is mainly used in a data federation scenario where parts of the requested information are stored in a triple store and the remaining information is requested from the SAP backend system. Today, federated queries\(^\text{19}\) as part of the SPARQL 1.1 specification are supported by major triple stores like Virtuoso Universal Server or Apache Fuseki\(^\text{20}\).

Figure 6 shows how the federated queries are processed within the data federation scenario employing a SPARQL server (e.g. Virtuoso) and the OData – LD Adapter using the SPARQL 1.1 protocol. The adapter retrieves the results from an OData service of a backend system (e.g. SAP CRM, Customer Relationship Management) and then translates these results into JSON or XML (as per the user’s selection). After the translation, which is carried out using the standard JSON and XML specification, the results are forwarded to the requesting client (e.g. the Domain Access Server or the triple store), which automatically transforms these results into Linked Data.

In the following, an example with respect to an SAP CRM system is outlined. Parts of the information with respect to temporary relations of sales opportunities to other business objects are stored in the triple store. Additionally, the opportunities master data and transactional data are stored in the SAP CRM backend system. Within the SPARQL query we want to do the data federation transparently, without additional translation of data and protocols.

```
prefix crm: <http://www.sap.com/crm/1.0/>
prefix xml: <http://www.w3.org/XML/1998/namespace>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
construct {
  ?opportunity
    crm:prospect ?prospect ;
    crm:confidence ?confidence ;
    crm:description ?description ;
    crm:origin ?meeting ;
    crm:currentStage ?stage ;
    crm:product ?product ;
    crm:account ?account .
} where {
  values (?guid) { ("5CF3FCDC-AE9C-1EE3-B2BF-CC9E6FEA9A8E") }
  SERVICE <http://odata-bridge.wdf.sap.corp:4080/sap/opu/odata/sap/CRM_OPPORTUNITY/Opportunities> {
```

\(^{19}\) [http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/](http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/)

\(^{20}\) [https://jena.apache.org/download/index.cgi](https://jena.apache.org/download/index.cgi)
The OData service is invoked with the SERVICE query pattern, which actually performs a SPARQL 1.1 Query Protocol query. Since the OData service can’t be directly accessed, the OData – LD Adapter is used to transform the SPARQL protocol to OData protocol. The following message will arrive at the OData – LD Adapter as a GET or POST request:

```
SELECT *
WHERE {
}
```

Currently only basic graph patterns (BGP) are supported, so we get a list of triples. Some of them have variables (e.g. ?prospect) in it. The left hand element of a triple is the searched subject. The middle element is the name of the property we want to filter and the right hand side is the value of the property. If an element is denoted by a variable, then it is expected to be the returned value. If it is a concrete value (e.g. string) it is to be used as a filter or service parameter. The request above expects a list of opportunities described by the opportunity URI (?opportunity), the prospect name (?prospect), the chance of success (?confidence), and the user status text (?stage).

The query is translated into a native OData request. Each OData service has a specific meta model for structuring and naming entities which need to be mapped to the semantic meta model used in the ComVantage architecture (as part of the data federation executed by the Domain Access Server and other Linked Data adapters).

The corresponding OData request for the above presented example is shown below:

```
https://<hostname>:<port>/sap/opu/odata/sap/CRM_OPPORTUNITY/Opportunities(guid"<guid>")
```

In order to have a generic and (semi-)automatic approach for the request and data translation between SPARQL and OData service, there are two areas which need to be investigated.

1. The (semi-)automatic generation and compilation of OData service proxy classes on the client side.
2. The (semi-)automatic semantic matching/mapping of OData service parameters (from OData service meta model) with the SPARQL query parameters (from semantic meta model of the ComVantage data federation infrastructure).
In order to address the above areas, one candidate is SAP’s Netweaver Gateway Productivity Accelerator (GWPA)\(^{21}\), which is a very handy plugin for Eclipse and deals with the first point. For handling the second point, the metadata information of an OData service can be retrieved easily (using $metadata at the end of a service URI) and processed using any XML DOM library. Moreover, vocabularies like VCARD are already used on both sides to facilitate mapping of meta models.

### 4.1.3.2 The Web Interface of the OData – LD Adapter

Additionally to the application web interface, a graphical interface for testing and demonstration was developed. Figure 7 shows the web interface of the OData – LD Adapter. It comprises a text area (for typing the N-Triples query) and a button “Submit Query”. The desired output of the query, e.g. JSON or XML, can be configured. If the output is set as XML, then after typing in the query and pressing the button, the output is downloaded as xml file, which can be opened in any text editor, as shown in Figure 8. If the output is set as JSON, then after typing in the query and pressing the button, the output is shown directly in the browser, as shown in Figure 9.

![Figure 7: Web Interface for N-Triples Query](https://help.hana.ondemand.com/gateway_gwpa/frameset.htm)

---

4.2 Middleware Adapters

Besides static data (as shown in section 4.1), live data is also necessary to provide value-added inter-organisational services. This live data has to be integrated into the static Linked Data network. This task is performed by middleware adapters.

4.2.1 Data Harmonisation Middleware (DHM) Adapter JobController

The DHM JobController (JC) is responsible for receiving incoming requests and organising and controlling the distribution of the tasks to the components of the DHM-Adapter (see Figure 10; more details can be found in D4.2.3 chapter 3.3 JobController). Requests for reading simple sensor data are dispatched to the...
LiveData Access, requests for performing tests are handled by the Test Execution Environment (see D 4.2.3 chapter 3.4 Test Execution Environment).

It is necessary to organise access to the data provided from the machines via sensors and actuators, because write access to a sensor or actuator must not be possible to concurrent applications. Therefore, it was necessary to add a component for this task to the DHM-Adapter. Thus, the JobController is not directly involved in mapping sensor values to the Linked Data Store, only blocking writing access to sensors and actuators while e.g. a test is running. Otherwise the test results would be unreliable and useless for diagnosis. Read access to sensors and actuators is allowed at any time. For a modular structure of the DHM-Adapter, this task was separated from the other DHM-Adapter components.

4.2.2 Test Execution Environment (Tee)
The Test Execution Environment (Tee) performs predefined tests and stores the results in a Linked Data Server. Tests are generated with the model based testing tool CETES\(^{22}\). For details of these processes see D4.2.3 chapter 3.4.

Linked Data is a very flexible technique for storing test results and related information for the DHM-Adapter component Tee and making the information available in the corresponding app.

By implementing the Tee component in Java and using the possibilities of Reflection it is possible to store also application related information in the Linked Data Store. This contributes to making the Tee component generic and useful in many environments within the Mobile Maintenance and Plant Engineering and Commissioning application areas. E.g., a test is a JUnit class to be executed at certain

\(^{22}\) See (Hartung, Prester, & Schachner, CETES - Cost Efficient Test System for Embedded Systems, 2010) and (Hartung, CETES – Model based testing involves further changes beyond test design, 2012)
intervals or when requested by the user. The information regarding this test is dependent on the embedded system which is used and the purpose for which it is used and possibly many more factors of a well-defined embedded system. All of this is stored in the Linked data store. The generic task of executing such test is done by the Tee component for all kinds of tests on different machines (see Figure 11).

The complexity of the technical analysis to be provided by the Tee is very high. Thus, the competency questions used as a first test for the Tee ontology have only provided very basic information about the usefulness of this ontology. While developing the Tee application, especially the apps which are showing the test results, it was necessary to enlarge and enhance the TEE and RPPM ontologies provided beforehand, because the questions evolved during the app development are much more detailed than any general questions about relations between e.g. tickets and the related machines or persons. For example, it is necessary to find the corresponding results to a single test execution on a certain date and the embedded system involved. For each part of the results the corresponding description in the test model must be found, while these descriptions should be stored only once in the Linked Data Store, because they will remain the same for all executions of one well defined test. The ramifications of these tasks have become clear only while implementing the corresponding applications.

Another aspect is the need for visual information which has become clear only during the implementation phase. Hence the RPPM ontology has been extended to comprise a logo in order to show a company logo next to a ticket. This company logo shows the owner of the faulty machine and helps the human actors to grasp the information provided very quickly and easily.

This confirms the general Linked Data best practice to start ontology development with very few elements and extend the ontologies later on as needed.

Since Tee implementation and the corresponding app development was both done by the same company, the task of aligning the data provided by the backend into the Linked Data Store and the data expected from the frontend from the Linked Data Store was much facilitated.
4.2.3 Predictive Active Machine Maintenance Support (PAMMS)

One of the major advantages of the ComVantage approach is the predictive maintenance support. It is possible to predict machine failures by analysing the past behaviour of a specific machine and other machines of this type. For details of this component refer to D4.2.3 chapter 3.5.

The Predictive Active Machine Maintenance Support (PAMMS) component is crucial in making the machine an active machine with predictive functions. All the intelligence of the failure prediction system is situated in this component.

All the advantages of using Linked Data for storing the necessary information of the Tee component apply also to the PAMMS system: I.e. the system is generic because specialised program related data can be stored and retrieved from the Linked Data store (see Figure 12). The ontologies have started with the basic elements from the TEE ontology and have been extended as needed during the implementation process. E.g. test results are stored in the Linked Data Store, it is necessary to correlate embedded system with the test results regarding this well-defined embedded system. Experimenting with the system has shown that it is easier to define tests for a group of embedded systems which are tested at the same time than defining a test for each embedded system a-piece, because all the test metadata are the same. Therefore the ontology was extended to store the correlation between values and embedded systems.

The PAMMS system is also providing information for the visualisation of the test results. In order to provide data ready to be shown in a graphical representation on the tablet the TEE ontology has been extended to provide all the necessary data to the app.

Aligning the data requested from the frontend has been much facilitated by implementing the backend in parallel at the same company.

A further advantage especially for the PAMMS component is the possibility to provide enhanced data for diagnosis and analysis ready to use for the app. In the complex environment of embedded systems simple sensor values have to be aggregated and programmatically interpreted to facilitate interpretation by human actors. Using the same techniques as for the test execution by storing specialised program information in the Linked Data Store, aggregated and interpreted data are created by the backend to be shown in a human readable way by the corresponding application.

![Figure 12: Communication between PAMMS component and Linked Data store](image-url)
4.2.4 The Extended Predictive Active Machine Maintenance Support (EPAMMS)

As described in D4.2.3 chapter 3.5, the PAMMS is a central component to provide Predictive Maintenance. The PAMMS is located at the customer’s company and is connected to the Active Machine via a Middleware.

The PAMMS has a counterpart at the site of the Support Company. This counterpart is called Extended Predictive Active Machine Maintenance System, or shorter EPAMMS.

Data are sent from the PAMMS in RDF format based on the TEE and RPPM ontologies at the customer site to the EPAMMS system at the Mobile Maintenance company (see Figure 13). The tasks of the EPAMMS are essentially the same as the tasks of the PAMMS. The difference is a larger data collection collected from different PAMMS components for statistical evaluations and analysis. Therefore the same principles for implementing a generic approach using Linked Data apply to the EPAMMS.

Figure 13: Relation between EPAMMS and PAMMS components

4.2.5 DhmAdapterLiveLDP

The Live Data Access has been merged with the Linked Data Publisher into the component DhmAdapterLiveLDP, as large parts of the code basis are identical.

4.2.5.1 Live Data Access

The component Live Data-Access is responsible for accessing up to date Sensor and Actuator data from the Machine. This data is acquired via HTTP-GET requests\(^{23}\) into middleware calls in order to read sensors and/or write to actuators. The request returns the current sensor data and sensor semantics in RDF/Turtle.

4.2.5.2 Linked Data Publisher

The component Linked Data Publisher is responsible for acquiring and exporting the Machine Semantics to the Linked Data Server. Its tasks are:

- Detecting newly attached & detached Gamma Systems by in-depth browsing for new attaching Gamma Systems

---

\(^{23}\) These requests should origin form the SupportAPP. During development of the Mobile Maintenance application area these requests may also origin from a simple web browser, like Firefox.
- Extracting **Machine Semantic** about all Info of **Machine/Sensor/Actuator** taken from the **Gamma Machine Configuration**.
- **Publish all** Msem **Semantic** of all detected Gamma Systems to the Linked Data Server

### 4.2.6 OPCUA2Gamma Adapter

In D4.2.3 the architecture of the DHM adapter has been described in detail. The following paragraphs are taken from that deliverable:

“A DHM-Adapter has two SPARQL Endpoints:

- the Linked Data Server (LDS, to the right) and
- the History Data Server (HDS, at the left side)\(^{24}\).

The DHM-Adapter uses a Middleware (in this example GAMMA V\(^{25}\) and OPC-UA). This middleware system is a combination of a middleware and a single or several underlying machines under control. The middleware provides (via a data-model driven interface) access to sensors and actuators of the underlying machines.

Each machine has a configuration file, which describes the data model of the available sensors and actuators in a so called Machine Semantic. This configuration file is loaded at the start of the Middleware Controller and so accessible for the components LDP and the LiveData access.

In a first step of the processing of this Machine Semantic the Linked Data Publisher (LDP) is extracting this Machine Semantic and as a second step it is exported to the Linked Data Server (LDS). In a third step, the SupportApp reads this Machine Semantic and uses it to directly access the Sensor/Actuator data via the LiveData Access as a fourth step.

The Middleware-Logger is responsible for logging all relevant data to the History Data Server in the cases when the middleware doesn’t implement this feature itself”

The PLCs\(^{26}\) are connected to the OPC\(^{27}\)-Server that is communicating via the OpcUa2Gamma Client to the GAMMA middleware as shown in Figure 14. The changes of the OPC UA Process variables are signalled to the Gamma Middleware. The components like the Tee can access all PV changes via Gamma callbacks.

All components of the DHM-Adapter from the Mobile Maintenance application area can be reused in the Plant Engineering and Commissioning application area in this architecture without changes.

---

\(^{24}\)In general, the LDS and the HDS may be integrated in a single SPARQL-Endpoint

\(^{25}\)(GAMMA , 2013)

\(^{26}\)Programmable logic controller

\(^{27}\)OLE for Process Control is a widely used middleware
4.2.7 OPC UA Linked Data Adapter

A direct integration of OPC UA middleware into Linked Data environments has advantages in terms of performance and configuration efforts. Furthermore, one could implement a SPARQL endpoint which is capable of directly performing a search inside the OPC UA information space. However, the available framework were too slow, not open enough and do cost a lot of license fees. Therefore, we decided to start an open and free software stack which implements OPC UA independent from the OPC Foundation. The project open62541 is about developing an open IEC-62541 compliant middleware research stack, which should be freely usable both for Linked Data environments as well as for other purposes. We are investigating enhancements, analysing vulnerabilities and learning how to implement a complex object-oriented/semantic middleware protocol with an imperative programming language like C. The sources are

[28] https://opcfoundation.org/
hosted at GitHub\textsuperscript{30}. The open62541 project will leverage a deep integration of OPC UA into Linked Data making it possible to combine the closed-world of OPC UA having high response times with the open world of Linked Data having seamless cross-domain integrations.

4.2.8 Silk

Silk\textsuperscript{31} is a tool for discovering relationships between data items within different Linked Data sources. While the most obvious use is finding entities referring to the same real-world object in different ontologies, it can also be used to find arbitrary relations between entities. The Silk engine works as a server offering a web frontend for defining rules describing when two entities should be considered as related. Once the rules have been defined, Silk connects with SPARQL endpoints and tries to find matches.

Within the Customer-oriented Production application area, we have employed Silk as a tool for finding possible replacements for shirt parts, in particular, zippers (see D7.3.2 for details). The local installation of the server was simple, and the syntax for defining rules is powerful: there is a large arsenal of conversion functions for numbers (e.g., arithmetic functions, removal of non-numeric characters) and strings (conversion to lower case, substrings, removal of prefixes) as well as comparison functions (Levenshtein distance, Jaro-Winkler, temporal or geographical distance). In order to aggregate the results of several comparisons into a single numerical score, there are aggregate functions like maximum or weighted average.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{silk.png}
\caption{Silk user interface showing the detection of possible replacements for shirt parts}
\end{figure}

\textsuperscript{30} https://github.com/acplt/open62541
\textsuperscript{31} http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/
These features can be accessed via a GUI that allows for dragging and dropping the different functions (see Figure 15), which makes it significantly easier to develop, verify and modify rules than a mapping file based on XML that has to be edited manually, like e.g. for D2RQ (see section 4.1.2).

In summary, Silk has proved to be easy to install and to use, and the feature range was sufficient for our purposes. Due to the large range of its functionality, it could be used in a large number of scenarios. However, LIMES\(^{32}\) (Link discovery framework for MEtric Spaces) is a well-established competitor which is also open source and provides faster algorithms and more enhanced self-learning functionality. So, LIMES is worth looking at in future projects.

4.3 Publishing Guidelines

It is important to ensure a specific quality of datasets in order to make them most useful for other partners and application developers. For this purpose, this chapter presents a publishing checklist for the provision of Linked Dataset inside ComVantage. The checklist is provided and maintained in the wiki (http://www.comvantage.eu/wiki/index.php?title=Linked_Data_Publishing_Guidelines). This checklist was intended specifically for the datasets published in LATC\(^{33}\). It was modified in order to meet the requirements of ComVantage and has established as helpful guideline for creating high quality datasets.

4.3.1 Introduction

This is a quality checklist for ComVantage-published datasets. Its purpose is to establish a baseline of expectations for the datasets we publish in the project. This doesn't necessarily mean that these datasets are available for public use. Thus, this checklist should also be applied for internal ontologies.

The quality criteria are grouped into three levels:

- **MUST** indicates criteria that are strictly necessary for a dataset to be called “Linked Data” and meet basic legal and social expectations. We will not announce or consider complete datasets that do not meet these criteria.

- **SHOULD** indicates criteria that are certainly a good idea, but there may be valid reasons (e.g., technical reasons or available resources) for not following them. Where these criteria are not met, this must be documented in a section on the dataset’s descriptions, called “Limitations” or “Future Work” or similar. If possible, the section should explain why each criterion was not fulfilled.

- **MAY** indicates criteria that are a good idea, but they are of lesser importance or often inapplicable. Publishers do not need to justify or document why they are not following them. These criteria are here mostly as reminders for the data publishers.

4.3.2 Documentation

Each dataset should have a wiki page that contains, or links to, the following information:

4.3.2.1 Basic documentation: Who, what, why, when, how?

This is about meta information about datasets

- **MUST**: What is the name of the dataset?
- **MUST**: What is this dataset about?
- **MUST**: Who has created the dataset? Who maintains the dataset?
- **MUST**: Under what license is the data released? There should be a link to a licensing/copyright statement by the original data publisher. If no such original statement exists, there should be some language that states that all rights to the data remain with the original data owner and use of the data requires their permission.
- **SHOULD**: What can users do with it? What sort of question can it answer? Hence, why is it here?

\(^{32}\) http://aksw.org/Projects/LIMES

\(^{33}\) http://latc-project.eu/
SHOULD: At what interval is the data updated (e.g., daily, monthly, continuously, never)? When has it been last updated?
MAY: How is the original data converted and processed?
MAY: Link to the code used for conversion and processing, and any software used in the publication
MAY: What is the source of the data? There should be a link to the original location of the data.

4.3.2.2 Documentation about data access

SHOULD: Link to a SPARQL endpoint and SPARQL query form
SHOULD: Download link for an RDF dump
SHOULD: What syntaxes are supported for the resolvable URIs (RDF/XML, Turtle, RDFa, others)?
MAY: Explain any difference between the data that is served via SPARQL, via dumps, and via resolvable URIs

4.3.2.3 Documentation on examples

MUST: URIs of example resources
SHOULD: URIs of example resources that contain links to other datasets
SHOULD: Example SPARQL queries
MAY: Example Turtle snippet showing the modelling of typical entities

4.3.2.4 Documentation on modelling and size

MUST: What other datasets does it link to? (How many links to each?)
SHOULD: How many triples?
SHOULD: What entities are described? (How many of each kind?)
SHOULD: Which vocabularies are used?
MAY: URI patterns for the various types of entities
MAY: Diagram (e.g., UML) of the dataset’s schema

4.3.2.5 Branding of the documentation

MUST: ComVantage branding, consisting of a ComVantage logo, a link to http://comvantage.eu/, and a sentence such as: “This work is supported by ...”

4.3.2.6 Documentation of limitations and future work

MUST: Are all SHOULD-level criteria that are not met documented?

4.3.3 Functional Test and Standards Compliance

URIs must actually resolve. The SPARQL endpoint must work. The returned RDF must be valid. Note that the Pedantic Web Group maintains a list of tools that can be used for testing compliance.

4.3.3.1 Resolvable URIs and content negotiation

MUST: Do all example resources resolve at all (return 200 OK) if the user is authenticated or the dataset is public?
curl -i http://example-uri

MUST: Can the returned RDF be parsed in an RDF parser or RDF validator?
rapper --gc rdf_file
riot --check --sink rdf_file

SHOULD: Does content negotiation for HTML yield HTML?
curl -H “Accept: text/html” http://example-uri

SHOULD: Does content negotiation for RDF yield RDF?
curl -H “Accept: application/rdf+xml” http://example-uri
4.3.2 Link check

- MUST: For all example resources that contain links to other datasets, does the target URI of the link resolve?

4.3.3 SPARQL endpoint

- SHOULD: Do all the example SPARQL queries from the documentation work?
- SHOULD: Do some simple test queries work?
  ```sparql
  SELECT * {?s ?p ?o } LIMIT 10
  SELECT DISTINCT ?type WHERE { ?x a ?type }
  ```

4.3.4 Human-readable Representations

Example URIs SHOULD yield a human-readable representation (HTML page) when accessed in the browser at least for debugging purposes.

- SHOULD: Do they link back to the dataset’s homepage?
- SHOULD: Do they explicitly state the URI of the accessed resource?
- SHOULD: Do they state, or link to, information about licensing and copyright?
- SHOULD: Do they have ComVantage branding?
- MAY: Do they state, or link to, the source of the data?

4.3.5 Good-practice Use of RDF

There are many aspects to good-practice use of RDF. These are just some basics.

- MUST: Are natural-language strings marked with appropriate language tags?
- MUST: Are numbers and dates marked with an appropriate XSD datatype?
- MUST: Are XSD dates and datetimes correctly formatted (e.g., 2012-02-25 and 2012-02-25T13:26:11.5Z)?
- MUST: Do resources have an rdf:type and a decent label (ideally using a standard property like rdfs:label, dc:title, skos:prefLabel)?
- SHOULD: Are blank nodes avoided for any resources of any importance?
- MAY: Does the dataset have a VoID34 file?
- MAY: Do all example resources have machine-readable metadata in the RDF representation (possibly a VoID file that contains the metadata)? License, source, and date are the most important bits of metadata.

4.3.6 Vocabularies

Vocabularies are essential machine-readable documentation of a dataset’s semantics.

4.3.6.1 Re-use of existing vocabularies

- MUST: Do re-used vocabularies use the correct namespaces? Watch out for missing hashes or slashes at the end. Use http://prefix.cc to look up namespaces.

---

34 Vocabulary of Interlinked Datasets: [http://www.w3.org/TR/void/](http://www.w3.org/TR/void/)
• MUST: Are the classes and properties actually defined in the existing vocabulary? Are class or property names misspelled?

4.3.6.2 Self-defined vocabularies

• SHOULD: Do the URIs of classes and properties resolve?
• SHOULD: Do the classes and properties have labels?
• SHOULD: Is the vocabulary registered on prefix.cc (search there for the namespace URI) if it is publically available?
• MAY: Are classes and properties mapped to other existing vocabularies?
• MAY: Do the properties have domains and ranges (Domains and ranges should only be used if they contain few (preferably: one) classes and are “natural” for the property)?
• MAY: Are there rdfs:isDefinedBy triples that point from class/property URIs to the vocabulary document?

4.3.7 Interlinking

Most ComVantage data is just a conversion of existing data, so no new information is generated. The production of links, on the other hand, often involves the generation of genuinely new information, and should be carefully documented.

• MAY: If Silk was used, are the link specs in the ComVantage repository?
• MAY: Do reference links exist and are they in the ComVantage repository?

4.3.8 Advertising the Dataset

Datasets have to be listed in the wiki. Public Datasets must be listed on the ComVantage homepage.

• MUST: Is the dataset documented and linked in the ComVantage wiki?
• MUST: Is the dataset announced via the WP4 mailing list?
• MUST: Is it listed on http://comvantage.eu/datasets if it is publically available?

4.4 Tool Support

4.4.1 Development

Jena35 is a Java framework for building Semantic Web applications. Jena provides a collection of tools and Java libraries to help you develop semantic web and linked-data apps, tools and servers. It has powerful APIs and command line tools for reading, processing and writing RDF data as well as for SPARQL querying (ARQ), RDF storing and drawing inferences.

For any tool development, for example a middleware-to-RDF adapter, Jena is a good choice for a development framework. This holds as long as the rest of the project can also be developed in Java which is very likely due to a large amount of freely accessible frameworks and libraries in the Java community. Additionally this framework is also well suited for many tasks in the development environment through its various command line scripts. This includes SPARQL queries, RDF file conversion, RDF file comparison, RDF file validation and RDF storage operations.

Small tools tend to be developed easier when using script language rather than Java. For this purpose, Python in combination with the rdflib36 library has been proven very useful. Rdflib is a pure Python package working with RDF supporting parsers and serialisers, a graph interface, store implementations and a SPARQL 1.1 implementation. It was for example successfully used in the UML RDF visualisation described in section 4.4.2.2.

35 http://jena.apache.org/
36 https://github.com/RDFLib/rdflib
4.4.2 Visualisation

During design and configuration of our global Linked Data backend a quick and intuitive visualisation of single and connected dataset was needed. Although Turtle provides a human-friendly serialisation of RDF, it is still not very easy to comprehend the structure of a dataset. However, exactly the task of getting a quick overview is really important for debugging purposes.

Thus, we have developed a visualisation tool set for Linked Data sets which can generate a visual representation of an RDF file with automatic graph layout. This makes it possible to have a brief overview of the connectivity and complexity of Linked Data sets supporting the developers. The use of GraphViz\textsuperscript{37} as visualisation engine revealed as good choice due to its cross-platform nature. The tool is described in detail in deliverable D4.4.3 – Linked Data support toolset.

There are three different types of visualisations possible:

1. Raw RDF triple visualisation
2. Instance diagram similar to UML
3. Class diagram similar to UML

4.4.2.1 Raw RDF visualisation

The first diagram type visualise each triple via with two nodes (subject and object) connected by a directed edge (predicate). All elements are labelled with their URIs (abbreviated with their prefix). An example for the RDF dataset from Figure 16 is shown in Figure 17.

Literals are displayed as rectangles whereas URIs are displayed as blue ovals making the distinction between these two types easy. At the bottom, all used prefixes are listed. This diagram type gives a brief overview over a dataset. However with an increasing number of triples it is getting less useful due to its huge size with many edges.

```
@prefix : <http://eatld.et.tu-dresden.de/sce-agents/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:prlt04 a foaf:Person, prov:Person;
   foaf:name "PRLT Group 4";
   foaf:mbox "group4@prlt.tu-dresden.de";

:markus a foaf:Person, prov:Person;
   foaf:name "Markus Graube";
   foaf:title "Mr.";
   foaf:firstName "Markus";
   foaf:surname "Graube";
   foaf:mbox "markus.graube@tu-dresden.de";

:comosExport a prov:SoftwareAgent;
   rdfs:label "Comos Export";
```

\textsuperscript{37} \url{http://www.graphviz.org/}
:tud a foaf:Organization, prov:Organization;
    foaf:name "TU Dresden".

Figure 16: Sample Data which should be visualised for debugging purposes

![Sample Data Diagram]

Figure 17: Raw RDF visualisation of exemplary dataset

4.4.2.2 RDF diagrams inspired by UML

A compression of the information density in the visualisation was needed to explore bigger graphs. Thus, we adapted some principles of UML\(^3\) (Unified Modelling Language). Class diagrams and object diagrams are the most used diagram types and fit also for visualising RDF.

An UML object diagram visualises all objects, their types, their attributes and their relations in one diagram. Objects are represented by a rectangle. In the top of the rectangle, the name of the object and its type are

\(^3\)http://www.uml.org/
displayed. Both are underlined. Furthermore the rectangle shows all attributes of the objects with their current values separated by a line from the name. The relations to other objects are represented by directed labelled edges.

Figure 18 shows an RDF object diagram which uses the main principles of a UML object diagram. This diagram shows the same information as Figure 17. It is clear, that the information is compressed in way that makes it easier to understand the data. In contrast to a UML object diagram, the object type is separated from the object name by parentheses and not by a colon. That’s because the names of objects and its types are represented by URLs which use colons for their integrating prefixes. Additionally a Namespace box is listed below the diagram in order to allow the reader to expand the prefixes.

![RDF object diagram](image)

**Figure 18: RDF visualisation in an UML object diagram style**

Furthermore, this diagram type supports multiple graphs. All nodes defined in one graph (which is usually a single RDF file) are collected in an additional rectangle with a description of the graph (file://sce-agents.ttl in the example in Figure 18). Figure 19 shows a more complex diagram with two named graphs and some nodes which aren’t defined themselves as object (meaning to have an rdf:type property) but are used in the other graphs. These are not covered in a graph rectangle but are freely placed in the diagram. As you can see also higher numbers of triples (in this example 73 triples) can be displayed in a useful and not confusing way.

A class diagram represents the hierarchy of a class structure and how the classes are connected to each other. They are displayed in rectangles with the name of the class at the top. Separated by a line you may find the attributes of this class. Classes are connected by edges which represent different types of relations between these classes. For example, inheritance is represented by an open arrow head.

Figure 20 shows an exemplary diagram which transformed the principles of a UML class diagram to a RDF class diagram. It basically supports the same functions as the RDF object diagram. Classes are detected when being of type rdfs:Class or owl:Class. The relation between classes as well as the class properties can be determined by the rdfs:domain and rdfs:range properties of properties of type owl:ObjectTypeProperty and owl:DataTypeProperty.
Figure 19: RDF object diagram with two graphs containing 73 triples
When using the UML-like diagrams for RDF, there is no good way to visualise instances and classes at the same time. However, in most cases, you won’t need such a visualisation at all. The visualisations are generated by a Python script which is provided as open source via GitHub (https://github.com/plt-tud/rdf-uml-diagram).

4.4.2.3 Graph Model Visualisation

In addition to OntoSketch and the UML-style graph visualiser, the tool chain for graphical exploration of semantic models has been complemented with the Graph Explorer.

The Graph Explorer supports technical users to assess the content, extent, quality, comprehensiveness and correctness of a semantic model to facilitate graph maintenance operations and query authoring. Therefore it offers a visual graph interaction and manipulation interface (Figure 21). It features multiple faceted filters which are extracted from the graphs metamodels and can be used to customise the users view on the graph model. Furthermore, the view can be customised by moving entities or selecting entities to open hidden details. Moreover, it visualises related instance data entities as well as connections in between. Additionally, the visualisations characteristic can be customised with respect to shown relation types, fisheye effects to strengthen detail-in-context effects and the depth of visualised nodes (number of direct relations).

Finally, several view modes are planned to enable users to look at the data from different pre-defined perspectives. These can be domain-specific or of general use. An example for perspectives with general validity can be time or location-based, e.g. “what happens now?” or “what happens where?” for specific location. A domain-specific example is the perspective of open maintenance issues that shows unresolved machine problems for which no service staff is currently assigned to.

Combined with the powerful filtering described above it allows to effectively select a relevant subset for detailed introspection. Useful starting points should guide the user. Moreover, the underlying SPARQL queries of executed visual interactions are revealed in order to make the graph structure and main entities more tangible. Users not familiar with SPARQL should be enabled to get an understanding of the language by example.

Figure 20: Automatic RDF visualisation in an UML class diagram style of the message ontology
4.4.3 Validation and Verification

The semantic debugging of Linked Data application gets much simpler when using a visualisation. However, this implies that the RDF datasets are syntactically correct. This has to be ensured by a formal validation which is a crucial point in the chain of tools for managing datasets. Subsequent tools can only operate on data in the correct way, if their input information is valid. In the following, some tools will be covered which have been proven useful to determine the validity of RDF datasets.

*Vapour*[^39] is a Linked Data validator. It checks whether public semantic web data is correctly published according to the current best practices, as defined by the Linked Data principles[^40], the Best Practice Recipes[^41] and the Cool URIs[^42]. Vapour can be used as a public web service validating the providing mechanisms of public data. All in all, this tool is really nice in order to check the configured content negotiation mechanisms and URI structures at least for data that should not be restricted by access control mechanisms like ontologies.

Another online tool is *rdf:about Validator*[^43] which has been the recommended web-based RDF validator in the Linked Data support toolset defined in deliverable D4.4.3. It has supported a bunch of different RDF serialisations like Notation3, Turtle and RDF/XML. Unfortunately the service has stopped because the developers didn’t want to maintain the code and service any longer. The proposal of *ComVantage* to support the maintenance has not been accepted.

[^39]: http://validator.linkeddata.org/
[^40]: http://www.w3.org/DesignIssues/LinkedData.html
[^41]: http://www.w3.org/TR/swbp-vocab-pub/
[^42]: http://www.w3.org/TR/cooluris/
[^43]: http://www.rdfabout.com/demo/validator/
Now, the best way of checking RDF datasets for their validity is the use of Rapper\(^{44}\). It is a command line tool from the Raptor RDF Syntax library and supports all kinds of RDF serialisations. It furthermore can convert to other serialisations as for example to the GraphViz language DOT (as already mentioned in section 4.4.2.1). It is the recommended tool in the Linked Data support toolset in D4.4.3.

Another alternative to Rapper is Riot\(^{45}\) which bases on the popular Apache Jena framework. It supports nearly the same features and can be a good choice when the whole development framework bases on Jena.

### 4.4.4 Storage and Querying

We have used Virtuoso OpenSource server\(^{46}\) for all of our triplestores. Virtuoso is an innovative industry standards compliant platform for native data, information, and knowledge management. It implements and supports a broad spectrum of query languages, data access interfaces, protocols, and data representation formats that includes: SQL, SPARQL, ODBC, JDBC, HTTP, WebDAV, XML, RDF, RDFa, and many more. The open source variant VOS (Virtuoso Open Source) includes a scalable high-performance RDF Triple Store with possibilities to divide the triples into named graphs.

We found it very easy to use Virtuoso for providing RDF via SPARQL as well as via HTML representations while keeping the system manageable via a user-friendly web interface. HTML representations can be quite useful for debugging purposes although the end user of such a collaboration space will probably never see an HTML representation but will access this information via mobile apps. The IRI Derefencing mechanisms could be successfully used for integrating transient process data from machines into SPARQL queries.

ARQ\(^{47}\) is a SPARQL processor for Jena. It is a query engine for Jena that supports the SPARQL RDF Query language. It contains also a command line tool for executing SPARQL queries on local files (http://jena.apache.org/documentation/query/cmds.html). The binary is bin/\texttt{arq} (Linux) or bat/\texttt{arq} (Windows). Thus, it is really a tool for supporting development and validating queries before all data has been published on a real triple store with an endpoint.

## 5 Conclusion

The best practices presented in this document are a consolidated result of the work from the generic work packages including Continuous Requirements Analysis and Overall Architecture (WP2), Secure Information Model (WP3), Business Process Modelling (WP3), Linked Data Integration (WP4) and Trustful Mobile Collaboration (WP5). Furthermore, results from collaboration with application areas Plant Engineering and Commissioning (WP6), Customer-oriented Production (WP7) and Mobile Maintenance (WP8) are incorporated.

In all these work packages experience on the handling of Linked Data and its integration to a common information space have been made which have been condensed into this deliverable pointing out guidelines and best practices from learning during the 3 years of continuous development. All in all, the integration of Linked Data was really successful and could be handled with the presented processes and tools.

---

\(^{44}\) [http://librdf.org/raptor/rapper.html](http://librdf.org/raptor/rapper.html)
6 REFERENCES


## APPENDIX I: Glossary

<table>
<thead>
<tr>
<th>Concept</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARQ</td>
<td>Part of the Jena tool chain. It is a tool to process a SPARQL Queries both on servers and on local files and triple stores.</td>
</tr>
<tr>
<td>Class</td>
<td>RDF allows the division of resources into groups called classes. The members of a class are known as instances of the class. Classes are themselves resources. The rdf:type property may be used to state that a resource is an instance of a class. A class is expressed as rdfs:Class in RDF.</td>
</tr>
<tr>
<td>COP</td>
<td>The Customer-Oriented Production application area that is the subject of WP7.</td>
</tr>
<tr>
<td>D2RQ</td>
<td>A tool for transforming information contained in a relational database into RDF. Described in detail in D4.4.1.</td>
</tr>
<tr>
<td>Dataset</td>
<td>A dataset is collection of related sets of information that is composed of separate elements but can be manipulated as a unit by a computer. A dataset is a set of information that is published, maintained or aggregated by a single provider in the context of Linked Data.</td>
</tr>
<tr>
<td>DHM</td>
<td>Acronym for Data Harmonisation Middleware</td>
</tr>
<tr>
<td>DHM-Adapter</td>
<td>Adapter to provide access to the Data Harmonisation Middleware as there are LiveData Access, control of Maintenance Test and diagnosis functionality.</td>
</tr>
<tr>
<td>DHM-JobController</td>
<td>The JobController is a component of the Data Harmonisation Adapter. It controls and if necessary blocks concurrent accesses to machines. It dispatches its commands to the components DHM-LiveData-Access or DHM-Tee or DHM-PAMMS</td>
</tr>
<tr>
<td>DHM-LDA</td>
<td>The Linked Data Adapter is a component of the Data Harmonisation Adapter. It is responsible for extracting the Machine Semantics from the Machines and publishing it to the Linked Data Server (LDS)</td>
</tr>
<tr>
<td>DHM-LiveData-Access</td>
<td>LiveData Access is a component of the Data Harmonisation Adapter. It provides access to live data like sensor and actuator data</td>
</tr>
<tr>
<td>DHM-MW-Log</td>
<td>The Middleware Log is a component of the Data Harmonisation Adapter. It collects all changes of the sensors and actuators in the Middleware and publishes them periodically to the History Data Store</td>
</tr>
<tr>
<td>DHM-PAMMS</td>
<td>Predictive Active Machine Maintenance System. Provides diagnosis functionality within the DHM-Adapter</td>
</tr>
<tr>
<td>DHM-Tee</td>
<td>The Test Execution Environment is a component of the Data Harmonisation Adapter. It provides control of Maintenance Test.</td>
</tr>
<tr>
<td>EPAMMS</td>
<td>Abbreviation for Extended Predictive Active Machine Mobile Maintenance System. Part of the Predictive Scenario. This is the Counterpart of a PAMMS located at the Service Company.</td>
</tr>
<tr>
<td>FOAF</td>
<td>The Friend of a Friend ontology that is used for representing personal data about people. Described in detail in D4.1.1.</td>
</tr>
<tr>
<td>Concept</td>
<td>Explanation</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Graph</strong></td>
<td>An RDF graph is a set of RDF triples. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph.</td>
</tr>
<tr>
<td><strong>Individuals</strong></td>
<td>Individuals are instances in RDF. Every resource which has a property rdf:type linking to an rdfs:Class is an individual.</td>
</tr>
<tr>
<td><strong>Jena</strong></td>
<td>Jena (<a href="http://jena.apache.org/">http://jena.apache.org/</a>) is a Java framework for building Semantic Web applications. Jena provides a collection of tools and Java libraries to help you to develop semantic web and linked-data apps, tools and servers.</td>
</tr>
<tr>
<td><strong>LDS</strong></td>
<td>Linked Data Server. Is responsible for providing the Machine Semantics to the Maintenance Personnel via the SupportApp.</td>
</tr>
<tr>
<td><strong>Linked Data</strong></td>
<td>Linked Data describes a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF.</td>
</tr>
<tr>
<td><strong>MMA</strong></td>
<td>The Mobile Maintenance application area that is the subject of WP8.</td>
</tr>
<tr>
<td><strong>Namespace</strong></td>
<td>Namespaces in RDF are used to distinguish between resources with the same (tag) name. To get the fully qualified URI of a resource, simply substitute the namespace prefix with the namespace URI.</td>
</tr>
<tr>
<td><strong>Ontology</strong></td>
<td>An ontology is a formal, explicit specification of a shared conceptualisation. In RDF ontologies contain information about classes and properties. RDF ontologies describe how these concepts have to be used for modelling individuals.</td>
</tr>
<tr>
<td><strong>Ontosketch</strong></td>
<td>A tool developed within the ComVantage project to support non-experts in extending ontologies.</td>
</tr>
<tr>
<td><strong>PEC</strong></td>
<td>The Plant Engineering and Commissioning application area that is the subject of WP6.</td>
</tr>
<tr>
<td><strong>Prefix</strong></td>
<td>Prefixes are used in RDF as an abbreviation for the namespace URI. To get the fully qualified URI, simply substitute the namespace prefix with the namespace URI.</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td>The concept of an RDF property describes a relation between subject resources and object resources. It is expressed as rdf:Property and is an instance of rdfs:Class.</td>
</tr>
<tr>
<td><strong>QUDT</strong></td>
<td>QUDT is short for: “Quantities, Units, Dimensions and Data Types in OWL and XML”. It is an ontology under construction (2014 currently version 1.2 is on the way). For more details have a look at: <a href="http://www.qudt.org/">http://www.qudt.org/</a></td>
</tr>
<tr>
<td><strong>RDF</strong></td>
<td>The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values.</td>
</tr>
<tr>
<td><strong>RDFS</strong></td>
<td>RDF Schema (RDFS) is a namespace for expressing basic concepts as classes and properties in RDF.</td>
</tr>
<tr>
<td><strong>Resource</strong></td>
<td>All things described by RDF are called resources, and are instances of the class rdfs:Resource. This is the class of everything. All other classes are subclasses of this class. rdfs:Resource is an instance of rdfs:Class. Usually resources are identified by URIs.</td>
</tr>
<tr>
<td>Concept</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RPPM</td>
<td>The RPPM-Ontology is short for Repair, Preventive and Predictive Maintenance (Scenario) Ontology. Its target is to provide an ontology to handle all three Maintenance Scenarios. For more have a look at this glossary on each of the three Maintenance Scenarios.</td>
</tr>
<tr>
<td>Semantic Web</td>
<td>The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on RDF.</td>
</tr>
<tr>
<td>Sensor</td>
<td>An Element that measures some physical Property. In the Machine Semantic Ontology describing the ComVantage Mobile Maintenance application area it is part of a Machine System.</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL (SPARQL Protocol and RDF Query Language) is the predominant query language in the semantic web. It is an RDF query language that is based graphs with powerful filter and aggregation functions.</td>
</tr>
<tr>
<td>SPARQL-Endpoint</td>
<td>A SPARQL Endpoint is a service endpoint which is able to process SPARQL queries. It is usually connected to a triple store holding the information.</td>
</tr>
<tr>
<td>SPARUL</td>
<td>SPARQL Update Language</td>
</tr>
<tr>
<td>SupportApp</td>
<td>The SupportApp is the Mobile User Interface of the Mobile Maintenance application area. It manages login and provides access to the LDS and the DHM-Adapter.</td>
</tr>
<tr>
<td>Taxonomy</td>
<td>Taxonomy describes the concept of identifying things in a specific domain and classifying them according to specific criteria. The arrangement in different categories and classes produces a hierarchical structure. Ontologies usually define a taxonomy of its containing concepts.</td>
</tr>
<tr>
<td>Tee</td>
<td>Abbreviation for Test Execution Environment.</td>
</tr>
<tr>
<td>Triple</td>
<td>The RDF data model is based upon the idea of making statements about resources (in particular Web resources) in the form of subject-predicate-object expressions. These expressions are known as triples in RDF terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. Databases capable of storing such data are called triplestores.</td>
</tr>
<tr>
<td>Triple store</td>
<td>A triple store is a database specialised for storing triples. Thus, it is core element for Linked Data application for static information.</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>Virtuoso is a server application with support for Features of SPARQL. Currently not the complete SPARQL 1.1 is provided.</td>
</tr>
</tbody>
</table>
DISCLAIMER

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.