A CONCEPT FOR SEMANTIC-BASED INFORMATION MANAGEMENT FOR CONTROL ROOM DEVELOPMENT

Eduard Gringinger*, Dieter Eier, Dieter Merkl*****
***Frequentis AG, **Frequentis USA Inc., *** Vienna University of Technology**

Keywords: *Ontology, Semantic, ATM, Control Room Framework, Modular Architecture*

Abstract

The global concept of the air traffic management (ATM) through the Next Generation Air Transportation System (NextGen) and the Single European Sky ATM Research Program (SESAR) will break up with the existing roles predicated by 50 years old technology [1], [2]. SESAR and, similarly, NextGen specify air traffic operations and management for the foreseeable future which is nowadays limited by out-of-date architecture. Current functionality is based on historical technical limitations. To achieve a performance-based and most efficient approach the further development has to be right balanced to prevent over-optimizing one area at the expense of others [3].

This paper describes a concept for an Ontology-based Control Room Framework (ONTOCOR) which allows more productive software code usage and easier development. It focuses on improving efficiency and increasing the code reusability in order to achieve SESAR's claim for a performance-based and cost-efficient system [2]. The main goal is to enhance software development in the area of ATM with semantic technologies and further, to enable an interchange of different domains with similar types of tasks. Therefore semantic standards and tools are briefly examined and the concept of the ONTOCOR project is explained.

1 Introduction

Control rooms are typically found in the Security, Public Safety (PS), Public Transport (PT), and ATM domains. Today, each of these sectors uses domain specific concepts of operation, which result in different solutions for every targeted environment. This limits the potential for cost efficient software development and increases the time-to-market. Information management, like systems for the ATM or other domains as mentioned before, typically consist of many heterogeneous sub components. Those sub components are mostly implemented with diverse types and structures of data, which result from the circumstance that such complex information management systems are developed for specific business needs. But, when the time has come and the business scope changes, for example to combine two existing parts, some sort of integration is needed [4]. To win the challenges of the data and system integration, a framework, which defines seamless information interchange, is needed. As described in the European Air Traffic Management Master Plan, "*the Information Management Work Package (…) defines the ATM Information Reference Model and the Information Service model (…) by establishing the framework, which defines seamless information interchange between all providers and users of shared ATM information"* [2]. A specific example to implement inter domain, is the European ATM Information Reference Model (AIRM). In general, domain independent implementation of components is a future goal and EUROCONTROL defines AIRM as a model, which contains all of the ATM information to be shared in a semantic way [5]. Exactly within these circumstances an ontology-based approach could bring the break through. Semantic structures will improve the productivity and increase the reusability of

software through a component based framework, which are both key goals of the Pan-European SESAR Joint Undertaking (SJU) [3].

Within ONTOCOR, an Ontology-based Control Room Framework will be developed in order to survey the potential benefits of such an approach over traditional software development. ONTOCOR aims to enhance software development with semantic technologies and further, to enable an interchange of different domains with similar types of tasks. An important aspect of a modular architecture is to gain control over accumulation and utilization of control room content. It is necessary to define analytic methods to describe the behavior of interfaces and to enrich the entire set of services semantically. Therefore, an exploration of existing ontology frameworks in the field of software-development in context to the ONTOCOR project is needed.

To address the identified issues, this paper analyzes reasoner and visualization tools that will consult with further investigation of ONTOCOR as a methodology. Another goal is to accentuate that ontology-based development could have the potential to develop, from a qualitative point of view, better software for mission critical environments in less time and at less cost. The paper will give an overview about the term "ontology" in context with semantic based-information management. Finally, the authors will draw a concept of ONTOCOR with focus on the methodological strategy.

2 Definitions of Ontology

A precise definition of an ontology is not a trivial task. The difficulty lies in the fact that the word ontology was first used in the field of philosophy. Therefore, it is important to go back in time. The term itself is loan from the Greek word *ὄν* (being) and λογία (science, study, theory) which has a different meaning in the philosophical context, where it refers to the study of being [6]. Greek philosophers from the Platonic school stated that some categories of being, are fundamental. Under the doctrine of Plato, Aristotle (384-322 B.C.) hypothesized four ontological dimensions in his Metaphysics book Theta [7]. In the middle Ages, European academics used ontological arguments to explain the existence of god in a scientific manner. The argument examines the concept of God, and states that the greatest possible being is on the top in a scale of terms ranging from the bottom to an infinity form of being. These ontological arguments are controversial in philosophy since then [8]. From a modern perspective this argument could be described through an ontology language in a way that God is the overall "Thing" class, and all other beings are underlying subclasses of "Thing".

2.1 Ontology in Computer Science

Computer scientists became interested into ontologies in the 1970s as the research in the field of Artificial Intelligence (AI) began [9]. They were tempted by the applicability to perform certain kinds of automated reasoning on ontologies as computational models, with mathematical logic [10]. Such an ontology could for example define classes, relations, formal functions with a concept description and axioms that constrain the interpretation. The first definition of ontology in terms of computer science was created by Tom Gruber in the early 90's. He defined ontology as an explicit and formal specification of a shared conceptualization [11]. The word "explicit" implies that the type of concepts and their constraints are explicitly defined. "Formal" connotes that the ontology is readable by a machine. And a "shared conceptualization" is specified to state axioms that do include the possible interpretations for the defined terms, which contain the knowledge of a specific domain and were accepted by a group. This early definition has kicked up much dust, therefore Gruber described the essential points of an ontology in the Encyclopedia of Database Systems in 2009 as a definition of "*concepts, relationships, and other distinctions that are relevant for modeling a domain*" whereas "*the specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use*" [12]. But there is no all-in-one

terminology. Often ontology is defined by its use or in context of the Semantic Web, where the World Wide Web Consortium (W3C) specified ontologies as "*formalized vocabularies of terms, often covering a specific domain and shared by a community of users. They specify the definitions of terms by describing their relationships with other terms in the ontology.* [13]"

Corresponding to Benjamin, Borst and Akkermans [14], first ontologies in technical domains were developed as reusable knowledge libraries. In the field of software engineering, ontologies are often used to refer to what exists in a system model [15]. Per default all software applications have an underlying ontology in form of standardized libraries, components, documentation and files, which tell the programmer what exists. However, often this is not enough or the description is poor for some reason, ontologies are precisely made for that specific purpose [16].

Through the initial work of Gruber and other computer scientists several markup ontology languages were developed. Most ontologies are based on Description Logics (DL), which are a conglomeration of knowledge representation formalisms [17]. Logical statements relating to roles in form of axioms are the fundamental of the modeling concept, which is the big difference to frame-based languages where a frame specification declares and completely defines a class. DLs are used in AI, information management and metadata integration. Within the context of the Semantic Web several languages based on DL were developed, like DARPA Agent Markup Language (DAML) [18], Ontology Inference Layer (OIL) [19], DAML+OIL [20], Simple HTML Ontology Extensions (SHOE) [21], Resource Description Framework (RDF) [22], Web Ontology Language (OWL) [23] et cetera. OWL for example, is still in a development phase, which means that the language is evolved by the W3C continuously. The first W3C recommendation of OWL came out in 2004 [24] and with the revision of OWL 1.1 in 2007 more expressiveness was added [25]. But OWL 1.1 was only another step to the further development which ended up in OWL 2. In October 2009 the W3C published a recommendation called OWL 2 [13], which obtains additional expressiveness through innovative ontological axioms to solve known problems that occur with OWL. Despite the new extensions main goal is to facilitate ontology development. The background logic of OWL is the DL $\mathcal{SHOIQ}$[24], and $\mathcal{SROIQ}$ [26] is used in OWL 2.

3 Semantic-based Information Management

Similar to object-oriented languages, a typical OWL ontology consists of instances to represent knowledge items, properties, and classes. But thinking in object-oriented terms during development with OWL will almost always lead you off target. You have to keep in mind that both modeling languages where developed among other circumstances and so have different semantic competence but there are some parallels. A comparison with the Unified Modeling Language (UML) shows that meta-models are closely related to ontologies and both are languages for modeling to describe and analyze the relations between concepts. However UML and OWL use classes in a significant different way [27]. In UML a class describes a set of software objects which entails the same specifications of features, constraints and semantics. Instance objects share their behavior from the class definition, and all objects in UML are general instances of titled classes. Instances of a class also have run-time semantics in a way that there are notions of static values and variables [28]. Contrary to this in OWL terms, a class is a labeled set of domain related things. Resources (individuals in OWL terms) are simply identifiers, not things with run-time semantics, state or storage. If an individual fits a criterion of the class, then it will be within the membership of that class. Through reasoning this could also include individuals, from whom you don't even know that they are in that class. As mentioned before, OWL has an ultimate class called "Thing", whereas all other classes are subclasses of it and individuals can only be instances of "Thing" [13].

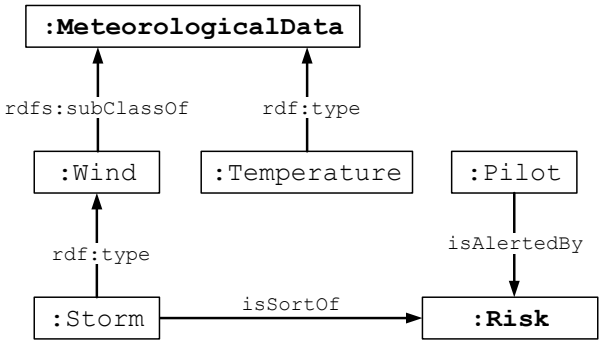

Fig. 1: A simple ontology

The real supremacy of an ontology-based approach lies in the capability to build relationships between instances and classes. The properties of those relationships will then allow reasoners to make suggestions about them. Consider a brief example (Fig. 1), `:Storm` (concept) is a specific type of `:Wind` (value of property `:MeteorologicalData`), `:Storm` could be a `:Risk` (relation), a `:Pilot` gets alerted through a `:Risk` (assertion). You can see the labeled relationships `isSortOf` and `isAlertedBy` infer the fact that a `:Pilot` is alerted by a `:Storm`, which is a specific `:type` of `:Wind` which in turn is a subclass of `:MeteorologicalData` (reasoning). This reasoning is possible because of the inverse property of `isSortOf`, which relates the two instances in the reverse direction. Several facts could be inferred from these relationships. Instances can either belong to a set of meteorological data or the set of risks, but only specific kinds of meteorological data is critical. In terms of ontology languages classes are disjoint to each other. There are no instances that belong to both. We can see from Figure 1 that a `:Storm` is some sort of `:Risk`, however with the knowledge of this example we could not conclude that some type of `:Temperature` is a `:Risk`. That is possible because OWL follows the open world assumption, which defines that any assertion not stated is indistinguishable. Individuals need not necessarily have a unique name because OWL does not use the unique name assumption.

Within the ONTOCOR project, different state-of-the-art ontology languages as well as relevant semantic environments for ontology development were analyzed and compared to each other [29], including languages like Frame Logic, RDF, RDF(S), OWL, OWL 2 and SPARQL, a standardized RDF query language. The knowledge in ONTOCOR is captured with OWL 2, due to the fact that it has the most complete set to express different concepts and relationships that occur within an ontology. As different ontology languages have different facilities, it was necessary to evaluate them. Of similar importance is the right choice of frameworks and tools. There are two different kinds of ontology related tools.

On the one hand, there are tools for the ontology development. A relevant mix of open source and commercial environments like Protégé[1], NeOn toolkit[2], Jena framework, OntoStudio, TopBraid Composer and Altova SemanticWorks were already analyzed for ONTOCOR [29]. Ontology environments like Protégé and semantic reasoners, such as FaCT++[3] or Pellet[4], are in the meantime adopted to support the ontology development process with OWL 2. The ONTOCOR framework is mainly supported by Protégé 4.1 in combination with the NeOn Toolkit. The need of selecting two environments, Protégé and NeOn, is that both support OWL 2 and have alternate strengths and weaknesses because of their different purposes. Together they are completing each other. And on the other hand, there are tools for the productive use like reasoner, alignment tools or visual representations.

3.1 Semantic Reasoning

One aim of the semantic Web is to offer machine readable metadata. Ontologies expressed by W3C's OWL 2 could improve that, in the engineering field of the semantic Web [13]. One key role of an ontology is the possibility to be processed by a reasoning system. To exploit such knowledge bases, semantic reasoning is essential as part of the ontology environment. The fact that relationships in OWL 2 are formal defined,

[1] http://protege.stanford.edu/
[2] http://neon-toolkit.org/
[3] http://code.google.com/p/factplusplus/
[4] http://clarkparsia.com/pellet

offers the possibility to use a reasoner [30]. One main service that such reasoning system can determine, is to test whether or not one class is a subclass of another class such as in Figure 1. `:MeteorologicalData` has the subclass `:Wind`. This relationship is called a necessary implication. So we could result that because `:Storm` is some sort of `:Wind`, and all types of `:Wind` are `:MeteorologicalData`, then a `:Storm` is also a type of `:MeteorologicalData`. A reasoner can conclude that the class of wind is a valid subclass of meteorological data, and that it contains at least one member. Such a test allows a reasoner to compute the ontology's inferred class hierarchy and could discover if a given class has any instances. If it cannot have any instances you can properly conclude that a class is inconsistent.

Protégé 4.1 enables the opportunity to take advantage of different OWL 2 reasoners as a plug-in. This all sounds great, but often semantic reasoners are incomplete in order to reach the required scalability, which means that they could not guarantee to provide only valid output. An excellent insight around that topic provides a paper [31] from Giorgos Stoilos et.al. published at the Oxford University Computing Laboratory. This chapter compares and describes the capabilities of some state-of-the-art reasoners, how support OWL 2 (Fig. 2).

3.1.1 Pellet

Pellet is an open source, OWL 2 reasoning system, written in Java. Original it was developed inside the MINDSWAP group at the University of Maryland, Institute for Advanced Computer Studies. Pellet is now commercial handled by Clark & Parsia LLC. The dual licensing model of Pellet allows using it for open source applications under the terms of the GNU Affero General Public License (AGPL)[5] version 3. For commercial usage it is recommended to get in contact with Clark & Parsia. In the beginning of March 2010 the release of Pellet 2.1.1 was announced and also the Pellet reasoner plug-in for Protégé 4.1 was updated. Pellet can be used directly via the Pellet interface in Jena. Based on the tableau decision procedure, which was developed for DL and Expression Language (EL), Pellet supports reasoning with the full expressivity of the description logic $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$ in order to support the OWL 2 specification. It implements procedures for general ABoxes and TBoxes.

The terms ABox (assertion box) and TBox (terminological box) in context with DL are used to describe two different types of statements in ontologies. TBox statements describe concept hierarchies, for example, relations between concepts [32]. An ABox, as compliant to a TBox, represents the statements about relations between individuals and concepts. Pellet also incorporates various optimization techniques described in the DL literature and contains several novel optimizations for nominal, conjunctive query answering and incremental reasoning [33].

3.1.2 FaCT++

FaCT++ is an efficient, open source DL reasoner for $\mathcal{SROIQ}$ compatible with OWL DL and OWL 2. It was initially developed within the WonderWeb project together with Ian Horrocks [30] and is now supported by the SEALIFE research project. It is implemented using C++ and licensed under the GNU Lesser General Public License (LGPL)[6]. Just as Pellet, FaCT++ implements optimized tableaux algorithms for ABoxes and TBoxes. One functionality of the tableaux calculus is, to check the consistency of an ontology. According to W3C's definition of OWL's semantic, a collection of ontologies "*[…] is consistent with respect to datatype map D if there is some interpretation I with respect to D such that I satisfies each ontology and axiom and fact in the collection*" [24]. FaCT++ can be used as back-end reasoner with the OWLAPI or as standalone via the DL Implementation Group (DIG) interface. As Protégé 3 uses DIG and Protégé 4 the OWLAPI, both are supported. The latest available version is 1.4.0, which was released in April 2010. Protégé 4.1 uses the

[5] http://www.gnu.org/licenses/agpl.html

[6] http://www.gnu.org/licenses/lgpl.html

Reasoner	Affiliation	Version	License	API	Expressiveness	Semantics	Rule Support	Conformance
Pellet	Clark & Parsia LLC	2.1.1	AGPL	OWLAPI, DIG, JENA	SROIQ(D)	direct	SWRL	full
FaCT++	University of Manchester	1.4.0.1	LGPL	OWLAPI, DIG	SROIQ(D)	direct	-	except keys and some datatypes
RacerPro	Racer Systems GmbH&Co.KG	pre-release 2.0	commercial, time-limited for education	OWLAPI, DIG, JENA	SHIQ(D-)	direct	SWRL, not full	not full
HermiT	University of Oxford	1.2.4	LGPL	OWLAPI 3.0	SROIQ(D)	direct	SWRL	full
OWLIM	Ontotext AD	3.3	LPGL, commercial	SAIL	OWL 2 RL	RDF-based	TRREE	full

Fig. 2: A comparison of semantic reasoning systems

plug-in version 1.4.0.1 of FaCT++ as default reasoner. Nevertheless OWL 2 is only partially supported. No support for keys or partial data types, are some of the missing things.

3.1.3 RacerPro

RACER[7] stands for Renamed ABox and Concept Expression Reasoner and was first introduced in 1997 within a cooperation of the Concordia University Montreal and the Hamburg University of Technology. RacerPro is the commercial derivate distributed by the Racer Systems GmbH & Co. KG. In addition to the commercial license, there are also a trail and a discounted license for time-limited educational purposes available. The current pre-release version 2.0 supports OWL 2 and uses tableau algorithms as inference engine. OWL 2 is only supported on syntactic level but is internally parsed as $\mathcal{SHIQ}$. RacerPro implements ABoxes for instance data and TBoxes to represent the knowledge axioms. It allows proving the consistency of these two boxes individually, computation of the subsumption hierarchy, finding inconsistent concepts, etc [34]. As the kernel operates with $\mathcal{SHIQ}$, new inventions of OWL 2 like axiom anti-reflexivity are not supported for reasoning. RacerPro could be exploited via DIG to use it with Protégé 3 and relies on the OWLAPI to use it with a RacerPro adapter[8] for Protégé 4. RacerPro embraces an own semantic query language for knowledge reasoning called new Racer Query Language (nRQL) [35]. Furthermore it offers the possible to perform queries in SPARQL syntax, whereas it is internally mapped to nRQL rules. Plug-ins allow extending RacerPro and with its own extension language called MiniLisp to define server functions.

3.1.4 HermiT OWL Reasoner

HermiT[9] was designed to process OWL and it offers the possibility to identify subsumption relationships between classes and determine whether an ontology is consistent or not. It is open source software under the terms of the LGPL version 3 and distributed by the Free Software Foundation. HermiT implements a novel hypertableau reasoning calculus for efficient reasoning, using the DL $\mathcal{SROIQ}$ with OWL 2 data type support. This approach allows a freer handling with nominals in the presence of number restrictions and inverse roles. The most important aspect is that the algorithm has much less non determinism than the previous tableaux algorithms [36]. To reduce the size of the models which are constructed, they are blocked anywhere. HermiT is pre-installed in Protégé 4.1 and the actual release is version 1.2.4, which makes use of OWLAPI 3.0. The semantic itself is processed directly as well as all conformance tests for OWL 2.

3.1.5 OWLIM

There are two different editions of OWLIM[10], SwiftOWLIM and BigOWLIM. They differ on separate Triple Reasoning and Rule Entailment Engines (TRREE[11]) and in terms of semantics, SwiftOWLIM does not support OWL 2. The different TRREE implementations have impact

[7] http://www.racer-systems.com/
[8] http://www.uni-ulm.de/in/ki/semantics/owltools
[9] http://www.hermit-reasoner.com/
[10] http://www.ontotext.com/owlim/
[11] http://www.ontotext.com/trree/

on performance and scalability [37]. Both are not open source, however SwiftOWLIM is free software under the LGPL version 2 and BigOWLIM requires a commercial license which is distributed by Ontotext. An exception is the usage in scientific environments, where the usage is free. The development of OWLIM is partly supported by the EU IST integrated project Semantic Knowledge Technologies (SEKT) and several other European research programs like the EU Sixth Framework Program (FP6). OWLIM is packaged with Sesame[12] and so benefits from the variety of supported query languages and ontology syntaxes (e.g. SPARQL, N3, Turtle, etc). The native rule-entailment engine of BigOWLIM can be configured through rule-set definitions. The rule-sets embrace RDF(S), OWL Lite and the OWL 2 RL profile. An OWL 2 profile is a synonymous for an OWL sublanguage. To improve the efficiency of reasoning, the W3C trimmed OWL 2 down to three different profiles with less expressiveness. Each one of them is made for different purposes [38]. OWL 2 RL focuses on scalability instead of expressive power.

3.2 Visual Representations

Visualization often deals with abstract data and offers a bundle of techniques to represent hierarchical or semi-structured data. There are several numbers of studies where different ontology visualization tools are compared [39], [40]. Considering the variety of methods and approaches to visualize ontologies, such tools can be separated into two big groups. One category uses variations of simple lists, the other uses simple types of visualizations like two-dimensional trees, node-links or even offers 3D information. As Protégé and the NeOn Toolkit were picked out to use within ONTOCOR, the following visualization tools are chosen to fit in that concept and therefore are compatible with Protégé 4.1 (Fig. 3).

[12] http://www.openrdf.org/

3.2.1 OWLViz

For instance OWLViz[13] is a simple visual representation tool to view class hierarchies in an ontology and is one of the further explained node-link and tree tools. OWLViz was established during the CO-ODE[14] project at the University Of Manchester. The visualization displays an ontology as a set of interconnected nodes, which is sometimes disturbing, namely if the number of nodes is very high. OWLViz hides role relationships, which is very useful. The color scheme is the native one from Protégé, so it is easy to distinguish primitives and classes. Inconsistent concepts are highlighted in red. A specific icon next to a class, signals if it is disjoint with the selected class. Particular views can be saved as image files including jpeg, png and svg. OWLViz is bundled with Protégé 4.1 and is licensed under the LGPL. OWLViz uses the GraphViz[15] algorithms delivered by AT&T, and take advantage of the Batik SVG Toolkit[16] from the Apache Software Foundation.

3.2.2 OntoGraf

OntoGraf[17] was invented at the Stanford University and is in a very early development stage. OntoGraf makes use of the visualization library from the Protégé 3 plug-in Jambalaya[18]. OntoGraf allows navigating through relationships of an OWL ontology. You can simple search or select a term in the tree. Hovering of edges shows the relationships between the terms and they can be explored through incremental expansion of the graph. Various layouts are supported and OntoGraf also allows zooming. Relationships can be filtered in order to help reducing graph complexity. For example you can narrow the focus by just showing the neighborhood of a term. Within the spring layout, which is a force-directed non-deterministic layout, each expansion re-orders the graph. OntoGraf can save a graph as a jpeg, gif, or png image.

[13] http://protegewiki.stanford.edu/wiki/OWLViz
[14] http://www.co-ode.org/
[15] http://www.graphviz.org/
[16] http://xmlgraphics.apache.org/batik/
[17] http://protegewiki.stanford.edu/wiki/OntoGraf
[18] http://www.thechiselgroup.org/jambalaya

Visualization Tools	Affiliation	Version	License	Visualization	Application
OWLViz	University Of Manchester, CO-ODE Project	4.1.1	LGPL	tree, node	Protégé 4.1
OntoGraf	Stanford University	0.0.3	-	tree, node	Protégé 4.1
SOVA	Gdansk University of Technology	alpha version	LGPL	tree, node	Protégé 4.1
Cloud View	University Of Manchester, CO-ODE Project	1.1.1	LGPL	list, tag cloud	Protégé 4.1
OWLDiff	Technical University in Prague	0.1.4	LPGL	comparison list	Protégé 4.1, NeOn Toolkit, standalone
Matrix	University Of Manchester, CO-ODE Project	1.1.1	LPGL	list, matrix	Protégé 4.1

Fig. 3: A comparison of visualization tools

3.2.3 Simple Ontology Visualization API

SOVA[19] stands for Simple Ontology Visualization API and is a brand new ontology visualization tool, which is developed at the Gdansk University of Technology. It is licensed under the terms of the LGPL and is made as a plug-in for Protégé 4.1. In this developmental stage, unfortunately it is still an alpha version, some bugs appear. But the ability to show ontology elements like (anonymous) classes, properties, individuals and relations between these objects, is very promising.

3.2.4 Cloud Views

In contrast to the previous tools, Cloud View[20] is not a standard visualization tool. It enables Protégé 4.1 to visualize an ontology as set of related tags with corresponding ratings, whereas the importance of a tag is shown with its font size. This type of visualization is called tag cloud [41]. The weight of a tag is based on the class usage, depth in the hierarchy and other criteria. The bigger the name, the higher is the rating. Cloud Views can easily filter out low ranking entities. It is available under the license of LGPL as plug-in for Protégé 4.1.

3.2.5 OWLDiff

OWLdiff[21] is one of the tools, which fall into the category of tools for list representation. The objective for OWLdiff is to compare OWL 2 ontologies and provide merging functionality for ontologies. It is developed under the LGPL at the Technical University in Prague. During the ontology development process, OWLdiff might help to maintain the overview. Similar to a versioning system OWLdiff provides abilities to compare changes and commit the resulting file. In combination with the Pellet reasoner, OWLdiff can show two ontologies, which are not semantically equivalent, in two separate trees. There are two algorithms in the background representing dissimilarities between two ontologies. To find axiom modifications, which are not visible in class hierarchies, OWLdiff uses CEX [42]. The second algorithm is much more trivial and finds simple added, missing, or changed axioms, but cannot expose complex dependencies. OWLdiff is offered as a standalone application and as plug-in for Protégé 4.1 and the NeON toolkit.

3.2.6 Matrix

Just like OWLdiff, Matrix[22] belongs to the category of list tools and was designed at the University Of Manchester within the CO-ODE project. It is available as plug-in for Protégé 4.1 under the LGPL. Matrix allows a tabular view for individuals, properties and classes. So it is possible to see either an item is in the same domain, range or if it is the inverse. Columns and values can be easily added by drag and drop.

[19] http://protegewiki.stanford.edu/wiki/SOVA
[20] http://protegewiki.stanford.edu/wiki/Cloud_Views
[21] http://krizik.felk.cvut.cz/km/owldiff/index.html
[22] http://protegewiki.stanford.edu/wiki/Matrix

4 ONTOCOR Concept

Through the static growth of data, which is processed within a control room, overlapping software components for different domains have been developed. It is often the case that the descriptive name is different, but the functionality of such components is quite the same. The motivation of this project is to show that the improvement of an ontology-based framework can be approved as a real business-case. Therefore, it is necessary to find the best fitting domains. For example, Maritime and ATM could be matched together because of their similar interfaces, or Rail and ATM, because of their comparable procedural architecture [43]. Communication, weather data, geographic information, tracking and tracing, network management and automatic identification systems are all together systems which have approximately the same requirements for different domains. Nevertheless, these components are often developed twice for each domain. The use of ontologies provides high flexibility for the future integration of new legacy applications, systems and services. Unified and open standards can raise the reuse of components for different applications in different domains. Ontologies, semantic annotation of content and semantic search are technologies, addressing the problems outlined above. They open up new ways of benefitting of already developed systems. The accessibility of knowledge and the contribution of reused software will increases the probability to deliver a project in time, in budget and with the specified capabilities. According to Dillon, "*this use of ontologies particularly when coupled with the philosophy of Web 2.0 is likely to have a profound effect on the nature of, consumption of and development of software. It is therefore important that the software engineering community takes this on board and plays a leading role in the developments that are taking place*" [44].

The fundamental concept underlying the ONTOCOR project is the Ontology-Based Software Development (OBSD) as you can see in Figure 4. OBSD covers various areas of software engineering like deployment, the definition of requirements, modeling, architecture and specification, reuse and reengineering and quality management.

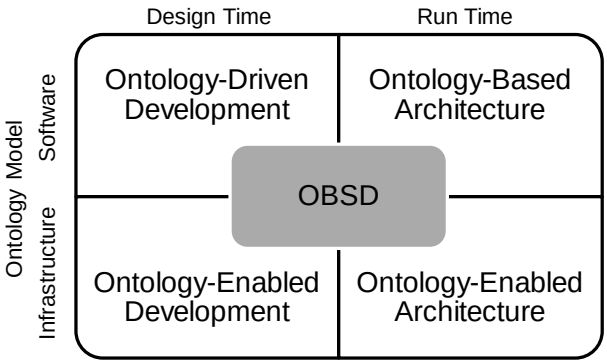

Fig. 4: Ontology-based software development

The four categories in the context of ontology-based software engineering are claimed by Happel et al [45]. OBSD extends the matrix above (Fig. 4) in order to exploit all four software engineering lifecycles. On the one hand there are two different ontology models, one for the infrastructure and one on software side, and on the other hand there are processes during design time and run time. The essential key of OBSD is to build software from reusable software components.

4.1 Methodology

In order to examine the ONTOCOR concept, a prototype has to be built. An important aspect to guarantee the success of ONTOCOR is a well-structured methodology and a detailed planned architecture of the project. The underlying architecture was already presented in a prior paper [29]. The choice of the right methodology depends on the needs of the particular ontology development. There exist a whole series of methodologies [46], [47], [48] and [49]. An important aspect is the granularity of such ontologies. To describe each and every detail is as useless as an imprecise and general description. All of the mentioned methodologies above have their pros and cons but most of them miss the ability to proof the captured knowledge against accuracy and consistency. Also very often spread is the fact that lessons learned are not even recorded or provide others as disposal. Based on the Domain Knowledge Acquisition Process (DKAP) [50], which covers most of

those issues, the methodology of ONTOCOR has slightly different steps built in. But most of all it covers the nine major steps of DKAP:

- Determine the purpose, domain and scope of the ontology
- Check availability of existing ontologies
- Organize the project
- Collect and analyze data
- Develop initial ontology
- Refine and validate ontology
- Check consistency and accuracy of ontology
- Collect additional data
- Incorporate lessons learned and publish ontology

As the ONTOCOR project is still in an early stage the definition of knowledge layers is needed. The project is separated up into different layers. It offers a primary version of a generic infrastructure for highly reliable information models between heterogeneous domains. ONTOCOR has an integrated infrastructure for a domain-specific layer, which defines patterns and configurations in any specific domain, such as Air Traffic Management. The main components of the ONTOCOR framework are split into design time and run time phase (Fig. 4). It also provides a layer for re-usable components, a layer for the domain knowledge model, and another layer for domain requirements.

The ONTOCOR framework relies on open source solutions only. Protégé 4.1 in complement with the NeOn Toolkit were selected [29] to develop the needed ontologies. In addition, the reasoner FACT++ and Pellet are selected for reasoning. The decision was based on the fact that both can get around with OWL 2 and are well supported. OWL 2 is currently the best choice, especially for building complex ontologies. It is also necessary to select visualization tools, which support OWL 2. OWLViz in combination with OWLdiff, Matrix and Cloud View will complete the list of tools which are used within the ONTOCOR project.

5 Conclusion

In order to meet the challenges of future aviation, new tools and methodologies for software development are presented in this paper. In future's environment, software development will profit from the reuse of code to improve the economic benefit. Modular architectures will be able to establish one approach for all control room environments. An Ontology-based approach will make possible advances in software-development, but to really achieve those benefits, specific Information Models for different ATM domains have to be developed. This not only requires simple UML-descriptions, but semantic logic to design and develop within a component based architecture. Such a venture has far reaching effects on systems, elements, procedures and regulations, but is necessary to achieve the benefits of an Ontology-based Framework to fulfill the key goals of NextGen and SESAR.

References

[1] Ulfbratt E, McConville J. Comparison of the SESAR and NextGen - Concepts of Operations. *NCOIC Aviation IPT*, 2008.

[2] SESAR Joint Undertaking. *European Air Traffic Management Master Plan*. 1st edition, 2009. http://www.sesarju.eu/sites/default/files/documents/reports/European_ATM_Master_Plan.pdf

[3] Joint Planning and Development Office. *Operational Concept for the Next Generation Air Transportation System (NextGen)*. 2nd edition, 2008. http://www.jpdo.gov/library/NextGen_v2.0.pdf

[4] Halevy A. Why Your Data Won't Mix. *Queue*, Volume 3, Issue 8, pp 50-58, 2005.

[5] EUROCONTROL. *Strategic Guidance in Support of the Execution of the European ATM Master Plan*. 1st edition, 2009.

[6] Wikipedia contributors. Ontology. *Wikipedia, The Free Encyclopedia*, Retrieved June 26, 2010. http://en.wikipedia.org/w/index.php?title=Ontology&oldid=369547074

[7] Aristotle. *The Metaphysics*: *Book Theta*. Cosimo Inc., 2008.

[8] Oppy G. Ontological Arguments. *Stanford Encyclopedia of Philosophy*, 2007.

[9] McCarthy J. Circumscription -- A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, Volume 5, Issue 13, pp 27-39, 1980.

[10] Hayes P J. The Second Naive Physics Manifesto, *Formal Theories of the Common-Sense World*. Norwood: Ablex, 1985.

[11] Gruber T. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal Human-Computer Studies*, Volume 43, pp 907-928, 1995.

[12] Gruber T. Ontology (Computer Science) – definition. *Encyclopedia of Database Systems*, Springer-Verlag, 2009.

[13] Motik B, Patel-Schneider P F, Parsia B. *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax*. World Wide Web Consortium (W3C), 2009.

[14] Benjamin J, Borst P, Akkermans J M, Wielinga B J. Ontology Construction for Technical Domains. *In Proceedings of the 9th European Knowledge Acquisition Workshop on Advances in Knowledge Acquisition*, pp 98-114, 1996.

[15] Wongthongtham P, Chang E, et al. Development of a Software Engineering Ontology for Multisite Software Development, Knowledge and Data Engineering. *IEEE Transactions*, Volume 21, Issue 8, pp 1205-1217, 2009.

[16] Chandrasekaran B, Josephson J R, et al. What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, Volume 14, Issue 1, pp 20-26, 1999.

[17] Baader F, Calvanese D, McGuinness D, Nardi D, Patel-Schneider P. The description logic handbook: theory, implementation, and applications. *Cambridge University Press*, Edition 2, 2007.

[18] Hendler J, McGuinness D L. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, Volume 15, pp 67-73, 2000.

[19] Fensel D, et al. OIL: An Ontology Infrastructure for the Semantic Web, *IEEE Intelligent Systems*, 2001.

[20] Harmelen F van, Horrocks I. *Reference Description of the DAML + OIL Ontology Markup Language*. 2001.

[21] Heflin J, Hendler J, Luke S, Qin Z. *SHOE: A Knowledge Representation Language for Internet Applications*. Institute for Advanced Computer Studies, University of Maryland at College Park, 1999.

[22] Klyne G, Carroll J. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium (W3C), 2004.

[23] Horrocks I, Patel-Schneider P F. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Web Semantics: Science, Services and Agents on the World Wide Web 1*, pp 7-26, 2003.

[24] Patel-Schneider P F, Hayes P, Horrocks I. *OWL Web Ontology Language Semantics and Abstract Syntax*. World Wide Web Consortium (W3C), 2004.

[25] Motik B, Patel-Schneider P, Horrocks I. *OWL 1.1 web ontology language structural specification and functional-style syntax*. World Wide Web Consortium (W3C), 2006.

[26] Horrocks I, Kutz O, Sattler U. The Even More Irresistible SROIQ. *In Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR2006)*, pp 57-67, 2006.

[27] Wongthongtham P, Dillon D, Dillon T, Chang E, Use of UML 2.1 to Model Multi-Agent Systems based on a Goal-driven Software Engineering Ontology. *Fourth International Conference on Semantics, Knowledge and Grid (SKG)*, 2008.

[28] Hesse W. Engineers Discovering the "Real World" - From Model-Driven to Ontology-Based Software Engineering. *Information Systems and e-Business Technologies*, pp 136-147, 2008.

[29] Gringinger E, Eier D, Merkl D. Ontology-based CNS Software Development. *Integrated Communications Navigation and Surveillance (ICNS) Conference*, 2010.

[30] Tsarkov D, Horrocks I. FaCT++ description logic reasoner: System description. *Lecture Notes in Artificial Intelligence*, Volume 4130, pp 292-297, 2006.

[31] Stoilos G, Grau B C, Horrocks I. How incomplete is your Semantic Web Reasoner? *In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 10)*, 2010.

[32] Wikipedia contributors. Ontology. *Wikipedia, The Free Encyclopedia*, Retrieved June 26, 2010. http://en.wikipedia.org/w/index.php?title=Description_logic&oldid=367053569

[33] Sirin E, Parsia B, Grau B C, Kalyanpur A, Katz Y. Pellet: A practical OWL-DL reasoner. *Web Semantics*, Volume 5, pp 51-53, 2007.

[34] Guohua S. Using Description Logics Reasoner for Ontology Matching. *Workshop Intelligent Information Technology Application*, 2007.

[35] Haarslev V, Möller R, Wessel M. Querying the Semantic Web with Racer + nRQL. *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics*, 2004.

[36] Motik B, Shearer R, Horrocks I. Optimized Reasoning in Description Logics Using Hypertableaux. *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction*, Springer-Verlag, 2007.

[37] Kiryakov A, Ognyanov D, Manov D. OWLIM - A Pragmatic Semantic Repository for OWL. *Web Information Systems Engineering - WISE 2005 Workshops*, pp 182-192, 2005.

[38] Motik B, Cuenca Grau B, Horrocks I, Wu Z, Fokoue A, Lutz C. OWL 2 Web Ontology Language Profiles. *World Wide Web Consortium (W3C) Recommendation*, 2009.

[39] Catenazzi N, Sommaruga L, Mazza R. User-Friendly Ontology Editing and Visualization Tools: The OWLeasyViz Approach. *13th International Conference of Information Visualisation*, pp 283-288, 2009.

[40] Lanzenberger M, Sampson J, Rester M. Visualization in Ontology Tools. *International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pp 705-711, 2009.

[41] Seifert C, Kump B, Kienreich W, Granitzer G, Granitzer M. On the Beauty and Usability of Tag Clouds. *12th International Conference in Information Visualisation (IV '08)*, pp 17-25, 2008.

[42] Konev B, Lutz C, Walther D, Wolter F. Logical difference and module extraction with CEX and MEX. *Description Logic Workshop*, 2008.

[43] Allmer G. Control Systems: Are Rail and Air so different? *IRSE News*, Issue 153, 2010.

[44] Dillon T S, Chang E, Wongthongtham P. Ontology-Based Software Engineering- Software Engineering 2.0. *Proceedings of the 19th Australian Conference on Software Engineering*, IEEE Computer Society, 2008.

[45] Happel H J, Seedorf S. Applications of Ontologies in Software Engineering. *In the Proceedings of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2006.

[46] Uschold M, Gruninger M. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, Volume 11, pp 93-136, 1996.

[47] Corcho O, Fernandez L M. Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowledge Engineering*, Volume 46, pp 41-64, 2003.

[48] Luczak-Rösch M, Heese R. Managing Ontology Lifecycles in Corporate Settings. *Networked Knowledge - Networked Media*, pp 235-248, 2009.

[49] Simperl E, Mochol M, Bürger T, Popov I. Achieving Maturity: The State of Practice in Ontology Engineering in 2009. *On the Move to Meaningful Internet Systems: OTM 2009*, pp 983-991, 2009.

[50] Sarder B, Ferreira S. Developing Systems Engineering Ontologies. *IEEE International Conference on System of Systems Engineering (SoSE '07)*, pp 1-6, 2007.

About the Authors

Eduard Gringinger, SESAR Project Member, Executive Assistant to the CEO at FREQUENTIS and Ph.D. Student at the Vienna University of Technology, Institute of Software Technology and Interactive Systems. He finished his master studies in media and computer science in October 2007 and his master studies in management and computer science in March 2008. In both studies he received distinction. His major research interests are in the areas of information processing, software development and new media. Mr. Gringinger works actively to apply modern technologies to SESAR, in areas of information modeling and flight data management, especially in the meteorological domain.

eduard.gringinger@frequentis.com

Dieter Eier is the Vice President, Business Development for FREQUENTIS USA. Mr. Eier has 18 years of experience in engineering analysis and product strategy in communications systems for mission critical applications. He was instrumental in the design of the groundbreaking new IVSR digital voice switching system for the NAS as well as the large scale MOVE voice conferencing system for all NASA space operations centers. As a proponent of networked operation Mr. Eier works actively to bring new technologies into the NAS.

dieter.eier@frequentis.com

Dieter Merkl is Associate Professor of Applied Computer Science at the Department of Software Technology and Interactive Systems, Vienna University of Technology. He received his master's and doctoral degrees in Social and Economic Sciences from the University of Vienna. His major research interests are in the areas of multi-user virtual environments, data mining and machine learning. He has published more than 140 papers in conference proceedings and journals.

dieter.merkl@ec.tuwien.ac.at

Copyright Statement