284928

Collaborative Manufacturing Network for Competitive Advantage

D8.3.1 – Mobile Maintenance Adaption of Linked Data Integration Concept (public)

Grant Agreement No.	284928
Project acronym	*ComVantage*
Project title	Collaborative Manufacturing Network for Competitive Advantage
Deliverable number	D8.3.1
Deliverable name	Adaption of Linked Data Integration Concept
Version	v 1.0
Work package	WP 8 – Mobile Maintenance
Lead beneficiary	TUD
Authors	Markus Graube (TUD), Frank Haferkorn (RST)
Peer Reviewers	Werner Altmann (K&A), Jan Hladik (SAP)
Nature	R – Report
Dissemination level	PU – Public
Delivery date	31/10/2012 (M14)

Executive Summary

This deliverable gives an overview how the Linked Data concept can be integrated in the application area Mobile Maintenance of work package 8. Thus, the used ontology engineering methodology to derive formal vocabularies is explained followed by a description of the application area with special focus an Linked Data relevant issues. This includes the informal description of the relevant application area terms and the existing infrastructure. Afterwards all aspects regarding the technology are handled. The concepts and constraints of the Linked Data adapters to provide current and historical data are explained and are aligned with the overall architecture from work package 2. Next, the informal description of the application area is transformed into formal ontologies with a description of its structure and main concepts. The deliverable closes with a discussion of open issues and on-going steps.

Table of Contents

List of Figures

1 OVERVIEW

1.1 Introduction

ComVantage strives for a dynamic and flexible collaboration network between different stakeholders on the basis of Linked Data. Therefore, WP4 has provided its primary concepts, vocabularies, technologies as well as tools. This document represents the results on the integration process of the generic results of WP4 in the application area of *Mobile Maintenance*. This deliverable includes the mapping of the proposed Linked Data concepts to this application area as well as detected limitations and borders of this approach. For the latter cases possible adaptations and implementation will be provided.

1.2 Scope of this Document

According to the description of work, "Based on the generic concepts of the Linked Data integration (WP 4) modifications and extensions required for this application area will be conducted". More specifically, activities performed in this task will cover two basic aspects:

- Harmonising the Middleware data model with the Linked Data model
- Adapt the testing model to Linked Data model

Thus, the introduction is followed by a refinement of the ontology engineering method provided by WP4 in chapter 2. This was aligned with the other Linked Data adaption deliverables (D6.3.1 and D7.3.1). Chapter 3 deals with an explicit description of the application area of *Mobile Maintenance* with special respect to Linked Data relevant issues. It is characterised by a hierarchy of technical systems with different sensors and actuators which each holds transient data. Furthermore this chapter introduces the existing infrastructure for this work package. Chapter 4 aligns the requirements with the technical concepts from WP4. It presents architectural constraints and the necessary adaptations with a description of new implementations. The used vocabulary to provide all these process data is explained in chapter 5 which formulises the informal terms from chapter 3. Finally, the deliverable closes with brief conclusion and outlook of further activities in the task T8.3 and the tasks from WP4.

1.3 Related Documents

The recent document is heavily included in the overall project context and thus references many other deliverables. Furthermore, it will also be an input for further deliverables:

- The starting point for this deliverable to identify application area issues with respect to Linked Data concepts and relevance according the mockup prototype has been provided by deliverable D8.1.1: *Mobile Maintenance – Scenario Specification and Refinement*.
- The first generic concepts about Linked Data, its vocabularies and ontology engineering are coming from deliverable D4.1.1: *Data Format Specification*.
- The generic concepts of Linked Data middleware concepts and technologies from deliverable D4.2.1: *Middleware Adapter Set* are used as input for the adaptations.
- D4.4.1: *Linked Data Support* toolset handles as input for the use of possible tools as well as for the integration in the lifecycle.
- The alignment in the overall *ComVantage* architecture bases on the results of deliverable D2.2.1 *ComVantage* A*rchitecture Specification***.**
- Furthermore all Linked Data adaption deliverables (D6.3.1: *Plant Engineering and Commissioning - Adaption of Linked Data Integration Concept*, D7.3.1: *Customer-oriented Production – Adaption of Linked Data Integration Concept*, and D8.3.1: *Mobile Maintenance – Adaption of Linked Data Integration Concept*) are linked to some extent. All are aligned to their specific contexts, but certain aspects are cross-referenced like the ontology engineering methodology.

- Both the developed vocabulary and the adaptations are input for the tasks in WP4. These results will settle down in the upcoming deliverables D4.1.2: *Data Format Specification*, D4.2.2: *Middleware Adapter Set* and D4.4.2: *Linked Data Support Toolset*.

2 ONTOLOGY ENGINEERING METHODOLOGY

For the first experimental prototype described in the deliverables D4.1.1, D4.2.1 and D4.3.1, we generated the ontologies in a rather ad-hoc way. In a meeting between the domain experts and the ontology engineers, we examined the scenario descriptions and then discussed the terms to use and their relations. For the second prototype, it is appropriate to employ a more structured approach to ensure the quality of the developed ontologies. Specifically, it has to

- satisfy all requirements occurring in the current scenario descriptions,
- be easily adaptable and extensible if the requirements change in the future,
- be well documented so that also people who use the ontology after a long time or who were not involved in the development process can understand the meaning of the terms and their relations.

In order to support a structured ontology development process, several methodologies have been developed, and also endeavours have been made to compare these methodologies with respect to their capabilities, applicability in specific scenarios, and general advantages and disadvantages (Lopez, 1999). There have also been attempts to unify different ontologies and combine their advantages (Uschold, 1996).

2.1 Methodology Comparison

Among the methodologies that were considered in the comparisons mentioned above, three deserve a closer examination because they have been used in practice for different projects and reached a level of maturity that warrants applying them in the *ComVantage* prototype. These are described briefly in the following paragraphs.

- The methodology developed by Grüninger and Fox (Grüninger & Fox, Methodology for the design and evaluation of ontologies, 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:

1. 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.
2. Coding: representing the conceptualisation produced in the previous stage formally in an ontology language.
3. 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.

- The Methontology, 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 cannot be changed once the phase is finished.

  The phases identified by the Methontology developers are:

  1. Specification: production of a natural language document describing purpose, scope, and level of formality of the ontology.
  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 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.

Regarding the usefulness of the three methodologies described above for the *ComVantage* project, 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 Linked Data. 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 much 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. We therefore decided that this approach is also not appropriate 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.

We therefore decided in favour of the Enterprise methodology. It 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. We also decided to adopt the competency questions suggested in the TOVE methodology, since they can help to ensure that the ontology not only represents the domain of interest, but can also be used in practice to fulfil the tasks arising in the application area.

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.

In the following, we will describe the Enterprise methodology in more detail.

2.2 The Enterprise Methodology

2.2.1 Identifying Purpose

This phase comprises answering questions like “Why is the ontology being built?” “Who will use the ontology for which purpose?” as well as general scenarios in which the ontology is used. For the three application areas of the *ComVantage* project, a large part of this work has already been completed for the document of work and the deliverables 6.1.1, 7.1.1 and 8.1.1. For example, the ontologies are being built in order to facilitate integration of data coming from different collaboration partners; they will be used by the personas in the scenarios described in the respective deliverables.

2.2.2 Building the Ontology

2.2.2.1 Capturing

The Enterprise methodology suggests the following phases for capturing the ontology:

- Scoping, i.e. identifying the key concepts and relationships in the domain of interest. This can be achieved e.g. by performing brainstorming sessions to produce those concepts, grouping them in sets of similar terms, and identifying relations between them.
- Producing definitions, i.e. writing down in natural language what these concepts and relationships mean. These definitions should be as precise as possible and consistent with other concepts and their definitions.
- Identifying terms for these concepts and relationships. This step should be performed after reaching an agreement on the definitions since, as the authors put it, "terms get in the way": pre-conceived ideas about the meaning of terms can undermine communication and prevent agreement. Moreover, for ambiguous terms, the different possible meanings should be selected, and only those meanings (or the single meaning) actually needed within the ontology should actually be included. For these meanings, the original term should be avoided if at all possible to avoid further ambiguity.
- Competency questions, i.e. formulate queries that the applications to be developed within the *ComVantage* project will make and ensure that the ontology can answer them appropriately. This also serves to ensure that the terms contained in the ontology are actually needed: if they do not occur in any competency question, they can be dropped.

For these steps, a middle-out approach should be used. This means that the developers should neither start with the most general nor with the most specific concepts but rather with the intermediate ones, i.e. those that are used most frequently and intuitively by the people working in the corresponding domain. The disadvantage of the top-down approach is that one starts with defining very abstract concepts that make it difficult for the domain experts to relate the concepts with the entities occurring in their everyday work. The bottom-up approach on the other hand increases the overall effort since it makes it difficult to spot communalities between related concepts and thus leads to double effort and possibly later to re-engineering of the ontology. The middle-out approach avoids both of these pitfalls.

2.2.2.2 Coding

In this phase, the terms and definitions developed during the Capturing phase are used to formally define the ontology. In the Enterprise methodology, this also involves choosing an appropriate knowledge representation language; in the *ComVantage* project proposal the partners have already agreed to use the Linked Data principles, which implies using standards like RDFS/XML (which includes the possibility of using OWL), hence there is no real decision to be made.

It is also necessary to choose the tools used for the development of the formal ontology. In *ComVantage*, we used the Protégé editor as we did in D4.1.1 for the first experimental ontology, since it satisfied all our requirements.

Although it is possible to merge the Capturing and the Coding phase into one and thus produce the ontology on the fly, this is discouraged. Since the domain experts are heavily involved in the Capturing phase, but to a much lesser extent in the Coding phase, where the ontology engineers do most of the work, this also seems appropriate for *ComVantage*.

2.2.2.3 Integrating Existing Ontologies

The developers of the Enterprise methodology consider giving useful guidance for this step as one of the biggest challenges in developing a comprehensive methodology. Consequently, they do not give clear guidance on how to proceed. Especially for cases where there are concepts in existing ontologies that are

closely related with the concepts identified for the new ontology but not identical, it is difficult to give general advice, especially if the ontology is intended to be published, because this would result in two competing concepts, which will appear to be synonymous for many users, and might even have the same identifier, except for the prefix. Since most of the *ComVantage* ontologies are not intended for publishing, this is not a major problem for this project. If we cannot find an existing term that satisfies our requirements, nothing speaks against creating a new, more appropriate term.

2.2.3 Evaluation

General criteria suggested for evaluation of the developed ontology are:

- Clarity: formal as well as natural-language definitions should be unambiguous, which can be supported by providing positive or negative examples. Underlying assumptions should be stated explicitly.
- Consistency and coherence: the ontology has to be internally consistent, i.e. free of contradictions, and also conform to external use of the terms. Circular definitions should be avoided.
- Extensibility and reusability: it is important to achieve the right balance between being specific enough for the required tasks, but not so specific to render the ontology useless for other tasks. Definitions for a term should not be biased, i.e. influenced by the specific environment of the ontology developers.

More specifically, the ontology should be able to answer all competency questions specified in the capturing phase. In order to avoid redundancy, one can also check if all terms contained in the ontology are necessary for answering at least one competency question, and remove those terms that are not.

2.2.4 Documentation

The Enterprise methodology does not provide clear guidelines for either the steps that should be documented or the documentation method. It is merely stated that both the formal and informal ontology should be documented since this is important for reuse.

In *ComVantage*, we will use the text deliverables D7.1.x and D7.3.x for documenting the informal natural-language ontologies and the RDFS vocabulary for documenting the formal ontologies.

3 APPLICATION AREA DESCRIPTION

This chapter describes the application area of work package 8 *Mobile Maintenance* with special respect to Linked Data related issues. This includes detailed scenario descriptions, the definition of the important terms as well as the existing infrastructure.

3.1 Terminology

3.1.1 Detailed Scenario Descriptions

3.1.1.1 Corrective Maintenance Scenario

The *Corrective Maintenance Scenario*, also called *Repair Scenario* is relevant for one or more factories with several machines each. **Error! Reference source not found.** shows such a scenario with a *Customer-Factory-* and a *Service Company* involved.

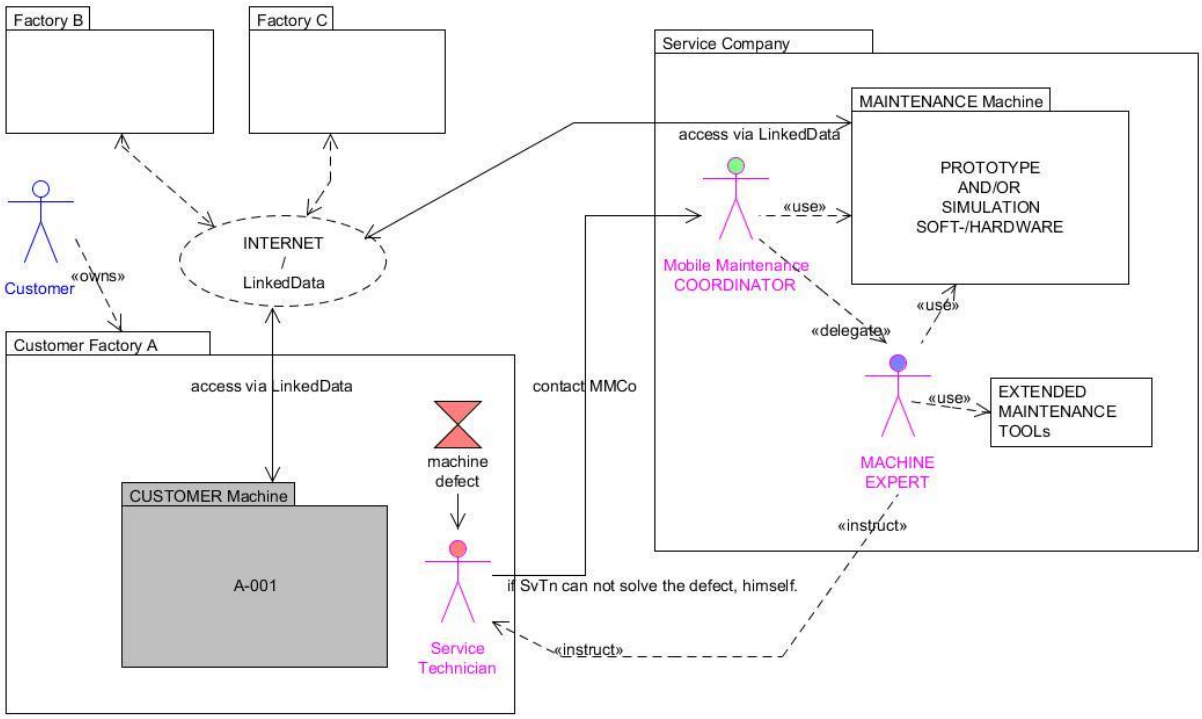

Figure 1:Corrective Maintenance Scenario

If a Machine[1] Defect (red) occurs, the customer (blue, at the top left side) reports the Machine Defect (red) to the Service Company. At first, a local Service Technician (SvTn, magenta with red head) tries to solve the defect, by himself. If he fails to solve the defect, he contacts a Mobile Maintenance Coordinator (MMCo, magenta with green head) at the Service Company. The MMCo has access to all relevant data about the Machine Defect. After a problem analysis, the MMCo delegates the problem to an appropriate Machine Expert (ME, magenta with blue head), who has access to Extended Maintenance Tools to analyse and solve the Machine Defect. The Machine Expert instructs the Service Technician, who is on-site at the Customer Factory. The Service Technician solves the defect with the help of the Machine Expert.

There is also the possibility that a Service Technician initiates the Corrective Scenario. In this case not the Customer, but an on-site Service Technician detects a Machine Defect and reports it to the MMCo at the Service Company. Then this alternative scenario follows the above described scenario.

[1] In general the word *machine* is written in lower cases. But in special contexts (*Machine Defect*, *Machine Expert*, or Active Machine, etc.) upper case initials are used.

3.1.1.2 Predictive Maintenance Scenario

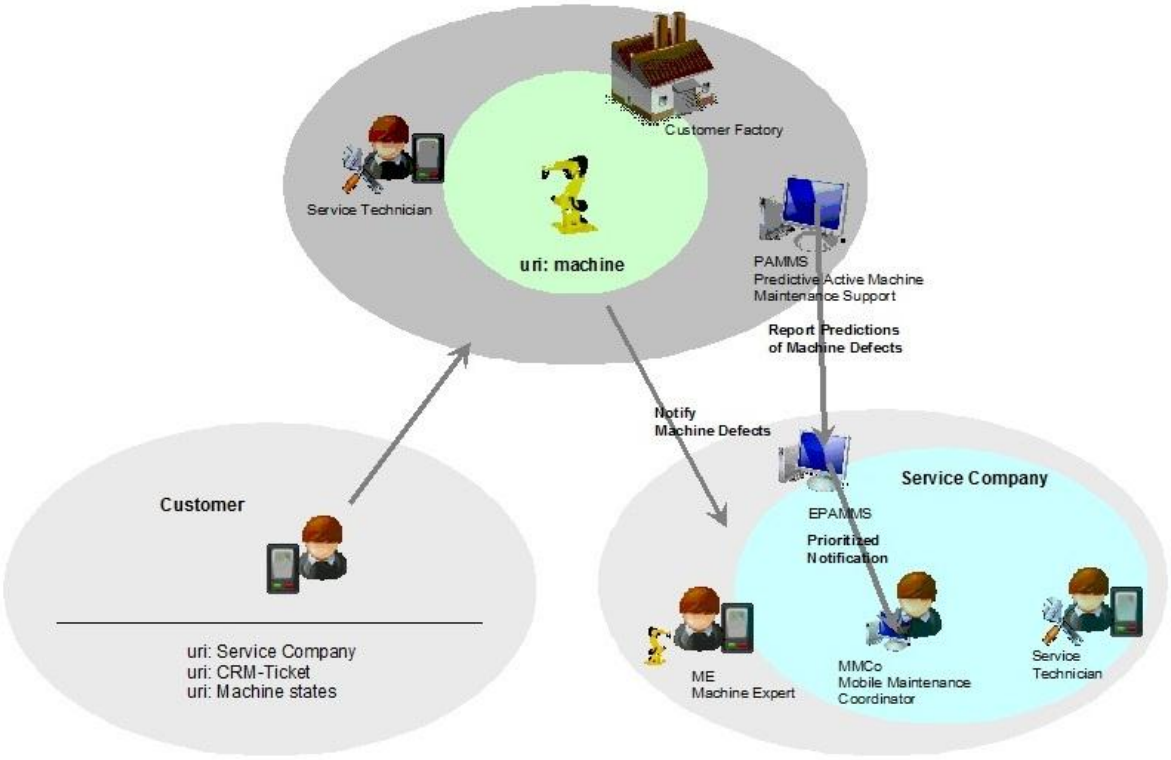

Figure 2: *ComVantage* Predictive Mobile Maintenance Scenario Vision

The intention of a *Predictive Maintenance Scenario,* depicted in Figure 2*,* is to predict and **fix a Machine Defect before it occurs** in order to avoid machines to break down.
This prediction is done by the *Active Machine* with the help of the component *Predictive Active Machine Maintenance Support (PAMMS)*. It evaluates the machine's sensors and process data and thus is event-driven. By avoiding machine breakdowns, blocking of complete machine lines in production is prohibited.

The *PAMMS* transmits the predictions to the component *Extended Predictive Active Machine Maintenance Support (EPAMMS)*, which is assigned to the Service Company. The EPAMMS notifies the Mobile Maintenance Coordinator (MMCo) about relevant predictions. The MMCo at the Service Company decides to start a Repair Scenario if necessary and starts a Corrective Scenario as described in chapter 3.1.1.1.

3.1.2 Application Area Terms

The most important application area terms for the Corrective and Preventive Maintenance Scenario are the following. They will later be, in chapter 4, integrated in the developed ontologies for WP8.

- Collaborative Partners:
  - The *Customer Factory* is owned by customer and has a Service Level Agreement (SLA) with the Service Company.
  - The *Service Company* is responsible for Maintenance Services for the Customer Factory. Some Service Technicians and Machine Expert are located at this company.
  - The *Machine Producer Company* developed and produced the machine. It is the last line of know how to find a Machine Defect. Some Machine Experts work for it.

- Human Actors
  - The *Customer* at the *Customers Factory* owns his factory and reports of an occurred Machine Defect to the Support Company
  - In the Corrective Maintenance Scenario the *Mobile Maintenance Coordinator* (MMCo) is notified about a Machine Defect and coordinates the involvement of adequate Service Technician(s) and/or a Machine Expert(s).

    In the Predictive Maintenance Scenario the *Mobile Maintenance Coordinator* is notified about predicted Machine Defects by the *EPAMMS*. For the predictions a classification and a prioritisation is generated (by probability, severity and impact). The MMCo has to decide whether and what measures have to be taken. If he decides so, he has to start a maintenance activity.
  - The *Machine Expert* (ME) is involved by the MMCo and has access to extended maintenance tools and expert know-how. He solves the problem in interaction with the (on-site) *Service Technician*.
  - The *Service Technician* (SvTn) is on-site at the Customers Company and in a first step he tries to solve the problem of the Machine Defect, himself. If he cannot solve the problem himself, he reports the Machine Defect to the Service Company and is instructed by the Machine Expert to solve the Machine Defect.
- Technical Actor
  - The *Active Machine is* the central part of the *Mobile Maintenance* application area. It provides the Maintenance Personnel with Technical Information about its internal Machine Configuration and contains descriptions and measures to access *sensor* and *actuator* hardware. In the Predictive Maintenance Scenario, the PAMMS is part of the Active Machine.
  - *The Machine Semantics* is the ontology that is use to describe the structure of the Customer Factory broken down from the Enterprise to sensor and actuators.
  - The *Predictive Active Machine Maintenance Support (PAMMS*) is part of the Active Machine. It is part of a system to detect and solve correct Machine Defects before they lead to production downtime. The *PAMMS* reads and analyses relevant sensor Information and actuator feedback. It tries to predict upcoming Machine Defects, before they occur. It sends the data to the *EPAMMS* located at the Service Company.
  - The *Extended Predictive Active Machine Maintenance Support (EPAMMS*) is a Technical Component at the Service Company. It receives data from the *PAMMS* at the Active Machines of all Factories of all Customers with valid Maintenance Agreements.

    This PAMMS data is analysed in order to detect upcoming defects before the defect has an impact on production. If an upcoming defect is detected, the defect is classified and prioritised (by probability, severity and impact) and sent to the human actor Mobile Maintenance Coordinator. He decides which Corrective Activities are taken (e.g. to exchange the affected (defective) parts during one of the next regular maintenance intervals).

    The data received from the PAMMS is analysed per single machine and additionally over all machines at all factories of all customers. This allows cross-factory analysis of each type of machine and can prevent the Service Companies from expensive product recall activities.
- Machine Detail Terms
  - The *Machine Park* is the set of all Machines at a Customer Factory.

- The *Machine Defect* is a malfunction of a machine at the Customer Factory. It might lead to breakdowns and to unplanned downtimes. As this is a big cost factor, Machine Defects should be detected and solved as soon as possible. The goal of Predictive Maintenance is to detect a defect and correct it before it has an impact and leads to machine downtimes.
- A *Solution* of a Machine Defect is the set of correction activities taken to fix a Machine Defect.
- A *Spare Part* is an interchangeable part that is kept in an inventory and used for the repair or replacement of failed parts[2].
- One or several *Defective Components* are the origin of a Machine Defect. If all *Defect Components* are replaced, the Machine Defect is solved.
- *Description of Machine's problem*: The Operator that has solved a Machine Defect has to describe the problem of the machine and the measures taken to solve the problem.
- A *Maintenance Test Case* Is a Test Script to verify correct operation of a (Active) Machine under Maintenance
- The *Identification Abilities* of the SupportApp of a Machine are via manual Entry, QR-Code, GPS and Camera
- A sensor is a subcomponent of a machine. "*It is a converter that measures a physical quantity and converts it into a signal which can be read by an observer or by an (today mostly electronic) instrument.*"[3] It may be a sensor, measuring a temperature or electric current or it might be a photo detector used for counting events.
- An actuator is a subcomponent of a machine, too. It is a technical component that allows "*controlling a mechanism or system*".[4] It may e.g. be a motor or a temperature control element. An actuator might provide have readable information like a sensor, too.
- Testing and Simulation Equipment
  - This term is specified in *ComVantage* Delivery D8.1.1 (D8.1.1, 2012):

    "In a Maintenance Scenario the Machine Expert uses his elaborated experience to solve the problem. All data concerning the service call, in particular acquired via the Linked Data Interface are available for the Machine Expert. After a first analysis he either solves the problem right now by delivering spare parts needed and instructing the Service Technician how to repair the machine. If a first analysis fails, he uses specialised Testing and Simulation Equipment and Compares the Simulated data to the data of the defect component."

3.2 Infrastructure Available before Mobile Maintenance Application Area

Before the adaption of to the Mobile Maintenance application area the already existing maintenance infrastructure had these components:

- Customers Factory
  - The Customers Factory is the site where the machines to be maintained is located.

[2] As described in http://en.wikipedia.org/wiki/Spare_part

[3] As described in http://en.wikipedia.org/wiki/Sensor

[4] As described in http://en.wikipedia.org/wiki/Actuator

- Machine(s)
  - A Customers Factory consists of several machines, each containing one or more sensor(s) and actuator(s), which are accessed directly by special maintenance tools.
  - Almost each machine type has its own maintenance tools like proprietary measuring and control equipment.
  - Almost each machine type has its own maintenance methodology.
- Maintenance Personnel
  - The maintenance personnel are responsible for maintenance (detecting and solving machine defects).
  - For each machine the personnel has to be trained separately, as the maintenance tools and used methodologies vary very strong over the different machine types. This leads to expensive and long training periods.

3.3 Future Infrastructure with Adaption of Mobile Maintenance Application Area

The adaption of the Mobile Maintenance application area has these components:

The Maintenance Personnel can do all maintenance activities based on a single, mobile device, the SupportApp. This reduces the cost of the training periods.

3.3.1 Servers

- SPARQL –History Data Store (Virtuoso Server)
- SPARQL –Linked Data Endpoint (Virtuoso Server)
- DHM-Adapter (based on Hiawatha Server)
- SupportApp is a small application on a Mobile Device used by the Maintenance Personnel (see also section 3.3.3)

3.3.2 Interfaces

- LDA to LDS
  - Extract the Machine Semantics from Machine Configuration
  - Publish it to the Linked Data Server (LDS)
- SupportApp reads from LDS
  - The SupportApp retrieves all relevant information about the currently maintained machine
- SupportApp accesses the DHM-Adapter
  - The SupportApp retrieves Live Data from the machines sensors
  - Or sets Live Data at the machines actuators
- SupportApp reads History Data
  - The SupportApp retrieves History Data about the currently maintained Machine.

3.3.3 Specific Hardware

- SupportApp
  - The SupportApp is the small application on a Mobile Device used by the Maintenance Personnel. It runs on a mobile tablet and is the central maintenance tool for the Mobile Maintenance application area.
- LDS – Linked Data Server
  - The Linked Data Server provides all available Machine Semantic of the intended Machine. It contains information about available sensors and actuators as well as information about available Maintenance Test-Scripts. It is accessed by the mobile device SupportApp. Because of security issues the personnel has to log in, first.
- DHM-Adapter - Data Harmonisation Adapter
  - The Data Harmonisation Adapter provides for the currently maintained machine access to the Live Data and control on Maintenance Test Scripts. It is accessed by the mobile device SupportApp.
- Middleware Controller
  - The Middleware Controller is the Data Abstraction Layer between the machines and the DHM-Adapter Application. In this context it is implemented using RST's GAMMA V [5]
- Machine(s)
  - The machine(s) contain(s) one or several sensors and/or actuators and are accessed via the DHM-Adapter using the Middleware Controller, which is accessed by the mobile device SupportApp.

4 LINKED DATA ADAPTERS

The Linked Data adapters for the application area Mobile Maintenance have to provide all relevant information about factories, machines and their states, personnel, maintenance issues and error handlings in RDF. All of this information is strongly interconnected. Since most of the data has a rather static nature, it can be stored in standard triple store and thus integrates very easily in the overall *ComVantage* concept from WP2.

The main challenge is to provide the huge amount of dynamic data for the end user applications. In a typical factory there exist thousands of machines and equipment each having one or more sensors and actuators. The maintenances worker in the field needs to know the current status of the machine he or she works on. Thus, he needs at least reading access to the sensors of this machine and probably also to sensors which are somehow connected to this machine. Since the values of the sensors are constantly changing, we call this data transient.

4.1 Transient Data in Linked Data

Linked Data has its main purpose in handling static data. But the application area of Mobile Maintenance requires the access to thousands of constantly changing process variables. This transient data from several types of sensors and actuators raises several new challenges to the existing Linked Data infrastructure.

The simple approach is pushing the data from the sensors to a Linked Data server at defined time intervals and overwriting the existing data. This way the server contains the most recent data after every update interval. This approach has very strong limitations. We conducted several benchmarks storing RDF files with

[5] See http://www.rst-automation.de/gamma-v-5

different sizes in a Virtuoso server. As **Error! Reference source not found.** shows, there is almost a linear elation between number of stored triples and the time needed for this operation. This linearity is also independent of the used method to store the triples in Virtuoso. But you can see that the Update via SPARUL (SPARQL/Update), marked as HTTP-SPARQL and ISQL-SPARQL, is about ten times slower than the native built-in methods of Virtuoso (ISQL-N3 and ISQL-TTL) which use the command DB.DBA.TTLP_MT. Both of them need a direct ISQL connection to the server.

If we assume that a typical factory has 1,000 sensors which have an update interval of 10 ms and we only want to store the current values, we would need 100.000 triples to be updated per second. In the tested configuration it was only possible to update about 15.000 triples in a second with the fastest function. The increase of the computing performance of the server is not a suitable solution since the change to higher update rate will again force us to increase computing performance. Furthermore we need also some computing time not only to store the RDF information but for our main task to answer SPARQL queries on this data.

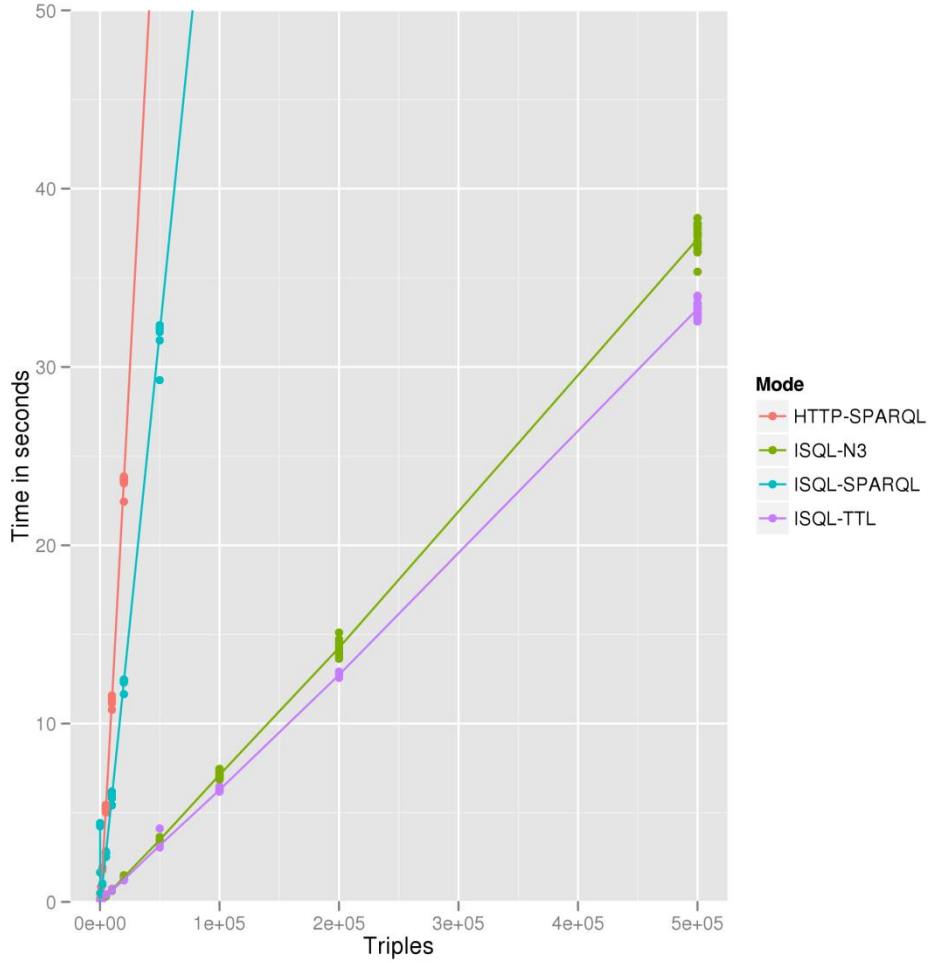

Figure 3: Benchmark for Updating RDF Triples

As further benchmark we investigated how Virtuoso scales if we upload not one single big RDF file but several small ones in several interactions like this would be in an environment where each machine updates the triple store with its own information on its own. For this purpose we created several random RDF files with each consisting of 10 triples representing a single process variable. Now we compared the time needed for an update of these files to different named graphs on the Virtuoso in one big transaction against several transactions with file per connection. The results depicted in Figure 4 show that the performance is about 3 times lower than in the first case. This makes the simple approach even more

unrealistic. Furthermore in the configuration for both benchmarks, the network latency was not considered at all because all processes run on the same machine. All in all, this simple approach doesn't work in this application area.

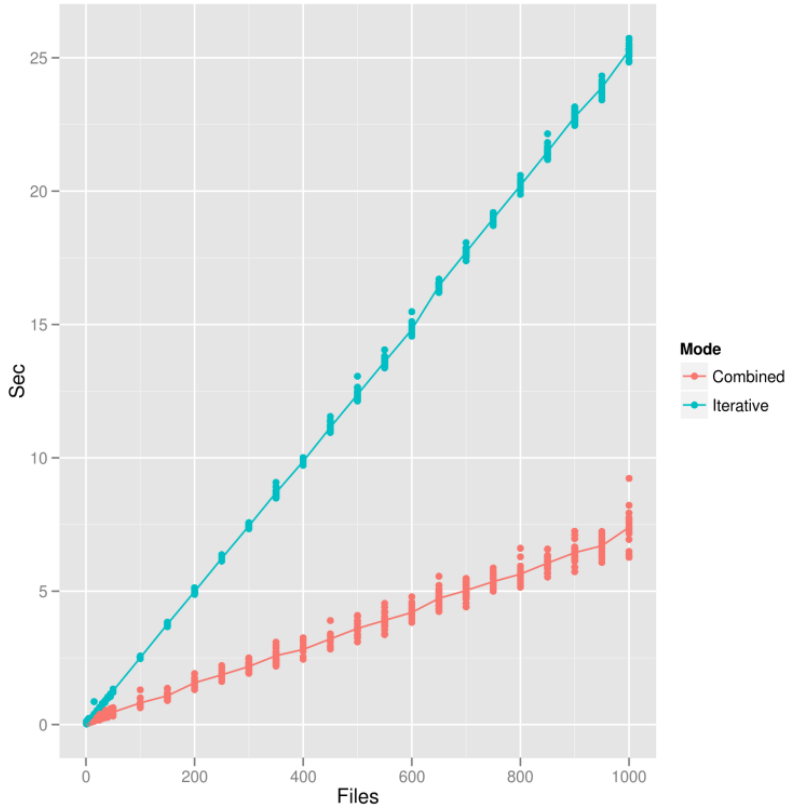

Figure 4: Combined vs. Iterative RDF Updates

Therefore we decided to use another approach which is shown in **Error! Reference source not found.**. ere, the triple store as part of the Linked Enterprise Data (LED) cloud holds only the static information about all machines. The dynamic data of the sensors is not stored in this place but can directly be fetched via a DHM Adapter (Data Harmonisation Middleware Adapter). This adapter is connected to the common Middleware of the different sensors and actuators and provides the current data in a harmonised format as RDF abstracted from the specific Middleware. The DHM Adapter offers this information via a HTTP interface which can be enclosed in the security mechanisms of the overall architecture from WP2. The information about the current state of a sensor or actuator is returned either as RDF in a suitable serialisation or as plain text for debugging purposes regarding the HTTP GET request settings. Even the specific address or namespace of an actuator is provided in the GET variables of the HTTP Call.

Thus, an end user application which wants to know the current data of a specific sensor has to contact the DHM Adapter with a HTTP GET request URI determining the specific sensor. The application itself doesn't need to store or compute this information about the correct namespace in the Middleware for this specific sensor. This Machine Semantic "knowledge" can be queried from the LED cloud which knows the structure of the machines and their sensors with their DHM adapter URIs which also enables several DHM adapters running in parallel on different servers.

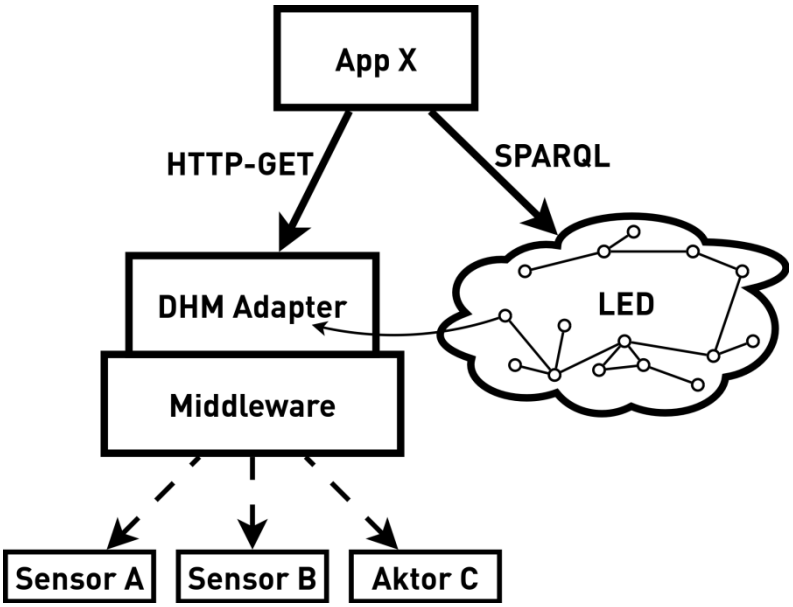

Figure 5: Transient Data Concept for Linked Data

This approach integrates well in the WP4 Linked Data concept. The information about the current state of process variables is provided as RDF and thus can be easily interlinked with other RDF information from the collaboration network. A small disadvantage can be seen in the fact that information about these transient data cannot directly be queried via SPARQL request. But a solution for this problem is under development. Virtuoso supports a feature called *Sponger* which includes a SPARQL extension for URI dereferencing of variables. This allows the fetching of additional RDF-resources and also non-RDF-resources in the execution time of the SPARQL request. This way the URIs for the current state of some sensors could be queried in a SPARQL request and then in the same SPARQL query the information from these URIs can be fetched and processed in the SPARQL query. First experiences show the approach working but further investigation will be made in WP4 with results in the upcoming deliverable D4.2.2.

4.2 Architecture of the Linked Data Adapter

4.2.1 Overall Architecture from WP2

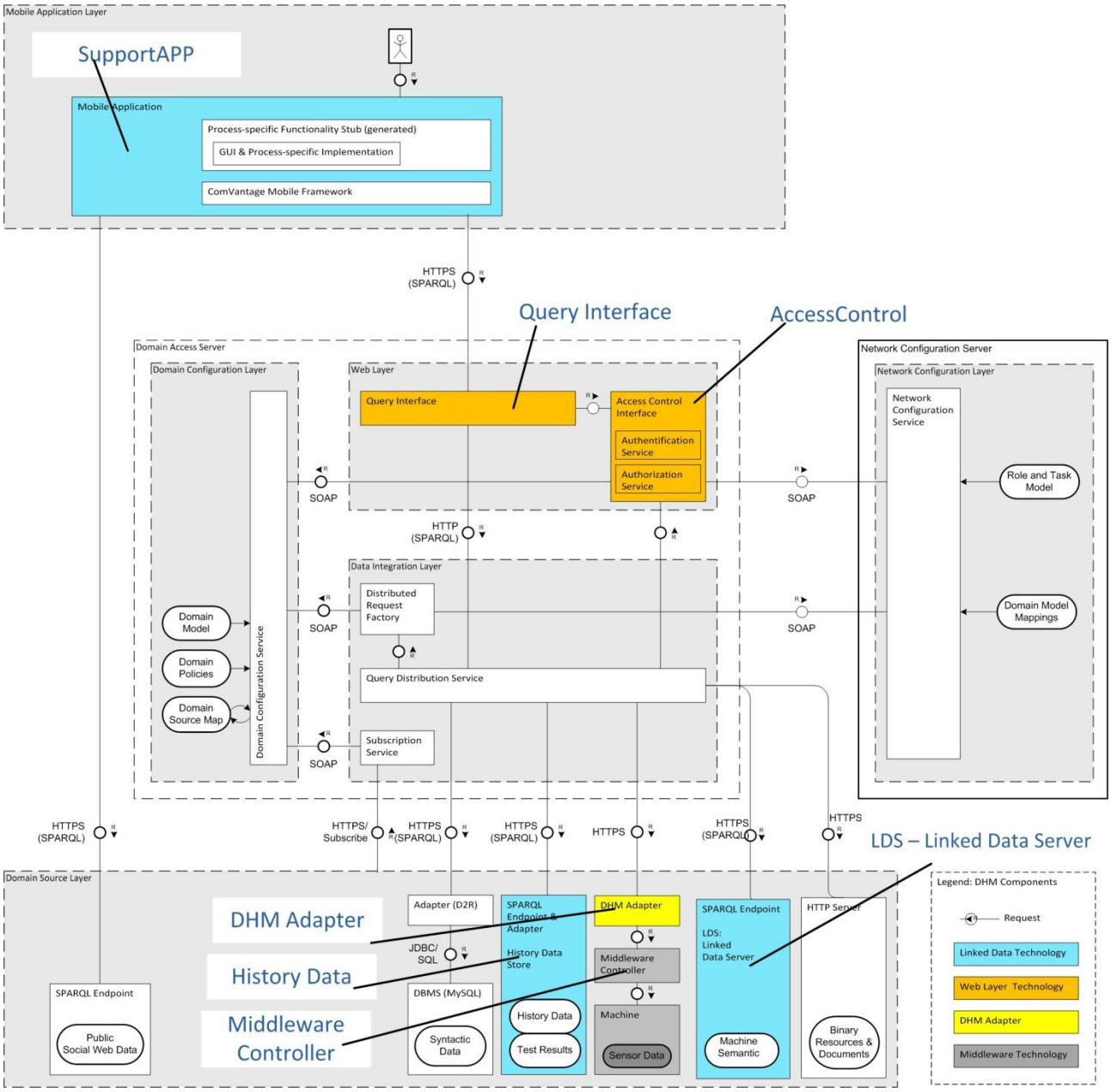

Figure 6: Overall WP2 Architecture, with coloured components of WP8

The WP8 architecture is well aligned to the overall *ComVantage* architecture. It has these main WP8 components, also shown in **Error! Reference source not found.**:

- The *SupportApp*, used by the user to do a Maintenance Task in a Repair Scenario (cyan, at the top)
- The *Query Interface* and the *Access Control* (orange, at the centre-top)
- The Linked Data Server (LDS) to store and forward the Machine Semantics (cyan, at the bottom-right)
- The DHM Middleware Controller and the machines (deep grey, at the bottom)

- The SPARQL Endpoint to store and access History Data (cyan, at the bottom)

For more details see below.

4.2.2 Complete WP8 Architecture

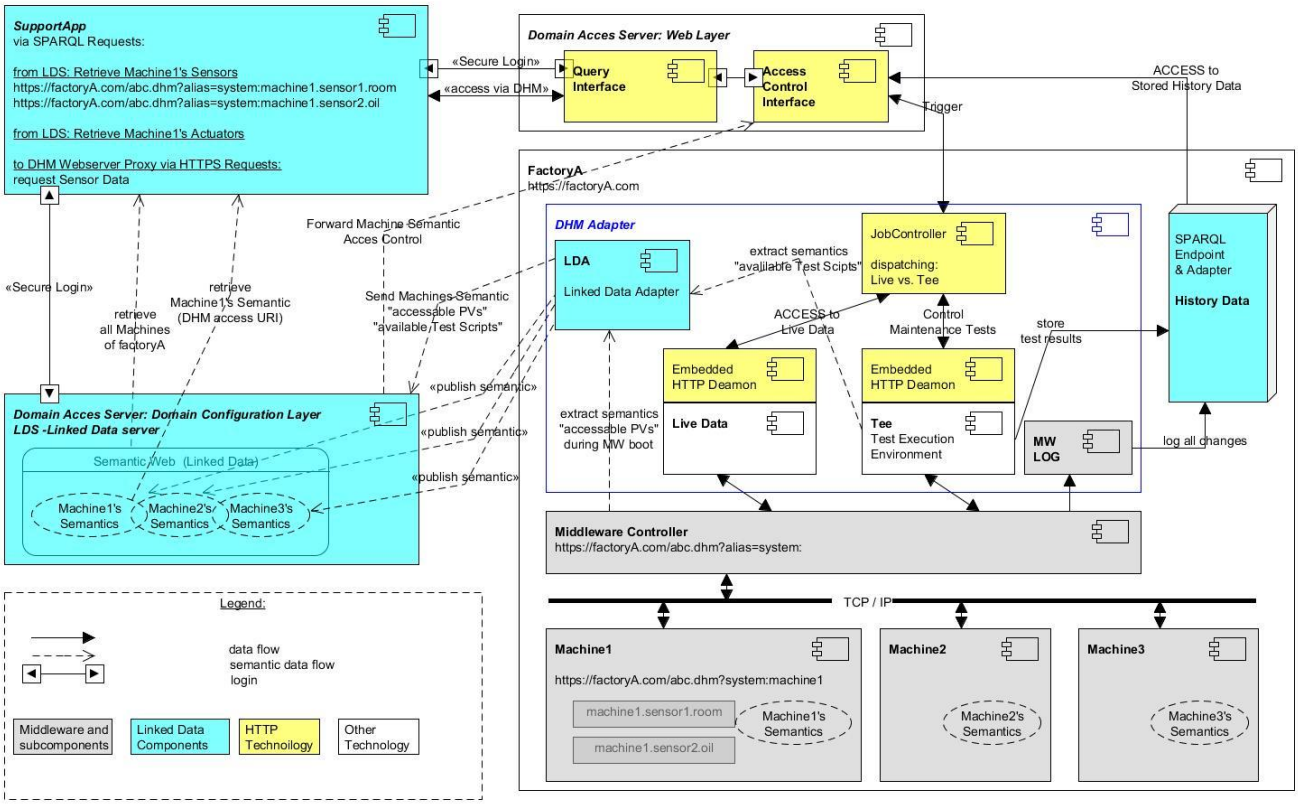

Figure 7: Detailed Architecture of Implementation of Prototype P-I

Here is shown the environment of the DHM Adapter (blue outline, at the right-centre) of the WP8 Architecture in a more detailed view:

- A *Middleware* (MW) controls one or more machines (marked grey, at the right bottom). These provide *sensors*/*actuators* that are described by a *Machine Semantics* for each machine.
- The *Linked Data Server (LDS)* (marked cyan, at the left bottom) receives the Machine Semantics during boot time in a machine readable format (RDF) from the *DHM-Adapter* subcomponent *LDA*.
- A *SupportApp* (marked cyan, at the top) is the User Interface to do the task of *Mobile Maintenance*. It retrieves all relevant *Machine Semantics* about the machine to be maintained from the *LDS.*
- The *Query Interface* and the *Access Control* Access controls and restricts the access of the *SupportApp*. These are part of the *Web Layer* in the *Domain Access Server* (marked yellow, at centre-top)
- If access is granted, History Data (marked cyan, at the right) can be retrieved using a SPARQL query. History Data comprises data logged by the MW Log and old test results stored during runtime. It is also possible to retrieve Live Data (like current sensor and actuator values) and to run maintenance test-scripts using Test Execution Environment (Tee). Yet, Live data retrieval and test runs need to be coordinated to avoid concurrency conflicts. This is the job controller's (JC) responsibility.

- After passing the *Access Control* Requirements
  - the *DHM-Adapter* (blue lined box at the centre) involves the JobController (JC) to check concurrency conflicts and to coordinates
    - retrieving *Live Data* (like the sensor and actuator Values) and running *Test Execution Environment (Tee)* Maintenance Test-Scripts and
    - retrieving some *History Data* (marked cyan, at the right) logged by the *MW Log* during runtime and

  All data is stored in a triple store and can be retrieved via SPARQL queries.

4.2.3 Architectural Details of DHM-Adapter

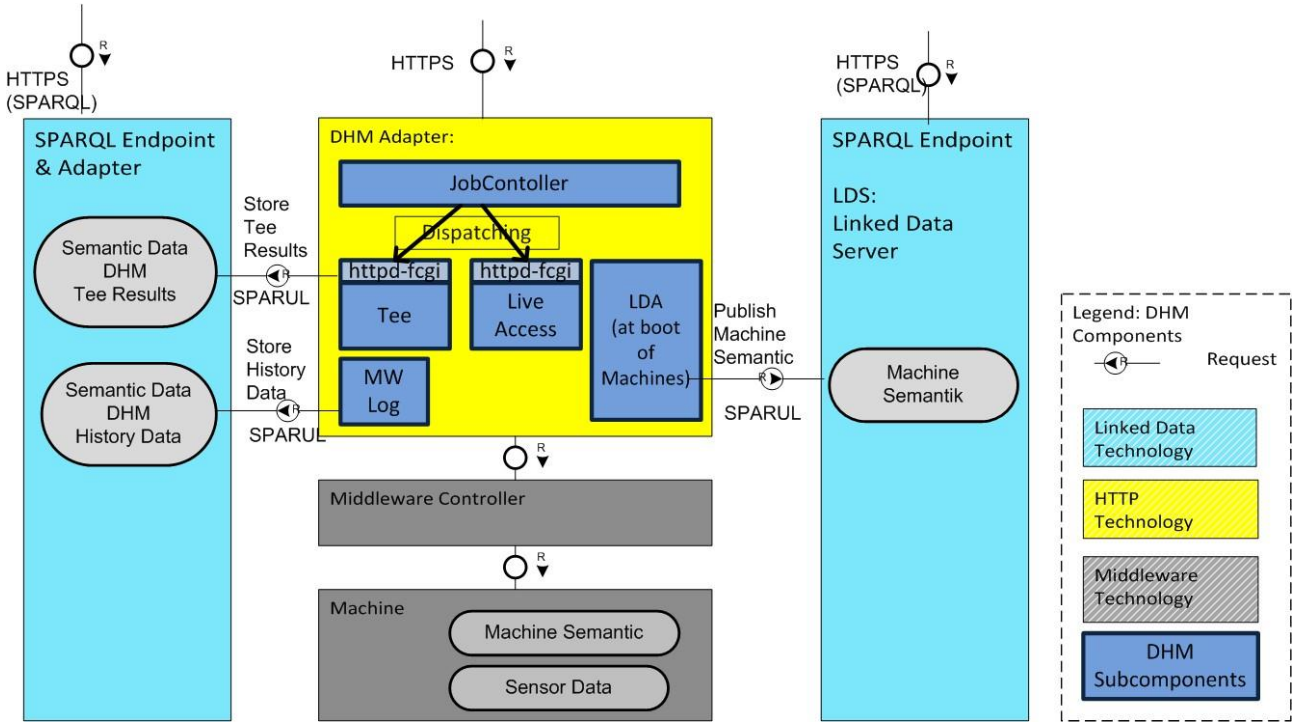

Figure 8: Detailed Architecture of the DHM-Adapter

The architecture of the DHM adapter has been implemented in a Prototype P-I whose structure is shown in **Error! Reference source not found.**. The DHM adapter accesses an underlying Middleware that controls ne or more machines (marked as grey).

In order to provide History Data[6], all changes of Process Variables of the Middleware are logged into a Linked Data Database (marked as cyan). The stored data is called *DHM History Data*. It is accessible via SPARQL Requests addressed to the SPARQL-Endpoint of that Database (marked cyan, at the left). The subcomponent of the DHM Adapter *MW Log* is listening to the Middleware for all changes and stores them periodically into the database via SPARUL Command.

At boot time of the machines controlled by the Middleware, the Linked Data Adapter (LDA) extracts the Machine Semantics for the machine(s). It publishes it to the Linked Data Server (LDS)

During runtime, the DHM Adapter is receiving commands via HTTP Requests. The subcomponent JobController controls and if necessary blocks concurrent access requests. It dispatches its commands to

[6] History Data is a name for earlier logged data, stored with timestamp. It is generated by the MW Log component of the DHM-Adapter.

components via an embedded FCGI[7] http daemon. The Access to current, live data is done via the DHM-Component *Live Access.* The component *Test Execution Environment (Tee*) is used to start and control Maintenance Scripts*,* named *Test Cases*.

4.3 Interface and Integration with Ontologies

The DHM Adapter the LDS and the History Data make use of the DS ontology as well as of the MSem Ontology in order to describe details of the machine structure and information about the transient data. Chapter 5 will explain these ontologies in detail. For details of the components SupportApp, LDS and DHM-Adapter, please have a look above at chapter 0.

4.3.1 Interfacing with the Linked Data Server

4.3.1.1 Exporting the Machine Semantics from the LDA to the LDS

- At the boot time of each machine, the Machine Semantics from Machine Configuration is extracted. This contains the structure of the Enterprise and for each machine all available sensors and actuators and the required access levels.
- The extracted information is expressed using the MSem semantics ontology in RDF.
- The extracted Machine Semantics is published to the Linked Data Server (LDS).

4.3.1.2 SupportApp Logs in at LDS

- The Maintenance Personnel has to use the SupportApp to be authenticated and authorised before accessing the LDS. This security issue is addressed in WP3.

4.3.1.3 SupportApp Retrieves Machine Semantics from LDS

- The SupportApp retrieves all relevant information about the currently maintained machine from the LDS.
- The MSem and DS semantics are used.

4.3.2 Interfacing the DHM Adapter

4.3.2.1 SupportApp Logs in at the DHM-Adapter

- The Maintenance Personnel has to use the SupportApp to authenticate and authorise before accessing the LDHM-Adapter. This security issue is addressed in WP3.

4.3.2.2 SupportApp Reads Live Data

- The SupportApp retrieves Live Data from the machines sensors or it reads/writes to machines actuators.
- The DS ontology is used to represent the results of Live Data.
- The DS semantics is used.

4.3.2.3 SupportApp Controls Maintenance Test Scripts

- The SupportApp can start, pause and stop Maintenance Test Scripts at the DHM-Adapter.
- All test activities as well as the test results are expressed with the TEE ontology.

[7] FCGI (FastCGI) is a protocol for webserver to interface interactive programs.

4.3.2.4 Job Controlling

- The Job Controller dispatches the Access to the machines at the DHM-Adapter.
- In order to beware of illegal, parallel access to the same machine and especially parallel writing access to the same Machine during a run of a Test Script, the JobController has to block conflicting requests.

4.3.3 Interfacing History Data

4.3.3.1 Reading History Data

- The SupportApp can retrieve History Data about a currently maintained machine from a SPARQL endpoint. The endpoint is connected to a triple store that is filled by the Middleware Log (MW-Log) as another component of the DHM-Adapter. The DS vocabulary is used to represent the retrieved History Data of sensors and actuators.
- The History Data provides also access to the results of all finished Maintenance Test Scripts. DS and MSem vocabulary is used to represent the Maintenance Test Results.

5 FORMAL VOCABULARY

The Mobile Maintenance application area of WP8 requires several well defined vocabularies in order to be able to express all relevant information for humans as well as machines. For that reason the following ontologies have been developed according to the process described in chapter 2:

- Data Series Ontology (DS)
- Machine Semantics (MSem)
- Repair and Predictive Maintenance Ontology (RPM)

All ontologies are in a rather early stage and might change in the future. Each ontology is described in the following subsection explaining its scope, purpose and main concepts. The ontologies have been developed in the OWL language with the tool support of Protégé.

5.1 Data Series Ontology (DS)

The Data Series ontology handles all aspects regarding the presentation of data series in Linked Data. It is suitable for expressing measurements in a time series with multiple dimensions. The prefix *ds* is used for this ontology as abbreviation for Data Series.

5.1.1 Domain and Scope of the DS Ontology

- What is the domain that the ontology will cover?
  - This domain will cover storing accessing and processing of small to large series of data.
- For what we are going to use the ontology?
  - It will be used in WP8 *Mobile Maintenance* and the related prototype. As this is a very general purpose ontology, it can be used for any purpose of storing sequential data series, which require a fast (indexed) access to the data.
- For what types of questions should the information in the ontology provide answers?
  - It allows efficient access to sequential data elements stored.
  - It should be optimised for accessing to the last elements in the data queue.

- Who will use and maintain the ontology?
    - The Data Series Ontology will be maintained by the *ComVantage* project, especially by RST & TUD. It will also be used by these stakeholders, but should be applicable to a broader range of applications.

5.1.2 DS Competency Questions

- Given: A Data Series of a fill level of depending on time t with 1hour of data.
    - What is the first/last value of fill level at time t?
        - What is the first value of fill level?
        - What is the last value of fill level?
        - What are the minimum and maximum values of fill level?
    - What are the data series of x between t_0 and t_1?
    - In which physical property is fill level measured?
    - In which physical unit is fill level measured?
    - What is the latest fill level of sensor A in inch as unit?
- What is the arithmetic mean of the fill level of sensor A over the last two hours?
- Merge two (or more) time series x(t), y(t) to a single xy diagrams
- What is the maximum/minimum value x in the given dataset x(t)

5.1.3 DS Elements

The goal of the DS (Data Series) Semantic is to create a lean and swift method of storing series of data like the dependency of voltage over time. This Data Series Semantic uses two mappings at the same data (ds:Node and ds:Leaf). It combines a single linked list mapping in combination with a tree mapping of the Data Series. The goal is to provide fast access to a requested subset of the data series.

The tree built by the ds:Tree provides fast access to any requested range in the Data Series (e.g. for a time period, like “the last 24 hours”). When the first element is identified by indexed access using the tree structure, the ds:List allows quick collection of the requested Data as the ds:Node(s) are linked sequentially.

This combination of indexed and sequential access methodologies promises good performance and effective memory requirements. The impact on speedup of this design will have to be validated, later on.

- *ds:DataSeries*
    - The *ds:DataSeries* class is the starting point of any data series. It contains a link to a *ds:List* and a *ds:Tree*. These allow access in two different ways to the set of *ds:Node*(s), the data values of the date series.
        - The list allows fast sequential stepping over the *ds:Node*(s).
        - The tree allows fast, indexed access to a selectable subset of the data series.

      Both (the *ds:List* and the *ds:Tree*) are ending in nodes that provide access to the data values of the data series.
    - For more information on header have a look in *ds:Header* below.
- *ds:Header*
    - The *ds:Header* class is the item in *ds:DataSeries,* where the type of data of the data nodes is specified. The number of arguments and the arguments themselves are specified. The name and the content of the arguments do not rely on fixed, predefined properties. They can be adapted to any combination of label, physical property and physical unit following

the context requirements. The arguments used in the *ds:Nodes* contain only the relevant data as properties and a link to the related ds:Header.
    - However, some frequently used arguments are predefined (e.g. *ds:argTimeStamp* as first argument).
    - The *ds:Header* is referenced in each o*f ds:DataSeries, ds:Tree* and *ds:List* and from each *ds:Node*/*ds:Leaf.* For all elements of a DataSeries the identical *ds:Header* has to be used.

- *ds:List*
    - The *ds:List* class expresses that the subject is the root list item of a data series. The list leads to a structure of a single linked list of *ds:Node*(*s*).
    - The list allows fast, sequential access to nodes of the data series.
- *ds:hasList*
    - The *ds:hasList* predicate references to the *ds:List* of each *ds:DataSeries*.
- *ds:Node*
    - The *ds:Node* class contains all the data values for a single data point of the data series. It is the basic element of *ds:List*. It uses the argument set as described in the ds:Header as referenced in its *ds:List*.
    - The nodes of a *ds:List* form a single linked list. This provides fast sequential access to the data value (e.g. all data since the last 2 hours).
    - The *ds:Leaf* item is used by *ds:Tree* and is derived from this *ds:Node*(s), that is used by *ds:List*. The combination of the *ds:Tree* and *ds:List* mappings allow quick indexed access to the data.
- *ds:Tree*
    - The *ds:Tree* class expresses that the node is the root tree item of a data series. The tree leads via a recursive structure of *ds:Limb*(s) to the *ds:Leaf*(s) that contain the data values.
    - The tree allows fast, indexed access to a selectable subset of the data series (see the comment at the begin of this section).
- *ds:hasTree*
    - The *ds:hasTree* predicate references to the tree for each *ds:DataSeries*.
- *ds:Limb*
    - The *ds:Limb* class expresses that the node is a Limb (that may have multiple sub-*ds:Limb(s)* and multiple *ds:Leaf(s)*)
- *ds: hasLimb*
    - The *ds:hasLimb* predicate describes that the object is a Limb.
- *ds:isLimbOf*
    - The *ds:isLimbOf* predicate describes that the subject is a Limb.
- *ds:index*
    - The *ds:index* predicate is part of the *ds:Limb* and allows ordered access to the sub limbs and allows ordered access to the *ds:Leaf(s)* at the end of the *ds:Tree*.
- *ds:Leaf* // inherited from ds:Node
    - The *ds:Leaf* is inherited from *ds:Node*. It is the endpoint of the *ds:Tree* structure that allows
    - The *ds:Leaf* class is inherited from ds:Node class and contains all the data values for a single data point of the data ser*i*es. It is the basic element of *ds:Tree*. It uses the argument set as described in the *ds:Header* as referenced in its *ds:Node* and *ds:Tree*.
    - The *ds:Leaf* of a *ds:Tree* build a tree structure that provides indexing for subsets of the data series with high performance.

- The *ds:Leaf* item is used by *ds:Tree* and is derived from this *ds:Node*(s), that is used by *ds:List*. The combination of the *ds:Tree* and ds:List mappings allows quick indexed and quick sequential access to the data. (see comment above)

- *ds:hasLeaf*
  - The *ds:hasLeaf* predicate describes that the object is a Leaf

- *ds:isLeafOf*
  - The *ds:isLeafOf* predicate describes that the subject is a Leaf
- *ds:argument*
  - The *ds:argument* specifies/describes the arguments of a ds:Node and a *ds:Leaf*.
  - As *ds:argument* has to be both predicate and object, a so called punning[8] technique has to be used.
  - In the *ds:Leaf* or the *ds:Node* structure a predicate derived from *ds:argument* specifies the data values of a data point of the data series

```
_:Leaf001
        ds:argTimeStamp         "2012-09-12T12:45:00";
        _:argFillLevel          0.5.
```

  - Besides the use as a predicate, the *ds:argument* can be used as a property, in order to configure an own, unique argument type. See the specification of *ds:argTimeStamp* and the custom argument *_:argFillLevel* below.
- *ds:argTimeStamp*
  - The *ds:argTimeStamp* predicate is a timestamp as first argument. The argument property must have a *rdfs:label*, a *ds:physicalUnit*, a *ds:physicalProperty* and a *ds:idOfArg*.

```
ds:argTimeStamp         rdfs:subPropertyOf ds:argument;[9]
        rdfs:label              "timestamp of measurement";
        ds:physicalUnit         "sec" ;[10]
        ds:physicalProperty     "timestamp"[11];
        ds:idOfArg              "1".
```

- *_:argFillLevel*
  - The property *_:argFillLevel* is specified like this:

```
_:argFillLevel  rdfs:subPropertyOf ds:argument;
        rdfs:label              "fill level";
        ds:physicalUnit         "mm" ;
        ds:physicalProperty     "fill level".
        ds:idOfArg              "2".
```

The UML class diagram in **Error! Reference source not found.** should help to understand the structure of he Data Series Ontology better:

[8] Punning is a semantic technique to use the same identifier (here ds:argument) for both predicate and for property. See http://www.w3.org/2007/OWL/wiki/Punning for more information about punning

[9] a _:timestamp should be a sub property of ds:argument

[10] it has unit "sec"

[11] It is a "timestamp"

A *ds:DataSeries* (marked blue) has the subcomponents: *ds:Tree* (marked yellow), *ds:List (marked red)* and *ds:Header (marked orange).* The *ds:Header* contains specifications of the number of arguments and the specifies arguments to be used (marked as purple).

The *ds:List* (marked red) is referenced from the *ds:DataSeries*. It contains a *ds:Header* (the same as in the related *ds:DataSeries* instance) and provides access to the data points via single linked list of *ds:Node(s)*. Each *ds:Node* uses the *ds:argument* predicate (marked cyan) to provide the data values for each data point. This allows fast, sequential access to the data points.

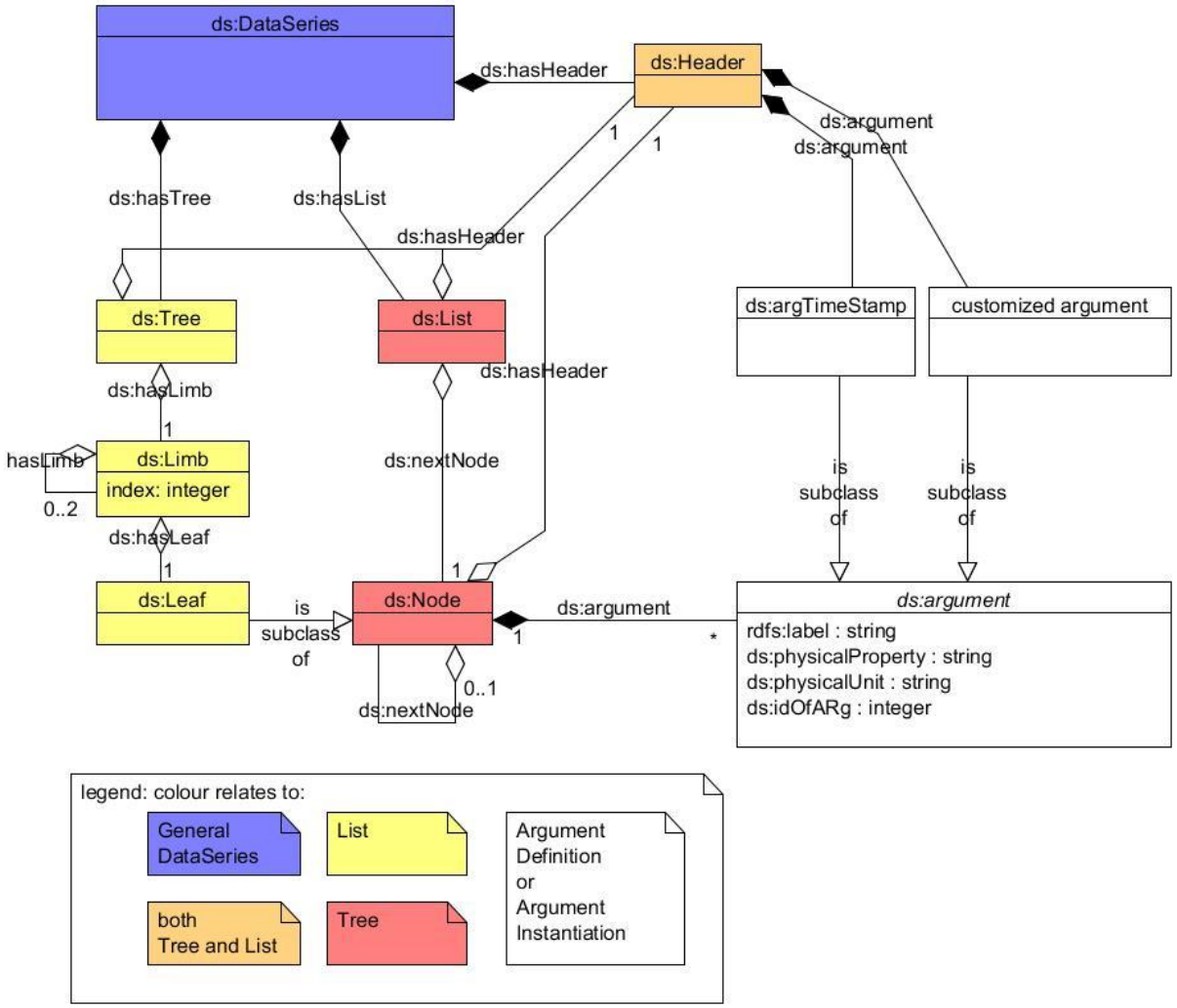

Figure 9: Structural description of the Data Series ontology

The *ds:Tree* (marked yellow) is referenced from *ds:DataSeries*, too. It contains a *ds:Header* item (the same as in the related *ds:DataSeries* instance) and an indexed tree structure. This index tree consists of an recursive tree of *ds:Limb(s)*. This tree structure of limbs is ending in a single *ds:Leaf, each.* This *ds:Leaf* inherits the data point argument representation from *ds:Node.* It is specified in the *related ds:Header.* This mechanism allows fast, indexed access to subsets of data values of the data series.

Error! Reference source not found. shows an exemplary use of the Data Series ontology, which will be escribed in the following lines:

A central subject *_:myDataSeries* (top left) of type *ds:DataSeries* has three subcomponents. An item of type *ds:Tree* , one of type *ds:List* and one of type *ds:Header.* The *ds:DataSeries,* the *ds:Tree* and the *ds:List* have a

reference to the same *ds:Header* instance (marked in orange). The *ds:Header* contains specifications of the number of arguments and specifies all arguments to be used (marked as purple).

The *ds:List* instance *_:myList* is referenced from the *ds:DataSeries*. Besides the reference to the *ds:Header* it contains (via *ds:nextNode*) a linked list of *ds:Node*(s) (marked as red/orange). These *ds:Node(s)* represent the data points of the data series (at the right side, marked as cyan). This allows fast sequential access to the data points.

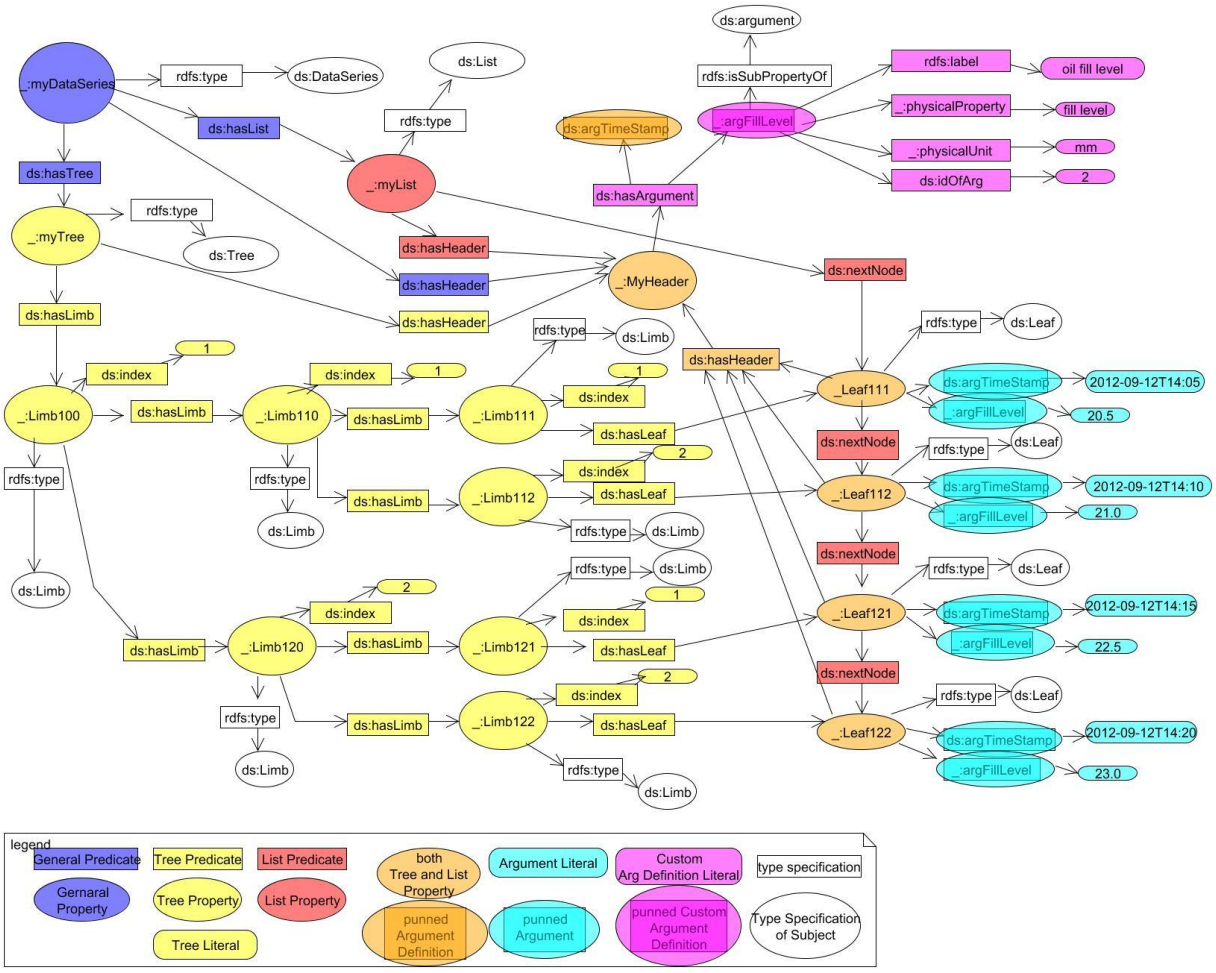

Figure 10: Example Configuration of Data Series ontology

The *ds:Tree* instance *_:myTree* contains (besides the *ds:Header* item) an indexed tree structure. This index tree consists of an recursive set of *ds:Limb(s)* (at the left/centre, marked as yellow). These limbs end in *ds:Leaf* (marked in orange), which is derived from *ds:Node*.

The organisation of the *ds:Tree* and the *ds:Limb* is done using the *ds:index* property of the *ds:Limb*. It can be organised as a B-tree[12]. It is possible to organise the tree/limb structure following native categories, e.g. for a time series it is useful to organise the tree by year, month, day, hour, minute and second.

The combined mechanism of indexed (via *ds:Tree*/*ds:Limb*) and sequential (via ds:List/ds:Node) access allows fast, indexed access to subsets of data values followed by scrawling the data series sequentially.

In the diagram, all of the predicates/objects that specify the type of the subject are marked white.

[12] A B-tree is like a binary tree, but with a bigger number of links in each node! For more info have a look at: http://en.wikipedia.org/wiki/B-tree

5.2 Machine Semantics Ontology (MSem)

The Machine Semantics has two major parts

- The *Enterprise Semantics*

  It describes the structure of a Virtual Enterprise broken down to ControlModules, sensors and actuators.

- The *Test-Execution-Environment Semantics*

  Provides the semantics to describe TestCases and to store the results of executed the TestCases as TestCaseResults.

Since both parts are interlinked they are put in a single ontology and use both *msem* as prefix for the ontology.

5.2.1 Enterprise Semantics

5.2.1.1 Domain and Scope of the Enterprise Semantics

- What is the domain that the ontology will cover?
  - This domain will cover the structure of a collaborating virtual enterprises broken down to each ProcessCell and each sensor and actuator in each Equipment- and Control-Module. It reclines to the Specification of ISA-88, also known as IEC 61512.
- For what are we going to use the ontology?
  - It will be used in *ComVantage* Mobile Maintenance (WP8) and the related Prototype.
- For what types of questions should the information in the ontology provide answers?
  - Queries should provide answers about all available sensors and actuator of an Enterprise at the machine that is target of the Maintenance Activity.
- Who will use and maintain the ontology?
  - The Data Series Ontology will be maintained by the *ComVantage* project, especially by RST & TUD.

5.2.1.2 Enterprise Semantics: Competency Questions

- What is the structure of a given Virtual Enterprise?
- What is the structure of a given Site, Unit or Process Cell?
- Which machines are available at a given Site of a given Enterprise?
- Where is the currently maintained machine of the Maintenance Task located in a given Enterprise?
- What types of sensors has a given machine?
- What sensors and actuators have a given machine?
- What temperature sensors have a given machine?
- What physical property and unit has a given sensor or actuator?
- What transient interval has a given sensor or actuator?
- What related URI to retrieve the current data has a given sensor or actuator?
- What is the historical information since timestamp of a given sensor?
- What is the current live data information of a given sensor?

5.2.1.3 Enterprise Semantics: Elements

The Enterprise Semantics is a subset of the Machine Semantics. It will closely follow ISA-88[13].

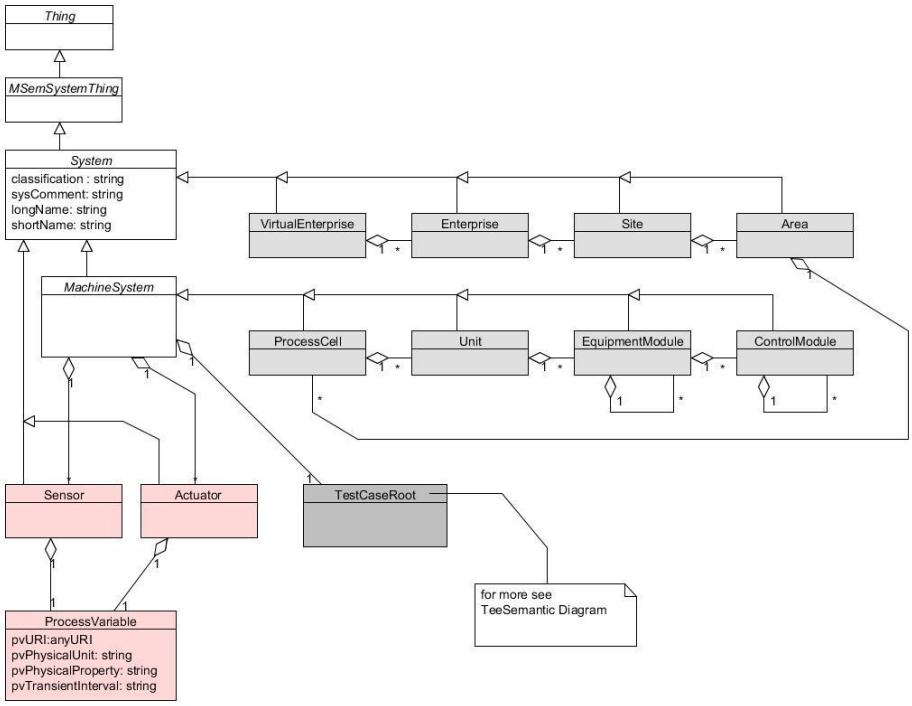

Figure 11: Structural description of Enterprise Semantic[14]s

The *Enterprise Semantics* is a subset of the *Machine Semantics Ontology.*

Almost all Elements are derived from the class *msem:System*. It provides a classification, a *msem:sysComment,* a *msem:longName* and a *msem:shortName* string property. There is a chain of elements which contain a *hasA* reference to its successor. The following elements are directly inheriting from *msem:System*.

- A msem:VirtualEnterprise has some msem:Enterprise(s)
- A msem:Enterprise has some msem:Site(s)
- A msem:Site has some msem:Area(s)
- A msem:Area has some msem:ProcessCell(s)

The following elements are inheriting from msem:MachineSystem that is derived from msem:System

Each class derived from msem:MachineSystem can contain actuators, sensors and a msem:TestCaseRoot node [15] for Maintenance Test Scripts. The following classes all are derived from msem:MachineSystem

- A msem:ProcessCell has some msem:Unit(s)
- A msem:Unit has some msem:EquipmentModule
- A msem:EquipmentModule has some msem:ControlModule and some recursive sub sets of msem:EquipmentModule
- A msem:ControlModule has some recursive sub sets of msem:ControlModule

Both msem:Sensor(s) and msem:Actuator(s) are also derived from msem:MachineSystem and allows access to so called transient Process Variables. This is represented by the class msem:ProcessVariable. Each has an

[13] For ISA-88 see http://de.wikipedia.org/wiki/ISA-88. It is also known as *IEC 61512.*

[14] Following the UML notification the classes marked with an italic class name are abstract and cannot be initiated.

[15] The class TestCaseRoot is described in the TEE semantics

URI, a physical unit (“K”) a physical property (“temperature”) and a transient interval (“0.5 sec”). The transient interval indicates the changing period of the transient ProcessVariable.

In order to learn about the mechanism of transient Process Variables see chapter 4.1.

This is a detailed description of the Enterprise Semantics, which is a subset of the Machine Semantics (msem:). The next two classes are parents of the classes following in this chapter:

- *msem:System*
  - The *msem:System* class is the central parent class for almost of the following classes

    Each system and all of its subsystems have these predicates:
    - *msem:classification* to classify the system
    - *msem:sysComment* to add a comment about the system
    - *msem:longName* long name of the system
    - *msem:shortName* short name of the system
  - Almost all classes of the Enterprise Semantics are inherited from msem:System directly or indirectly (via msem:MachineSystem). This allows iterative querying along all systems and subsystems.
- *msem:MachineSystem* derived from *msem:System*

  The Machine System class inherits from *msem:System* and extends the System by *msem:Sensor* and *msem:Actuator* support. It allows the specification of a TestCaseRoot to handle Testing via Testing Scripts.
  - The *msem:MachineSystem* property is a parent of the classes
    - msem:ProcessCell,
    - msem:Unit,
    - msem:EquipmentModule and
    - msem:ControlModule.

The following classes are derived from *msem:System* and may have *msem:Sensor* and *msem:Actuator* components. They may use the predicates *msem:classification, msem:longName, msem:shortName, msem:sysComment*

- *msem:VirtualEnterprise* derived from *msem:System*
  - The msem:*VirtualEnterprise* class describes a collaboration of several Enterprises that build a virtual enterprise
  - It contains links to one or more (collaborating) msem:*Enterprise(s)*
- msem:*Enterprise* derived from *msem:System*
  - The *msem:Enterprise* class describes a Company/Enterprise and all of its sub-structures
  - It contains links to one or more *msem:Site(s)*
- *msem:Site* derived from *msem:System*
  - The *msem:Site* class describes a Site of an Enterprise and all of its sub-structures
  - It contains links to one or more *msem:Area(s)*

- *msem:Area* derived from *msem:System*
  - The *msem:Area* class describes an Area of an Enterprise's Site and all of its sub-structures
  - It contains links to one or more *msem:ProcessCell*

The following classes are derived from *msem:MachineSystem* and all may have *msem:Sensor*(s) and *msem:Acuator*(s)

- *msem:ProcessCell* derived from *msem:MachineSystem*
  - The *msem:ProcessCell* class describes a process cell of an Enterprise's Area and all of its sub-structures and its subcomponents.
  - It contains links to one or more *msem:Unit*
- *msem:Unit* derived from *msem:MachineSystem*
  - The *msem:Unit* class describes a Unit of an Enterprise's process cell and all of its sub-structures and its subcomponents.
  - It contains links to one or more *msem:EquipmentModule*
- *msem:EquipmentModule* derived from *msem:MachineSystem*
  - The *msem:EquipmentModule* class describes an equipment module of an Enterprise's Unit and all of its sub-structures and its subcomponents.
  - It contains
    - recursive links to *msem:EquipmentModule* (itself). This allows more complex equipment module structures
    - and several *msem:ControlModule* subcomponents
- *msem:ControlModule* derived from *msem:MachineSystem*
  - The *msem:ControlModule* class describes a control module of an enterprise's unit and all of its sub-structures and its subcomponents.
  - It may contain other *msem:ControlModule (itself), t*his allows more complex control module structures

The following classes describe sensors and actuators properties. They are used in *msem:MachineSystem.*

- *msem:Sensor* derived from msem:System
  - The *msem:Sensor* class describes the instance to be a sensor
  - It has a process variable
    - *msem:hasProcessVariable*
- *msem:Actuator* derived from msem:System
  - The *msem:Actuator* class describes the instance to be an actuator
  - It has a process variable
    - *msem:hasProcessVariable*

Both *msem:Sensor* and msem:Actuator have a subcomponent *msem:ProcessVariable*

- *msem:ProcessVariable*
  - The *msem:ProcessVariable* class describes a process variable and has this subcomponents:

- *msem:pvPhysicalProperty* describes the physical property (e.g. “temperature”)
- *msem:pvPhysicalUnit* describes the physical unit (e.g. “K” – “Kelvin”)
- *msem:pvTransientInterval* describes the changing interval
- *msem:pvURI* describes the URI that links to the transient data.

Here are some *predicates* provided by the Enterprise Machine Semantics

- *msem:hasSubSystem*
  - The transitive predicate *msem:hasSubSystem* describes a subsystem. This object can be any of type :
    - *msem:VirtualEnterprise, msem:Enterprise, msem:Site* or *msem:Area,*
    - *msem:ProcessCell, msem:Unit, msem:EquipmentModule* or *msem:ControlModule*
    - *msem:Sensor* or *msem:Actuator*.
- *msem:classification*
  - The *msem:classification* predicate classifies the type of a sensor of type (e.g. “oil temperature”)
- *msem:pvPhysicalProperty*
  - The *msem:pvPhysicalProperty* predicate describes the physical Property (e.g. “temperature”) related to the subject (sensor or actuator).
  - The predicate will refer to the QUDT[16] ontology which handles Quantities, Units, Dimensions and Data Types in OWL and XML. Thus the *msem:pvPhysicalProperty* predicate will have *qudt:QuantityKind* as a range.
- *msem:pvPhysicalUnit*
  - The *msem:pvPhysicalUnit* predicate describes the physical Property (e.g. “Kelvin”) related to the subject (sensor or actuator)
  - The predicate will also refer to the QUDT ontology and uses *qudt:PhysicalUnit* as a range.
- *msem:pvTransientInterval*
  - The *msem:pvTransientInterval* predicate describes how often the measured property changes in seconds.

5.2.2 Test Execution Environment Semantics: Elements (Tee)

5.2.2.1 Domain and Scope of the Ontology

- What is the domain that the ontology will cover?
- This domain will cover
  - Storing the descriptions and the results of TestCases of the Test Execution Environment.
  - Accessing this information with SPARQL Queries.
- For what we are going to use the ontology?

[16] http://www.qudt.org/

- It will be used in WP8 Mobile Maintenance application area and the related WP11 Prototype. It is used for selecting the TestCases to be stated and handling the results of the executed TestCases.

- For what types of questions the information in the ontology should provide answers?
  - It is used to retrieve information about the available TestCases,
  - It is used to store, access and process the Results of executed Test Scripts.
- Who will use and maintain the ontology?
  - The Test Execution Environment Semantics ontology will be maintained by the *ComVantage* project team, especially by RST & TUD.

5.2.2.2 Test Execution Environment Semantics: Competency Questions

- Which are the available TestCaseRoots at a given MachineSystem?
- Which machine is in focus of the Test Execution Environment?
- Which TestCases are available at a given TestCaseRoot?
  - What are the short and long names of a given TestCase?
  - What are the classifications and the comments of the available TestCases?
- Which TestCasesSteps are available for a selected TestCase?
  - What are the short and long names of all available TestCaseSteps?
  - What are the classifications and the comments of all available TestCaseSteps?
- Is an earlier started TestCase already finished?
- Which are the last TestCaseResults with result, short and long name?
- What is the information about the TestCaseStepResult for a given TestCaseResult?
- What are the timestamp and results of all previous Starts of a given TestCase, since time t_0?
- How often has a given TestCase been started with which results?

5.2.2.3 Test Execution Environment Semantics: Elements

The *Tee Semantics* (short for Test Execution Environment Semantics) is a subset of the Machine Semantics ontology. The entry point is an instance of msem:TestCaseRoot. It contains all available *TestCases* with their subcomponents *msem:TestCaseStep*. The results of a finished run of *msem:TestCase* (and *msem:TestCaseStep*) are stored in *msem:TestCaseResult* (and *msem:TestCaseStepResult*).

Both *msem:TestCase* and *msem:TestCaseStep* are derived from *msem:TeeThingDescription* that provides additional information strings like a classification, a comment, and a long- and short-name. The *msem:TestCaseStep* contains an index, in order to describe the sequential order of the *msem:TestCaseSteps* of a given *msem:TestCase*.

The TestCase Results are derived from *msem:TeeThingResult*(s) and inherit a Result Literal to indicate the Result. The *msem:TestCaseResult* is labelled with an *msem:ExecutionStartTimeStamp* and an *msem:ExecutionDuration*. Both *msem:TestCaseResult* and *msem:TestCaseStepResult* are linking to the related instances of *msem:TestCase* and *msem:TestCaseStep*. This reduces redundancy in the representing the *msem:TestCaseResult(s)* and *msem:TestCaseStepResult(s)*.

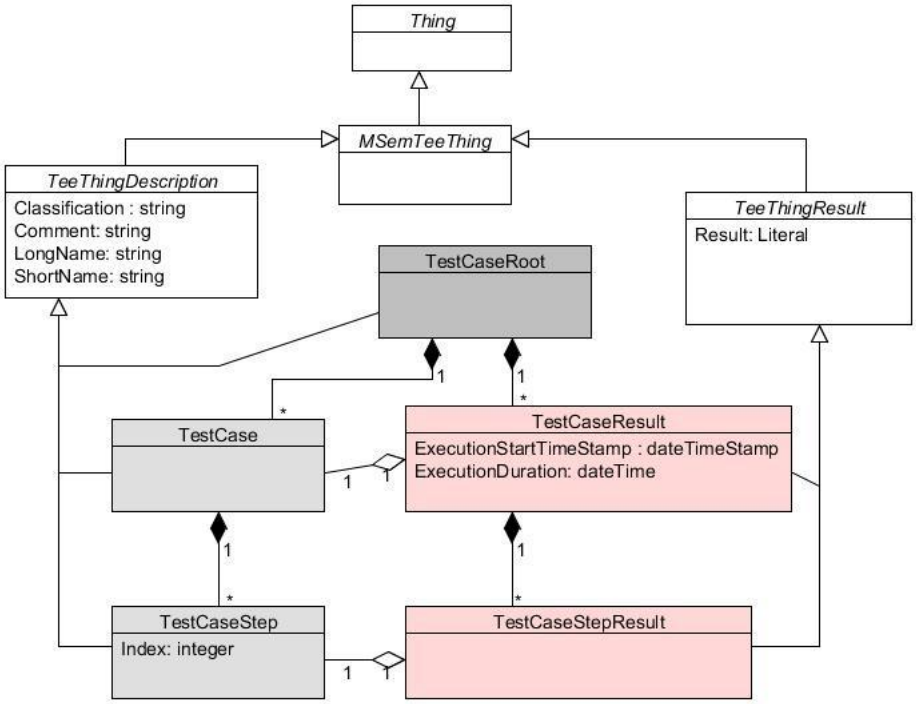

Figure 12: Structural description of Test Execution Environment Semantics.[17]

Detailed description of the *Test Execution Environment (Tee) Semantics*

- *msem:TestCaseRoot*
  - The *msem:TestCaseRoot* class describes a Central Root-Node of Test Execution Data Representation. It can be used in all classes derived the from *msem:MachineSytem*.
  - It consists of several TestCases and TestCaseResults. The TestCaseResults are related to one of the TestCases.
- *msem:TestCase* *// is derived from msem:TeeThingDescription*
  - The *msem:TestCase* class describes a complete Test Case.
  - A TestCase consists of several *msem:TestCaseSteps*.
  - It also contains some classification, comment long name, and short name strings.
- *msem:TestCaseStep* *// is derived from msem:TeeThingDescription*
  - The *msem:TestCaseStep* class describes a sub step of an *msem:TestCase*.
  - It also contains some classification, comment long name, and short name strings.
  - Additionally it has an Index to indicate the sequence the Test Cases were executed.
- *msem:TestCaseResult*
  - The *msem:TestCaseResult* class describes the result of an executed *msem:TestCase*.
  - It contains a *msem:executionStartTimestamp* and a *msem:executionDuration* and a *msem:TestCaseResult*
  - It contains a link to the related *msem:TestCase* and a Result literal
- *msem:TestCaseStepResult*
  - The *msem:TestCaseStepResult* class describes the result of a *msem:TestCaseStep* of an executed *msem:TestCase*.
  - It contains a result literal
  - It has a link to the related *msem:TestCaseStep*.

[17] Following the UML notation, classes with italic names are abstract and cannot be initiated. This is relevant for the classes *msem:Thing*, *msem:MSemTeeThing*, *msem:TeeThingDescription* and *msem:TeeThingResult*.

5.3 Repair and Predictive Maintenance Ontology (RPM)

The Repair and Predictive Maintenance Scenario ontology handles all aspects to understand and represent a typical scenario in the WP8 application area. This ontology is in a rather early stage and not all details are specified yet. It uses the prefix *rpm*.

5.3.1 Domain and Scope of the RPM ontology

- What is the domain that the ontology will cover?
  - This domain will cover the aspects to understand the scenarios in work package 8. Thus, it is grounded in the field of Mobile Maintenance of factories.
- For what we are going to use the ontology?
  - It will be used in WP8 *Mobile Maintenance* and the related prototype. This ontology is very specific to the application area of WP8 in *ComVantage*. It should present to some extent the relation between different stakeholders in a Virtual Enterprise with respect to the application area of Mobile Maintenance. It is highly unlikely that this ontology can be used for other purposes without modifications. But there can maybe some parts for a Virtual Factory Stakeholder ontology extracted.
- For what types of questions the information in the ontology should provide answers?
  - It allows tracking and manipulating the process of maintenance scenarios.
  - It should describe the relations between the actors and their environment.
- Who will use and maintain the ontology?
  - The Data Series Ontology will be maintained by the *ComVantage* project, especially by RST & TUD. It will also be used by these stakeholders.

5.3.2 RPM Competency Questions

- Is Actor A a Service Technician?
- Does Company A have the correct Service Level Agreement to inspect machines in Site C?
- Is the ticket regarding Machine Defect Report F still open?
- Who has reported the Machine Defect for machine C?
- Who is assigned to Ticket ABC?
- Which machines are affected from Machine Defect C?
- Which company employs Service Technician X?
- When was the Machine Defect A predicted and by which EPAMMS system?
- What is the description of all open Machine Defects?

5.3.3 RPM Elements

A detailed description of the Repair and Maintenance Scenario ontology is given in the following lines beginning with the classes and then followed by the object properties of the ontology. The data properties are not described in detail but can be found in the RDF ontology.

- *rpm:CollaborativePartner*
  - This class is the common parent class of *rpm:CustomerFactory*, *rpm:ServiceCompany*, *rpm:MachineProducerCompany.* It can furthermore hold all different kind of stakeholders in a Virtual Factory.

- *rpm:CustomerFactory*
  - The *rpm:CustomerFactory* represents a stakeholder which produces something with *rpm:Machines* from a *rpm:MachineProducerCompany* and supported by a *rpm:ServiceLevelAgreement* with a *rpm:ServiceCompany*.
- *rpm:ServiceCompany*
  - A *rpm:ServiceCompany* provides services for a *rpm:CustomerFactory, as negotiated in a rpm:ServiceLevelAgreement*.
  - The *rpm:MobileMaintenanceCoordinator(s)*, some *rpm:ServiceTechnician(s)* and some *rpm:MachineExpert(s)* works for the *rpm:ServiceCompany.* They have to provide service to a *rpm:CustomerFactory.* The *rpm:MachineExpert(s)* may be employed by the *rpm:MachineProducerCompany*
- *rpm:MachineProducerCompany*
  - The *rpm:MachineProducerCompany* has produced some *rpm:Machine*(s) and sold to a *rpm:CustomerFactory* and has deep knowledge about these machines.
  - On request it supports the rpm:ServiceCompany with an enhanced service of one or more *rpm:MachineExpert(s),* which are specialised in the relevant *rpm:Machine*.
- *rpm:Actor*
  - This class is the parent for both *rpm:HumanActor* and *rpm:TechnicalActor* and describes an actor.
  - It describes the role of human or technical stakeholder(s) in the maintenance scenario.
- *rpm:HumanActor*
  - The class *rpm:HumanActor* bundles all common properties and abilities of human *rpm:Actors*, such as name and contact information. Its sub-classes are *rpm:Customer*, *rpm:MobileMaintenanceCoordinator*, *rpm:ServiceTechnician* and *rpm:MachineExpert*.
- *rpm:Customer*
  - A *rpm:Customer* is a *rpm:HumanActor* which owns (*rpm:own*) a *rpm:CustomerFactory* to produce something (with machines).
- *rpm:MobileMaintenanceCoordinator*
  - A *rpm:MobileMaintenanceCoordinator* is a *rpm:HumanActor.* He *rpm:worksForCompany rpm:ServiceCompany.* In the Repair Scenario he delegates each *rpm:DefectReport* to an adequate *rpm:ServiceTechnician* or *rpm:MachineExpert.* In the Predictive Scenario, he additionally is notified about all predictions of *rpm:MachineDefect(s).* These predictions are classified and prioritised. He will decide on adequate measures and can initiate a Repair Scenario to fix the Defect before it occurs.
- *rpm:MachineExpert*
  - A *rpm:MachineExpert* is a *rpm:HumanActor*. He represents a staff member from the *rpm:ServiceCompany* or the *rpm:MachineProducerCompany,* who has deep, specialised knowledge about specific *rpm:Machines* and their possible *rpm:MachineDefect*(s).
- *rpm:ServiceTechnician*
  - A *rpm:ServiceTechnician* is a *rpm:HumanActor*. He represents a staff member from the *rpm:ServiceCompany* or the *rpm:CustomerFactory,* who is trained to maintain specific

machines. If he cannot solve a defect himself, he is supported and instructed by a *rpm:MachineExpert* to solve the problem.

- *rpm:TechnicalActor*
  - An *rpm:TechnicalActor* describes all actors that are not human but a technical system that has to some extent some intelligence and can act on its own.
- *rpm:Machine*
  - *A rpm:Machine* is a technical equipment to produces something. It is constructed and built by a *rpm:MachineProducerCompany*. It is serviced by a *rpm:ServiceCompany*. For each type of Machine there are some specialised *rpm:MachineExpert*(s) available.
- *rpm:ActiveMachine*
  - A *rpm:ActiveMachine* is a *rpm:Machine*, located in a *rpm:CustomerFactory* and has information about its internal machine configuration, represented partly in the MSem ontology as *msem:MachineSystem*.
- *rpm:PAMMS*
  - The *Predictive Active Machine Maintenance Support (PAMMS)* is part of a *rpm:ActiveMachine.*
  - Its task is to try to analyse all *sensor* and *actuator d*ata and to predict upcoming *rpm:MachineDefect*(s), before they arise and cause a machine breakdown. It sends the predictions to the *rpm:EPAMMS* component at the *rpm:ServiceCompany*.
- *rpm:EPAMMS*
  - The Extended *Predictive Active Machine Maintenance Support (EPAMMS)* is part of the *rpm:ServiceCompany* and is used to evaluate the predictions about Machine Defects transmitted from the *rpm:PAMMS* at the *rpm:ActiveMachine*.
  - Its task is to classify and prioritise (by probability, severity and impact) the Machine Defects prediction of the *rpm:PAMMS*. It will notify the *rpm:MobileMaintenanceCoordinator*, who has to decide which repair activity is appropriate and eventually initiate a Repair Scenario.
- *rpm:ServiceLevelAgreement*
  - A *rpm:ServiceLevelAgreement* (SLA) is a part of a service contract between different *rpm:CollaborativePartners* where the level of service is formally defined[18].
- *rpm:MachineDefect*
  - A *rpm:MachineDefect* can refer (*rpm:defectRefersToMachine*) to one or more instances of *rpm:Machine*. It has a description of the malfunction (*rpm:defectDescription*) and is used for both *rpm:DefectReport* and *rpm:DefectPrediction* as a parent class.
- *rpm:DefectPrediction*
  - A *rpm:DefectPrediction* was predicted by the appropriate *rpm:EPAMMS* for a specific *rpm:ActiveMachine*. At this time no defect has been occurred, but this can be used for creating a *rpm:Ticket*.
- *rpm:DefectReport*
  - A *rpm:DefectReport* describes an occurred defect that was reported (*rpm:defectReportedBy*) by a *rpm:Actor*.

[18] As described inhttp://en.wikipedia.org/wiki/Service-level_agreement

- *rpm:Ticket*
  - A *rpm:Ticket* refers to a reported or predicted *rpm:MachineDefect*. It is assigned (*rpm:isAssignedTo*) to a *rpm:HumanActor* which tries to solve the task of the ticket. A ticket has additionally a *rpm:ticketStatus* to be able to follow the process of the ticket.
- *rpm:ticketStatus*
  - *A rpm:ticketStatus* refers to a *rpm:Ticket.* It states the status of the ticket (e.g. open, pending, closed, ...).
- *rpm:owns*
  - This predicate states that a *rpm:Customer* owns a rpm:CostumerFactory.
- *rpm:hasServiceLevelAgreement*
  - This predicate states that a *rpm:Company* has a *rpm:ServiceLevelAgreement* with a *rpm:ServiceCompany*.
- *rpm:worksForCompany*
  - The predicate *rpm:worksForCompany* indicates that the subjected *rpm:HumanActor* is working for the objected *rpm:CollaborativePartner*
- *rpm:hasProducedMachine*
  - This predicate states the a *rpm:MachineProducerCompany* has produced a *rpm:Machine* and thus has deep knowledge about it.
- *rpm:isPartOf*
  - This predicate states that some entity is a part of another entity. For example, a *rpm:PAMMS* is part of a *rpm:ActiveMachine*.
- *rpm:defectRefersToMachine*
  - This predicate states that a *rpm:MachineDefect* refers to one or more *rpm:Machines*.
- *rpm:wasPredictedBy*
  - This predicate states that a *rpm:DefectPrediction* was created by a *rpm:EPAMMS*.
- *rpm:wasReportedBy*
  - This predicate states that a *rpm:DefectReport* was created by a *rpm:Actor*.
- *rpm:isAssignedTo*
  - This predicate states that a *rpm:Ticket* is assigned to a *rpm:HumanActor* which is responsible for this ticket.

6 CONCLUSION AND OUTLOOK

In this deliverable, we have provided the results of the integration process of the generic WP4 concepts into the application area *Mobile Maintenance* of WP8.

The results of this deliverable are a crucial part of the first prototype mockup for *Mobile Collaboration*. Therefore we concentrated on the important aspects for this prototype. We have detected the problem of handling transient data in triple stores and found an alternative solution with a DHM Adapter which provides the specific Middleware data in a generic RDF format via HTTP request. The concept with this DHM Adapter as a central component can be well aligned to the WP2 global architecture and the WP4 Linked Data concepts. The data provided from the DHM Adapter uses the DataSeries (DS) ontology which has also been described in this deliverable besides the ontologies for the Machine Semantics (Msem) which are necessary to find the currently maintained machine to be maintained with all its sensors and actuators. Additionally a first draft of the Repair and Predictive Maintenance (RPM) ontology was described in order to model all relevant aspects for the tracking of the repair scenarios.

For the remainder of the project, the findings of this deliverable can hopefully be integrated into the generic WP4 Linked Data concepts to form better vocabularies and enhanced data integration concepts. Thus the next iteration of this deliverable can build upon an enhanced version of these concepts which take the detected issues and limitations into account.

7 REFERENCES

Altmann, W., Hartung, J., Voss, W., Schachner, R., & Haferkorn, F. (2012, feb). Mobile Maintenance: Scenario Specification and Refinement. *(D8.1.1)*. ComVantage: D8.1.1.

Buchmann, R., & Münch, T. (2012, feb). Functional and Technological Requirements. ComVantage.

Fernandez-Lopez, M., Gomez-Perez, A., & Juristo, N. (1997). METHONTOLOGY: from Ontological Art towards Ontological Engineering. *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering.*

Graube, M. (2012, jun). Linked Data support toolset. *(D4.4.1)*. ComVantage: D4.4.1.

Grüninger, M., & Fox, M. (1995). Methodology for the Design and Evaluation of Ontologies.

Grüninger, M., & Fox, M. S. (1995). Methodology for the design and evaluation of ontologies. *Workshop on Basic Ontological Issues in Knowledge Sharing, International Joint Conference on Artificial Intelligence.*

Haferkorn, F., & Schachner, R. (2012, jun). Middleware adapter set. *(D4.2.1)*. ComVantage: D4.2.1.

Hladik, J. (2012, jun). Business and engineering software adapter. *(D4.3.2)*. ComVantage: D4.3.2.

Hladik, J., Graube, M., & Weber, C. (2012, apr). Data format specification. *(D4.1.1)*. ComVantage: D4.1.1.

Lopez, M. F. (1999). Overview of Methodologies for Building Ontologies. *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem Solving Methods (KRR5).*

Uschold, M. (1996). Building Ontologies: Towards a Unified Methodology. *Proceedings of Expert Systems '96, the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems.*

Uschold, M., & King, M. (1995). Towards a Methodology for Building Ontologies. *Workshop on Basic Ontological Issues in Knowledge Sharing; International Joint Conference on Artificial Intelligence.*

8 APPENDIX: GLOSSARY

Concept	Explanation
(Active) Machine	Target of the Mobile Maintenance application area.
ARQ	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.
Corrective (Maintenance) Scenario	Maintenance scenario with "maintenance [as a] task performed to identify, isolate, and rectify a fault so that the failed equipment, machine, or system can be restored to an operational condition"[19]. See Preventive and Predictive Maintenance Scenario, too.
Customer Factory	Part of the Maintenance Scenario. This is the site, where the machine, targeted of the Maintenance Scenario is located.
Dataset	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.
DHM	Short for Data Harmonisation Middleware
DHM-Adapter	Adapter to provide access to the Data Harmonisation Middleware as there are Live Access and control of Maintenance Test Scripts
DHM-JobController	The Linked Data Adapter is a component of the Data Harmonisation Adapter. It controls and if necessary blocks concurrent accesses to machines. It dispatches its commands to components via DHM-Live-Access or DHM-Tee
DHM-LDA	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)
DHM-Live-Access	Live Access is a component of the Data Harmonisation Adapter It provides access to live data like sensor and actuator data
DHM-MW-Log	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 it provides periodically to the History Data Store
DHM-Tee	The Test Execution Environment is a component of the Data Harmonisation Adapter It provides control of Maintenance Test Scripts.
Graph	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.
Jena	Jena (http://jena.apache.org/) 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.
LDS	Linked Data Server. Is responsible for providing the Machine Semantics to the

[19] As described in http://en.wikipedia.org/wiki/Corrective_maintenance

Concept	Explanation
	Maintenance Personnel via the SupportApp.
Linked Data	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.
Machine Defect	The *Machine Defect* is a malfunction of a Machine at the Customer Factory. It might lead to breakdowns and to unplanned downtimes. To solve all Machine Defect(s) is the target of all Maintenance Scenarios.
Machine Expert	Human Actor at the Corrective (Repair) Scenario and the Predictive Maintenance Scenario The Service Technician is the off-site responsible to fix occurring Machine Defects. He instructs the on-site Service Technician to fix the Machine Defect.
Maintenance Personal	Personal responsible for detecting and handling Machine Defects. May be a Service Technician, a Machine Expert or the Customer or one of his employees.
Maintenance Test Script	Part of the Test Execution Environment. The Test Script consists of several Test Cases that are split into several Test Case Steps. The Test Script runs autonomous until it ends and returns a value indicating success or fail of the Maintenance Test.
ME	Abbreviation for Machine Expert
MMCo	Abbreviation for Mobile Maintenance Coordinator.
Mobile Maintenance	Central issue of WP8.
Mobile Maintenance Coordinator	Human Actor at the Corrective (Repair) Scenario and the Predictive Maintenance Scenario. Coordinates all incoming reports about machine defects and delegates the repair tasks to adequate Machine Experts and/or Service Technicians #MMCo at the Predictive Scenario
off-site	Not located at the customers factory. Opposite of “on-site”. Accessing to maintenance target machine via external internet.
on-site	Located at the customers factory. Opposite of “off-site”. Accessing to maintenance target machine via local Intranet.
Predictive (Maintenance) Scenario	The intention of a *Predictive Maintenance Scenario* is to predict and fix a Machine Defect before it occurs in order to avoid machine break downs See Corrective and Preventive Maintenance Scenario, too.
Preventive (Maintenance) Scenario	Predictive Maintenance is “Maintenance, including tests, measurements, adjustments, and parts replacement, performed specifically to prevent Machine defects from occurring.”[20] The maintenance activities are taken periodical and at special scheduled intervals. Replaced parts are tested and if possible repaired and reused.

[20] As described in http://en.wikipedia.org/wiki/Preventive_Maintenance

Concept	Explanation
	See Corrective and Predictive Maintenance Scenario, too.
RDF	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.
RDFS	RDF Schema (RDFS) is a namespace for expressing basic concepts as classes and properties in RDF.
Repair Scenario	Short name for Corrective Maintenance Scenario
Service Company	Part of the Maintenance Scenario. This is the Maintenance Service partner of the Customer Factory. This Company is responsible for quick and immediate solving of Machine Defects. The Mobile Maintenance Coordinator and the Machine Expert and in some cases the Service Technician are working for the Service Company.
Service Technician	Human Actor at the Corrective (Repair) Scenario The Service Technician is the on-site responsible to fix occurring Machine Defects
SPARQL	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.
SPARQL-Endpoint	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.
SPARUL	SPARQL Update Language
SupportApp	The SupportApp is the Mobile User Interface of the Mobile Maintenance Scenarios. It manages login and provides access to the LDS and the DHM-Adapter.
SvTn	Abbreviation for Service Technician
Test Case	A Test Case is a part of the Test Execution Environment. It is a script to perform tests on a Machine. Contains Execution Start Time, Execution Duration and overall result.
Test Case Step	This is a sub step of a Test Case. Contains a Result.
Tee	Abbreviation for Test Execution Environment.
Test Script	A generic term for TestCase and TestCaseStep
Triple	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.
Triple store	A triple store is a database specialised for storing triples. Thus, it is core element for Linked Data application for static information.
Virtuoso	Virtuoso is a server application with support for Features of SPARQL. Currently not the

Concept	Explanation
	complete SPARQL 1.1 is provided

Table 1: Glossary

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.