FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Ontology-Based Software Development

Semantically Enhanced Information Management and Software Component Reuse in the Air Traffic Management Industry

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

Mag.rer.soc.oec. Dipl.-Ing. Eduard Gringinger, Bakk.techn.
Matrikelnummer 0126915

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag. Dr. Dieter Merkl

Diese Dissertation haben begutachtet:

Ao.Univ.Prof. Mag. Dr. Dieter Merkl	Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gerald Futschek	Professor Markus Stumptner

Wien, 27.04.2017

Eduard Gringinger

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Ontology-Based Software Development

Semantically Enhanced Information Management and Software Component Reuse in the Air Traffic Management Industry

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Mag.rer.soc.oec. Dipl.-Ing. Eduard Gringinger, Bakk.techn.
Registration Number 0126915

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Mag. Dr. Dieter Merkl

The dissertation has been reviewed by:

Ao.Univ.Prof. Mag. Dr. Dieter Merkl	Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gerald Futschek	Professor Markus Stumptner

Vienna, 27.04.2017

Eduard Gringinger

Erklärung zur Verfassung der Arbeit

Eduard Gringinger
Aichhorngasse 5/20, 1120 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27.04.2017

Eduard Gringinger

Acknowledgments

“Chi trova un amico, trova un tesoro!”

Italian proverb [Spencer and Hill, 1981]

First of all, I sincerely thank my parents for their support and motivation during my whole life. From the beginning they gave me the freedom and time to learn, explore, and supported me in whatever I did. In addition, I am very grateful for the patience, the support, and the incessant encouragement of my grandmothers.

This thesis would have never come to live without the ongoing encouragement and help of my adviser Dieter Merkl. I am really thankful for his guidance and to seed the claim to improve my scientific work principles. I’m also very grateful for the advice of my mentor Hannes Bardach. He gave me the freedom to pursue my own research interests while working full-time.

I want to thank all my colleagues and scientists for their creative input preparing papers and there economic know-how. Furthermore, I want to thank, the various members of the international scientific air traffic management community I had the privilege to meet, for all the interesting discussions and valuable comments on my thesis. This includes the anonymous peer reviewers who did not make me happy all the time, but nevertheless contributed to the quality of the work. And finally I would like to thank my friends, without whom the world would not be such a great place.

Abstract

This thesis addresses the reusability challenge to reuse software components across various domains. By using ontology-based technologies, software development costs shall be lowered and the semantic interoperability gap shall be closed. The main goal is to improve the quality of the software products in order to save costs. Ontology-Based Software Development tries to address significant problems of traditional software development in order to improve the code quality and to avoid redundant development. It focuses on improving efficiency and increasing the code reusability. The thesis follows the design science research paradigm to answer the defined research questions. Domain-specific development leads to similar software solutions for different areas. Usually this raises the costs for efficient software development and increases the time-to-market. The motivation for this thesis is to show the benefits of an ontology-based methodology to enhance software development with semantic technologies and further, to avoid redundant development of similar software-components within large companies.

To meet the requirements, state-of-the-art information management techniques are evaluated and picked to conquer the reusability challenge. The operational context and the special requirements, a safety critical environment is accompanied by, are described. The Ontology-Based Software Development life-cycle is outlined in detail including the methodology, semantic concept, and techniques for the underlying processes. The semantic information is captured in models which are then transferred into the solution model. The various roles and responsibilities which are required by the different processes are mentioned. The key principles used to design the knowledge base are described. The ontology mediation further details the reasoning, the semantic mapping and the semantic description transformation process. This thesis examines if ontology-based development methods are capable of reusing software components efficiently. A case study was carried out to evaluate and analyze the differences between classical development processes and the newly introduced concept. The evaluation framework is intended to provide evaluation measurements with appropriate metrics, which should enable an assessment of the presented approach. The goal is to incorporate feedback from the evaluation into the Ontology-Based Software Development processes in order improve the existing processes.

Kurzfassung

Diese Dissertation befasst sich mit der Wiederverwendbarkeit von Softwarekomponenten über verschiedene Domänen hinweg. Darüber hinaus sollen Softwareentwicklungskosten gespart und die semantische Interoperabilitätslücke durch Ontologie-basierte Technologien geschlossen werden. Das Hauptziel ist, die Qualität der Produkte zu verbessern, um Kosten zu sparen. Ontology-Based Software Development versucht, wesentliche Probleme in der traditionellen Softwareentwicklung zu lösen, um die Codequalität zu verbessern und eine redundante Entwicklung zu vermeiden. Das vorgestellte Konzept konzentriert sich auf die Verbesserung der Effizienz und die Erhöhung der Code-Wiederverwendbarkeit. Die Arbeit folgt dem Design Science Research Paradigma, um die definierten Forschungsfragen zu beantworten. Domainspezifische Entwicklung führt oftmals zu ähnlichen Softwarelösungen in unterschiedlichen Bereichen. In der Regel steigen dadurch die Kosten für die Softwareentwicklung und die Entwicklungszeiten erhöhen sich. Die Motivation hinter dieser Arbeit ist es, die Vorteile einer Ontologie-basierten Methodik zur Verbesserung der Softwareentwicklung mit semantischen Technologien zu demonstrieren und die redundante Entwicklung ähnlicher Software-Komponenten in großen Unternehmen zu vermeiden.

Um diesen Anforderungen gerecht zu werden, wird der letzte Stand der Technik hinsichtlich Information Management-Techniken ausgewertet und dementsprechend ausgewählt, um die Herausforderung einer sinnvollen Wiederverwendbarkeit zu meistern. Um die speziellen Anforderungen zu verdeutlichen, die ein sicherheitskritisches Umfeld mit sich bringt, wird der operationelle Kontext beschrieben. Der gesamte Ontology-Based Software Development Prozessablauf wird ausführlich beschrieben. Dabei werden die angewendete Methodik, das semantischen Konzept und die verwendeten Technologien erklärt. Die semantischen Informationen werden in Modellen erfasst, die dann in ein sogenanntes Lösungsmodell übertragen werden. Darüber hinaus werden die verschiedenen Rollen und Verantwortlichkeiten definiert, die von den unterschiedlichen Prozessen benötigt werden. Beschrieben werden auch die wichtigsten Grundsätze für die Erstellung der Wissensbasis. Der Transformationsprozess generiert aus der semantischen Informationsbasis ein Solution Model, das jeweils auf konkreten Anforderungen zugeschnitten ist. Diese Arbeit untersucht ob Ontology-Based Software Development in der Lage ist Softwarekomponenten effizient wieder zu verwenden. Eine Fallstudie soll die Unterschiede zwischen klassischen Entwicklungsprozessen und dem neu vorgestelltem Konzept evaluieren und analysieren. Das Evaluierungsframework soll mit entsprechenden Metriken Auswertungskennzahlen liefern, die eine Bewertung des vorgestellten Ansatzes ermöglichen. Ziel ist es Feedback der Evaluierung in die Ontology-Based Software Development Prozesse einfließen zu lassen um Verbesserungen zu ermöglichen.

Contents

CHAPTER 1

Prologue

"Basic research is when I am doing what I don't know what I am doing."

[von Braun, 1957]

This chapter introduces Ontology-Based Software Development. It provides an overview about Ontology-Based Software Development issues, being the problem statement and motivation for the choice of the subject of this thesis. Furthermore, the scientific context is explained and the research question is defined. Applied evaluation methods for the accomplished case study are described. Last but not least it gives a condensed overview about the structure of this thesis.

1.1 Motivation

A couple of years ago handling large sets of data was made possible through the enormous performance growth of computer hardware. During the last years big data has become a new ubiquitous buzz word. Gartner defined Big Data as

> "*high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization*" [Gartner, 2012].

The use of ontologies for software development is an attempt to handle big data in a smarter way [Rus and Lindvall, 2002]. Knowledge-based approaches gained popularity during the 70s in combination with artificial intelligence [McCarthy, 1987]. The conventional approach focuses on how to enrich the software development process through extra knowledge in order to make software development an efficient, predictable activity which reuses already gained knowledge for further projects and/or products. Earlier ontology-based procedures address organizational, individual, and technology issues like how to store requirements and how to share knowledge across developers [Gašević et al., 2009], [Siricharoen, 2007].

The main disadvantage of all these approaches is that they do not support domain independent software development. The possibility to reuse software components is the main research goal of this thesis but all those knowledge-based approaches are built around the software development to support it from the outside. As software engineering is a knowledge-intensive domain the main idea is to support the software engineering process during the implementation phase in order to improve software development in its core tasks. In addition this cognizance can be used to link it with the software development on the level of components. With this support, software architects can identify reusable components across domains. Moreover it can be used to support offers as customer requirements can be matched not only to data elements but also to software elements. Figure 1.1 demonstrates what the customer expects from a software product and on the other hand what the developer has to consider. The International Organization for Standardization (ISO) 25010 standard[1] was developed for the evaluation of software quality. The fundamental objective of this software quality model is to address some of the quality characteristics of the four main columns. Ontology-Based Software Development (OBSD) can be deployed to increase the quality of the product, product strategy, product line, and architecture life-cycle as shown in figure 1.1. The information exchange between applications and platforms can raise the interoperability, flexibility and expandability.

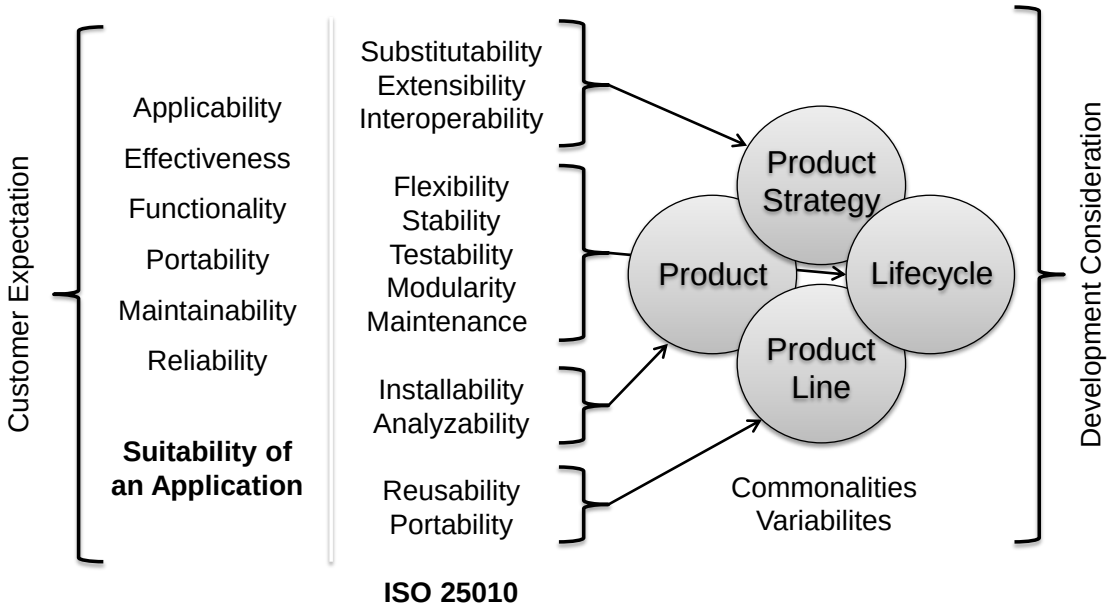

Figure 1.1: How to create quality software.

This thesis addresses the combination of OBSD and semantically enhanced information management beyond the work currently envisaged by the industry and several research projects and programs. To verify, validate, and evaluate the domain independent idea of OBSD within a case study, the Air Traffic Management (ATM) domain was chosen. There is a tremendous need

[1] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733, accessed: 2016-11-27

from the industry as well as from the scientific community to develop better solutions through scientific results. The static growth of data which is processed within a control room results in the development of overlapping, or in the worst case, identical software components for various domains. Often it is the case that the descriptive name is different but the functionality of such components is quite the same. Motivation of this thesis is to show the improvement which an ontology-based life-cycle can offer in a real business-case. Components dealing with communication, weather data, geographical information, tracking and tracing, validation and mediation are all artifacts which have approximately the same requirements for several 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, open standards raise the reusability of components for different applications in different domains. This thesis contributes to such solutions through innovative research integrating both areas. To reach this goal, a shift in software development paradigms has taken place. New emerging Ontology-Based Software Development paradigms are dealing with the underlying data itself. They place less emphasis on the process, but focus on domain independent code reuse ability and efficiency starting at the core of software development covering the information management.

Furthermore, an ontology-based approach tries to address significant problems in traditional software development [Calero et al., 2006]:

- **Loss of Knowledge**: Due to steep learning curves and attrition a substantial part of the knowledge gets lost. Miscommunication and the lack of semantic interoperability means that information is lost or altered.

- **Single Point of Knowledge**: A serious internal problem of an organization are single individuals with key knowledge as they have an incredible amount of leverage but might not be able to share it.

- **Reinvention of the Wheel**: When companies grow knowledge interchange is a serious issue. It is more than common that one department is developing software components which are already implemented or covered by another division.

- **Domain Specific Development**: This leads to the development of domain specific software resulting in different solutions for each specific environment. Usually this raises the costs for efficient software development and increases the time-to-market.

- **Lack of Feedback Cycles**: In software engineering, inefficient feedback cycles increase costs and cause delays. Processes should be able to handle changes and allow rapid feedback.

1.2 Definition of Ontology-Based Software Development

Ontology-based technologies are used for acquiring and storing semantic descriptions. Ontologies allow computer machines to process domain overlapping information, which up to now only was defined in operational procedures independent of automation systems. An ontology does not necessarily contain all information but can be split up in multiple ontologies referencing and inheriting properties from other ontologies. Ontologies can be used to enhance decision making and enrich existing data with more logic [Gartner, 2012]. From a marketing perspective, one would speak of smart big data [Buhl et al., 2013]. This smart way is the main driver for this thesis. Section 2.3 provides a definition of ontology in computer science and gives an historical insight.

For a better understanding among the various ideas how ontology-based approaches could be used [Happel and Seedorf, 2006] created four categories for ontologies in the context of software engineering (cf. figure 1.2). This simple classification scheme provides an overview about different semantically enriched software engineering life-cycles. The Ontology-Driven Architecture [Tetlow et al., 2006] from W3C was the starting point and attempts to outline how semantic Web technologies can be applied in the field of software engineering. The basic properties of ontology-based software engineering can be described in a 2x2 matrix. On the one hand there are processes during run-time and design-time and on the other there are two different ontology models, one capturing the software components and one describing the domain. [Happel and Seedorf, 2006] describe an Ontology-Driven Development, which is processed at design-time, covering the domain itself. The Ontology-Enabled Development in contradiction, supports the developer during the design-time (e.g. component search). At run-time Ontology-Based Architectures are used for business rule approaches whereas the ontology is the main part of the application logic. Ontology-Enabled Architectures supply software infrastructure to support a system at run-time. Through an additional semantic layer provided by ontologies, web services gain automatic discovery and compose service-based work-flows.

OBSD [Gringinger et al., 2010a] extends the matrix outlined by combining Ontology-Driven and Ontology-Enabled Development (cf. figure 1.2). It considers different fields of software development during design time, including the definition of requirements, architecture and specification, modeling, reuse and re-engineering, quality management, maintenance and deployment [Dillon et al., 2008]. Core focus of OBSD is to compose software from reusable software components. More and more software projects reuse code instead of developing it completely new from scratch. To fulfill the dream of code component based software engineering the developers need the corresponding knowledge about the reusable parts in relationship to the requirements. Often ambiguity requirements cause minimal errors, which can endanger a whole project. However, a semantically enriched modeling process improves the mix-up of requirements and reusable code components by using ontology techniques. OBSD makes it easier to select the precise requirements with the best fitting existing code components.

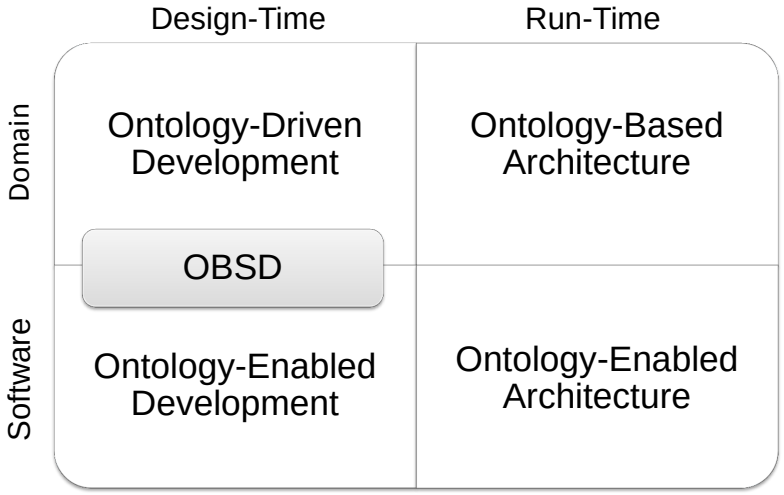

Figure 1.2: OBSD in the context of [Happel and Seedorf, 2006].

1.3 Problem Statement

When trying to integrate ontology-based information management into software engineering several integration problems appear. This section covers these problems, describing the problem statement and the motivation for the choice of the subject of this thesis.

1. **The Reusability Challenge**

   Although the reusability of software components is not a new concept it is a constant problem. To achieve the possibility to reuse software components and interconnect them with domain information models and requirements is complicated. There are numerous reasons why it is not a trivial problem.

   The major difficulties lie in the technical areas, in particular:

   - First of all mapping knowledge from a well-understood base to a new target domain is not an easy task [Lung et al., 2007]. Domain specific development leads in the worst case to various solutions for each specific environment. This increases software development effort and lowers the efficiency. Software architects have to think further than to the upcoming deadline. The target to achieve is reusable, modular software components rather than monstrous and monolithic code.
   - Another important aspect of reusable software components is to design them as a black-box [Lung et al., 2007]. This means that each component can interact with the rest with an interface as an abstraction. As a result, one does not necessarily have to know the internals of a component to be able to reuse them. The right level of abstraction is important.

- Semantic interoperability issues can cause the redundant development of components in various domains, due to different definitions. Not to reinvent the wheel is one of the biggest challenges.

Common libraries and components are usually developed with the consumer in focus instead of the industry.

2. **The Semantic Interoperability Gap**

   Insufficient knowledge distribution is a problem with many manifestations. For example there are several standards which have various definitions for one and the same thing. Consequently that the information cannot be distributed. The cultural gap between Stakeholders due to the lack of semantic interoperability means that information is lost or altered. As a consequence it is not verified that the exchanged information share the same meaning at both their origin and their destination which is necessary to receive, combine, and process information from many sources. Overlapping data models from several domains which define the same information differently may have serious implications. To find a reliable, common definition between the various stakeholders is not an easy task. Moreover it is most of the time not a technical but a political problem. Semantic information needs to be collected from the operational as well as from technical and political side to find a common agreement.

3. **The Modularity Fallacy**

   To find the right granularity while building software, based on Service Oriented Architecture (SOA) principles, is not an easy task [Zdun et al., 2007]. Defining services on the SOA patterns is one thing but finding the right modularity from components down to packages is another story. Implementing SOA modularity creates new and additional challenges for people, processes, and technology that must be addressed through sound and effective governance. Without such governance, business agility is impossible, service ownership remains locked within silos, service portfolio management remains ineffective, and security is not providing an enterprise-wide view [Papazoglou and Heuvel, 2007]. Not only on the governance side things have to change, also the meta-data and the software documentation has to come to the next level to support the idea of modularity.

4. **Insufficient Requirements**

   In order to be able to infer from the requirements to the reusability of software components, the requirements must be of excellent quality. Software requirements are usually dynamic, pervasive, and hardly specified. Consequently, applications are difficult to be well-defined entirely in advance. If the requirements are unclear, incomplete, too general, or simply not testable they cause trouble. A good requirement analysis includes the interaction between customers and developers. Unfortunately, this is still often underestimated.

5. **Software Incompatibility**

   Software incompatibility is the inability of independent, heterogeneous systems to work together as seamlessly as possible to replace or provide the necessary information in an efficient and usable manner to the user without the need for separate agreements between the systems. The worst case is that applications cannot process the same data within the same product family. On the other hand with interoperability in place the best possible case would be that an application is able to communicate seamlessly without additional interfaces and data mediation with third party data systems. It is definitely a challenge to ensure maximum compatibility.

6. **Weak Offer Management Support**

   Without the knowledge of the capabilities of a software product, presales and offer management is incapable of estimating correctly the effort needed when reading the customer requirements. The more accurately the estimation of the requirements for the costs of software development is, the better. Of course it also helps if someone knows which components are in development or are already developed. A weak offer management prevents an application to develop from a single implementation towards an own product.

7. **Dispensability of Ontology-Based Information Management**

   Some software architects and managers cannot afford to spend too much time on information management. They worry that ontology-based information management will cost more effort than it will bring back over time. First of all, there should be measurable parameters in place to quantify what information management really caused at additional costs. And Secondly the long-term factors of the benefits ontology-based information management should be kept in mind.

8. **The Lack of Ontology-Based Software Development Tools**

   Ontology-based techniques and tools are primarily developed for the Semantic Web. Addons, extensions, and plug-ins are mainly implemented to support the specific needs of the Semantic Web. Most often software developers working with OBSD complain that these tools are scarce. But even if there were already enough theoretical concepts, there is a need for OBSD tools.

9. **The Need of Standardized Domain Knowledge**

   In various domains appropriate domain models do not exist. But without them it is not possible to conquer the semantic interoperability gap. Even if there are standardized exchange models most of the time the knowledge is not captured as an ontology. It is essential that there are suitable semantic models for each standardized data and service model.

1.4 Scientific Context

The idea for this thesis was born in 2009 and took place between January 2010 and December 2016. The work was embedded in the science framework of an international research program and several projects. The **Single European Sky ATM Research program** (SESAR) was founded by the European Union, European Organization for the Safety of Air Navigation (Eurocontrol) and fifteen industry partners. The development phase of the SESAR program started in June 2009 in order to ensure the modernization of the European ATM system. The work was conducted in two work packages (WP) of the System Wide Information Management (SWIM) thread, in the Information Management (WP 8) and System Wide Information Management Technical Architecture (WP 14) WP. SWIM establishes concepts and mechanisms which combine the forces of all suppliers of shared ATM information in terms of Eurocontrol's strategy [Eurocontrol, 2009]. SESAR addresses four key goals according to the masterplan [SJU, 2015]:

- Improve Safety by a factor of 10.
- Enable European sky to handle 3 times more traffic.
- ATM costs reduction by 50%.
- Reduction of the environmental impact per flight by 10%.

Information management establishes the governance framework in terms of regulatory and support functions needed to perform system-wide information sharing. Sensitivity with regard to some information continues to exist and is managed within the information management WP. ATM information changes over time, but to varying degrees in terms of frequency or magnitude, varying from almost static to very dynamic. Information management recognizes and accommodates this temporality of information for the aeronautical, weather, flight and environmental domain in the ATM Information Reference Model (AIRM). It also covers the semantic description used in this thesis to build the ontologies. Another main purpose is to develop information services based upon Service Oriented Architecture (SOA) principles for current and assumed foreseen ATM business needs. The diversity of services are collected in the Information Service Reference Model (ISRM). The information management strategy is based upon the following main objectives [Gringinger et al., 2010b]:

- Seamless information interchange through globally and universally defined interfaces by removing existing barriers between systems.
- Open standards such as information models published in the public domain shall be used.
- Information on demand and filtered by geographic and temporal information.
- Quality of information by ensuring information timeliness, accuracy and quality.

The SWIM Technical Architecture WP establishes and validates the infrastructure solution of SWIM [Gringinger et al., 2012b]. The primary objectives are architectural description, technological options and system solutions regarding the requirements provided by the information management WP. This includes the development of technical services and data models defined in the information management WP.

Additional semantic work was accomplished during the science research project "SemanticNOTAMs: Ontology-based representation and semantic querying of Digital Notices to Airmen", funded by the Austrian Research Promotion Agency (FFG) in 2012 (cf. [Gringinger, 2014], [Burgstaller et al., 2015], [Burgstaller et al., 2016], [Steiner et al., 2016b], and [Steiner et al., 2016a]). Digital Notices to Airmen (DNOTAM) were exchanged in teletype format in the past. Supporting the move to digital ATM along a common AIRM, Eurocontrol and FAA drafted event specifications for DNOTAM according to the Aeronautical Information eXchange Model (AIXM) 5.1. Intelligent filtering and querying of Digital DNOTAMs was identified as important but is still an unsolved issue to fully exploit their potentials in future Air Traffic Management. This is where the project sets in. The project intends to use a knowledge-based approach exploiting semantic technologies for tackling the problem of efficient, flexible and context-aware filtering and querying of DNOTAMs. Its results are part of new support systems (such as digital briefing) that assist flight controllers and pilots in reducing their information overload. The project complements the key technologies of SESAR, AIRM and ISRM, in that it investigates querying and notification of DNOTAMs so far not considered in this frame and uses a knowledge-based approach, novel to this domain. By employing a knowledge-based approach, knowledge about what DNOTAMs are relevant for and how important they are in a given context (e.g. flight phase, aircraft, time, space) is represented as data (and not in program code). Thereby, systems can be easily modified and personalized. Moreover, not only DNOTAMs themselves but also knowledge about filtering and querying can be digitally exchanged in the SWIM of SESAR between different services, thus, enabling cross-system reuse of filtering and query rules without any need for reprogramming. The insights gained and results achieved in this project on filtering and querying time- and safety-critical announcements to global air traffic are also valuable for future adaption of terrestrial and nautical traffic management.

The principle goal subsumes three sub-goals, including the support of climate-protection-oriented air transport systems, the support of efficient, secure and comfortable air transport systems, and the intensification of cooperative and challenging research projects at national and international level.

- Intelligent filtering and querying of DNOTAMs based on contextual relevance in conjunction with immediate digital delivery reduces the time to promulgate information concerning airspace status. This will allow more effective airspace utilization and allow improvements in trajectory management. Optimized arrivals, departures and reduced queues help to decrease environmental and noise pollutions.

- Intelligent filtering and querying of DNOTAMs reduce information overload of air traffic controllers and pilots and assist them in organizing relevant DNOTAM information at hand.

- Reduced stress level and situation-aware DNOTAM notification at situation-appropriate alert levels contribute to avoid air traffic incidents and thus, to air safety.

1.5 Research Question and Research Approach

All these problem statements emerge from one main research question:

Is it possible to solve the reusability challenge to reuse software components across various domains?

The main research question can further be divided into two sub-questions. As a result the main research question will be satisfied by giving a solution to the two following problem statements:

1. Can software development costs be lowered with the help of ontology-based technologies?
2. Which strategy closes the semantic interoperability gap?

This work mainly focuses on the main research question covered in chapter 2, 4, and 5. The first sub-question is examined in detail in chapter 4, and 5. The second sub-question is described in chapter 3 and 4.

1.5.1 Information Systems Research

This thesis follows the design science research paradigm described by [Hevner and Chatterjee, 2010] to investigate the research questions above. The information systems research addresses controversial discussions about the role of the information technology artifact and the lack of information systems research in a professional context cf. [Vaishnavi and Kuechler, 2013]. Since the late 40s research in design methodologies were adopted from other domains for the information technology and information systems domain. More and more technical disciplines and specializations such as computer science and electrical engineering were added over time. In the early 90s the information systems community recognized the importance of design science research and after a number of review cycles [Hevner et al., 2004] published the design science research paradigm. The design science research paradigm tries to solve a problem by building an application of the designed artifact. Within this work the development of new ideas are accomplished by the creation and evaluation of prototypical artifacts. The design science research cycles are shown in figure 1.3. The environment part on the left side defines the problem statement within a specific domain including the involved people, organizational, and technical

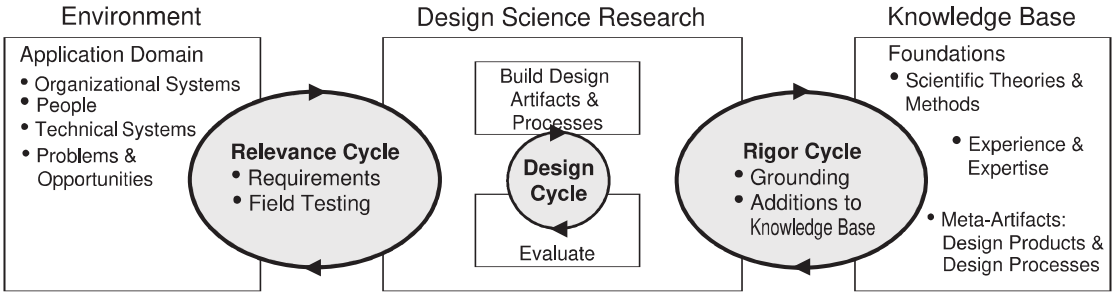

Figure 1.3: Design science research cycles [Hevner and Chatterjee, 2010].

systems. The relevance cycle iterates between the domain environment and the design science part of the research project. Not only does it provide requirements for the project as input (e.g. problem statement), it also defines verification and validation rules for the final evaluation of the research results. The design science research part identifies in an iterative way theories and artifacts. The design cycle connects the building and validation of the design artifacts in a design science research project. The rigor cycle bridges these activities with the knowledge base of scientific methods, meta-artifacts experience, and expertise. These three cycles are the main part of design science research.

Furthermore, as part of information systems research, [Gregor and Hevner, 2013] proposed a knowledge contribution structure for design science research (cf. figure 1.4). *Improvement* specifies new solutions for known problems, whereas an *Invention* creates new solutions for new problems. *Exaptation* conclude non-trivial extension of known solutions for new problems and research contributions while *Routine Design* is rarely acknowledged as such. It is a known issue that software components are developed for a certain domain and that even within the same company in another domain more or less the same software components are built from scratch just because the knowledge about the specific artifact was not shared. As Ontology-Based Software Development is a quite new research area, new issues and problems have surfaced. To raise the awareness of reusable software components inter-domain specific, this thesis uses novel techniques, such as OBSD and semantically enriched information models. According to the knowledge contribution structure, the research work in this thesis can be classified as an *Improvement* and *Invention* providing new solutions to known and new problems (cf. figure 1.4). The OBSD methodology was awarded with the innovation idea award[2] at a conference hosted by the Institute of Electrical and Electronics Engineers (IEEE) and the American Institute of Aeronautics and Astronautics (AIAA) cf. [Gringinger et al., 2010b]. As this thesis also tries to improve the semantic interoperability gap, parts of it are located in the quadrant of improvement.

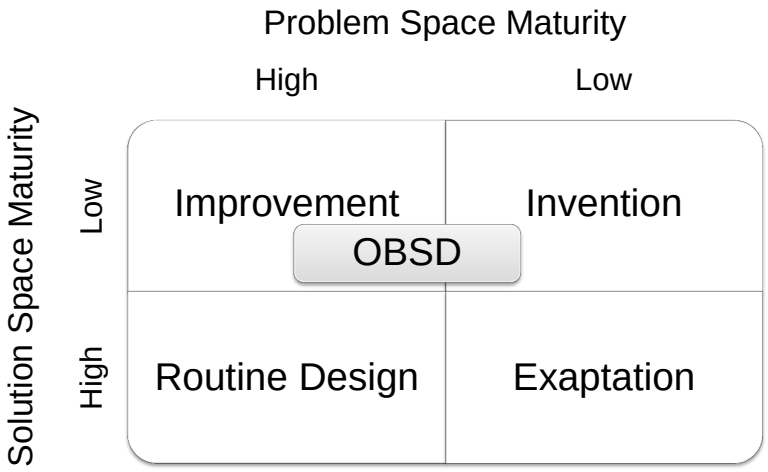

Figure 1.4: Design science research structure adopted from [Gregor and Hevner, 2013].

[2] http://i-cns.org/2010/student-paper-award-winners-announced-for-icns-2010/, accessed: 2016-10-23

Figure 1.5 visualizes the iterative methodology process model. It offers the possibility to start the research from a variety of contexts: problem-centered, objective-centered, design and development centered, or client/context based. For this work the problem-centered initiation in the gray oval was used as a starting point defining the problem statement and its motivation. The next step was to think about appropriate artifacts to face this problem, followed by a design and development phase. During the demonstration phase two artifacts were selected (cf. section 1.4). Verification and validation of these artifacts were done to evaluate the efficiency and effectiveness. Finally the results of the research work were presented through various publications, patents and this thesis.

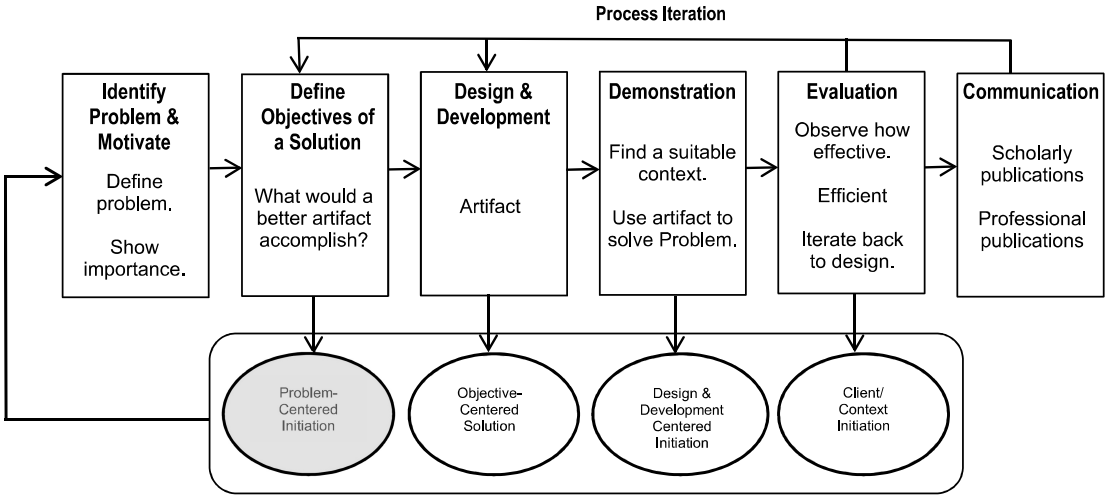

Figure 1.5: Adopted design science research methodology process model [Peffers et al., 2006].

[Hevner et al., 2004] defined seven guidelines to assist researchers and readers for conducting top quality design science research. Based on those guidelines a more specific checklist of questions was developed by [Gregor and Hevner, 2013], to evaluate a design research project. The checklist ensures that a project addresses the key aspects of design science research. These eight questions are briefly answered in section 6.1, accompanied by an explanation, how the thesis answers these questions.

1. What is the research question?
2. What is the artifact? How is the artifact represented?
3. What design processes will be used to build the artifact?
4. How are the artifact and the design processes grounded by the knowledge base?
5. What evaluations are performed during the internal design cycles?
6. How is the artifact introduced into the application environment and how is it field tested?
7. What new knowledge is added to the knowledge base and in what form?
8. Has the research question been satisfactorily addressed?

1.5.2 Evaluation Framework for Case Study

When starting to work on a case study, or any form of research, it is important to consider the validity of that research [Basili, 1992]. For this thesis evaluation metrics are used to demonstrate artifact utility and improvement by analyzing the case study with ontology-based approach against the previous developments without OBSD of the same product. For the purpose of comprising the result of the ontology-based approach, the case study is evaluated by means of using an adopted version of the Extreme Programming Evaluation Framework (XP-EF) [Layman et al., 2004b].

The XP-EF provides a benchmark for expressing agile case study information for researchers and users to assess concretely the extent new software development practices and the results of those adoptions [Layman et al., 2004b]. The XP-EF is a compilation of validated and proposed metrics and was designed to allow companies or organizations recording case studies, the differences of changes or adoptions which took place, and capture results of this adoption regarding the agile software development process [Layman et al., 2004a]. All metrics are parsimonious and lightweight so that the data can be collected even without dedicated metrics knowledge [Layman et al., 2004b]. The framework offers the possibilities for combining various case studies. Within the context of this thesis the XP-EF was used exclusively to compare prior software development against the OBSD processes.

The framework consists of three parts:

1. **Context Factors**

   Since evaluation and empirical studies in the field of software development are non trivial due to the variable framework conditions the context factors record fundamental information about the background of the respective project cf. [Layman et al., 2004a]. This is essential for the comparison purpose of the case study to completely understand the differences and similarities between these different approaches.

2. **Adherence Metrics**

   The adherence metrics are the second part of the evaluation framework and trace how far the measures were taken on board. It is often the case that software development teams are not as pleased with reusing components as stated in the first place. Whereas the reasons for this can be manifold the XP-EF adherence metrics comprises objective and subjective measures to qualitative analyze the component integration practices of the team and to which extend the OBSD methodology was used.

3. **Outcome Metrics**

   Finally the outcome measures evaluate business-oriented metrics [Layman et al., 2004a]. Conventional software development metrics like quality, productivity, etc. are taken into account to assess and evaluate the projects outcome. As stated before, within this work the XP-EF is used exclusively to compare old software development methodology against the OBSD process. Therefore, the outcome metrics of the framework are an important aspect.

1.6 Structure of this Thesis

Most work in this thesis was published as conference papers, in journals, white papers, technical reports, as book chapters or are still under review. These core papers build the foundation of this thesis. The published ones are also listed in the Appendix B. The following list contains a short description of the contents of the chapters of this thesis. In addition the used methodology is described in section 1.5.

- The prologue, chapter 1, examines the motivation and definition of Ontology-Based Software Development. Furthermore, it provides the problem statement, the motivation for the choice of the subject of this thesis and an overview about the scientific context. This thesis follows the design science research approach by [Gregor and Hevner, 2013] that contains good guidance on how to conduct, evaluate and present a scientific project. The evaluation method and research contributions, following the design science research guidelines, are outlined.

- Chapter 2 covers the technical context in which this research took place. The development of ontology-based tools was steadily increased and enabled the idea of Ontology-Based Software Development. Different ontology languages, as well as relevant semantic tools for ontology development, are analyzed and compared. In order to meet the challenges outlined in the prologue considerable extensions and improvements are necessary for an efficient Ontology-Based Software Development environment. This chapter is a refined version of the publications [Gringinger et al., 2010b] and [Gringinger et al., 2010a].

- Chapter 3 introduces the operational field of this thesis and discusses recent aspects of information management within the Air Traffic Management domain. It reflects on the used concept, design, and technologies of a System Wide Information Management infrastructure based on a Service Oriented Architecture approach. The operational background for the validation implementation is discussed. Specific information models for different ATM domains are introduced and the way of how to derive related ontologies on top of them is shown. This chapter is a refined version of the publications [Gringinger et al., 2011], [Gringinger et al., 2012b], and [Gringinger et al., 2012a].

- In chapter 4, the concept of Ontology-Based Software Development is presented. The approach describes the usage of ontologies for data representation and the semantic interface for accessing the information stored in the Ontology-Based Software Development ontologies. This covers the used methodology, semantic concept, and techniques that are used for the development process, which derives a logical solution model. This model is calculated from the semantic description based on software components model, domain knowledge model, and customer requirements. This chapter is a refined version of the publications [Gringinger et al., 2011] and [Gringinger, 2017] which is still under review for the 7^{th} SESAR Innovation Days.

- Chapter 5 describes the development and validation of an OBSD relevant case study analyzed with the evaluation framework XP-EF. For the case study of this thesis two product developments were chosen using the Ontology-Based Software Development methodology and compared the results of prior development of the the same product. It is shown how an semantic-based development process facilitates software development and preserves development costs. Furthermore, a summary of the results of the reusability case study of the porducts developed by means of the adapted development process is outlined. This chapter is a refined version of the publications [Gringinger et al., 2013] and a proceeding which is still under review for the SESAR Innovation Days 2017.
- Finally in the epilogue, chapter 6, general conclusions about the achieved results are drawn and interesting topics for future research are outlined. In addition the eight design science research questions raised in the prologue chapter 1 are answered and considerations about possible future work and future research are given. The reflections represent the lessons learned after the case study presented in chapter 5.

CHAPTER 2

Information Management Engineering

"Hell! There ain't no rules around here! We are tryin' to accomplish somep'n!"
[Edison, 1903]

The development of ontology-based tools is steadily increasing, especially due to the success of the Semantic Web. This chapter analyzes and compares different ontology languages as well as relevant semantic tools for ontology development. Therefore semantic standards, tools, and extensions are examined in a technology evaluation for Ontology-Based Software Development. Results show that considerable extensions and improvements are necessary for a successful setup of an Ontology-Based Software Development life-cycle.

2.1 Introduction

Control rooms are typically found in the security, public safety, public transport, 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 business scope changes, for example to combine two existing parts, some sort of integration is needed [Halevy, 2005]. To win the challenges of the data and system integration, a life-cycle, which defines seamless information interchange and connects it with reusable components, is needed. A specific example inter domain development, is the European 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 [Gringinger et al., 2012b].

Exactly within these circumstances an ontology-based approach can bring the break through. Semantic structures improve the productivity and increase the reusability of software through an ontology-based based life-cycle. This chapter discusses corresponding software languages and development tools, which are a critical part in building up an ontology-based life-cycle. Unified and open standards raise the reuse of components for different applications. One issue was to evaluate the best fitting ontology language. Some languages are very complex and need tools for editing and developing Ontologies. More often, these tools are extensions or plug-ins for an Integrated Development Environment (IDE). Existing and approved standards were compared to make the right choice. The first chapter summarizes related work regarding ontology languages, presenting generic ontology languages and tools, and conclude a short description of existing ontologies which are related to Ontology-Based Software Engineering. In addition reasoner and visualization tools have been investigated for OBSD.

2.2 Definition of Information Management

Information management is more and more important to assimilate the enormous data collected. Standardized data models are developed to support information systems by providing the definition, relationship and format of data, controlling, processing, and evaluating [Haag et al., 1997]. The term “information” has various associations and definitions depending on the context. Originally, it comes from the Latin word “informatio” which means a certain image and concept. In everyday language, the term is used nonspecific. Information simply tries to cover knowledge about facts, processes or ideas. In the computer science industry “information” is used among the terms “data” and “character”. Whereas “characters” serve as the basis and “data” formalizes information for transfer, interpretation, or processing. It does not matter whether the processing of data is handled by humans or automatically. The concept “information” originally came from mathematics and has been further developed in the gray field of statistics and computer science and was introduced by Claude E. Shannon as quantitative information theory [Shannon, 1948]. After that, a variety of ideas were developed explaining the value of information. Shannon was one of the first who understood the value to describe information in a binary way. Since than the progression in this domain is simply incredibly. The diversity and variety of information management definitions means that this concept can be perceived in many ways. Based on the subject of this thesis, “information” plays a particularly relevant role, as information models, specifying data semantics, are created and matched with the corresponding data models to gain more knowledge from the existing data. A new concept of global information management is the Semantic Web. The emerging approach of the Semantic Web is an information enhancement of the existing web with semantic techniques. The underlying approach is to enrich web-based data with well-defined semantics to make it machine-readable. Semantic data is then prepared through the use of ontologies to be processed by web applications [Allemang and Hendler, 2008].

The heart of ATM and Air Traffic Control (ATC) lies in the control room, in the ATC en route center, Terminal Radar Approach Control (TRACON), and ATC tower facilities. However, control rooms are also used in other mission critical domains such as public safety or emergency control centers. In the past this led to the development of domain specific control rooms resulting in different solutions for each specific environment and raised the cost for software development. Today, ATM is situated in a highly complex environment, requiring well-defined but flexible means for communication and cooperation that can be more easily adapted to future changes. These prerequisites are the foundation why the ATM domain is utilized in this thesis as playground for validation and verification of the generic OBSD concept. The Aeronautical Information Services-Aeronautical Information Management Study Group (AIS-AIMSG) organized by International Civil Aviation Organization (ICAO) defines information as:

> "*Data that has been verified to be accurate and timely, is specific and organized for a purpose, is presented within a context that gives it meaning and relevance, and which leads to increase in understanding and decrease in uncertainty. The value of information lies solely in its ability to affect a behavior, decision, or outcome*" [ICAO, 2010a].

and further determine information management as

> "*The management of resources and processes for the timely collection, integration, exchange and delivery of quality-assured data, information and services*" [ICAO, 2010a].

The objective of information management is to enable digital, interactive and on-demand information interchange between the stakeholders of the global aviation community to support safe, efficient, and environmentally sound performance based ATM operations. This empowers the decision makers and information users with the right information at the right time and the right place. The scope of information management includes all types of shared ATM information including trajectories, surveillance data, meteorological, flight and aeronautical information of all types. The principles of ATM information management are supported by the SWIM Infrastructure that consists of the implementation aspects, covering the definition of the physical network, protocols and systems facilitating ATM information management. The SWIM infrastructure is an essential enabler for ATM providing the interoperable technical infrastructure (ground/ground and air/ground) over which the information gets distributed. The ATM information that is shared is provided from technical systems on the SWIM Infrastructure, and enabled through services. All information exchanges between the participants' technical systems will be performed using services. This infrastructure provides the means to transition from point-to-point message exchanges towards system-wide information provision and usage based on common definitions of data and associated services. The implementation of information management, including the management of its security and its safety is a cornerstone of the future ATM system.

The benefits of semantically enhanced information management are:

- The provision of standardized and reusable information elements for the description of the information exchanges between information providers and consumers in process models.
- The availability of high quality interoperable data unlock new automation potentials, e.g. automated calculation of optimal trajectories.
- Improved decision making for the participants during all planning phases and operations (pre-flight, real-time and post-flight). This is achieved by providing and integrating all ATM information between the participants. Better quality of information will lead to greater confidence in the decision making process.
- Semantic interoperability within ATM for a better understanding between the different stakeholders.
- Enhanced up-to-date information distribution between all ATM participants provide a better view of the operations for ATM service providers, airport operators, etc.
- To provide a base for harmonization, consolidation, review and change management activities for various initiatives and realization efforts related to ATM information.
- A set of models to be used for consolidation activities, development of services and information systems.
- Enabling the identification of gaps and overlaps in information and other models and provides a semantic reference for linking models from different domains.
- It can be used as the reference for the definition of the payload (information content) of a shared ATM service.
- Worldwide promotion of ATM information standards to ensure global harmonization.

2.3 Definition of Ontology in Computer Science

A precise definition of an ontology is not a trivial task. The origin of the word "ontology" is the field of philosophy. Therefore, it is important to go back in time. The term itself is loan from the Greek word $\check{o}v$ (being) and $\lambda o\gamma\iota\alpha$ (science, study, theory), which has a different meaning in the philosophical context, where it refers to the study of being[1]. 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 [Aristotle, 2006]. 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 [Oppy, 2007]. 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`.

[1] https://en.wikipedia.org/w/index.php?title=Talk:Ontology&oldid=726018876, accessed: 2016-11-27

Computer scientists became interested in ontologies in the 1970s within the research field of artificial intelligence [McCarthy, 1987]. They were tempted by the applicability to perform certain kinds of automated reasoning on ontologies as computational models, with mathematical logic [Hayes, 1990]. Such ontologies 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 90s. He defined ontology as an explicit and formal specification of a shared conceptualization [Gruber, 1995]. The word explicit implies that the type of concepts and their constraints are explicitly defined and formally connoted, so that the ontology is readable by a machine. 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 caused a great stir, 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*" [Gruber, 2009].

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*" [Motik et al., 2009b].

Corresponding to Benjamin, Borst and Akkermans [Benjamin et al., 1996], 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 [Wongthongtham et al., 2009]. Per default, all software applications have their own underlying paradigms in form of standardized libraries, components, documentation and files, which also acts like an ontology. However, often this is not enough or the description is poor for some reason, ontologies are precisely made to support under those specific purpose [Chandrasekaran et al., 1999].

Through the initial work of Gruber and other computer scientists several markup ontology languages were developed. Most ontologies are based on Description Logic (DL), which is a conglomeration of knowledge representation formalisms [Baader et al., 2003]. Logical statements relating to roles in form of axioms are the fundamental 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 artificial intelligence, information management and meta data integration. Within the context of the Semantic Web several languages based on DL were developed, such as DARPA Agent Markup Language (DAML) [Hendler and McGuinness, 2000], Ontology Inference Layer (OIL) [Fensel et al., 2001], DAML+OIL [van Harmelen et al., 2001], Simple HTML Ontology Extensions (SHOE) [Heflin et al., 1999], Resource Description Framework (RDF) [Klyne and Carroll, 2004], Web Ontology Language (OWL) [Horrocks et al., 2003] 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 [Patel-Schneider et al., 2004] and with the revision of OWL 1.1 in 2007 more expressiveness was added [Patel-Schneider et al., 2008]. But OWL 1.1 was only another step to the further development which ended up in OWL 2, which was published by W3C in October 2009 and obtains additional expressiveness through innovative ontological axioms to solve known problems that occur with OWL [Motik et al., 2009b]. Despite the new extensions, the main goal is to facilitate ontology development. The background logic of OWL is $\mathcal{SHOIQ}$ [Patel-Schneider et al., 2004], and $\mathcal{SROIQ}$, [Horrocks et al., 2006] which is used in OWL 2.

2.4 State-of-the-Art of Ontology Engineering

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 off target. It is important to understand that both paradigms are 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 model languages to describe and analyze the relations between concepts. However, UML and OWL use classes in a significantly different way [Wongthongtham et al., 2008]. 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 [Hesse, 2008]. In OWL terms, a class is a labeled set of domain related things. Resources (individuals in OWL terms) are simply identifiers, without 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. Reasoning allows identifying individuals, from whom you don't even know that they are within a 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` [Motik et al., 2009b].

The real supremacy of an ontology-based approach lies in the capability to build relationships between instances and classes. The properties of those relationships allow reasoner to make suggestions about them. Consider a brief example (cf. figure 2.1), `: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 information events or a set of risks, but only specific kinds of events are critical. In terms of ontology languages, classes are disjoint to each other. There are no instances that belong to both. In figure 2.1 a `:Storm` is some sort of `:Risk`. However, with the knowledge of this example one can 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.

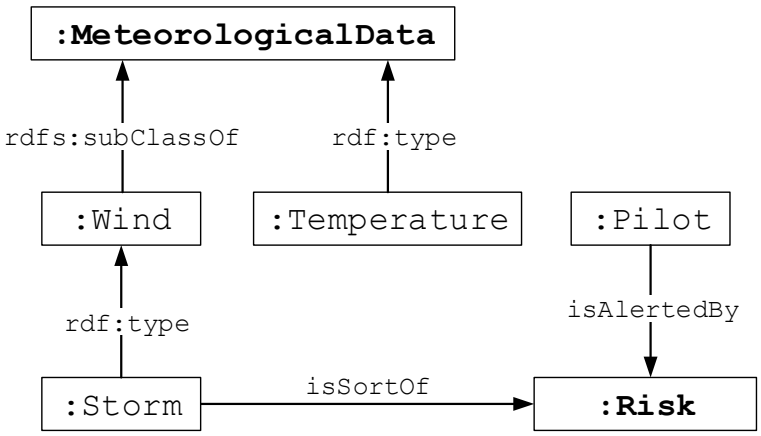

Figure 2.1: A simple ontology.

2.4.1 Ontology Languages

In the early 90s a couple of artificial intelligence based languages were developed. Knowledge representation languages were first-order logic based such as Knowledge Interchange Format (KIF) [Genesereth et al., 1992] or Ontolingua [Gruber, 1993]. In 1989 Frame Logic (F-Logic) [Kifer and Lausen, 1997] was introduced by Michael Kifer at New York State University and Georg Lausen at the University of Mannheim. It combines first-order logic with a frame-based approach. In order of its object-oriented concept it includes revolutionary features such as complex objects, inheritance, axioms and deductive rules. Even now F-Logic is supported by modern ontology tools such as OntoStudio or Jena. SHOE [Heflin et al., 1999] was developed in 1996 at the University of Maryland. It was created as an extension of HTML to combine machine-readable semantic knowledge in HTML documents. SHOE enabled the possibility to gather meaningful information about Web documents and their knowledge congregation by adding additional tags from those of the HTML specification. As this chapter focuses on state-of-the-art languages related with the Semantic Web, earlier ontology language approaches are not described in detail.

In the last few years several ontology languages have been developed and many of them are well known in the context of the Semantic Web, especially those created by the W3C. Such languages are commonly called web-based ontology languages or ontology markup languages. These languages are still in development phase, which means that they are continuously evolving. Most of them are based on the Extensible Markup Language (XML) syntax [Bray et al., 2008]. XML was specified as an open standard by the W3C to improve the information exchange via the Web. Despite the fact that XML was designed for the electronic processing of

documents, it is widely used in a different range of application (e.g. for web services). Therefore, the SHOE syntax was extended to use XML and later on, other ontology languages were built on the XML syntax as well. Other languages also have been used for building ontologies, such as the OIL [Fensel et al., 2001] or DAML+OIL [van Harmelen et al., 2001] which was replaced by OWL. Contrary to traditional ontology languages the Resource Description Framework and RDF Schema are markup ontology languages. OWL is built on the top of RDF(S), which is the union of RDF and RDF Schema. The stack of ontology markup languages and the relationships among them are shown in figure 2.2.

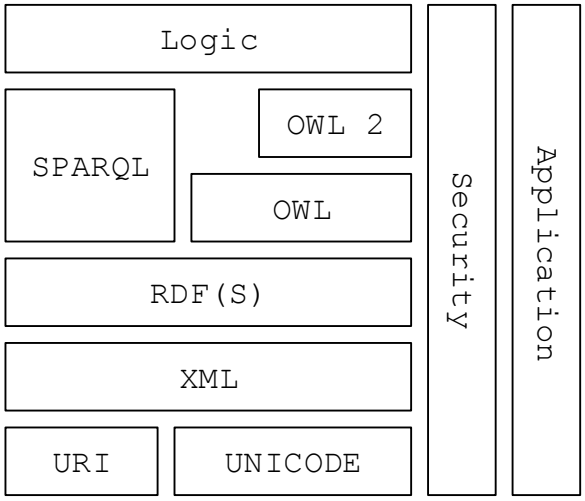

Figure 2.2: Stack of ontology markup languages.

Ontologies represent the information in a semantic space and contain following data:

- **Concepts**: To collect and process data, predefined structures must be established to store the relevant information. First, concepts are defined as basic entities of the system. Examples for concepts are messages or network connections.
- **Hierarchy of concepts**: Concepts are further arranged in a hierarchy.
- **Relation of concepts**: Relations exist between single concepts that are called properties.
- **Hierarchy of relations**: Like concepts, also relations can be arranged in hierarchies inheriting properties from superior relations.
- **Instances**: Instances are the actual representations of concepts. The above entities (concepts, hierarchy of concepts, relations and hierarchy of relations) form a structure (or vocabulary) that is filled with the names and types existent in the system.
- **Relation of instances**: Instances of a specific concept are interconnected by property instances. However, it is not necessary that predicates always link instances of concepts.

According to the "*comparison criteria*" identified by Gomez-Perez and Corcho [Gomez-Perez and Corcho, 2002] the following sections will identify different concepts and components in each language. Furthermore, the availability of tools such as editors, semantic reasoner etc. is an important aspect for an adequate usage.

2.4.1.1 Frame Logic

F-Logic is a former deductive, object-orientated and frame-based ontology language. More precisely it combines a trivial higher-order syntax and declarative first-order semantics. As mentioned before, F-Logic was introduced first in 1989 [Kifer and Lausen, 1997] as a language for deductive databases, but the theoretical description for formalizing ontologies to fit into the semantic technology pool was renewed in 1995 in the F-Logic report [Kifer et al., 1995]. F-Logic embraces the closed world assumption. In context of formal logic the presumption is that any statement that is unknown to be true is false. F-Logic offers the possibility to develop knowledge representations about objects, classes and their inward relationships. Comparable to the pure logical languages a fundamental representation is based on the notion of terms and predicates. F-Logic uses Prolog as base and so all objects have names and are specified via logic terms to represent object identities. There are three main terms: constants, variables and functions. A constant consists out of a starting letter, followed by letters, digits or the underscore symbol out of the American Standard Code for Information Interchange (ASCII) character set (e.g. Ontology_1). It is subdivided into strings, numbers and symbols. Variables are built up in the same way with the distinction that they are followed by a logical quantifier `FORALL` or `EXISTS`. Functions are made up of a symbol and a list which represents the arguments. Additionally expressions are class status e.g. `eclipse:application` which means that eclipse is an application, or types to specify that the name of an application is type of string `application[name?string]`. There are many more which can be specified, such as subclasses, modules, etc. Several modern development environment tools exist, such as NeOn Toolkit or OntoEdit which support F-Logic (cf. section 2.4.2).

2.4.1.2 Resource Description Framework

RDF was standardized through the W3C as an official recommendation [Klyne and Carroll, 2004]. It was developed to represent XML based meta data on the web. But the data does not necessarily contain web information, it could contain other meta data as well. The data model is shown as a set of resources. The Information itself is stored through RDF statements, which are the fundamental structure of every expression in RDF. W3C formally defines a RDF statement as a collection C of triples $\langle \mathtt{s}, \mathtt{p}, \mathtt{o} \rangle$ consisting of a subject "s", a predicate "p" and an object "o" (cf. figure 2.3). The elements of these sets are Uniform Resource Identifiers (URI), literals (L), plain literals (pL) or blank items (B) with $\mathtt{s} \in \mathtt{URIs} \cup \mathtt{B}$, $\mathtt{p} \in \mathtt{URIs}$, $\mathtt{o} \in \mathtt{URIs} \cup \mathtt{B} \cup \mathtt{L}$.

Since OWL is used in the Semantic Web, a generalization of the URI is used called International Resource Identifier (IRI). These IRIs can be very long and therefore are often abbreviated. It is possible to generate a graph of any RDF data set by representing all elements as an edge which connects subject and object via a predicate (cf. figure 2.3). The subject denotes the resource, the predicate denotes properties or aspects of the resource and expresses a relationship

between the subject and the object. The RDF format can be visualized as graph. In this thesis the graph visualization has been used to demonstrate different parts of ontologies using the OntoViz and the OWLViz plug-ins for Protégé. Additional ontology languages such as RDF(S), OIL or OWL are built upon RDF. The circumstance that the RDF data model does not allow to describe the relationships between the resources $\langle$`s,p,o`$\rangle$ itself, was the reason for W3C to introduce RDF Schema [Brickley and Guha, 2004]. The idea was to map RDF statements, which are mere notations, to some reasoning model in order to fix the truth-value. It is a simple, not very expressive, entailment to RDF with frame-based primitives. RDF(S) is the combination of RDF with RDF Schema which is very common as a representation language in various tools (cf. section 2.4.2).

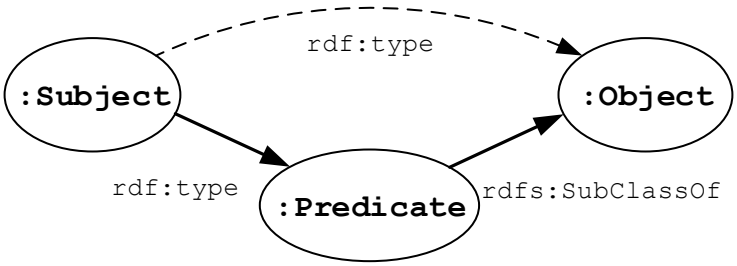

Figure 2.3: A basic RDF triple as RDF(S) graph.

But there are different syntaxes for RDF, Beckett and McBride specified a XML Syntax for RDF in 2004 [Beckett and McBride, 2004]. Others are Terse RDF Triple Language (Turtle) and Notation 3 (N3) or RDFa. RDFa is the abbreviation of Resource Description Framework – Attributes and its purpose is providing a way to embed RDF meta data to Extensible HyperText Markup Language (XHTML) documents. In October 2008 RDFa reached recommendation status [Adida et al., 2008]. Main goal of N3 is to set up a readable alternative to RDF's XML syntax and to extend the expressiveness [Lee and Connolly, 2008]. Turtle, again, is a subset of N3 which wrapped the most useful things to a human-machine readable language whereas the main goal of the W3C was to keep it sufficient inside the RDF model [Beckett and Berners-Lee, 2008]. Turtle simplifies to code a RDF graph by enabling some nice shortcuts. For example you can group same subjects triples together with ";" and ",". Blank nodes can also be grouped together by using "[" and "]". There are also tweaks so that untyped literals can have a language tag and instead of writing `rdf:type` every time "a" is used as abbreviation. The Turtle syntax style for the Triple patterns is also basis for the SPARQL Protocol And RDF Query Language (SPARQL) [Prud'hommeaux and Seaborne, 2008]. Fundamental concepts of RDF Schema are introduced in the following example. `rdfs:subClassOf` describes a relation between classes, that the items of one class are inherited in the sub class. `rdfs:domain` and `rdfs:range` express a property that determines class membership of individuals related by that property. Figure 2.4 and 2.5 explain examples starting from from RDF, RDF(S) to OWL. Assuming the following starting triples:

```
:WeatherData a rdfs:Class .
:ContentType a rdfs:Class .
:has rdfs:domain :WeatherData .
:has rdfs:range :ContentType .
:WeatherShort :has :UTF .
```

As a result one can conclude that `WeatherShort` is a subclass of weather data and has the type UCS Transformation Format (UTF):

```
:WeatherShort a :WeatherData .
:UTF a :ContentType .
```

To define a relationship of weather data in more detail, `WeatherData` can have a subclass `WeatherPicture`and `SatellitePicture` is a subclass described by the `ContentType` class.

```
:WeatherPicture a rdfs:Class ;
   rdfs:subClassOf :WeatherData .
:SatellitePicture a rdfs:Class ;
   rdfs:subClassOf :ContentType .

:WeatherLong a :WeatherPicture ;
   :has :Bitmap .
```

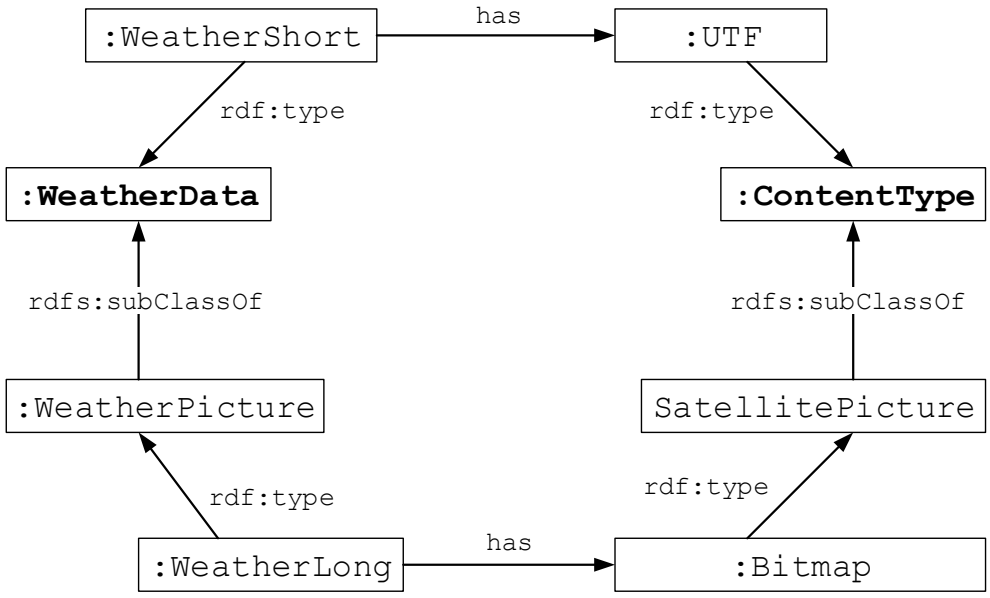

Figure 2.4: An example for a mixed usage of RDF and RDF(S).

With the further adoption one can infer that a satellite picture is a type of `:Bitmap a :SatellitePicture`. But the corresponding inference does not make sense for weather data because UTF is not a type of bitmap. For a correct model an `owl:Restriction` is needed, which will be described later on. These languages are the fundamental keys for the success of the semantic Web. It enables an evolutionary stage of the World Wide Web, where automated software is used to distribute machine readable data to maximize efficiency and knowledge. The trivial data model of RDF with its ability to design heterogeneous concepts is widely used in ontology related applications.

2.4.1.3 Web Ontology Language

In 2001 the W3C sets up the web ontology working group to develop a more expressive markup language for representing and sharing ontologies over the semantic Web and so OWL was born. A first working draft about the syntax came out in 2002, which became afterward a W3C recommendation [Patel-Schneider et al., 2004] in 2004. In comparison to F-Logic, OWL attempts the open world assumption which embraces that a statement with an unknown value can't be proven to be true or not with the current knowledge. It can only be inferred something, if it is explicitly defined as present or absent. If something is not defined as right, it can not be concluded that it is wrong. Reasoning in DL is monotonic: If one can deduce that an individual is an instance of a class and then more information will be added to the model, it can not happen, that this knowledge is wrong. It is clearly the opposite of the closed world assumption. Furthermore, OWL does not include the unique name assumption. Individuals represent objects in a domain. If the unique name assumption does not apply, this means that an individual by two or more different names can be referred. This further means that two individuals may be the same until they are explicitly defined as not equal. OWL adds the possibility to extend some DL axioms (TBox, ABox) to RDF and defines a rewriting how to express such DL axioms in RDF. OWL also supplies the possibility to add some DL statements to an ontology. OWL DL is based on the description logic $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$ [Horrocks et al., 2003]. Ian Horrocks et al. defined $\mathcal{SHOIN}$ as

> "*most expressive means that one could reasonably expect from the logical basis of an ontology language, and to constitute a good compromise between expressive power and computational complexity/practicability of reasoning*" [Horrocks et al., 2006].

The whole semantics of OWL are defined in terms of its RDF reading (OWL Full semantics), and in terms of its description logic reading (OWL Lite and OWL DL) [Patel-Schneider et al., 2004]. Both, RDF Schema and OWL ontologies are RDF graphs themselves. OWL and RDF Schema provide more or less a RDF vocabulary to describe RDF vocabularies. To express unique relationships, something more than the RDF Schema vocabulary is needed. This is offered by an uniquely identifying property: `owl:InverseFunctionalProperty`. OWL is again an RDF vocabulary, extending RDF(S), with fixed semantics that adds more expressivity on top of RDF Schema. OWL can be divided into three sub-languages, whereas the expressiveness increases beginning with OWL Lite to OWL DL and OWL Full. Using OWL Lite allows

the modeling of a classification hierarchy and simple constraints with cardinality 0 to 1. OWL DL provides maximum expressiveness for complete processability and decidability. All OWL language constructs can be used under certain conditions. It also requires a type discrimination, as a class can not be a property of an individual or simultaneously. Furthermore, properties may be either object or data related. OWL Full provides the greatest expressive power, but does not guarantee the process-ability. This means that not all conclusions can be drawn. The three basic elements of the model are axioms, entities, and expressions. Axioms are statements within the ontology, while entities are elements which are used to reference objects in the real world. Expression are combinations of entities in order to achieve a higher complexity [Hitzler et al., 2012].

In this section, classes, properties and individuals are explained. Further, the modeling with these elements will be described. Classes are generally used to group individuals with the same properties. An individual is an instance of a class. Membership in a group is not exclusive, meaning that an individual may also occur in several classes. The following conditions represent a class and an individual of this class:

```
:MeteorologicalData a owl:Class ;
  :rdfs:subClassOf owl:Thing .
```

The classes `owl:Thing` and `owl:Nothing` are predefined. Classes may also be a specialization of other classes. This means that a specialized class can be assigned to a general one.

```
:Wind :rdfs:subClassOf ::MeteorologicalData .
```

A reasoner can infer that each type of `:Wind` is a `:MeteorologicalData` with the axiom shown in figure 2.1. Individuals can be members of multiple classes. Sometimes it is desirable that there is an exclusive member. This is for instance the case when classes are incompatible such as `:Wind` and `:Temperature`. The class `:MeteorologicalData` is a disjoint class. Defining as a disjoint classes ensures that an individual can not simultaneously be assigned to two classes. If there are individuals who are members in two disjoint classes, the knowledge base is inconsistent:

```
:Wind  owl:disjointWith :MeteorologicalData .
```

Object properties can describe relationships between individuals. Each object can have a property domain and range. The characteristics of object properties can be functional, inverse functional, symmetric, and transitive. A functional property specifies that a maximum of one individual can be linked through this property with another individual. With the information about the relationships between two individuals, conclusions can be made on the individuals. Through object properties individuals of a range can be associated to individuals of a domain. The domain allows an assignment of types to subjects, which occur together with a specific predicate in the triple. OWL does not make the assumption that two individuals are different just on the basis of their relationship. Whether it is equal or inequal must be defined as there is no ‘unique name assumption in OWL.

Individuals can be described on the base of their class affiliation and/or the relationship with other individuals. Individuals may, however, be described more specifically by their associated

properties, such as, for example, `:WindSpeed` or `:AirPressure`. The scope and range of values can also be assigned to properties of data types. `:AirPressure` has an integer value operatively associated with the type of `:MeteorologicalData`. To express greater complexity, OWL provides the construct of a complex class. To combine atomic classes logical expressions such as `averages`, `union` and `complement` can be used. The expressions `and, or` are used in Protégé to define `union` or `average`. A main functionality of OWL is the ability to define `owl:Restriction`. Using this mechanism via OWL, values could be restricted for certain properties. For example the restrictions described in figure 2.4 can be used to model the tricky situation. `owl:allValuesFrom` is another OWL restriction from that all values for a specified property must come. Figure 2.5 defines restriction for a set of content types via `:allValuesFrom` that only use bitmaps. Then it has to be declared that any weather picture accomplish this condition through the `rdfs:subClassOf`:

```
:WeatherPicture rdfs:subClassOf
  [ a owl:Restriction ;
    owl:onProperty :has ;
    owl:allValuesFrom :SatellitePicture ] .
```

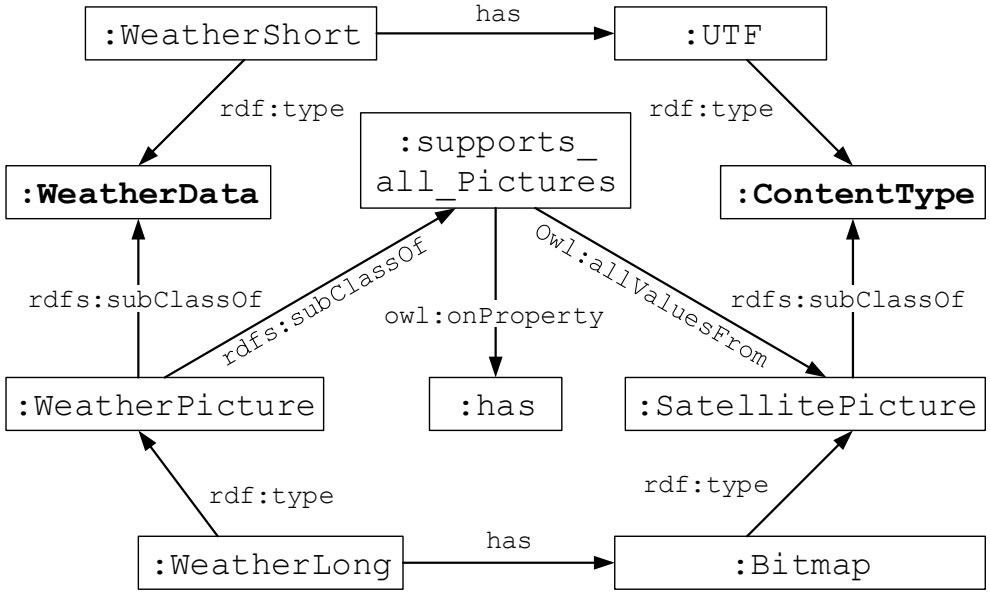

Figure 2.5: OWL enriched example.

In assumption `:WeatherLong a :WeatherPicture.` and `:WeatherLong :has :Bitmap.` it is possible to conclude that:

```
:WeatherLong a [ a owl:Restriction ;
  owl:onProperty :has ;
    owl:allValuesFrom :SatellitePicture] .
```

```
:Bitmap a :SatellitePicture .
```

Now `WeatherShort` is independent of `WeatherPicture`, so there is no relationship that could assume `WeatherShort` has a type of picture, it is just contains text in UTF format. To be correct all RDF(S) classes defined before should now be OWL classes (e.g. `:WeatherData a owl:Class`). OWL 1.1 was established in 2007 to extend ones more the expressiveness [Patel-Schneider et al., 2008]. But OWL 1.1 was only a sub step to the further development which ended up in OWL 2.

2.4.1.4 Web Ontology Language 2

In October 2009 the W3C published a recommendation called OWL 2 [Motik et al., 2009b], [Hitzler et al., 2012]. The new extensions are based on the experience made with OWL and to solve problems which occur with OWL. OWL 2 obtains additional expressiveness through innovative ontological axioms. It also contains non-logical expansions such as a changed syntax, new data types and declaration, a function to comment and easier ontology import. Figure 2.6 gives an overview about the OWL 2 class hierarchy with no claim to completeness. It shows the RDF(S) basis, for example that the OWL Property is an extension to the RDF Property. The description logic behind OWL 2 is called $\mathcal{SROIQ}$ [Horrocks et al., 2006], which is a continuation of $\mathcal{SHOIN}$ [Horrocks et al., 2003]. $\mathcal{SROIQ}$ was extended to solve the lack of some main problems with $\mathcal{SHOIN}$. For example $\mathcal{SHOIN}$ only supports simple number restrictions, e.g. `WeatherData` $\sqcap \geq 3$ `ContentTypes` specifies all classes of weather data with three or more types of content. Instead, $\mathcal{SROIQ}$ supports qualified number restrictions like `WeatherData` $\sqcap \geq 3$ `ContentTypes`.(`Bitmap` $\sqcup$ `UTF`), which implies a class of weather data with three or more types of content that are in the format bitmap or UTF. Nevertheless $\mathcal{SROIQ}$ is still decidable [Grau et al., 2008]. Ontology tools like Protégé and semantic rea-

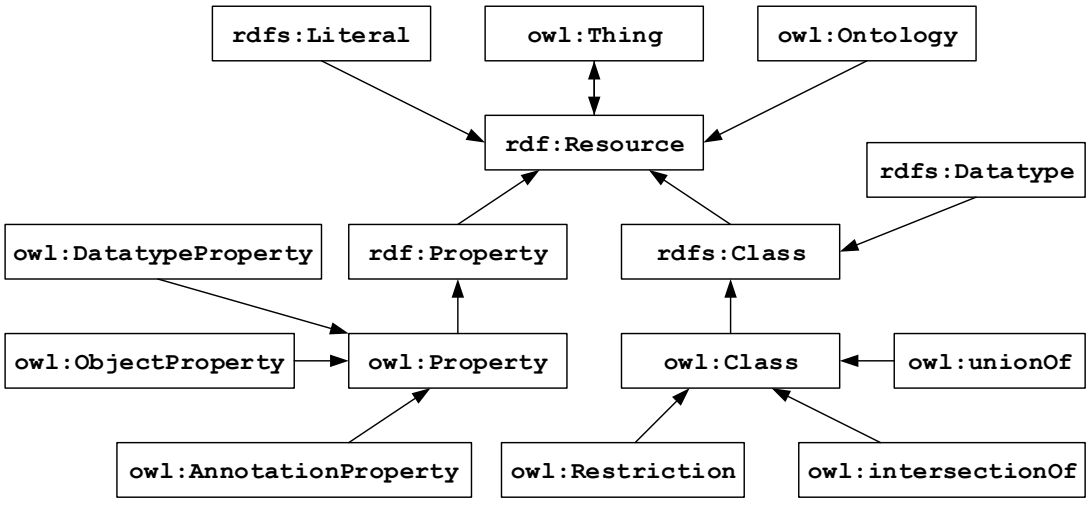

Figure 2.6: OWL 2 class hierarchy.

soners such as FaCT++ [Tsarkov and Horrocks, 2006] or Pellet [Sirin et al., 2007] are in the meantime adapted to support the ontology development process with OWL 2.

2.4.1.5 Semantic Web Rule Language

The aim of the Semantic Web Rule Language (SWRL) is to facilitate the interchangeability of rules and their processing through various rule engines [O'Connor et al., 2005]. This brings benefits for business processes such as work-flow management, diagnostic investigations and compliance monitoring. These business processes are already supported by rule-based systems, whereas the rules are limited in their interchangeability. SWRL is designed as control and exchange language for the Semantic Web and has the status of a W3C submission. It combines concepts of OWL with Datalog. SWRL rules can be used to derive new knowledge from an OWL ontology. The rules are stored in an OWL knowledge base and can access OWL individuals, classes, properties and literals and are thus compatible with OWL. The specification of SWRL does not pretend who the conclusions should be performed by the reasoner. This depends on the implementation of the reasoners or the rule engine used. The so-called SWRLTab, a SWRL editor, which is integrated in Protégé can be used to edited SWRL rules with Protégé. In addition, SWRL tab accesses via the Protégé SWRL Application Programming Interface (API) the ontology and can extend the derived knowledge. The API can also be used by plug-in developers. Furthermore, SWRL tab provides a way to integrate a rule engine for executing the rules with the built-in SWRL Factory. In this case, the Jess rules engine is used[2]. Thus, Protégé can expand with SWRL Tab and Jess to enable rule-based reasoning of the ontology. However, SWRL Tab is not the only way to exchange SWRL rules, as rules can be saved as *.owl* file to import and process them. When using SWRL, it is important to note that, like OWL, the open world assumption is in place. Therefore no weak negation (negation as failure) is supported. Like many other rule languages, SWRL consists of an antecedent and a consequence. The two parts of the rule are called body (antecedent) and head (consequence). There is currently no way to create complex constructs except the combination of atoms. A rule is true if the queried antecedent, or precondition returns true. Therefore it is necessary that all the atoms of the precondition are true. If the condition is true, the consequence is true. The components of a rule, called atoms, can either be restricted classes, individuals, object properties, data properties, built-ins or data-types. A class atom consists of an OWL class with an argument. The argument can either be a variable or an OWL individual. A variable name must begin with a question mark. The following class assignment specifies that all individuals of the class `:Wind` are also within the class `:MeteorologicalData`.

```
Wind(?n) → MeteorologicalData(?n)
```

If a class atom is used in the rule body, it determines whether the variable or the individual is a member of the class. The classes atom only returns true if the variable is a individual of the class. If the class atom occurs in the head of the rule, an allocation of the variable or the individual to the class takes place. It should be noted that each variable which occurs in the

[2] http://herzberg.ca.sandia.gov, accessed: 2016-11-27

body must also occur in the rule head, as it will be attached in the rule body. An object property atom consists of the OWL `ObjectProperty` and two arguments which represent either a variable or an individual. Data Property atoms consist of the OWL `DataProperty` and two arguments, the first of which is an individual or a variable, and the second is a (variable) data value. The following example shows portion of a rule, where the value of the data properties `hasValidTime` is bound to the variable `?vFrom`. The data type is a `DateTime` in this case. For SQWRL queries, string values may be relevant, these can be represented with double quotes.

```
MeteorologicalData(?n) ∧ hasValidTime(?n, ?vt) ∧ ValidTime(?vt)
∧ validFrom(?vt, ?vFrom)
```

Since SWRL does not support the unique name assumption, it offers use of the atom `differentFrom` to specify that the selected individuals are different. In contrast, one can establish with `sameAs` that there are two equal individuals. SWRL also supports so-called built-ins that further enhance the expressiveness of the language. The number of arguments is not limited to two but supports a plurality, or an optional number of arguments depending on the selected built-in. Firstly, there are the core built-ins which are also part of the W3C submission. There are also built-ins which can be developed custom wise. Core built-ins provide, for example, functions for manipulating strings, to perform comparisons and mathematical tasks. Since the W3C submission more built-ins were published in the scientific field. [O'Connor and Das, 2011] developed a library that provides the possibility of temporal comparison. It offers the ability to calculate the duration between two points in time in a desired granularity.

```
MeteorologicalData(?n) ∧ hasValidTime(?n, ?vt) ∧ ValidTime(?vt)
∧ validFrom(?vt, ?vFrom)
```

2.4.1.6 Semantic Query – Enhanced Web Rule Language

Semantic Query Enhanced Web Rule Language (SQWRL) allows the possibility to retrieve information from existing ontologies which are defined in OWL [O'Connor and Das, 2008]. SQWRL is built on the top of SWRL and so gain the advantages and characteristics of the SWRL rule language. These advantages are distinguished by the consistency, readability and semantics. Furthermore, SQWRL provides various functionalities to deal with the already described open world assumption. However this query language also has some limitations, which will be discussed here briefly. According to wiki of Protégé it is not possible to use the results of SQWRL for further ontology development. In other words such a mechanism could affect OWL's open world assumption and lead to a non-monotonic knowledge base. A SQWRL query basically consists of two distinctive parts. The left part of a query consists of a SWRL body, which restricts the query area in advance. Through rules, a certain part of the ontology can be selected, on which queries can be executed. In addition variables are bound, which can be repeatedly used in a query. Furthermore, all available SWRL or SQWRL built-ins can be used. The right part of a query reflects the actual query operation. It specifies which variables are selected, or how to sort or group them. Basically, there are two different groups of operators. On the one hand there are core operators and on the other hand there are collection operators.

Core operators allow rather simple queries like selection, counting, aggregation, grouping, and sorting. If these operators are not sufficient, you can use the collection operators. These support for negation, disjunction and advanced concepts of grouping and aggregation. The selection operator is used to query and output objects of a certain ontology. It should be noted that this operator can only access variables that are already bound. This operator returns results as a table in which the queried variables form the columns of a table. In the following example, all types of wind are selected, which are defined in the ontology.

```
Wind(?a) → sqwrl:select(?a)
```

To specify an additional variable as a parameter, first an additional variable must be bound by the rules. For this, the query is restricted to all kinds of wind which have a specified wind type.

```
Wind(?a) ∧ WindType(?at) ∧ hasWindType(?a, ?at) →
sqwrl:select(?a, ?at)
```

With class descriptions it is possible to interrogate individuals who meet certain properties without knowing these individuals in advance. This could be useful to retrieve all types of wind, which have a certain wind speed.

```
(WindSpeed ≥ 20) → hasWindType(?at)
```

It is possible to use already existing SWRL statements in a SQWRL. This may contribute significantly to structure and readability of the rules and queries. The makeSet operator provides the basis to convert an open world assumption into a closed world one. This operator creates a set, and can insert various individuals in this set. All individuals in a set represent a closed world.

```
sqwrl:makeset(<set>, <element>)
```

The first argument is a variable that refers to the newly created set, the second argument represents the individuals added to this set. It should be noted that with respect to a set only built-ins are available that have the name-space `sqwrl`. The following example shows how such a set can be applied. All kinds of wind are added to a set, and then printed out:

```
Aircraft(?a) ° sqwrl:makeSet(?s1, ?a) ° sqwrl:size(?size, ?s1) →
sqwrl:select(?size)
```

To check the status of a set of individuals (e.g. all kinds of wind from a set) the operator `element` can be used. Negation as failure assumes that everything that can not be proven is invalid. This approach encapsulates the open world assumption. With this construct, it is now possible to retrieve all wind types that have not occurred. Because this operator is, as the name suggests, the difference between two sets, and supplies all the individuals which occur in the first set, but not in the second set:

```
Sqwrl:difference(<set>, <set>, <set>)
```

The first argument reflects the set in which the result is stored. The second argument is the set which contains this value, whereas the third argument includes the set which does not reflect this value. It is important that the second argument contains the values to remain after the negation. The third argument will contain the values that are to be deducted from the second argument. This sequence must be followed, as it may lead to unintended results otherwise. Since SWRL only supports the conjunction, it is obvious, that the insertion of the `set` operator , now supports the disjunction . This operator returns as a result of a union of two sets:

```
Sqwrl:union(<set>, <set>, <set>)
```

Last but not least the operator `groupBy` can now be explicitly applied in a set, and provides a wider range of applications and also contributes to the clarity and readability of queries.

2.4.1.7 SPARQL Protocol and RDF Query Language

SPARQL Protocol and RDF Query Language (SPARQL) is a standardized RDF query language based on graph pattern matching. In the beginning of 2008 SPARQL became an official W3C Recommendation [Prud'hommeaux and Seaborne, 2008]. Version 1.1 was released in 2013 by the W3C [Harris et al., 2013]. It could accomplish the dream of the Semantic Web to utilize it as a big data, where unambiguous queries could be made around the globe. With the use of an URI every identifier in SPARQL is unique. A merge of several graphs can be queried at once and a data set can also be implicit, depending on the implementation. One of the interesting features of SPARQL is that a query may retrieve data from different sources with the `GRAPH` statement. The formal semantics of SPARQL is over all based on simple entailment which means that there are no special interpretations of the RDF(S) vocabulary except some extensions. But there are also peculiarities in SPARQL's semantics such as multi set semantics, joins over unbound variables, etc [Prud'hommeaux and Seaborne, 2008]. The semantic itself is specified as an operational way based on [Pérez et al., 2006], which first defined a relational algebra for SPARQL. The basic framework of SPARQL has three query parts with different query statements which take advantage of the solutions from pattern matching to form result sets as shown in table 2.1. The prologue contains the `PREFIX`, while the head embraces either the `CONSTRUCT`, `ASK` or `SELECT` statement. For example, the `SELECT` statement returns all, or a subset of the variables bound in a query pattern match. `ASK` queries are so called yes or no queries without explicit output. With the `CONSTRUCT` statement an RDF graph is returned on the basis by substituting variables in a set of triple templates. The graph patterns at the `WHERE` part allow all Turtle shortcuts [Angles and Gutierrez, 2008]. Different patterns can be expressed by following functions: `FILTER`, `UNION`, `OPTIONAL`, `GRAPH`, `NOT EXISTS`. `OPTIONAL` allows partial variable bindings in the solutions. The negated `bound()` function in the `FILTER` allows to suppress unbound values. Extensions of SPARQL such as update statements (e.g. `DELETE`, `INSERT`, etc.) and aggregate functions (e.g. `MIN`, `MAX`, `SUM`, `AVG`, etc.) have been introduced by the W3C with the introduction of SPARQL 1.1 [Harris et al., 2013]. SPARQL is a language

Prologue	PREFIX *foo: <namespace-URI>*

Head	CONSTRUCT { *template* } SELECT *variable list* ASK

Body	FROM *<dataset-URI>* FROM NAMED *<dataset-URI>* WHERE { query *pattern* } ORDER BY *expression* LIMIT *integer* > 0 OFFSET *integer* > 0

Table 2.1: SPARQL - a standardized query language.

to query information from RDF. Since OWL builds on RDF, it is obvious that OWL can be converted into RDF. This had several disadvantages as due to the necessary conversion from OWL to RDF the knowledge of what is behind the individual constructs, which were defined in OWL, is lost. Also, SPARQL can only query on this data base, but can no longer connect to this lost knowledge. Another problem was the lack of standardized conversion. Different tools produce different RDF data while the converted OWL ontology remains the same. This finally led to different query results. To circumvent these problems, it is of advantage that the query language used can deal directly with OWL ontologies. Besides SQWRL (cf. subsection 2.4.1.6) there are languages such as OWL-QLand ASK 2.0 DL interface which satisfy these properties. According to [O'Connor and Das, 2008] there neither exists a useful implementation of OWL-QL, nor does the DL Implementation Group's (DIG) ASK Protocol represent a language that is as expressive to define queries based on OWL ontologies. With the introduction of SPARQL 1.1 this changed, as it does include OWL entailment regimes (cf. [Glimm et al., 2013]). [Horridge and Musen, 2016] presented a plug-in for Protégé which uses the entailment regime to query OWL ontologies.

2.4.2 Ontology Editors

Not only the comparison between different languages is important, also the right choice of the editor, plug-ins and extensions around a language is a significant task to be accomplished. In general one can divide between two kinds of ontology related tools. Ones for the development process and others for the productive use like reasoner or alignment tools. To the purpose that the open source community pushed the ontology development in the last years a growing number of free and open source ontology-engineering environments have been developed to support the needs of Semantic Web. They provide support for each different ontology life cycle step and most of them offer a component-based design to extent more functionality to the basic environment. Import and export opportunities often allow language independent use of ontology-models. For instance FlexViz[3] is a Flex-based application for ontology modeling with great potential. Of course some of those tools and frameworks deserve a closer look. Table 2.2 indicates a comparison between those editors.

		Protege 3	Protege 5	Jena Framework	OntoStudio	NeOn Toolkit	TopBraid Composer	Altova SemanticWorks
Tool	Developer	University of Manchester, Stanford University	University of Manchester, Stanford University	Apache	Ontoprise	EU-funded project 14 partners	TopBraid	Altova
	Version	3.5	5.1.0	3.1.1	3.2.0 - Build 353	2.5.2	5.2	2012
	Policy	Mozilla Public License (MPL)	Mozilla Public License (MPL)	Apache License 2.0	Commercial	Eclipse Public License (EPL)	Commercial	Commercial
	Platform	Java	Java		Eclipse	Eclipse	Eclipse	.NET
	Plug-ins	++	++	~	+	+	+	no
	API	Protege-OWL	OWL	RDF, OWL	NeOn API (OWL)	OWL, F-Logic	Jena API (RDF, OWL)	?
Languge	Import	RDF, RDF(S), OWL	OWL 2, RDF/XML, Turtle, OBO	RDF(S), N3, OWL, Turtle	F-Logic, RDF(S), OWL, OXML, UML 2.0, Database Schema, Excel	OWL 2, F-Logic	RDF(S), N3, OWL, XML, UML, Database Schema	RDF, RDF(S), OWL, RDF/XML, N-Triples
	Export	RDF, RDF(S), OWL	OWL 2	RDF(S), N3, OWL	F-Logic, RDF(S), OWL, OXML	OWL 2, F-Logic	RDF(S), N3, OWL	RDF, RDF(S), OWL, RDF/XML, N-Triples
	SPARQL	yes	yes	yes	yes	Plugin	yes	no
Reasoner		through HTTP DIG interface	FaCT++, Pellet, other	internal reasoners, Pellet	embedded reasoning	reasoner plugin	Pellet	built-in semantic reasoner
Database Storage		Files, JDBC	yes	JDBC	MS SQL Server, JDBC	OBDA plugin (JDBC)	Jena, AllegroGraph, Oracle, Sesame	Files
Multi User Support		yes	yes	no	yes	yes	yes	no
User Interface		Form, Text	Form, Text	Form, Text	Form	Form, Text	Form, Text , (UML-like) Graph	Form, Text , Graph

Table 2.2: Comparison of ontology editors.

2.4.2.1 Protégé

A well-known and most widely used open source ontology development tool is Protégé[4]. Protégé is based on Java and uses Swing for its user interface. The project was started at Stanford University in cooperation with the University of Manchester. Comparable to the IDE Eclipse, the open source community provides Protégé plug-ins, which range from visualization to mapping tools, to expand its capabilities. The plug-in library is one of Protégé's powerful features. It shows an ontology as a mind map. For example the OwlViz plug-in is a mapping visualization designed for Protégé. Version 3.5 of Protégé allows ontology modeling via the Protégé-Frames or the Protégé-OWL editors. The Frames editor has not been migrated into Protégé 4.3 yet. The

[3] http://sourceforge.net/projects/flexviz/, accessed: 2016-11-14

[4] http://protege.stanford.edu, accessed: 2016-11-14

main distinction of both versions is on the one hand the support of 3.5 of OWL and RDF(S) including OWL Lite, DL and FULL and on the other hand that 4.3 uses OWL 2. Users and developers, who still have the need to access RDF or require a particular tool which is not already ported, have to fall back to 3.5, others should give 4.3, or the fresh released 5.1.0 a try. There is also a web-based editor called WebProtégé to simplify the collaborative ontology development. Protégé can be extended by a set of plug-ins in order to allow the export of the data stored in an ontology. The following sections list some of these plug-ins and show their export capabilities.

SWRLTab is an extension[5] for Protégé to create, edit and read SWRL rules. It supports, with a few exceptions, all OWL expressions. With the tool it is possible to access directly to the OWL classes, properties and individuals. The tight integration of the OWL knowledge database allows to perform syntactic and semantic checks already during the input of rules with the SWRL editor. It ensures that all references to OWL entities are valid, and that all variables occurring in the rule head also occur in the rule body. If an error is present, the input box is grayed out and an error message appears. It is impossible to store faulty rules. The development process of rule creation also allows the ability to activate certain rules or disable them. In the ontology itself the SWRL rules are stored as OWL individuals. The "Hidden Display Frames" option in the Protégé project options allows to view the classes and the associated SWRL rules. The class `swrl:Imp` contains the rules whereas the subclasses of `swrl:Atom` will allow to find the individual atoms of the rules. To access the SWRL Editor with Java, the SWRL Factory API is provided. This makes it possible to create rules within a Java program. Also the SWRLTab itself uses the SWRL Factory API to create rules. To execute the rules, and produce conclusions the SWRLTab alone is insufficient. However, a rule engine can be integrated which then performs these tasks. There are many rule engines that work very well with Java. As discussed further the Jess rule engine has been selected. Version 3.5 of Protégé for example, permits the possibility to use Drools. With the installation of Jess, SWRLJessTab and SQWRLQueryTab are automatically activated. These two tools are plug-ins for the SWRLTab and are already included in the Protégé installation. It is important that SQWRL queries can use the derived knowledge of SWRL rules. Consequently, using SQWRLQueryTab is sufficient for the needs of an Ontology-Based Software Development Life-Cycle.

The SWRLTab just supports version 3 of Protégé, version 4 is not supported but there is a Protégé 5-based version. And there is an editor within Protégé 4 which is able to create and edit SWRL rules but the editor is not as mature as SWRLTab and does not offer the possibility to integrate a rule engine. On the other hand the editor allows the possibility that the used reasoner can derive results through the SWRL rules. The main argument against the Protégé 4 editor is that SQWRL queries are not supported. With the built-in Bridge of SWRLTab own built-ins for SWRL can be created using Java. Following procedure is needed to create a built-in:

[5] http://protegewiki.stanford.edu/wiki/SWRLTab, accessed: 2016-11-27

1. **OWL Generation** At the beginning it is necessary to create an ontology for the desired built-ins. Therefore a predicate name or the Java method name has to be defined as an individual in the class `swrl:BuiltIn`. Usually built-ins which belong together are defined in a set.

2. **Built-In Creation in Java** The actual implementation of the built-in is achieved using Java. It should be noted that the package name should consist of a qualifier and the Java package name `edu.stanford.smi.protege.owl.swrl.bridge.builtins`. The dynamic loading and execution process of the built-in Bridge demands that the class in which the built-ins are defined, is derived from `AbstractSWRLBuiltInLibrary`.

3. **Deployment of a Java ARchive (JAR)** Before a built-in can be used it must be provided as a JAR file in Protégé plug-in folder. Protégé then automatically loads the added class and runs the selected as needed.

4. I**mport OWL ontology** To use the new built-ins, the OWL ontology with the built-in OWL individuals must be imported in the desired project. After that it can be accessed via the qualifier in the SWRLTab.

Self-developed built-ins can handle multiple parameters. Anyway, they must return either true or false. The return value indicates whether the predicate is satisfied. In addition, there is the possibility of binding values to a parameter to use them in a SWRL rule. Thus, calculated values from a rule can be returned by parameter binding. The parameters are passed in a list of `BuiltInArgument` objects. The most common types are `IndividualArgument`, `DataValueArgument`, `ClassArgument` and `PropertyArgument`. For individuals, classes and properties, only the appropriate name is handed over. Please note that only individual and data value arguments are permitted in OWL DL. The transfer of a class or property argument is only allowed in OWL Full ontologies.

SQWRLQueryTab can define, edit and execute queries in Protégé. Already defined SWRL rules can be enabled or disabled and have a direct effect on the queries. Creating queries is supported by a graphical editor that makes it easier to access SWRL and SQWRL built-ins, and offers the ability to perform syntax checks. The results of the queries are clearly displayed in tabular form. The SQWRLQueryTab, in the version 3 of Protégé uses the Jess rule engine to run queries. Furthermore, it is possible to execute queries using the SQWRLQuery API in Java programs.

Snap-SPARQL was introduced by [Horridge and Musen, 2016] and has been implemented as a side project of Protégé. It is open source and available via github[6] and supports Protégé 5. With SPARQL 1.1 comes a specification of entailment regimes that includes OWL ontology support. Combined with a reasoner (cf. subsection 2.4.3) the Java framework offers a GUI for processing SPARQL 1.1 queries and the asserted reasoner is constantly improved and supports

[6]https://github.com/protegeproject/snap-sparql-query, accessed: 2016-11-27

OWL API 4.1.3. But there are some features that are not supported by Snap-SPARQL. For example `property path expressions` or SPARQL `Update` are not supported. While the `property path expressions` seems not a big deal when it comes to OWL the `Update` function to add and remove axioms to an OWL ontology would make sense. Also complex OWL class expressions are not supported.

2.4.2.2 Jena Semantic Web Framework

Jena is a Java-based framework for building ontologies and was initialized by the HP Labs Semantic Web Program[7] and lies now in the Hands of the Apache Software Foundation and offers good support. The open source environment provides a RDF and OWL API to read and write RDF, RDF(S) and OWL in RDF/XML, N3 and N-Triples. A model can also be queried through SPARQL. Developers have the ability to exploit Jena through plug-ins arbitrary. Through the inference API various reasoners can be accessed in Jena.

2.4.2.3 OntoStudio

OntoStudio[8] is a commercial engineering environment based on the NeOn toolkit. Therefore OntoStudio has a modular design with a rich set of features that can be easily extended. It was originally developed for F-Logic but it now supports import and export of OWL, RDF and XML. OntoStudio includes a patented evaluator to implement rules during the modeling process and provides the ability to query via SPARQL or F-Logic queries. Execution and testing can be done through its embedded reasoning capabilities. OntoStudio also supplies a tool for testing and debugging entire ontologies. The internal ObjectLogic is based on F -logic and allows a much more compact representation of rules than Datalog and the model of object structures. Benefits are a direct import of UML and Oracle, export of rules with the Rule Interchange Format (RIF) and support of existing modeling-tool. Again, there is the possibility to define appropriate built-ins, with the difference that one can model directly and without detours complex objects and defined rules and *closed world* must not be simulated with *open world* languages. More difficult is the formation of conceptual hierarchies (subsumption reasoning). Therefore a transformation in OWL would be necessary.

2.4.2.4 NeOn project

The NeOn project[9] was co-funded by the European Commission's Sixth Framework Program with 14 European partners involved. It started in the beginning of 2006 and had a duration of 4 years. Like Protégé it is an open source ontology engineering environment which supports the whole life-cycle process. NeOn toolkit inherits an open and modular architecture from its underlying platform, the Eclipse IDE. The NeOn toolkit provides an extensive set of plug-ins for visualization, matching, reasoning, reuse, etc. The core installation set provides basic ontology functionality such as editing, browsing, ontology and project management. Similar to Protégé

[7] http://jena.apache.org/ accessed: 2016-11-27

[8] http://www.semafora-systems.com/en/products/ontostudio/ accessed: 2016-11-27

[9] http://www.neon-project.org/, accessed: 2016-11-27

4.3 the NeOn toolkit is build on-top of the OWL API, de facto standard for implementing OWL based application. As a result the newest ontology language OWL 2 is supported by the NeOn toolkit.

2.4.2.5 TopBraid Composer

TopBraid Composer[10] is a commercial visual modeling environment built on top of Jena discussed above and is a component of the TopBraid Suite. TopBraid Composer is built as an Eclipse plug-in based on the Eclipse 3.6 platform and was designed for RDF Schema, OWL and SWRL. It accompanies the developer as a multi-user modeling tool during the creation and maintenance process of ontologies. SPARQL queries and rules are supported as well as scalable database back-ends like Jena, Oracle, AllegroGraph or Sesame. Through its component-based design TopBraid Composer provides extension points to integrate third-party Java components. There is also a free version available with limited features. Finally, TopBraid Composer includes the Pellet reasoner, which was developed at the University of Maryland. Additional features are consistency checking, data source mapping, geography and location mapping.

2.4.2.6 Altova SemanticWorks

Another commercial tool is Altova SemanticWorks 2012[11] for ontology editing and creation. SemanticWorks supports visual document creation with RDF, RDF(S) and all three OWL dialects (Lite, Full, & DL). Syntax and semantic consistency checking is instantaneous. It automatically generates RDF/XML or N-Triples for export.

2.4.3 Semantic Reasoner

One aim of the semantic Web is to offer machine readable meta-data. Ontologies expressed by W3C's OWL improved this in the engineering field of the semantic Web [Motik et al., 2009b]. 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 or OWL 2 are formal defined, offers the possibility to use a reasoner [Tsarkov and Horrocks, 2006]. 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 table 2.3. `:MeteorologicalData` has the subclass `:Wind`, this relationship is called a necessary implication. 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 4.3 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

[10] http://www.topquadrant.com/resources/products/docs/TBC-Getting-Started-Guide52.pdf, accessed: 2016-11-27
[11] http://www.altova.com/documents/SemanticWorks.pdf, accessed: 2016-11-14

that they could not guarantee to provide only valid output. An excellent insight provides a paper from Giorgos Stoilos et.al. [Stoilos et al., 2010] published at the Oxford University Computing Laboratory. This subsection compares and describes the capabilities of some state-of-the-art reasoners, which support OWL and OWL 2 (cf. table 2.3).

		Affiliation	Version	License	API	Expressiveness	Semantics	Rule Support	Conformance
Reasoner	Pellet	Clark & Parsia LLC	2.3.1	AGPL	OWLAPI, DIG, JENA	SROIQ(D)	direct	SWRL	full
	FaCT++	University of Manchester	1.6.2	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	1.3.8	LGPL	OWLAPI 3.0	SROIQ(D)	direct	SWRL	full
	OWLIM	Ontotext AD	3.3	LPGL, commercial	SAIL	OWL 2 RL	RDF-based	TRREE	full

Table 2.3: A comparison of semantic reasoning systems.

2.4.3.1 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 and Parsia LLC. The dual licensing model of Pellet allows using it for open source applications under the terms of the GNU's Not Unix! (GNU) Affero General Public License (AGPL) version 3. For commercial usage it is recommended to get in contact with Clark and Parsia. In the beginning of August 2011 the release of Pellet 2.3.1 was announced and also the Pellet reasoner plug-in for Protégé 4.3 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 expressiveness 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 Assertion Box (ABox) and Terminological Box (TBox) 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. 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 [Sirin et al., 2007].

2.4.3.2 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 [Tsarkov and Horrocks, 2006] and is now supported by the SEALIFE research project. It is implemented using C++ and licensed under the GNU Lesser General Public License (LGPL). 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 data type 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*" [Patel-Schneider et al., 2004].

FaCT++ can be used as back-end reasoner with the OWL API or as standalone via the DIG interface. As Protégé 3 uses DIG and Protégé 4 the OWL API, both are supported. The latest available version is 1.6.2, which was released in February 2013. Nevertheless OWL 2 is only partially supported. No support for keys or partial data types, are some of the missing things.

2.4.3.3 RacerPro

RACER 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 derivative 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 [Guohua et al., 2007]. 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 OWL API to use it with a RacerPro adapter for Protégé 4. RacerPro embraces an own semantic query language for knowledge reasoning called new Racer Query Language (nRQL) [Haarslev et al., 2004]. 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.

2.4.3.4 HermiT OWL Reasoner

HermiT is 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 hyper-tableau reasoning calculus for efficient reasoning, using the DL $\mathcal{SROIQ}$ with OWL 2 data type support. This approach allows free 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 [Motik et al., 2007]. To reduce the size of the models which are constructed, they are blocked anywhere. HermiT is pre-installed in Protégé 4.3 and the actual release is version 1.3.8, which makes use of OWL API 3.4.3. The semantic itself is processed directly as well as all conformance tests for OWL 2.

2.4.3.5 OWLIM

There are two different editions of OWLIM, SwiftOWLIM and BigOWLIM. They differ on separate Triple Reasoning and Rule Entailment Engines (TRREE) and in terms of semantics, SwiftOWLIM does not support OWL 2. The different TRREE implementations have impact on performance and scalability [Kiryakov et al., 2005]. 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 European Union IST integrated project Semantic Knowledge Technologies and several other European research programs like the European Union Sixth Framework Program. OWLIM is packaged with Sesame and so benefits from the variety of supported query languages and ontology syntax (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 [Motik et al., 2009a]. OWL 2 RL focuses on scalability instead of expressive power.

2.4.4 Visualization Tools

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 [Catenazzi et al., 2009], [Lanzenberger et al., 2009a]. 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. The following visualization tools are chosen to fit in that concept and therefore are compatible with Protégé 4.3 (cf. table 2.4).

		Affiliation	Version	License	Visualization	Application
Visualization Tools	OWLViz	University Of Manchester, CO-ODE Project	4.1.2	LGPL	tree, node	Protégé 4.3
	OntoGraf	Stanford University	1.0.1	-	tree, node	Protégé 4.2
	SOVA	Gdansk University of Technology	0.8.1	LGPL	tree, node	Protégé 4.2
	Cloud Views	University Of Manchester, CO-ODE Project	1.1.1	LGPL	list, tag cloud	Protégé 4.2
	OWLDiff	Technical University in Prague	0.1.5	LPGL	comparison list	Protégé 4.1, NeOn Toolkit, standalone
	Matrix	University Of Manchester, CO-ODE Project	1.1.2	LPGL	list, matrix	Protégé 4.2

Table 2.4: A comparison of visualization tools.

2.4.4.1 OWLViz

OWLViz 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 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 4.3 and is licensed under the LGPL. OWLViz uses the GraphViz algorithms delivered by AT&T, and take advantage of the Batik SVG Toolkit from the Apache Software Foundation.

2.4.4.2 OntoGraf

OntoGraf 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. 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.

2.4.4.3 OWLDiff

OWLdiff 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 a compare-exchange algorithm [Konev et al., 2008]. 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 4.3 and the NeON toolkit.

2.4.4.4 Simple Ontology Visualization API

SOVA 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. 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 useful.

2.4.4.5 Cloud Views

In contrast to the previous tools, Cloud View is not a standard visualization tool. It enables Protégé 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 [Seifert et al., 2008]. 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 4.3.

2.4.4.6 Matrix

Just like OWLdiff, Matrix 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 4.3 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.

2.5 Conclusion

In order to meet the challenges of OBSD, seven different semantic languages and six different modeling tools were presented in this chapter. Furthermore, a detailed review about state-of-the-art semantic reasoner and corresponding visualization tools was given. An Ontology-Based Software Development approach allows to improve software development. If benefits are wanted the right choice of specific tools and languages must be selected. This does not only require the simple use of such tools, but also the development of new language and tool extensions to design and develop the needed environment. To be able to solve all these problems in the best suitable way, OWL in combination with SQWRL and SPARQL was selected for this thesis. As main tools Protégé was chosen, since the use of an open world language in a closed world but with subsumption reasoning (OWL/Protégé) fits the OBSD methodology more, than for example OntoStudio which directly uses a closed world language with a necessary transformation in OWL to form conceptual hierarchies (ObjectLogic/OntoStudio). As mentioned above, within this thesis, the chosen ontology environments were extended or tailored to meet the requirements of OBSD. Existing extension mechanisms (e.g. the SWRL Built-in Bridge) however, are in some cases insufficient for the needs of this thesis. In these cases, extensions of the OBSD ontologies and rule language along OWL and SQWRL were needed, to use a more general language for implementation. SPARQL was taken on board, as version 1.1 adds more functionality even the Snap-SPARQL plug-in is in an early development stage. It turns out that none of the existing ontology-based environments offer the all-in-one solution for OBSD. For the purpose of this thesis it was necessary to implement extensions to better integrate the ontology-based tool chain and future ontology-based tools shall be studied to identify potential new solution, that increase the productiveness of the OBSD methodology.

CHAPTER 3

Operational Context

"The important thing is not to stop questioning."

[Einstein, 1955]

This chapter presents an overview about the operational context in which the case study evaluation took place. It is essential for this thesis to introduce the concept, design, and technologies of the generic System Wide Information Management concept based on Service Oriented Architecture principles which roughly scratches the surface of the relevant ATM domain. The SWIM infrastructure, dedicated to the definition of the European SWIM technical architecture, allows seamless interoperability and information sharing for future European ATM Systems. Furthermore, standardized ATM data and service models and their semantic enhancements, which were used as a baseline for the semantic domain knowledge in this thesis, are explained.

3.1 Introduction

> "*The European Commission shares the industry's frustration with the failure of European states to progress the single European sky. Every year that single European sky languishes in limbo is a EUR 5 billion knock to European competitiveness and costs the environment 8.1 million tonnes of wasted carbon emissions,*" [1]

said Tony Tyler, International Air Transport Association (IATA) Director General and CEO. EUROCONTROL's SESAR program and Federal Aviation Administration's (FAA) Next Generation Air Transportation System (NextGen) are transforming the global ATM as we know it today and will break up with existing roles predicated by 50 year old technology [Ulfbratt et al., 2008], [SJU, 2015]. Similar to SESAR, NextGen right now runs the risk to specify ATM systems based on architectures already out-of-date. While current functionality is based on historical grown technical restrictions, a performance-based and most efficient approach requires new

[1] http://www.iata.org/pressroom/pr/Pages/2013-06-11-01.aspx, accessed: 2016-11-14

paradigms and eventually has to lead to a balanced approach to prevent over-optimizing one area at the expense of others [JPDO, 2011]. The main objective of ATMs information management is to provide a framework which defines seamless information exchange between all providers and users to share the information within a SOA. To cover the needs of existing research projects like NextGen, SESAR or Collaborative Actions for Renovation of Air Traffic Systems (CARATS), OBSD achieves the goal of reusability across domains. The OBSD approach, based on semantically enriched service models, addresses these issues and allow easier development and modular applications for multiple domains.

This chapter describes the current results of SESAR SWIM in order of requirements and capability definitions that a compatible information system needs to fulfill. Those capabilities are basically derived from major topics of interest within the SWIM technical infrastructure such as interoperable communication, security, and governance. Following requirements and capabilities, conceptual architecture was created utilizing artifact types from several commonly used architectural views (structural, behavioral, deployment) depicting logical structure and interaction among the major sub-systems and their components, as well as information exchange flows. Based on previous logical architecture considerations, a list of appropriate technology standards is given. The system will be deployed either as an extension to new services of certain service providers or in case of existing legacy services as a SOA enabler. Therefore, the possible fields of use are beyond the SESAR Air Traffic Management SWIM scenario.

The next section describes how Aeronautical Information Services (AIS) are connected to a newly established technical platform that needs to be compatible with the SWIM infrastructure in respect to its implementation and technology requirements. This creates the link between the technical infrastructure and the business service identified and developed by SESAR. As an important first step, the benefits of applying SOA for ATM opens the door for further capabilities of using different traditional data domains in a SOA environment beyond the scope of traditional AIS. The rest of the chapter introduces main concepts for the SWIM technical infrastructure and provides further details on the architecture that served as a basis for OBSD.

3.2 Information Services moving towards Information Management

The Air Navigation Service Provider (ANSP) industry has only recently realized that the exchange of information has become an important influence on their business and will have an even greater influence on ATM performance in the future. Various events (e.g. Icelandic volcano Eyjafjallajokul) have involuntarily highlighted this to all stakeholders. Today many AIS remain focused on providing narrowly defined legacy aeronautical products as specified in International Civil Aviation Organization (ICAO) Annex 15 [ICAO, 2010b]. ICAO has a number of Annexes, several of them are relevant for AIS. The ICAO standards are the lowest common denominator and all other standardization committee build extension on the top (e.g. country-specific extensions). Yet, there is an awareness among providers of aeronautical information that the industry must transform from AIS to Aeronautical Information Management (AIM) to meet the growing demands of the ATM business. The transformation includes a number of changes such as the transition from manual, paper product environment to a digital, standards-based environ-

ment [SJU, 2015]. As mentioned in the Civil Air Navigation Services Organization (CANSO) AIM business model,

> "*the mission of AIM is to provide interactive, on-demand aeronautical information interchange between the global aviation community to support safe, efficient and environmentally sound flight operations that maximizes system capacity.*" [CANSO, 2013].

The challenge for AIM is to consider information, processes and services needed by ATM today and in the future [SJU, 2015], [Ulfbratt et al., 2008]. One of the crucial elements of ATM capacity, especially in the implementation and operational phase, is the availability of quality assured aeronautical information. ANSPs face the need to replace the traditional product-centric provision of aeronautical information by a data-centric and systems-oriented solution that continuously provides reliable data to all ATM users and applications everywhere in the system, at the right time in consistent high quality. If ANSPs have to raise airspace capacity, they will have to adopt new and improved data management practices and incorporate this change into planning. One key enabler for ATM will be a proper and flexible AIM, fully integrated within the future ATM system based on SWIM principles.

The traditional product-centric provision of the current AIS will be replaced by a data-centric environment, in which timely and reliable data is made available permanently and dynamically for use in applications that perform the required tasks, be it flight planning, flight management, navigation, separation assurance or any other strategic or tactical ATM activity. Although the automation of AIS is in progress, not all relevant information is currently available in a form suitable for automatic processing. These technical constraints are further impaired by human behavior. Some data owners are extremely reluctant to share information and to adopt new information management practices because of the safety-critical nature of the ATM business, and a fear that real-time requirements cannot be met. Therefore, SESAR provides an initial AIRM representing the data shared between the stakeholders, which is updated periodically during the project. The AIRM is intended to represent all the possible ATM information constructs that can be present in the ATM world and part of the SESAR concept. The AIRM conceptual model is neutral and may support any possible external view (i.e. ATM domain models) within the range it is valid. This implies that the AIRM must stand clear from constraints with respect to the ongoing ATM business which may change over time [Gringinger et al., 2010a]. The main objective of ISRM is the service view by identifying the logical shared information services and specifying the information models relating to ATM services. The purpose of the ISRM is to develop aeronautical and meteorological business information services based upon SOA principles as well as current and foreseen ATM business needs. The ICAO annex 15 [ICAO, 2010b] step by step transition of the traditional AIS service provision into an aeronautical information business service based upon SOA principles ensures the states responsibilities for AIS services in both the foreseen SOA for the European ATM network as to cater for global requirements outside an SOA framework [Gringinger et al., 2011].

3.3 System Wide Information Management

According to the SESAR SWIM *Concept of Operations*, SWIM consists

> "[...] *of standards, infrastructure and governance enabling the management of ATM information and its exchange between qualified parties via interoperable services*" [Cruellas and Roelants, 2013].

The SWIM concept is defined by the information elements (AIRM), the services (ISRM) and technical infrastructure. The SWIM infrastructure consists of SWIM access points and (federated) SWIM capabilities like messaging, recording, registry, security, and supervision. The SWIM Governance on the other hand defines trust, policies, validation criteria, security, management and the service life-cycle [Gringinger et al., 2012b]. The SWIM concept is encompassing all the processes and changes to assure that the full life-cycle of all ATM information is managed systematically and in a system wide manner throughout the ATM enterprise ensuring that the right information is available at the right time and at the right location.

3.3.1 Principles

The SWIM Concept of Operations document [Cruellas and Roelants, 2013] defines a set of guiding principles for all stakeholders in respect of the design, provision, evolution of SWIM, and with regard to the information/solution providers and consumers, the processes, standards, and technologies. The ten SWIM principles are aligned with the operational business to supply the right information, to the right people, at the right place, and at the right time. SWIM combines the ATM stakeholders, shared information, common processes, and the underpinning use of technology as indicated in the SWIM Concept of Operations document [Cruellas and Roelants, 2013]:

1. **Accessibility**: All involved stakeholders are able to consume and publish ATM information via the SIWM infrastructure using common service and data standards.
2. **Equity**: No stakeholder dominates, or constrains, what may be offered by others.
3. **Flexibility**: The solution offers various capabilities to ensure that changes of the involved stakeholders are covered.
4. **Performance**: Certain levels of performance and resilience have to be provided to cover the requirements from various ATM stakeholders.
5. **Quality, Integrity and Security**: Each ATM stakeholder is owner of the information created and therefore responsible for the data quality and integrity. Security metrics have to be applied to ensure the safe exchange of the information via the applied services over the technical infrastructure. An appropriate policy system ensures the correct use provides and consumed information.
6. **Implementation and Evolution**: Release and roll-out is following a clear road-map to ensure the operational, technical and governance functionality.

7. **Cost**: The main benefit is to reduce costs through open standards and common interfaces, to lower the integration and implementation effort of all ATM stakeholders.

8. **Service orientation**: Service orientation principles like SOA are the fundamental part of the SWIM concept to support the exchange of information via services between the different ATM stakeholders.

9. **Open standards**: SWIM relies on applicable open and internationally recognized standards for the information, the content, the processes, and the provision of services.

10. **Global applicability**: SWIM relies on international and local agreements, to achieve a seamless ATM information environment with adequate governance.

3.3.2 Concept of Operations

ICAOs global ATM operational concept foresees the roll-out of SWIM concept [ICAO, 2005]. The SWIM concept concerns and involves all participants who have a stake in, or a right to the shared and exchanged ATM information. The scope of the SWIM concept covers the collection and management of ATM information from one or more sources and the interoperable distribution of that information to one or more consumers. This implies SWIM activities which entail retrieving, acquiring, maintaining and exchanging the shared ATM information on a system wide basis. SWIM activities are performed with the support of a set of orthogonal SWIM functions which concern the collaborative management of organizing and controlling ATM information sharing and distributing. Furthermore, running SWIM relies on the availability of a SWIM technical infrastructure and its services for exchanging ATM information between the SWIM participants. The SWIM technical infrastructure is the technical enabler of SWIM. As presented in figure 3.1, SWIM starts with the standardization of the information on the participants' side, which is referred to as SWIM data management.

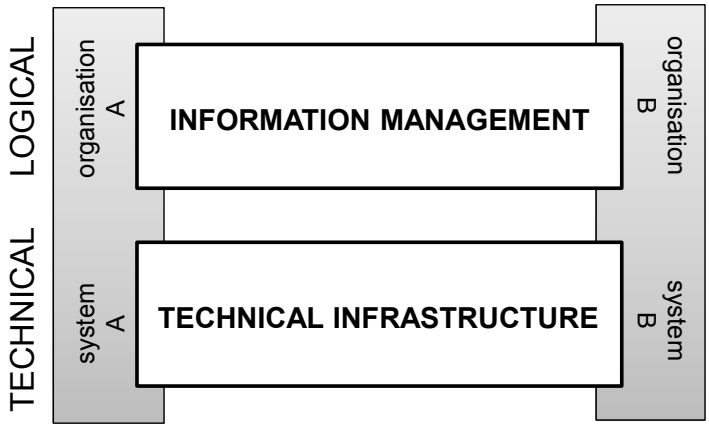

Figure 3.1: SWIM = The ATM implementation of information management.

It covers the interoperability, through the alignment of processes and procedures. The SWIM information itself may come from data-sets maintained and operated by different organizations. The SWIM technical infrastructure covers the technical aspects, from the system interface standardization to the interaction, transport and the enabling technical services, such as registration, authentication, supervision, and others.

- The **SWIM information management** deals with SWIM functions addressing the broad range of processes that need to be put in place to deliver the SWIM concept. It offers general principles, rules and recommendations regarding ATM information and contains information modeling conventions and common meaning within the AIRM. Furthermore, it offers the information services specifications called ISRM and a registry services.
- The **SWIM technical infrastructure** deals with all ATM data exchanged through the whole SWIM network, although this information itself may come from data-sets maintained and operated by different organizations. It supports the technical implementation and services to exchange ATM Information.
- The **SWIM concept** addresses information management, the technical infrastructure and the information management functions. In order to exchange information, the participants need to enable information sharing through services. These services are exposed on the SWIM technical infrastructure.

3.3.3 Technical Architecture

This subsection describes the conceptual architecture, the concept, design and implementation of SESAR SWIM [Trausmuth and Klopf, 2010], [Di Crescenzo et al., 2010], [Gringinger et al., 2012b]. The communication infrastructure follows the widely accepted concepts of SOA based on the HTTP/SOAP communication protocol and technology specification stack commonly known as Web Services. This platform provides the best fit to majority of aviation use cases in consideration of acceptance, technology maturity, and the adoption costs [van Meenen et al., 2013]. The communication protocol of choice for AIM communication via SWIM is the Hyper Text Transfer Protocol (HTTP) over the Transmission Control Protocol / Internet Protocol (TCP/IP). On top of HTTP, the Simple Object Access Protocol (SOAP) provides the means to exchange object data in XML format via the concept of services. Each service provides functionality and data according to a definition called Web Service Description Language (WSDL). Protocols based on SOAP, such as WS-Security and WS-Policy, provide additional functionality and features including encryption, authentication, and quality of service. These protocols can be selected as needed, allowing the provided feature set to be tailored to specify the needs of each service. The preferred information exchange model is either request/response or publish/subscribe via the Web Services Notification (WSN) standard. The Organization for the Advancement of Structured Information Standards (OASIS) specification for WSN defines a web service-based mechanism for the distribution of information based on the idea of each interested client subscribing to whatever information is desired, and a notification broker distributing incoming updates to every subscribed client. SOAP and Web Services constitute a proven, industry-standard and vendor-neutral interface for integration of systems via services.

The technical features listed here have been prototyped in so-called SWIM access points in the context of SESAR [Gringinger et al., 2012b].

The application aware infrastructure based on SOA principles consists a distributed communication platform semantically enriched with knowledge about interactions among collaboration applications including the types of executed processes and exchanged data. The SWIM infrastructure provides capabilities to manage inter-system collaboration. It enforces constraints and monitors and manages the information exchange according to the prior defined interaction/business rules. Such rules are aggregated to policies. The SOA network elements apply restrictions through Policy Enforcement Points (PEP). The policy descriptions are composed of atomic rules called Policy Assertions. The SWIM infrastructure is being built from a network of ATM application-aware communication elements, the underlying general purpose network infrastructure and SWIM access points at the boundary. A SWIM access point can either be deployed individually as stand-alone computer systems or embedded as part of ATM systems.

The network of SWIM access points (cf. figure 3.2) enables a high level of interoperability among connected ATM systems regardless of utilized technology, design and implementations. From an architectural point of view, the interaction dependencies are described through the concepts of wire and application interfaces: The wire interfaces are essential for the system interoperability and are mandatory in the inter-system communication. The application interfaces, which are supposed to remain an internal characteristic of an ATM system, describe the interaction between an ATM business logic implementation and a SWIM access point instance. Wire and application interface might be of the same or of different types. For example, the wire interface might be the SOAP/HTTP and the application interface might be built upon the same technology, when the back-end business logic already provides the service endpoints of that type. The major advantages of such organized communication architecture are in decoupling of inter organizational B2B collaboration from particular business services implementations while the systems remain autonomous and still ready to integrate with reasonable integration expenses.

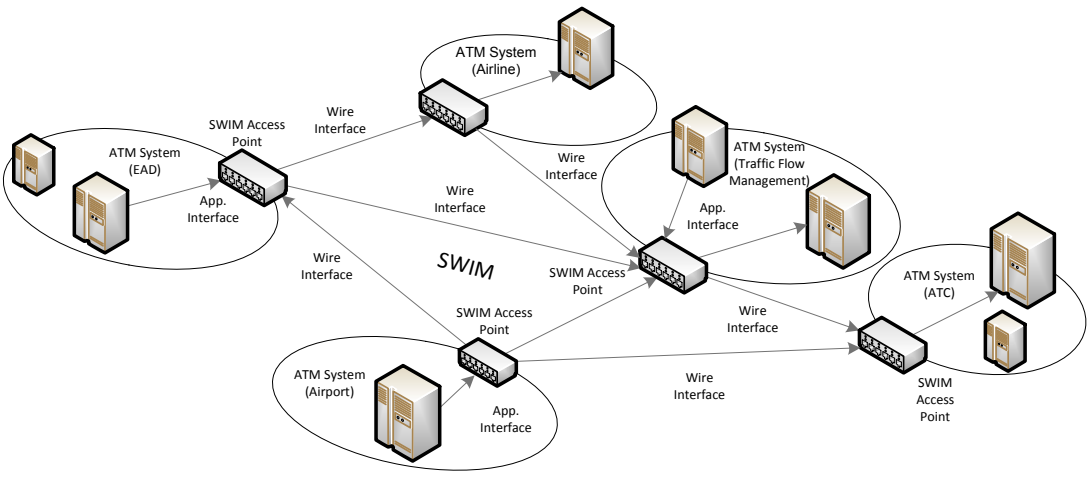

Figure 3.2: High level overview of the SWIM network.

The communication responsibility is outsourced to a dedicated communication sub-system, a SWIM access point instantiation.

3.3.4 Access Point

The SWIM infrastructure is a logical network of PEPs, which proxy the access to business services and resources provided by ATM systems. PEPs control the whole information exchange among the interconnected systems. Being equipped with additional functionality, such as communication protocol mediators and message transformation engines, these access points act as intermediaries between resources provided by service providers and their business partners. Figure 3.3 describes the structure of a SWIM access point, one key building block of the SWIM technical infrastructure. Interactions between other access points (left side) and back-end service implementations (right side) are added in order to provide a fully coherent system overview. The green colored components implement the messaging capability according to the current SWIM platform development within the SESAR program. The supervision, messaging, interface management, and security components are colored differetn gray scales. All those components are reusable blocks in itself, e.g. a weather data model compliance component can be reused in various domains.

- The **Service Endpoint** component manages the access to business logic service implementations (reverse proxy), or assists during consumption of an external service (proxy). Service Endpoints use different communication stacks, one defining the message ex-

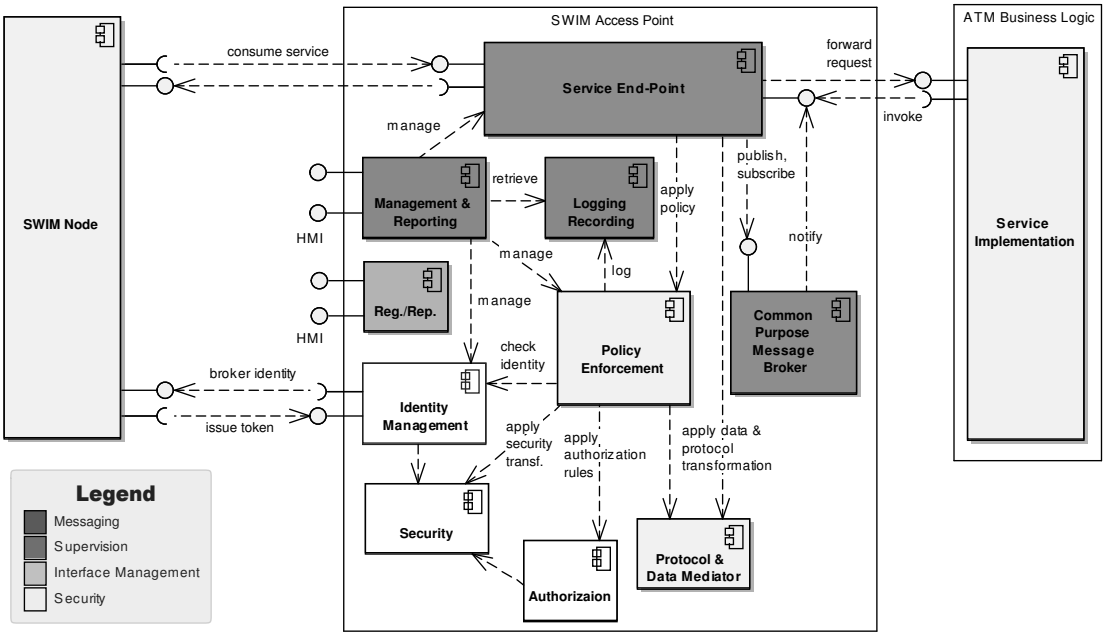

Figure 3.3: Structure of a SWIM access point.

change pattern used, and another one for service meta-data describing the policy rules. The SWIM access point maintains many instances of the service endpoint component for both communication directions (proxy, reverse proxy). During the communication process, these instances collaborate with the policy enforcement component and are supervised by the management component.

- The **Policy Enforcement** subsystem applies the policy definitions configured for service endpoints. This includes evaluation of authorization rules, message structure, service types, applied security measurements, level of quality of service, etc. Furthermore, a policy might define message transformation execution or raise monitoring events. Based on the OASIS eXtensible Access Control Markup Language (XACML) there are several sub components which actively participate in policy enforcement.

- The **Protocol and Data Mediator** is a component similar to an Enterprise Service Bus (ESB) but focused on single interaction mappings. It is used by PEP and service endpoint components in order to modify the structure of the communication payload or to switch from one communication protocol to another. This approach is used whenever the back-end service implementations are not fully compatible with business domain data and service types.

- The **Identity Management** handles consumer authentication, which includes attribute and pseudonym services, as well as identity token creation and evaluation. Furthermore, it collaborates with external identity managers building security federation.

- The **Security** component implements basic security concepts such as message integrity and confidentiality. The recording component is responsible for data collecting and reporting.

- The **Management and Reporting** component uses data previously collected by the auditing component and provides it to the SWIM environment for further integration. It can be accessed either via a Human-Machine Interface (HMI) or via specialized interfaces exposed to other SWIM participants.

- The common purpose **Message Broker** is a subsystems which implements a communication subsystem that is based on Message Exchange Pattern that decouples systems and allows payload agnostic inter-system information exchange.

3.4 Standardized Air Traffic Management Data Models

The main goal of AIM is to provide the right information to operators at the time and place when they need it. To ensure this, the concept of net-centric ATM operations is used, supported by potent information filtering. It is important to select the essential data out of the huge space of ATM information that is available before distributing it. Current aeronautical information requirements are declared by ICAO standards and recommended practices, especially in Annex 15 [ICAO, 2010b], which defines the aeronautical information services. These requirements in combination with the digital age a model for the European AIS Database (EAD) called AIXM was created in Europe. In 2003 EUROCONTROL and FAA established a cooperation, which led to the current AIXM version 5.1. The actual version overcomes limitations of the previous version 4.5, as it models more than only static data. In 2008, an ICAO Study Group has been formed whose main objective is to globally harmonize aeronautical information data exchange. AIXM 5.1 also covers temporary data updates such a Notices to Airmen (NOTAM). As described in the European ATM Master Plan [SJU, 2015], digital NOTAM is a primary objective within SESAR. The European Commission delivers additional requirements through the aeronautical data quality interoperability regulation, which was developed for the single European sky phase 1 project.

3.4.1 Air Traffic Management Information Reference Model

As described in the European Air Traffic Management Master Plan [SJU, 2015], the information management work package defines the AIRM and the information service model by establishing a framework, which defines seamless information interchange between all providers and users of shared ATM information. A specific example implementing inter-domain data models is the European AIRM. SESAR defines the AIRM as a model which contains all of the ATM information to be shared in a semantic way [Gringinger et al., 2012a]. The AIRM represents civil, military, and civil-military information constructs relevant to the ATM domain. In a service-oriented approach, data exchange and its fundamental data models and the processing of data are decoupled, in order to prevent over-reliance by the end-user applications on the data models. To fulfill these SOA requirements, known elements such as airspace, aerodrome, flight procedure as well as common definitions such as geometry and time are considered in the frame of the AIRM. The baseline of information management was defined by the SESAR program covering more or less all the various information, logical and physical model standards into one common reference model called the AIRM. Through harmonized conceptual and logical data models AIRM provides a definition of all ATM information. It is used as a common reference for the different models (cf. figure 3.4) that are developed as part of SESAR.

A first version of the AIRM, so called initial load, was established in summer 2010 based on existing data models such as the ICAO Weather Information Exchange Model (IWXXM), and the Weather Information Exchange Model (WXXM), which are both maintained by the World Meteorological Organization (WMO), the Flight Information Exchange Model (FIXM), AIXM, and industry standards, like the Geography Markup Language (GML), [OGC, 2007] or ISO 19103, the geographic information conceptual schema language, ISO 19107, the geographic information spatial schema, and ISO 19108, the geographic information temporal schema. In fact

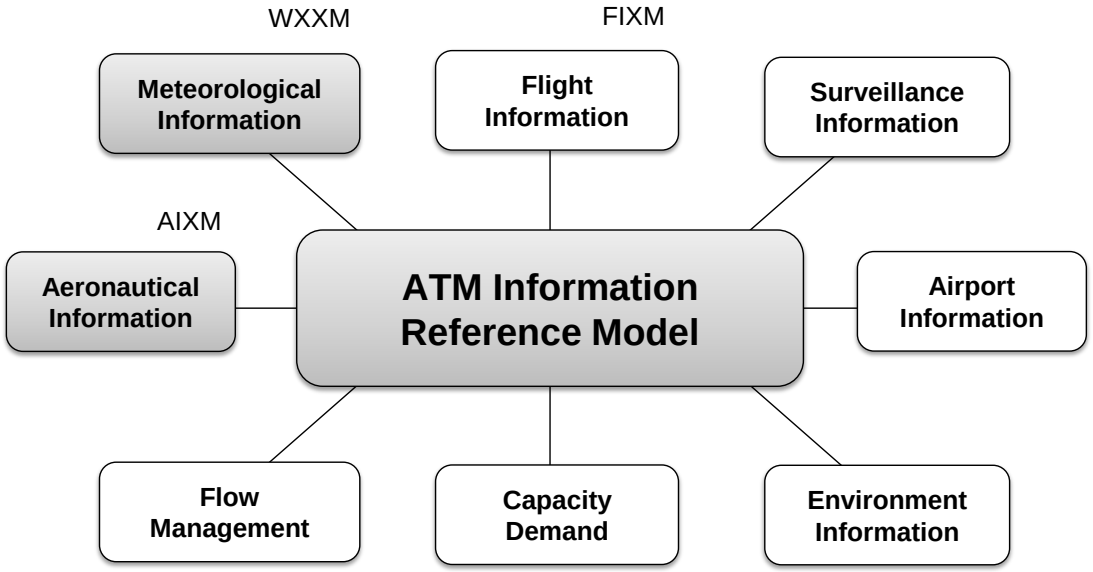

Figure 3.4: AIRM semantic and common syntax.

GML is not part of the AIRM. AIRM uses only the conceptual schemata from the ISO 19100 series e.g. ISO 19108. GML appears at the physical modeling level. Further increments of the AIRM are planned for release every six months. The AIRM is defined using UML and used in the context of service modeling based on the Service oriented architecture Modeling Language (SoaML). It also includes the consolidated logical data model, which provides a reference model of the SESAR data concepts for service architects and system implementers. The ISRM models the information to be exchanged between ATM systems, and defines interfaces and interactions at the logical level. It derives service payloads from the AIRM, and is fully SOA based upon the North Atlantic Treaty Organization (NATO) service-oriented view defined in the NATO Architecture Framework (NAF) version 3. The idea has always been to verify and potentially evolve or replace it with content derived from actual operational requirements, so that it can fulfill its role as a documentation of the semantics used in the SESAR operational documents to become a valid implementation of NATO Operational View[2] (NOV) 7 in the SESAR Enterprise Architecture (EA) framework.To facilitate reusability and software development itself, ontologies are used during the prototype development. Semantic interoperability is significant for the ATM system development. In this regard the AIRM will be open to a change in its scope and will be capable of absorbing new requirements as they are identified. The European AIRM is used as a specific example for inter domain implementation. Referring to EUROCONTROL,

> "*AIRM is required to define the semantics of all the ATM information to be shared*" [Eurocontrol, 2009],

an ontology-based approach is a breakthrough. Improving efficiency, increasing reusability of

[2]http://nafdocs.org/meta-model/, accessed: 2016-11-14

code through semantic structures and a component based framework are among the key goals of the Pan-European SESAR Joint Undertaking (SJU) as well as the FAA's NextGen program. Eurocontrol plans to evolve the AIRM up to ICAO level, which will satisfy the dream of OBSD to cover various domains on a global level. To implement a successful case study around the AIRM, OBSD is used to gain reusable software components. In a first step the Aerodrome Map Information Service (AMIS) was implemented to validate and demonstrate the usefulness the ontology of the underlying domain model and to have a first case study in place (cf. chapter 5).

3.4.2 Information Service Reference Model

The logical shared information service model is the main objective of the ISRM, and specifying the information models relating to ATM services (cf. subsection 3.4.1). ISRMs purpose is to develop aeronautical and meteorological business Information Service based on SOA principles as well as current and foreseen ATM business needs. A step by step transition of traditional AIS provision [ICAO, 2010b] into an aeronautical information business service based on SOA principles will ensure the individual responsibilities for AIS services in both the foreseen SOA for the European ATM network, and to cater for global requirements outside an SOA framework [Eurocontrol, 2009]. The service modeling process translates operational requirements into a construction plan for the SWIM infrastructure implementation. Service modeling defines the information exchange protocols (service contracts) required to support the operational processes and provide the technical capabilities derived from operational requirements. The ISRM [Pola and Solberg, 2013] focuses on the logical service architecture providing a logical viewpoint on services, comprising the service contracts, interfaces, operations, behavior (interactions), input and output parameters and orchestration of services. The outcome of service modeling is a comprehensive description of services on a logical level and constitutes the major part of the contribution of service modeling domain projects to the ISRM.

3.4.3 European Air Traffic Management Enterprise Architecture

In SESAR, the European Air Traffic Management Enterprise Architecture (EAEA) [Vaudrey, 2013], based on the NATO Architecture Framework (NAF) Version 3.1 [NATO, 2010] defines the artifacts of the ATM target architecture. NAF defines several views, each addressing a specific perspective where the ISRM is created under the service-oriented views. The EAEA metamodel is the key enabler to architectural coherency within SESAR. The coherence of the EAEA architecture is achieved by defining which architectural meta-elements can be used in which EAEA views and sub views. Within ISRM, only a subset of EAEA views, sub views and architectural elements are created and/or used, namely:

- NATO Operational View (NOV) identify information exchange needs and corresponding services and is used, analyzed, and refined as ISRM input. This was essential input for the rules and policies used in OBSD.

- NATO Service-Oriented Views (NSOV) are descriptions of services needed to directly support the operational domain and are implemented within SESAR as the ISRM

- NATO System Views (NSV) are split into NSV-11b, the physical data model represents the payload of the service messages to whom the ISRM is referring to. NSV-11a is considered during the service message modeling and represents the logical data model part of the European AIRM. The association between NSV-11a and NSV-11b is extremely significant to gain the benefits from the ontology-based approach.

While ISRM services are using the UML/SoaML [OMG, 2012] modeling languages for expressing the service-oriented views, the domain language elements are used in coherence with the EAEA meta-model [Vaudrey, 2013], to ensure a seamless integration into the overall EAEA. For the NAF views on services that require formal notation, the service models are described in the UML/SoaML languages. SWIM applications have to be complaint with the ISRM model.

3.4.3.1 Relation to Service-Oriented Architecture

The logical information service models which are integrated into the ISRM are just one building block of the entire service-oriented architecture and services landscape. Consequently, they will be tightly connected and interrelated with other building blocks and have to be set in the context of an overarching SOA strategy, a common service life-cycle model and according collaboration and governance procedures.

3.4.3.2 Relation to Operational Focus Area

The concept of the Operational Focus Area (OFA) is defined by the SESAR Program Management Plan [SJU, 2012] as:

> *"A limited set of dependent operational and technical improvements related to an Operational Sub-Package, comprising specific interrelated operational improvements designed to meet specific performance expectations of the ATM Performance Partnership."*

The key challenge of ISRM modeling is to derive a single service portfolio consistent with the OFA requirements. From a SESAR working perspective, the OFAs are suitable as a means for identification and refinement of operational requirements, non-functional requirements and derivation of related operational processes and capabilities as a basis for service model creation. The SESAR program comprises 16 work packages which split up into more than 300 sub-work packages. All of these sub work packages are working in projects with defined responsibilities, tasks and deliverables. These tasks are ideally performed in tight collaboration and communication with the SESAR operational work packages, system work packages and the SWIM work packages (WP 8, 13 and 14) [Gringinger et al., 2012b]. ISRM projects are collaborating informally with the other work packages in order to align to the OFA and acquire the needed input for ISRM service modeling. ISRM related work packages implement a pragmatic service modeling approach based on a proposal which comprises:

- Extraction and refinement of relevant requirements from OFA related documentation such as Detailed Operational Description, Operational Services and Environment Definition

- Definition of detail operational processes (where needed and feasible) to identify interaction and information exchange needs and potential for support through services

- Definition of a hierarchy of capabilities as a means of structuring requirements and identify candidate services to support interaction needs

3.4.3.3 Service Implementation Process

Services can be applied whenever two or more ATM stakeholders are interacting with each other. Given the number of stakeholders in ATM and the amount of interaction needed to perform ATM, one can see that applying a service-oriented way of collaborating is something that will affect ATM at its core. Due to this complexity and many involved stakeholders it is vital to have a structured and well-governed way of identifying, designing, and developing services.

Currently this process is, in Europe, performed as part of the SESAR program. To support governance of the process and provide structure, a technique called EA is applied. EA is the process of translating business vision and strategy into effective enterprise change by creating, communicating and improving the key requirements, principles and models that describe the enterprise's future state and enable its evolution[3]. In case of SESAR this provides a best-practice framework for describing how the federated enterprise of the European ATM shall evolve during the coming years.The service-related activities are very important inputs to service identification since they provide the business and operational requirements and contexts. More specifically, the business modeling provides a high level description of how the EAEA evolves over time by describing what capabilities the different ATM stakeholders will have during the different evolutionary steps. The operational modeling provides a description of who the actors are and how they will collaborate with each other. This is done by modeling the operational activities they perform and describing the information exchanges that takes place between actors.

It is of importance that data and service related work packages [Blomqvist et al., 2012] are involved in the operational modeling in order to ensure that the operational modeling deliverables are of such quality that they can be used to identify and design services. Also, it is important for data and service related work packages to support the operational modeling by providing expertise in the area of information modeling and ensuring alignment with the AIRM. The service modeling process is divided into four major steps and will be described in detail afterward:

- Requirements Analysis

- Service Identification

- Service Design

- Development and Implementation

[3] http://www.gartner.com/it-glossary/enterprise-architecture-ea/, accessed: 2016-09-27

In the **Requirements Analysis** step the inputs are analyzed and essential requirements regarding interaction and information exchange are extracted and refined. From a modeling point of view, detailed process diagrams representing specific parts of operational processes are created where necessary and reasonable to analyze interaction of activities and identify corresponding information exchange needs. Operational requirements and derived information exchange requirements are initially created as model elements of a requirements tracing model to ensure back-traceability from identified services, based on refined operational processes and refined operational requirements. Identified information exchange needs are then transformed into a hierarchical set of interrelated (service level) capabilities to be potentially provided by services. To ensure the back-traceability of those capabilities to requirements and thus enable requirements coverage checks, a mapping of capabilities to requirements is created. The capability modeling step is not needed if the requirement represents a single capability of the service. In this case the requirement can be directly traced to the service contract.

The **Service Identification** step performs an iterative drill down from process level to services level and aims to produce a portfolio of candidate services. During the drill down, parts of the operational processes to be potentially supported by services will be identified and classified. If needed, the high level capability hierarchy provided by the Requirements Analysis step will be refined to capabilities that can be immediately exposed by single services. This will be the input for assessment, decision, and responsibility assignment of services through the ISRM governance process. To ensure the back-traceability of candidate services to the capabilities they expose a mapping of candidate service contracts is created.

The **Service Design** step further elaborates on the services of the initial service portfolio that have been decided to be exposed. It produces an integrated and consolidated services specification comprising the static structure as well as dynamic and behavioral aspects of the service. It therefore creates a detailed description of the service interfaces, intended service contracts, detailed interaction behavior, as well as the Quality of Service (QoS) aspects offered and requested, and service policies to be obeyed.

The **Service Development/Implementation** describes the implemented Aerodrome Map Information Service, abbreviated AMIS, which supports pilots, air traffic controllers, airport personnel, as well as pre-flight planning personnel. The four sub services are described in detail in section 5.2.

3.4.4 Aeronautical Information

The AIXM is a specification to enable the encoding and distribution of aeronautical information in accordance with the ICAO convention. In order to enable the transition to the AIM, AIXM has evolved from version version 4.5 to 5.1 [Eurocontrol, 2010]. The AIXM conceptual model is specified using UML. AIXM UML provides a semi-formal description of the AIXM data model. For encoding and exchanging aeronautical information, the AIXM XML schema defines how to represent aeronautical information in an XML format and serves as the exchange model for aeronautical data. It is an implemented XML schema of the conceptual model. Therefore, it can be used to send aeronautical information to others in the form of XML encoded data, enabling systems to exchange aeronautical information. This information combined with implicit operational knowledge is key for OBSD. The ISO provides standards e.g. for meta-information

or, Geographic Information System (GIS). Open Geospatial Consortium (OGC) defines geospatial standards, e.g. OGC Web services like the Web Map Service (WMS) or the Web Feature Service Interface Standard (WFS). As shown in figure 3.5 AIXM is built on top of those layers.

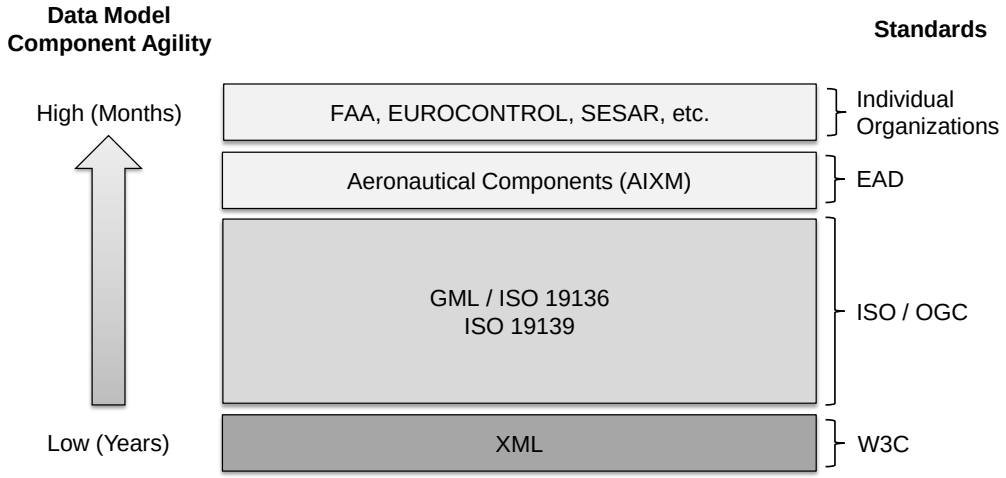

Figure 3.5: Layers of the Aeronautical Information Exchange Model (AIXM).

For the aeronautical domain AIXM is the starting point to build an aeronautical ontology. An ontology-based approach has the capability to build relationships between instances and classes. The properties of those relationships allow reasoner to make suggestions about them. Figure 3.6 shows a small sample: `:RunwayClosure` (concept) is a specific event of `:Notam` (type of property `:AeronauticalInformationEvent`), `:RunwayClosure` could be an `:Issue` (relation), an `:Airmen` gets alerted through an `:Issue` (assertion). The labeled relationships `isSortOf` and `isAlertedBy` infer the fact that an `:Airmen` is alerted by a `:RunwayClosure`, which is a specific `:type` of `:Notam`, which in turn is a subclass of `:AeronauticalInformationEvent` (reasoning). This reasoning is possible because of the inverse property of `isSortOf`, which relates the two instances in the reverse direction.

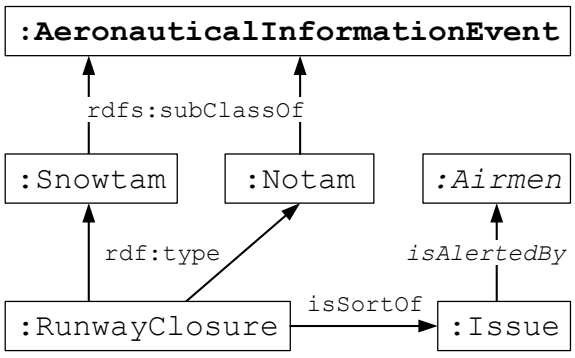

Figure 3.6: Snippet of the aeronautical ontology.

Several facts could be inferred from these relationships. Instances can either belong to a set of aeronautical information events or the set of risks, but only specific kinds of events are critical. In terms of ontology languages, classes are mutually disjoint. There are no instances that belong to both. In figure 3.6, a `:RunwayClosure` is some sort of `:Issue`. However, with the knowledge of this example we could not conclude that some type of `:Snowtam` is an `:Issue`. 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.

3.4.5 Meteorological Information

WXXM supports data-centric environment, meteorological information collection, dissemination and transformation throughout the data chain. Similar to the AIXM, the conceptual model is a high level view of the meteorological domain related packages (e.g. ICAO Annex 3, ISO 191**), that make up the data model. The conceptual model is implementation independent using a combination of plain text and UML package diagrams. The logical data model is technology agnostic as well and capable of supporting multiple physical representations. It describes in abstract form (UML) the association between exchange and information model. The physical representation is mapped to the logical data model by the exchange schema, which is implementation-dependent. It is a XML based schema which is a machine-generated "physical" representation of the model. The geospatial aspects are modeled using version 3.2.1 of GML [OGC, 2007]. For this work the conceptual and logical model are used to build up the necessary ontology. The base layer of the conceptual model serves as a foundation of basic concept packages (cf. figure 3.7).

For example, the ISO data types package contains '`Speed` as a type, which is a very generic

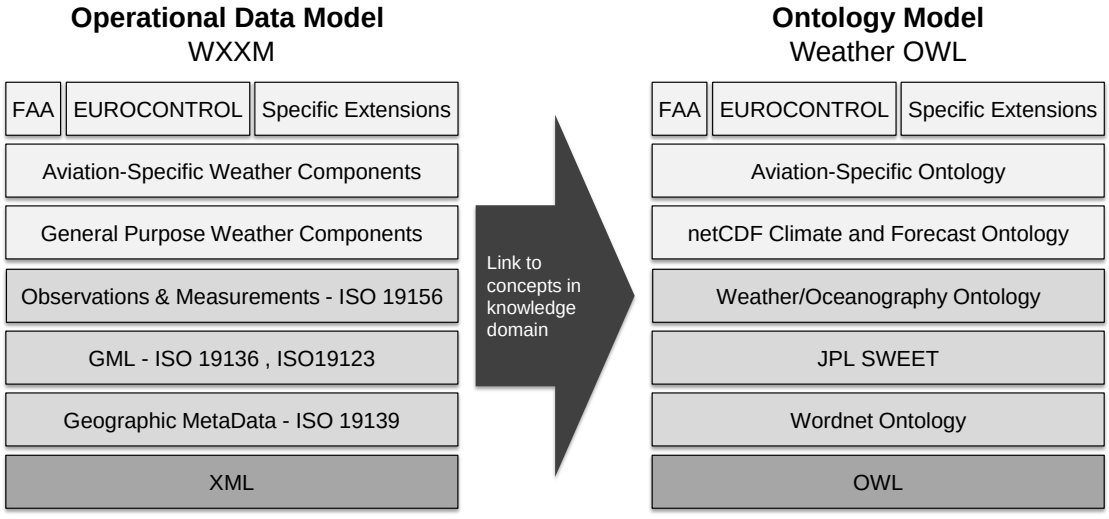

Figure 3.7: Operational and knowledge domains.

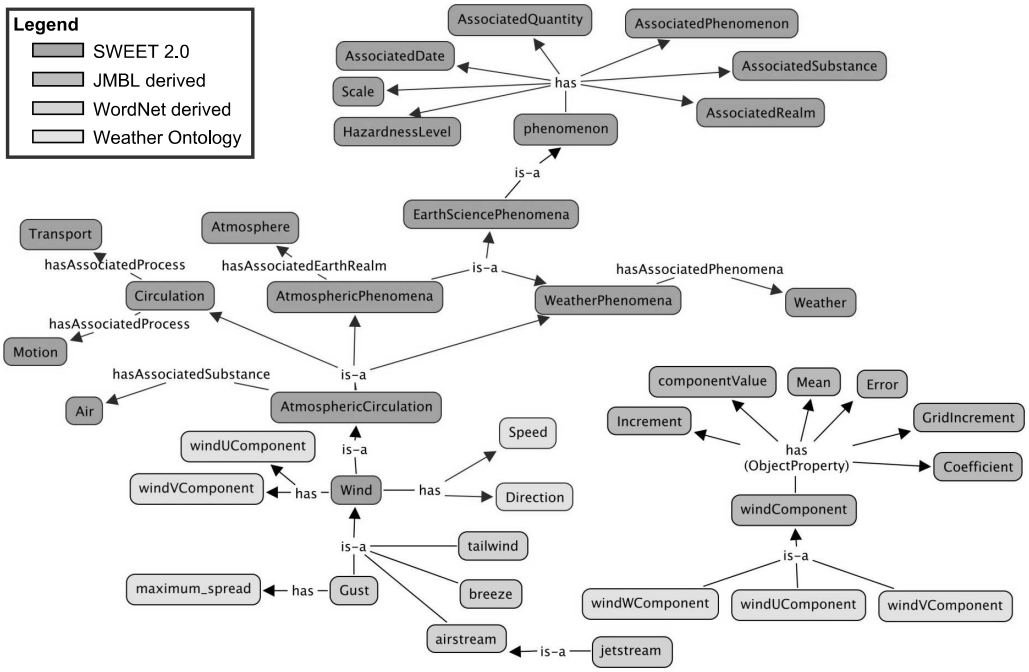

Figure 3.8: Constructing the weather ontology.

type. The weather layer above than specifies a type `WindSpeed`, which would provide the base layer's `Speed` type with some weather-related specificity. More and more specific packages like an aviation weather layer or organization add-ons are put on top of it. Just as the general weather layer's packages build on the concepts encapsulated in a base layer package, so do even higher-layer packages build on the concepts present in lower layers (cf. figure 3.7). The purpose of these layers (indeed, of the weather conceptual model as a whole) is to provide a common vocabulary and taxonomy for logical and physical representation of the weather data model. FAA's NextGen Network Enabled Weather (NNEW) program constructed a first weather ontology to enhance semantic interoperability of WXXM users and to enable the reuse of domain knowledge (cf. figure 3.7). NNEW uses WordNet, a large domain-independent lexical database, as base layer. Also, the suggested upper merged ontology[4] is used here as it contains a WordNet mapping. The base ontology covers general high-level concepts, but not necessarily domain-specific. The mid ontology contains domain-specific concepts, which are not necessarily tied to aviation weather. Therefore the Jet Propulsion Laboratory's (JPL) Semantic Web for Earth and Environmental Technology[5] (SWEET) version 2.3 is used. Domain-specific concepts are captured at a lower level of. Different weather phenomena and their quantitative measurements are represented in climate and forecast meta-data of network Common Data Form (netCDF)[6]. Even though all ontologies store the same fundamental ideas, the terminology and definitions

[4]http://www.ontologyportal.org/, accessed: 2016-10-10

[5]http://sweet.jpl.nasa.gov/, accessed: 2016-10-10

[6]http://www.opengeospatial.org/standards/netcdf, accessed: 2016-10-10

are slightly different. During this work a mapping transformation algorithm was implemented (cf. section 4.4), which can create alignments that connect single concepts in one ontology to multiple concepts in another. A search for all the data pertaining to `WindSpeed` would then return, not only its related data from `Weather`, but also the data categorized under `Wind` and `Speed` in `Meteorological Phenomena` (cf. figure 3.8).

3.5 Conclusion

This chapter gave necessary insight about the special requirements for a safety critical environment. More precisely it highlighted the great benefit of reusing components in the Air Traffic Management domain, in which products have a very long life cycle. Standardized information management ontologies, data, and service models based on EA and SOA principles, require joint forces between ANSPs and the industry. This chapter presented an extended conceptual architecture of the European SWIM infrastructure in order to meet the challenges of future aviation. Design and technologies of a system wide information management based on SOA principles were shown. The approach presented a technical infrastructure with interoperable communication, security, and governance components. The focus of this chapter is mainly technical, but the implementation of such a system will be deployed either as new application (or extension) of certain service providers or in case of existing legacy services as a SOA enabler. The possible fields of use are beyond the ATM SWIM scenario. Such software components may be the information exchange enabler in a number of other domain areas such as public safety, public transport, and other security systems. In a modern environment, software development profits from the reuse of code which improves in the best case the economic benefit. To achieve those benefits, specific information models for different ATM domains have been developed and the development will make further progress in the future. To accomplish this goal it requires more than simple UML-descriptions. Semantic logic is necessary to design and develop a component-based architecture. Such a venture has far reaching effects on systems, elements, procedures and regulations, but is required to achieve the benefits of an OBSD, to conquer the dream of generating directly out of a data model useable lines of code. The next steps in terms of global standardization are to bring the AIRM to ICAO level, which will ensure the the long-term future. More details about the domain knowledge is presented in the next chapter.

CHAPTER 4

Ontology-Based Software Development

"Any sufficiently advanced technology is indistinguishable from magic."

[Clarke, 1973]

Today semantic technologies are widely used in web application development. The main benefit of using ontologies is to combine knowledge from different domains and assume new facts based on them. The lightweight and flexible handling of ontologies allows to make extension simple, which supports the reuse of existing and upcoming implementations. Nevertheless, through erroneous ontology design and an inaccurate semantic methodology, the benefits can turn into the opposite. One way to alleviate those issues is to have a quality assurance process in place during the whole life-cycle. This chapter describes the methodology, semantic concept, and techniques of the Ontology-Based Software Development (OBSD). The first section outlines the OBSD methodology including the project and technological approach. In the following sections the life-cycle and process of OBSD is presented. Moreover, the usage of OBSD semantic techniques in a software development project are covered. In the section, ontology management, the usage of ontologies for data representation is motivated and the set of ontologies used in OBSD are described. Finally the semantic interface, a way of accessing the semantic description stored in the OBSD ontologies, is introduced. The ontology mediation process derives a logical solution design from the prior selected semantic sub-description of the software components and domain knowledge in combination with the requirement patterns among the stakeholders.

4.1 Introduction

Today the industry and the ANSPs operate in a non trivial domain which requires stable, but dynamic methods for communication and information exchange that can be easily adapted to potentially new business processes. Traditionally most organizations have developed software products consisting of numerous components. Such product developments cannot always support increasing demands for more flexibility and in particular, more interoperability. Over the

last years, several approaches were adopted to solve this architectural problem (i.e., enterprise application integration as a concept and the SOA [Zdun et al., 2007]). These approaches provide mechanisms for a flexible development of various business applications in one domain, but none of them was proven entirely satisfactory. The demand for an improved solution in the ATM domain generates the need for the development of an OBSD based concept. OBSD has to enable the sharing of information in a highly distributed environment, taking demanding requirements regarding performance, scalability, maintainability, safety and security into account. According to [Lung et al., 2007]:

> "*Software reuse and/or software components will not solve all problems we encounter in software engineering, but they will contribute to an important step towards more flexible software systems that are constantly evolving and adapting*".

Moreover, in a multi-user environment, OBSD allows decoupling of technical solutions and generic services for information sharing from business-driven user-specific applications. Each developer can then use these basic OBSD capabilities to make the best possible use of available information for their own development. OBSD aims to serve as software reuse knowledge base within the Air Traffic Management domain. This domain is characterized by very demanding safety and security requirements as well as the need for high availability, leading to conservative structures at present (cf. chapter 3). Data are exchanged among many stakeholders like ANSPs, airports, airlines, military users, general aviation, air traffic flow management instances, providers of meteorological, aeronautical or flight data, and many others. In the past, their actions and decisions were more or less decoupled from each other. However, the expected growth of air traffic in the next decades forces all ATM actors towards co-operative handling of virtually shared information during the entire life-cycle of a flight. The concepts of SWIM are seen as key enablers for sustained growth of air traffic management domain. These high-level concepts imply that an appropriate underlying technical solution for such co-operative handling of information is in place and that operational interoperability was established between the involved actors. In other words, for the high-level concepts to work, low-level mechanisms for information sharing have to be established and the corresponding operational procedures and practices agreed on and installed at all actors' premises.

Currently, only relatively conservative software component reuse capabilities and mechanisms can be found in some domains that cannot be really seen as a mean for software component reuse. The approach investigated aims to provide a scientifically robust and practicable method for establishing software component reuse among various applications and products based on standardized services and data models over a heterogeneous infrastructure using the aviation domain as an example. This includes the development of a concise semantic description to capture existing and reusable components in terms of data requirements and functional range as well as constraints of the payload, including algorithms to deduce the reusability from the semantics. This defines an automated way to acquire the canonical data exchange model as well as the resulting data flows and interface functions and description of the solution architecture with all functional components. The key motivation for OBSD is to elaborate fitting components based on existing SOA patterns to address the following issues [Zdun et al., 2007]:

- Automatically discover necessary information during design phase.
- Reduction of design effort for integration processes, data and service models.
- Methodology to precisely describe reusable software components.
- Manage sources of wrong, incomplete and outdated information.
- Allow multiple data sources for the same data.
- Reuse of reliable software components in various products and projects.

4.2 Ontology-Based Software Development Methodology

The static growth of data, which is processed within a control room, led to the development of overlapping software components for different domains. It is often the case that the descriptive name is different, but the functionality of such components is quite the same. The motivation of OBSD 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. AIM services in the domain of weather data, geographic information, briefing, tracking and tracing are among them. These are all together software products which share similar requirements but cover different sub-domains. Nevertheless, such components are often developed twice for each domain for example the GIS component for visual representation or data mapper. 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 to benefit from already developed systems. To prove the Ontology-Based Software Development methodology, the ATM domain was chosen as a first testbed. Today, industry and ANSPs operate in a non-trivial environment requiring stable and standardized but also versatile means for information exchange that can be easily adapted to changing business needs. Semantic techniques are an adequate means of bridging the knowledge gap (e.g. different names for the same entities) between organizations and improving information sharing and business processes (cf. chapter 2). Through the mapping the semantic description with the help of model rules and operational requirements, an easy identification of reusable parts between different development team within large companies is applied.

A significant point for the whole project is that it follows the idea of open standards. OBSD mainly relies on open source solutions. The OSBD is supported by different state-of-the-art ontology languages as well as relevant semantic environments for ontology development. In chapter 2 necessary tools and languages like RDF, RDF(S), OWL, OWL 2 and SPARQL, a standardized RDF query language were identified. As different ontology languages have different facilities, it was necessary to evaluate them. The knowledge in OBSD is captured with OWL, due to the fact that it has the most complete set of language features to express different concepts and relationships that occur within an ontology. Of similar importance is the right choice of frameworks and tools (cf. subsection 2.4.2). An ontology is a information model derived from the semantic description and represents a certain domain. Domain ontologies are used to

represent the information objects and their specific properties in that domain and the relations between them. Information in the semantic web model is stored in form of triples (instead of conventional key-value pairs) with a *Subject* as resource to be described, a *Predicate* as property of the resource, and an *Object* with a property value of the resource. This information is used to build vocabularies that describe a domain and the relationship between those resources. Based on the relationships, implicit hidden information is derived and inconsistencies are identified. The information stored in ontologies is accessed through the semantic interface (cf. subsection 4.5.3). The use of ontologies for modeling data instead of traditional data modeling using UML/XML (e.g. entity relationship diagrams) is the key factor for future reusability. Since ontologies are per definition never complete, future concepts or adaptations can be integrated without the need to change the whole model. This provides high flexibility for the future integration of new applications, products and services. Within OBSD, the following ontologies are used:

- **Domain Information Ontology**: Defines the elements that represent a specific domain used within the OBSD by adding individuals (instances) for the software components described in a software component ontology. This ontology also defines the concepts and therefore the taxonomy for the domain-specific knowledge. The main knowledge for this ontology is extracted from existing information models which were used as baseline for the initial semantic load.

- **Software Component Ontology**: Contains the basic concepts for a OBSD-based scenario (e.g. the integration of different legacy applications in the ATM domain), the reusability specification. This includes concepts for modeling the software components of various applications and products which shall be reused. A perfect generic example for a possible software component, which is recyclable, is a data mapper/validator for a specific information model translating from one version of the model to another one (e.g. AIXM 4.5 to AIXM 5.1).

For this thesis an initial version of a state-of-the-art information management was provided. In addition a highly reliable data improving software component reuse with heterogeneous data and service sources was implemented (cf. table 4.1). The generic OBSD domain information layer captures the information management of a domain itself, such as AIXM, WXXM, FIXM, or information reference models like AIRM or ISRM (cf. section 3.4). Below this layer the

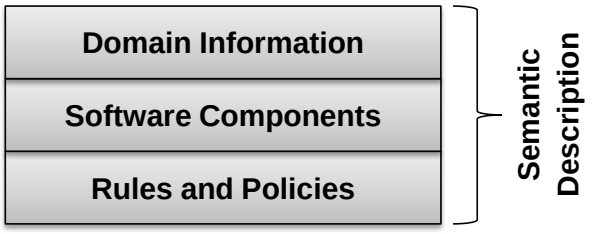

Table 4.1: The semantic description layers.

software component layer defines the software elements and components, that shall be reused in various applications and products within a company or organization. The bottom layer is the rules and requirements layer that customizes the mapped semantic description of OBSD to the environment for a specific customer or product via rules built from the requirements. Those three layers build the so called semantic description of OBSD. This approach provides a common foundation for integration projects where only domain specific functions or customer requirements have to be adapted while other basic components can be reused. All three layers exist in versions that evolve concurrently over time. Thus, for this thesis an initial implementation of the generic OBSD layers and the case study in the ATM domain was performed. In addition to the improvement of intra-organizational integration, OBSD enables inter-domain development as well.

4.2.1 Project Approach

The implementation of OBSD requires significant research in several fields of semantics and software architecture. Therefore, the project was broken down into the following tasks:

1. Elaboration of the requirements for the generic software components to accommodate the needs of an initial case study performed with the OBSD methodology.
2. Diverse discussions between experts from different domains to extract the semantic description of highly complex relationships and their definitions between existing information models.
3. Development of the semantics, usage and extension of existing domain ontologies.
    a) Use of the already established SWIM data models and FAA ontologies as an initial baseline for the OBSD ontologies. Integration of properties between OBSD external data format (UML/XML) and OBSD internal data format (OWL/XML) including the specification of OBSD data sources and targets.
    b) Specification of OBSD attributes for software reuse determination in a given time frame and for OBSD operations.
    c) Mapping the reusable software model with the domain information software model.
4. Elaboration of the semantic mediation process for transforming the semantic models into a list of reusable components ready to use. Provide rules and queries gained from operational requirements to build a solution model with the reusable components to use.
5. Establish filters and query algorithms for determining the semantic rules in the pool.
6. Implementation of a Quality Assurance (QA) concept (software component monitoring and empirical evaluation) for assessing the core qualities of a OBSD solution during design-time.
7. Development of real products using the OBSD methodology to proof the concept. This shall be a case study for the current version of the OBSD methodology to show the benefits and also issues that may have emerged by software component reuse.

8. Scientific evaluation of OBSD in comparison with the former software development process through the Extreme Programming Evaluation Framework [Layman et al., 2004b] (cf. subsection 1.5.2 and chapter 5).

4.2.2 Technological Approach

Table 4.2 shows the structure of the technical layers of OBSD, divided into the domain information, software components, the operational needs, and technical infrastructure which lead to a so called semantic description. This structure allows modeling each layer of OBSD (in gray) separately to enhance the flexibility of the solution. OBSD provides the mechanisms and methods for creating the necessary semantic description, coordination and matching the knowledge bases. In operational environments, existing technical infrastructure, transport layers, and software standards are given. The abstraction of the information management from the physical transport allows OBSD to operate within a highly heterogeneous technical infrastructure and is based on the software component selection only.

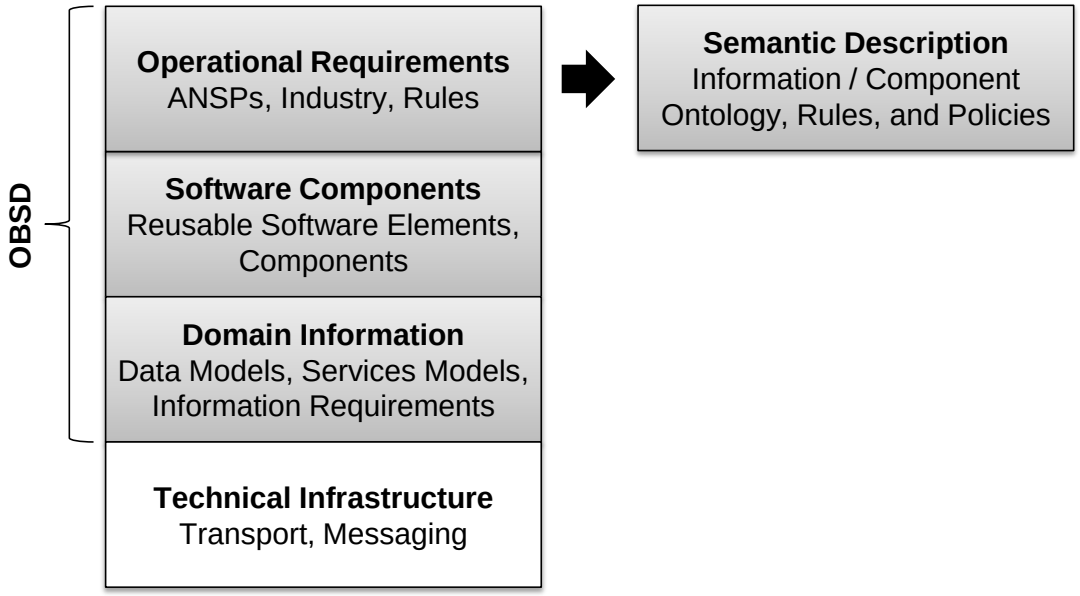

Table 4.2: OBSD technical layers.

OBSD distinguishes between two major phases in the life-cycle of a solution: design-time and run-time. First a set of two models is implemented, one for the components that can be reused and another one capturing the semantic information which uses the data models AIXM, WXXM, FIXM, and information reference models, like AIRM and ISRM, as baseline. During the design phase the solution model (the logical model of the OBSD based calculation, containing all attributes and software component information available for reuse) is deduced from the semantic description of the system. This flexible process allows quickly changes and easy

development of extensions. It contains a list of possible reusable software components, which code artifacts can be accessed via the repository of available software components after selection through the semantic mediation process. The generated solution model is then mapped to used technical infrastructure. Thus, the OBSD solution model covers mainly standard components (i.e. an data mapper, validator, etc.), which provide optimal reuse capabilities during run-time (e.g. less integration effort, etc.). An OBSD solution normally is operated without any outages. Therefore changes and extensions to the solution are introduced into the solution model via changing the semantic description leading to a system redesign. The resulting changed solution model can be used by the run-time part without any interruptions.

OBSD uses different components to implement the goals described above. These layers reflect the usual approach of an integration project by first describing requirements, then developing a solution model that meets the demands, and finally rolling out the reusable components. The solution model provides the selected software components for reuse indirectly within an IDE plug-in to improve the usability for software developers. Figure 4.1 gives an overview of the main OBSD components and shows the split between design-time and run-time. During the design phase a solution model is developed based on the semantic description without interfering with the operational development system. This allows changes or updates to the models that lead to the semantic description without interfering with the operational development system. Software requirements for the new development are captured as rules and then performed by the semantic mediation task to create the solution model. After deployment the solution model is executed within the operational development framework during run-time. This architectural design simplifies version tracking and plays well with agile software development methods.

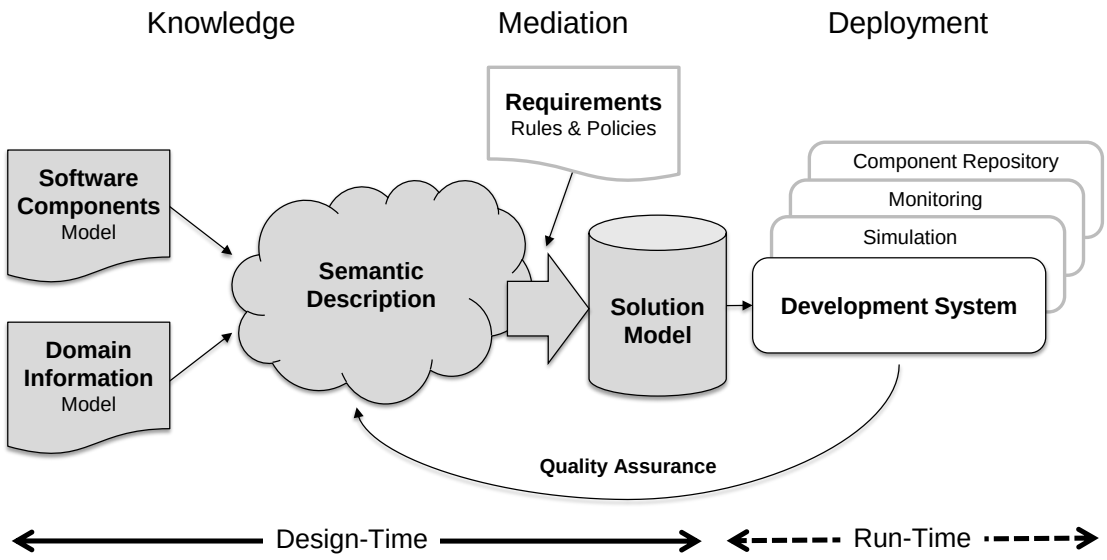

Figure 4.1: Overview of the Ontology-Based Software Development methodology.

4.2.3 Life-Cycle

As shown in figure 4.1 the OBSD life-cycle consists of three main tasks, collecting the **knowledge**, the model **mediation** process and the **deployment** phase. The semantic description captures all the knowledge. The solution model is generated during the design-time by queering the knowledge of the needed ontology models through the semantic mediation in an iterative way. The solution model acquires rules and special configuration setup to meet the requirement that are given for a software component to be selected as reusable. Below the three phases are split up in various sub-processes:

- The semantic description represents the **knowledge** unifying the domain information and reusable software components which is the core of OBSD. Various experts collect existing knowledge or import already existing models to the core. In the process of building a coherent semantic description the following tasks have to be achieved:
  - Ontology Modeling
  - Ontology Verification
  - Ontology Reasoning
  - Ontology Optimization
- The model **mediation** process generates a solution model, which contains a complete set of reusable software components ready to use by the developer. To calculate the solution model the customer requirements are used to select the right components. In an iterative process a solution model is created by the following steps:
  - Customer Requirement Specification
  - Definition of Requirements Queries and Restriction Rules
  - Solution Model Calculation
  - Solution Model Optimization
- The **deployment** phase of the solution model production can be separated into two tasks. The deployment itself and the QA task. Simulation is needed to evaluate a solution model before deploying it to the development environment and improve it over time. The deployment process is monitored to provide iterative feedback to the knowledge and mediation process:
  - Deployment Simulation
  - Deployment Monitoring
  - Solution Model Deployment
  - Semantic Description/Solution Model Optimization

The life-cycle can be split up into design-time and the run-time processes which are described below. The modeling and mediation part at design-time is performed in multiple iterations to identify issues and problems before rolling out the solution model to the development system. These steps may be executed several times before it is ready to use to evolve the quality of the models. Since the design-time is completely independent of the run-time, when a version of solution model is established, there are no constrains that interfere between design-time and run-time processes. During design-time the models are implemented, compatibility tests, reasoning and refinement of the ontologies are executed. Domain and component experts provide expertise to support these sub-processes. Versioning is handled to maintain the models and improve the quality of them. Semantic sub-descriptions are then transformed into the solution model. Requirement rules and queries are used to calculate the solution model.

During design-time the mapping process identifies reusable components, while at run-time the solution model can be accessed for deployment. QA is also part of the design-time processes including monitoring and evaluation of the models. The OBSD life-cycle ends with the run-time process by deploying the reusable software components from a solution model. An XML Schema defines the selected reusable artifacts which can than be handled by the IDE plug-in to access the code from the repository. For QA the behavior of the developers is monitored to feedback improvements back to the semantic description to improve the creation of a solution model. All reusable software components are then available via the component repository. Figure 4.2 shows the OBSD life-cycle in more detail and describes the necessary steps to create a set of reusable software components in form of the solution model which than can be accessed by the development system. The processes of the semantic description, the semantic mediation, the solution model and the QA aspects for all these phases are described on the next pages.

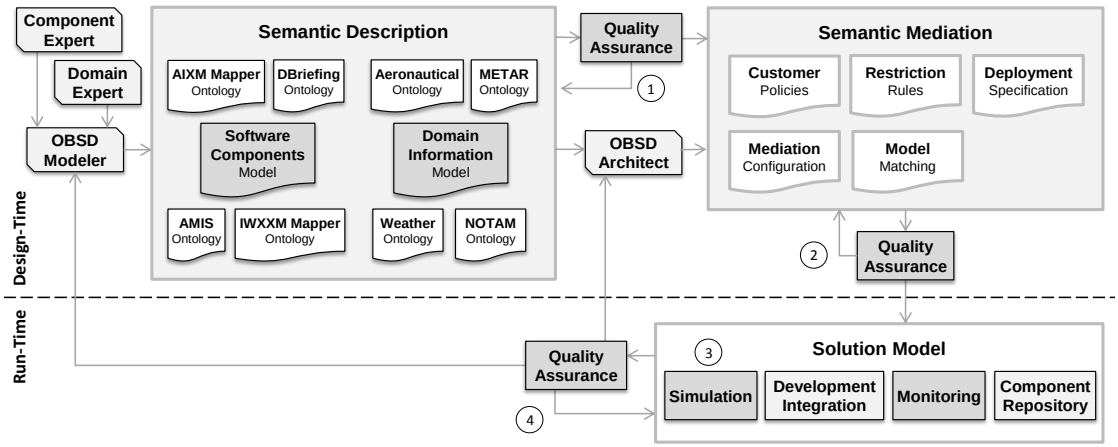

Figure 4.2: Ontology-Based Software Development life-cycle.

4.2.3.1 Capturing the Knowledge

The OBSD life-cycle begins with capturing the knowledge into a set of ontologies split-up into the domain information model and the software components model by OBSD modelers. The domain information model is described in detail in subsection 4.4.1 and inherits all semantic descriptions of a specific domain to discover redundancies of definitions. It is a known issue that in multiple operational scenarios the same information is shared. But because of the specific context, the definition name is different or they share the same name but the definitions are divergent. The domain information model also captures domain related messages, e.g. weather messages like the Meteorological Terminal Air Report (METAR), Terminal Aerodrome Forecast (TAF), Significant Meteorological Information (SIGMET) or aeronautical messages like a NOtice To AirMen (NOTAM) (cf. figure 4.2). The software components model on the other hand describes all artifacts that are planned to be reusable. It should be noted that the decision which artifacts are carpentered in the model can be influenced either from technical or from political perspective (cf. section 4.4.2). The variation of selected components to be reused be as diverged as possible but it should kept in mind that the more complex the selected software artifacts, the more complicated the calculations of the solution model. Examples for such artifacts described are exchange model mapper or services which can be encapsulated as a single component. With a semantic editor the ontology models are validated and invalid statements are updated. The quality of the models is checked by a specified QA gate which takes place before the hand-over to the semantic mediation process. These two models contain several ontologies which form the semantic description. A selected semantic sub-description may than be transformed, with the help of the requirement rules, into a solution model which can be deployed. Determined by the requirement rules and other configuration settings the ontologies are matched to create the solution model.

4.2.3.2 Information Mediation

The information mediation process receives the domain ontologies from the domain information model due to the requirement queries and match them with the ontologies from the software component model together with the provided rules. The main goal of the semantic mediation task is to find reusable software artifacts from the software component model on the basis of provided semantic mediation configuration implemented by the OBSD architect (e.g. restriction rules, customer policies, deployment specification, and model matching). Additional tasks are focusing on fulfilling all software components demands and requirement attributes to offer the developers the best suitable and effortless integration of those artifacts into their development system (IDE). Through the versioning concept, components can be reused without any influence on the further software development of the components themselves. The goal is to optimize the reuse of software components according to company criteria, and reduce integration and implementation cost at the same time. The outcome of the semantic mediation process is the solution model which declares the artifacts that are reusable within a new project. With the implemented feedback loop, which is part of the QA concept of OBSD, the result is optimized iteratively.

As already mentioned, the information mediation process generates a OBSD solution model in form of an XML file that specifies the reusable artifacts from the component model. An XML schema is provided to define the XML structure, which can than be used to access the right components from the repository. The calculated XML can be used to load the identified reusable components and initialize the roll-out process for the IDEs. The QA process tests the OBSD solution model if the links to the reusable components repository and the description of the technical and logical functionality are correctly described. The correct integration of logical and technical dependencies is a very important task to assure a certain amount of quality. A repository with all common reusable software components that can be reuse shall be provided and is not further investigated in the course of this thesis, nevertheless it was used for the case study to gain information about the advantages and disadvantages of the OBSD. The advantages of a clear split between design-time and run-time environment are trivial changes of specific settings, different parameters and faster evaluation than within a coupled architecture. The OBSD monitoring process provides a way to evaluate the solution model (testing coherency and completeness) before it is rolled-out to the development system. Software architects shall check if the queried items exist and are correctly linked to the real software artifacts. Any issue is analyzed and wrong behavior is fed back to the OBSD configuration, whereas oversimplification should be avoided. An in-depth analysis brings redundancies and critical elements within the solution model to the surface (e.g. wrong domain overlaps, false descriptions, wrong system behavior, etc). The feedback loop can also go directly to the OBSD modeler to improve the semantic description.

4.2.3.3 Deployment Phase

The solution model roll-out is deploying the OBSD solution model to all developers. If a developer uses a special kind of IDE which is not supported by the OBSD solution model roll-out service (publication/subscription), it can be imported in a manual way. As part of the case study Eclipse and Visual Studio were used and linked to the reusable components repository. For selecting the queried components, the roll-out service publishes necessary information to select the correct components from the repository available in the corresponding IDE. A new solution model is exchanged automatically after major changes in the semantic model description (e.g. component changes, new components are added, etc). OBSD handles multiple versions of a solution model concurrently based on the deployment progress of a new version. It is important to take dependencies into account if different versions of a solution model exist, so that at least one instance covers the requirements (if possible). By comparing various versions, diversities can be identified and lead to a required update of the components (e.g. different exchange model support, etc). To handle those requirements a repository allows tracing back deployments and provides the configuration settings to switch back to a previous working configuration in case a rollback from a solution model becomes necessary.

To optimize or solve a problem with a created OBSD solution model the semantic description can be simplified by limiting the domain or software component models used. As a result only working semantic sub-description can be composed into a bigger description. This was very important especially for the early development and testing phase of OBSD. The solution model is the outcome of the semantic mediation process and can be used in a first step within the

simulation process to imitate a real world software development environment to proof functionality of the IDE plug-in that is used to access the reusable component repository or to improve the semantic quality that is provided by the OBSD. Simulation offers the possibility to check the solution model upfront the deployment integration. A regular test regarding coherency and completeness of the knowledge is done with real data from the repository. Another simulation is to analyze false OBSD behavior by using reduced sub-description selections to narrow down the error. The important advantage of this specific strategy is the opportunity to easily adjust minor configuration settings when the selected input knowledge is smaller. And of course it is a much fast way to process the semantic mediation and speeds up the evaluation process of the whole OBSD life-cycle. Meticulous reasoning helps to find bottlenecks and other practicalities within the the whole process.

4.2.3.4 Quality Assurance

Software development for safety-critical environments requires high standards for quality measurement, Quality of Service (QoS), and auditing. Several Quality Assurance (QA) related steps are necessary within the OBSD life-cycle to guarantee this (cf. figure 4.2):

- **QA 1:** The semantic description has to pass semantic validation tests before being handed over to the semantic mediation process. This includes formal consistency and validation checks of the models. In case of an inconsistency error, messages are created which than shall be edited by the model owners in an semi-automatic way to update the models accordingly.

- **QA 2:** Another QA check is performed after the semantic mediation process has calculated a solution model. The generated model has to pass again some validation tests (e.g. if there is one valid component matching with the requirements at all, otherwise minimize the requirements to detect at least some overlapping parts). Issues or problems are reported back and used to improve the configuration of the semantic mediation process led by the OBSD architect.

- **QA 3:** During the simulation process the compiled solution model and its selected set of reusable components from the component repository can be tested in a test developer environment before the actual roll-out where the actual software developer get there hands on the results of the OBSD system. Performance values are monitored which may lead to changes of the configuration of the semantic mediation model or to improvements of the semantic description to minimize the gap between the knowledge description and the real development process.

- **QA 4:** The values measured by the simulation and monitoring process are categorized by the QA process and forwarded either to improve the quality of the solution model, the mediation process or the semantic description.

4.3 Ontology-Based Software Development Processes

The OBSD processes can be divided into three main ones, the production of the semantic description, the identification of reusable components using the mediation process, and the creation of the solution model. This section discloses those processes including a definition of the required experts involved. Each process step originating the semantic description and the solution model is explained in detail from a process-oriented view. In the first subsection the semantic description origination process is characterized. The second subsection provides insight on the solution model origination process identifying possible software components during the semantic mediation. The pre-conditions to initialize the OBSD processes are an accessible repository with the reusable software component, an semantic interface defined to access the knowledge in the semantic description, and experts available to perform the manual parts. Finally responsibilities and roles are outlined in detail.

4.3.1 Semantic Description Origination

Figure 4.3 gives an overview of the semantic description process. The sub-processes involved, the knowledge that is built, and the experts involved are described below in detail:

1. First step is to construct the **domain information model**, consisting of various ontologies from ATM sub-domains. This task is accomplished by modeler experts in a joint undertaking with domain experts to capture the knowledge. Together they build a board of experts and are the owner of this process. They discuss which information should be added, updated or deleted from the model. But not everything is built from scratch, a lot of existing ontologies were taken on board or exchange models were converted into an ontology with minor changes (cf. subsection 4.4.1).

2. A group of software architects together with a model expert built the semantic concepts behind the **software component model**. Together they build a board of experts and are the owner of this process. They discuss which information should be added, updated or deleted from the model. All relevant software artifacts that are planed for reuse have to be modeled. This includes the provided and/or consumed message type, the sending/receiving frequency, and the type of the service (e.g., request/reply), etc. For more details see subsection 4.4.2 which defines what is meant be the term “component” and which substructures were captured.

3. The **operational requirements** are captured as additional ontologies to the existing models. Subject-matter experts own this task, which is iterative for each model of the semantic description.

4. In an iterative way, while modeling the semantic description, the ontologies are checked regarding their completeness based on configuration criteria (e.g. to remove conflicting or redundant information) to support the reasoning processes. Additionally ontology consistency checks are executed to ensure a certain amount of quality in the description of the semantic knowledge (cf. subsection 4.4.5). Last step of the **quality assurance process**

is to perform the repository synchronization check. For each software component in the model there should be at least one code artifact in the reusable software repository. This is an automatic sub-process to prove the reciprocal existent of the semantic description and its linked, real software equivalent.

4.3.2 Semantic Mediation

1. **Global model restriction rules** have to be defined in order to apply certain dependencies and restrictions for the specific model (e.g. a specific exchange model format which is only valid in certain components). The group of domain and component experts are responsible for those rules.

2. The **customer policies** in form of queries have to be defined to cover the specific policies and needs of the customer. The policies specify the phrased attributes and customer requirements in order to select the right choice of reusable software components.

3. For each OBSD project a **semantic sub-description** is produced. Such a subset of the semantic description is an alignment of the needed ontologies extracted from the domain information model and the software component description taken from the corresponding model (cf. subsection 4.5.1). In combination with the related rules, requirements and policies another round of consistency checks are performed on the semantic sub-description.

4. A further step is the alignment and mapping of the program artifact concepts described in the software components model with the reciprocal domain information ontologies. **Ontology mapping** can be a very tricky and fussy especially in the initial phases of OBSD, as a result the domain and software experts support this task (cf. section 4.5). The last step of this process is the filtering of redundancies or collaboration errors. In addition, available global rules and policies are applied.

5. The last step is to perform the **quality assurance** process by a group of experts to review the ontologies. Hereby the definitions and matches are checked if they are semantically valid. Especially the mapping between the domain information model and the software component model is in the focus of this quality check (cf. subsection 4.2.3.4). Last but not least the consistency of the ontologies is proven again after the mapping process.

4.3.3 Solution Model Deployment

1. The solution model deployment process begins with the **solution model origination**, which defines a set software components that shall be reused with the started project, selected through the processes executed before. The identification process of these reusable components fetches all possible matched regarding the requirements, rules, and policies (cf. subsection 4.2.3.3.

2. **Simulation** is performed in order to improve the solution model deployment. The process is running in a virtual machine and is a 1:1 copy of real world IDE setup and can be run to improve, test and upgrade the current solution.

3. The solution model provides an XML document specifying possible reusable components. This list of **software components** is loaded into the IDE through an plug-in install.

4. For every component concept, the particular **reusable software component** code segments are imported for further processing from the repository that is available within the IDE. Therefore every possible component selected is checked if it exists in the repository.

5. The developers can now select those elements, preview them and then select which one of them they want to use within the new project. In addition the developer can submit bug-fixes, improvements and extension to the specific component. New versions are submitted to the reusable software component repository and after a successful review OBSD processes can start from the beginning to capture the new available knowledge.

6. As part of the **feedback** loop often reusable software components were requested but don't exist are noted for further investigation and can be used in a process out of the OBSD lifecycle.

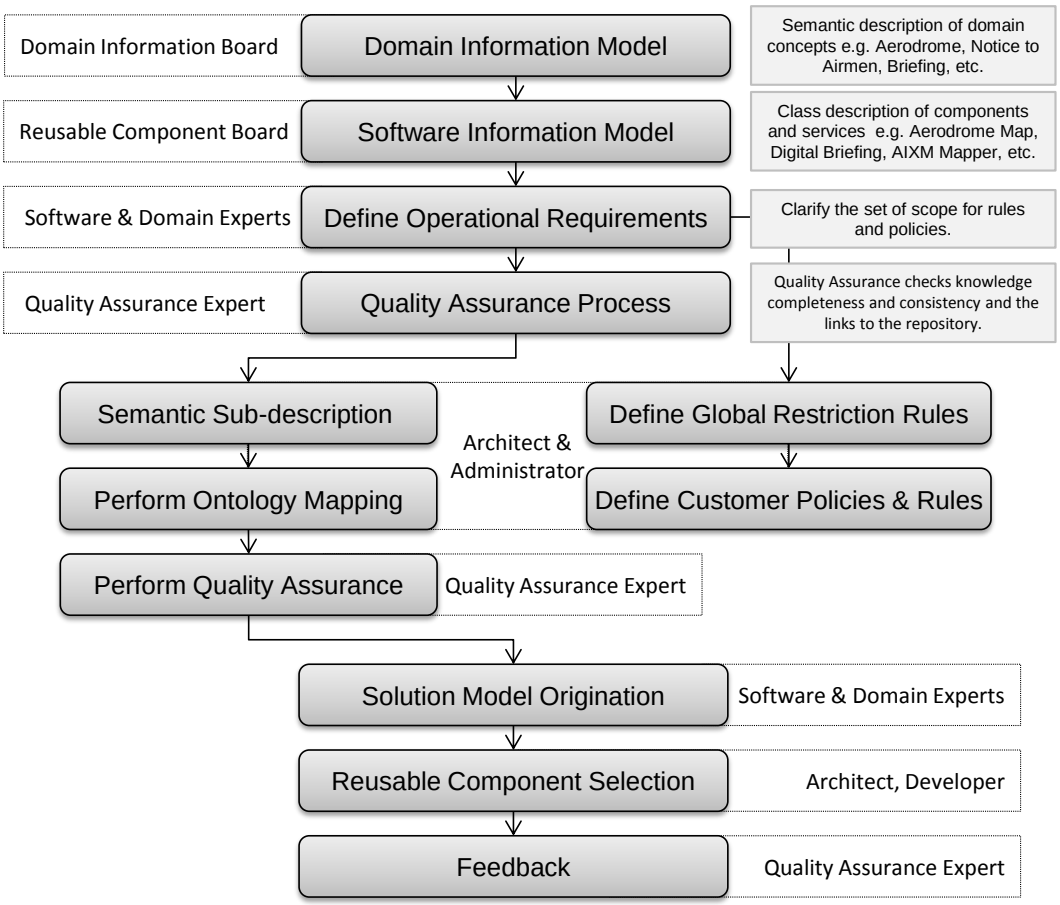

Figure 4.3: Overview about the whole OBSD processes.

4.3.4 Roles and Responsibilities

During the OBSD life-cycle many roles with various responsibilities are required by the different processes mentioned in the sections above.

The **Data Modeler** is responsible for the organization of the knowledge models. He manages the right modularization of the ontologies, the versioning, refinement, and the consistency of the models described in section 4.4. The data modeler supports the work of the domain information expert and the software component expert and is member of the domain information board.

Each domain concept has its lead expert that works together with the data modeler. The **Domain Information Expert** is the primary source of knowledge for the specific domain, manages the domain ontologies, and integrates the operational requirements and concepts into the domain information model (cf. subsection 4.4.1). On request he can coordinate the software component expert in regards of the semantic description and possible selected sub-description. Board approved changes have to be assessed and conducted by the domain information expert. He is also member of the domain information board.

The **Software Component Expert** has the responsibility for the software component model. He is the key source about functionality, architecture, and use of a components selected by the board to be reusable. The expert delivers information about the technical interface of a component and the data exchanged. Information about quality of service, quality and security test, and a basic understanding of the operational use-cases are necessary. He is member of the reusable components board.

The **Quality Assurance Expert** is accountable for quality measurement, Quality of Service (QoS), and auditing (cf. subsection 4.2.3.4). He is the supervisor of all OBSD processes described in section 4.3, the documentation and he checks the completeness and persistence of the semantic description. As member of all boards he ensures that all processes, simulations, and monitoring are performed correctly.

The OBSD **Architect** contributes knowledge about semantics and ontology management in general. As such he is responsible for the maintenance and evolution of the core architecture of the OBSD-based methodology (life-cycle and processes).

The **Administrator** has knowledge about information exchanged, the operational use-cases, and the OBSD architecture to supervise the whole OBSD life-cycle. He steers all the mentioned experts from above, is in a close cooperation with the architect, and is member of all boards.

The **Domain Information Board** consists of a group of experts i.e. data modeler, domain information expert, QA expert, and administrator. The board is the owner of the domain information model and responsible for the domain concepts that are captured. Changes, updates, deletions, etc. have to be approved by the board and redirected to the appropriate expert to deal with it.

The **Reusable Components Board** is the owner of the reusable components model and decides which components are taken on-board of the reusable repository and will be modeled as an ontology, and which are not. As mentioned before not only technical but also political interests can be the foundation of such a decision. Members of the board are software component experts, data modeler, senior lead architects (non OBSD role), QA expert, and the administrator.

4.4 Ontology Management

This section outlines the key principles used to design the OBSD ontologies. The modular design is based on the concept of Thomas Gruber following the minimal ontological commitment to support the knowledge base in an adequate way [Gruber, 1995]. The ontology profiles define just as many statements as really needed to enable the instantiation and specialization of the ontology itself. As most of the Semantic Web languages discussed in subsection 2.4.1 OWL offers a native way to extend the vocabulary to reach the goal of minimal commitment. This possibility was used within OBSD to change OWL properties and those link properties with specific formal axioms. The minimal ontological commitment approach also allows the reuse of the ontologies themselves between different domains. The semantic description is part of the logical layout of OBSD and contains all available information about the software components captured in the software component model and the behavior of the domain described in the domain information model. Ontology-based technologies are used for acquiring and storing the semantic description. The reasons for using ontologies are to make OBSD information machine process-able and to provide a knowledge resource that maps this specific information from different sources. Chosen tools and standards are described in chapter 2 and support the collection and processing of the required information. In order to meet the challenges of OBSD Protégé was selected as main ontology tool. As mentioned earlier within this thesis the chosen ontology environments were extended or tailored (e.g. Protégé with OWL, SWRL and SQWRL support) to meet the requirements of OBSD. Existing extension mechanisms (e.g. the SWRL Built-in Bridge) are sometimes insufficient for the needs of this thesis. In such a case, for the OBSD methodology, ontologies and rule language were extended or adopted.

One of the challenges in today's integration projects is to extract the required information from the available documentation to create a ontology model that can serve as baseline for subsequent implementation. In this context the following requirements emerge:

- **Completeness and consistency**: Creating a semantic model for an integration project is a complex and error-prone process. Logical additional deducible facts from already available information need to be derived from OBSD automatically to avoid duplication of work. This includes the UML/XML import of existing and already standardized data models like AIXM, WXXM, FIXM, and information or service models like AIRM and ISRM (cf. [Reiss et al., 2006], [Jie-ning et al., 2009], or [Keller, 2016]).

- **Maintainability**: The format needs to be adapted to changing requirements during life-cycle of an integration solution. If new components are developed or information elements are added or updated, OBSD must support adapting the elements, new features, etc. into the semantic models. Versioning of the gained semantic knowledge is an essential feature of OBSD.

- **Machine readability**: Once the semantic description is available, the mapped elements appear in a format that can be processed by algorithms to create a solution model. Further the context of the ontology model must allow the algorithm to produce human readable warnings or error messages.

Another important part is that the OBSD semantic interface that interacts with the semantic description and allows accessing and manipulating the information stored there. The semantic mediation process retrieves data from the semantic description by using this API when generating the solution model. Algorithms are used to build and validate the semantic description. The solution model contains all available software components in a repository ready to use for the development system (cf. section 4.1). It is generated during design-time by processing the information of the semantic description through the semantic mediation in an iterative process. The complete solution model can be checked against an existing solution in a simulation environment. A test bed built up by information gained from the domain information model and from monitoring data of the component model. The roll-out of the solution model for software developers requires a set of algorithms/rules and will than be loaded plugged into an IDE ready-to-use in a development project. A representation of an ontology model that satisfies these three requirements is the so-called semantic description as mentioned before. Ontologies represent information in a semantic description with the following data:

- **Concepts**: To collect and process data, predefined structures must be established to store the relevant information. First, concepts are defined as basic entities of the system. Examples for OBSD concepts are data structures, or software components including classes and their payload and attributes.

- **Hierarchy of concepts**: Concepts are arranged in a hierarchy. Examples for a hierarchy are consumer- and provider-services that share properties of the superior concept service (e.g. ISRM service catalogue and its sub-subsidiary services, object-oriented software design, or the information hierarchies used in the AIRM).

- **Relation of concepts**: Relations between single concepts are called properties. An example of a relation is a specific service with data model elements. Relations can be expressed between concepts or individuals (instances).

- **Hierarchy of relations**: Like concepts, relations can be arranged in hierarchies inheriting properties from superior relations. The concept of hierarchies is essential for OBSD as it allows to identify reusable components and for each part its sub-components.

- **Instances**: Instances are the actual representatives of concepts. The above entities (concepts, hierarchy of concepts, relations and hierarchy of relations) form a structure (or vocabulary) that is filled with the names and types existent in the system. Orders, product descriptions or status information are examples for the reusable component concept.

- **Relation of instances**: Instances of a specific concept are interconnected by property instances. However, it is not necessary that predicates always link instances of concepts, i.e. the object of the S-P-O triple can also be an enumeration of property values, a string or an integer.

Protégé is the ontological editor selected for this thesis (cf. 2.4.2.1) as it offers back-ends for UML classes and their relations using the standard UML and XMI format that is compliant with the Protégé Meta-Object Facility (MOF) meta-model. Since it does not load the model with

extensions like a UML profile, Protégé is not importing map class relations like associations. Its nontrivial architecture allows easy-to-use extensibility of third-party components. Some of these provide interfaces to other ontology-based tools as mentioned in subsection 2.4.2, 2.4.3, and 2.4.4. Many of those support various ontology languages like RDFS, OWL, DAML+OIL, OIL or XML. Protégé is based on the MOF meta-model, which is expansible and offers the ability to implement own extensions and meta-modeling concepts by adding knowledge primitives.

The right modularization of an ontology is a very important point. An ontology does not contain necessarily all information but can be split up into multiple ontologies (which is the case for OBSD) referencing and inheriting properties from other ontologies to capture all the important knowledge. Therefore, the information to describe a OBSD project is divided into the domain information layer, the software components layer, and the operational requirements layer as depicted in table 4.1. The domain information layer consists of different ontology models. Richard Keller, chief scientist for Information Management Technologies within the Intelligent Systems Division at NASA presented a good overview about resent use cases for ontologies relevant to aviation information management and summarized state-of-the-art semantic prototypes [Keller, 2016]. The AIRM (cf. subsection 3.4.1) was used as baseline for building those ontologies. The software components related information is also split up in a hierarchy of ontologies clustered by components, services, objects and the transported payload (which is than the major link to merge them with the domain related information. Operational and business requirements map the proprietary information, which is defined by a set of rules to the standardized information in order to ensure interoperability between reusable components. Compiling the requirement rules on the merged ontologies from the domain information model with the ontologies from the software component model creates the solution model which is a list of reusable components. Building the semantic description for a specific scenario, a subset of needed ontologies is chosen. Instead of conventional key-value pairs, information in ontologies is stored in triples and follows the RDF/OWL standard, which specifies an XML serialization form to provide machine-readable information. To build up the complete information about an OBSD system based on ontology schemata, the semantic editor Protégé is used to assist the data input process. In the course of data input, a user gets feedback about the completeness and consistency of the data. The problem of inconsistency can arise in situations where two ontology schemata define the same information. The semantic editor provides mechanisms to identify those conflicting statements. The OWL format allows visualization using a graph for maintenance [Lanzenberger et al., 2009b].

In addition, modularization enables maintenance of the ontologies at an acceptable level. Other performance issues can rise during the modeling, query, reasoning, and visualization process. This already was addressed in chapter 2 as current reasoners are handling small ontologies quite well but almost collapse on large scale ontologies. To keep up the performance, ontologies shall be small and modular by building such slighter subsets. Querying just on a small number of ontology subsets boost up the calculation time. Modularity is also a mechanism to enable reusability and compact subsets increase the chance of reuse. OBSD is described by a set of ontologies filled with information from the software component model and domain information model. To describe various aspects of an integration environment (i.e. different scenarios) there exist multiple versions of ontologies (ontology schemata) at each OBSD layer. All these on-

tologies together are forming the so-called semantic description. Dependent on the goal for an integration scenario (e.g. remodeling of OBSD during maintenance, or additional components added), the ontologies are chosen from there. By the semantic editor, the subset of ontologies is validated and missing or incomplete statements are edited. Having the ontology schemata available, the documentation is generated considering the expectations of all stakeholders. The information stored in ontology models is described in detail in the following subsections.

4.4.1 Domain Information Model

The domain information model of the OBSD methodology is based on the data and information models outlined in section 3.4. The model includes the main knowledge of a specific domain exchanged between different SWIM stakeholders. It represents a collaborative view on the exchanged data. To capture the logic in a precise way it is necessary to come up with a hierarchical structure having multiple domain ontologies which are all part of the domain information model. In addition, the domain ontologies are the place to model standardized, domain specific, information. All information of a particular domain required in a OBSD-based scenario is described as part of one or more domain ontologies. This includes, among others, the different concepts of the domain and their intra-domain relationships, the description of standards of the domain as well as the definition of the relevant data types and services. This is achieved by adding individuals of domain concepts that are referenced by a defined component. The domain concepts may or may not be arranged in a hierarchical way, like the concept flight related weather data may define a number of sub concepts like aircraft relevant meteorological data. In addition, the domain may specify a predefined set of data elements. This domain specific knowledge is used to identify semantic redundant information provided or consumed by any software component. The identification of possible reusable software components is made possible by extracting the semantic overlapping and identical knowledge of the OBSD ontologies in combination with the rules created from the requirement specification. As a result it is possible to detect semantic redundant information even if identifiers, format, or structure are different. This generic concept allows the reuse of components in a domain independent way. The area specific knowledge described in the domain ontology simplifies the identification of reusable parts by semantically connecting the domain with the software components information query them based on the requirements.

The essential part of the information domain model is coming from the AIRM, especially the knowledge of existing standardized information exchange models [Burgstaller et al., 2015]. The three main data models where also the first OBSD ontologies model, beginning with the weather, aeronautical, and flight related information. The domain information models tried to cover spatial representation, aviation equipment, flight and airport constrains, airspace infrastructure, ATM meteorology, flight and navigation relevant information. The OBSD sub-domains are defined by adding individuals to the concepts of the respective ontology, describing the particular data elements used. It is important to note that the borders between different domain ontologies are not set in stone, moreover they are based on the scenario and the selected information elements (e.g. some parts of the flight ontology can be transferred to the weather ontology and back forth). This allows an evolution of concepts as the number of reusable ontologies will grow and extensions and updates will change the domain ontology. A group of domain experts

is responsible for QA and advancement of the domain model and its ontologies, and the group is responsible for this evolution process, too. It is very important to establish such a domain information board to undermine proliferation (cf. subsection 4.3.4). This can easily lead to huge waste of effort building and modeling ontologies of various existing standards, some might are overlapping, which then end in an overwhelming diversity that can not be handled at all. The ATM related case study is a grateful scenario as the safety critical environment is first of all not evolving very quickly and secondly has already a standardized data models in a structured way which allows a smooth transition of the knowledge into ontologies.

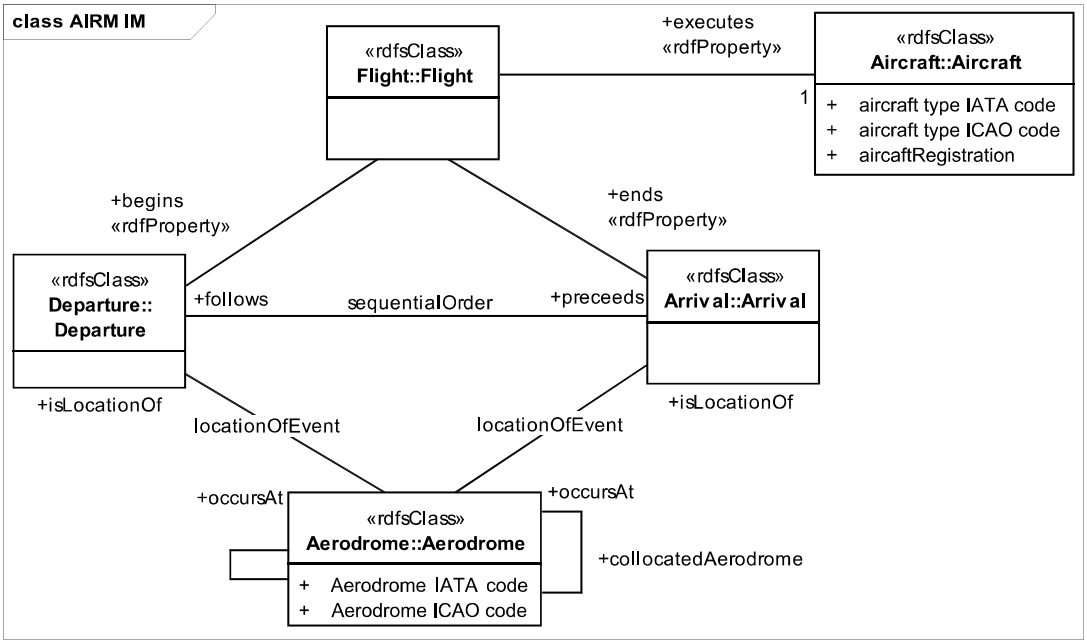

Figure 4.4: AIRM information model example transformed into an OWL ontology.

As initial load parts of the AIRM information model and the AIRM consolidated logical data model was loaded into the OBSD domain information model. The meteorological related part was used for the weather ontology which is based on WXXM/IWXXM parts of the AIRM (cf. subsection 3.4.5). Another one is capturing aeronautical information based on AIXM related parts of the AIRM (cf. subsection 3.4.4). Flight related information was captured mainly on the information elements described in the AIRM which is based on FIXM. For future work it is also planned to have an ontology dealing with environmental information. The AIRM information model currently is entity based. The reason is that the UML notation of the information model is best suited to provide the semantic underpinning of technical data structures. However, the operational discourse proceeds in plain language sentences. Consequently, the information model should be able to provide a bridge between natural operational language and the information and service modeling domain. This allows modelers to extract full statements from the operational discourse and cast them into a UML format (cf. figure 4.4). Reversing this process, subsets of the AIRM information model are extracted into RDF/OWL triples (cf. subsection 2.4.1.3). These

are used for the validation of the model by operational experts in a first step. In a further step this extraction is automated by a script. It is then used to map it against the component ontology (cf. section 4.2 and subsection 4.4.1). The restriction to a subset of OWL implies also that the AIRM does not attempt to be a full ontology model itself. Rather it provides a repository of standardized resources to be used in domain specific ATM ontology models derived from the AIRM (cf. subsection 3.4.4 and 3.4.5). The AIRM information model remains to be a formally consistent UML model and so fully usable in the technical domain. The proposed NATO extensions to the NOV-7 information model could be accommodated within NAF boundaries. Modeling rules for the use of OWL stereotypes and the AIRM name-space are defined (i.e. definition of the AIRM URI as AIRM unique name). In addition rules and practices are defined for resolving the URIs of the OWL elements to resources.

4.4.1.1 Classes, Properties and Specialization

Any information model class may be stereotyped as an `rdfsClass` or `owlClass`. If an information model class is stereotyped as a `rdfsClass` its definition is available in the "notes" property, and the following RDF triple (i.e. statement in the operational language) is valid:

```
airm:imClass.name rdfs:isDefinedBy airm:imClass.Note
```

Information model class attributes may not be stereotyped as an `rdfProperty`. Information model class roles may be stereotyped as an `rdfProperty` or `owlProperty`. In this case both entities connected by the associations shall be stereotyped as `rdfsClass` or `owlClass` and the name of the property shall be an active verb. If `relatesTo` is a role of class `imClass1` in `imClass2`, the following RDF triple (statement) is valid:

```
airm:imClass1 airm:relatesTo airm:imClass2
```

Modelling OWL/RDF stereotypes allows distinguishing "nouns" and "verbs" in UML class diagrams in a well-defined formalism. It thus combines the advantages of the existing solutions discussed above simple solution approaches with the use of an established standard. If two classes stereotyped as `rdfsClass` are related by a `UML::Generalization`, this generalization is implicitly typed as `rfdsSubClassOf`. To illustrate this claim the rules stated above are applied to figure 4.4, representing the statements presented in table 4.3.

4.4.2 Software Components Model

Per definition an OBSD software component wraps a group of functions or data elements to implement a particular functionality. It is structured in a modular way and usually packaged to be easily integrated. A component per definition can be a software package, a service or a trivial software module [Caldiera and Basili, 1991]. Since a service can be made up of a number of components it is a special form also captured in the software components model. Generic components can be reused almost without effort, specific ones most of the time only suit to particular systems. The communication between such software components is enabled through interfaces allowing the encapsulated modules to interact 3.3. According to [Sametinger, 1997],

"*Components are artifacts that we clearly identify in our software systems. They have an interface, encapsulate internal details and are documented separately*".

The software components model describes all available software services, components, and data elements of existing applications and products that shall be reused. Software components cannot be defined on the basis of a particular programming language or technology [Szyperski, 2002]. It is important in which manner the component will be used. Also essential is the context and the composition without any adaptions [Hamlet, 2010]. The software components model describes how components interact, how they can be composed, and which payload will be exchanged. Similar to the domain information model a group of experts are formed to a governance board (e.g. senior and lead software architects). The board has to provide the information needed for the description of the software components and act as a change control board with the power to vote about updating, adding or deleting components within the model. The experts are the primary source of information about functionality, architecture, and use of the applications. They are also able to provide information about the technical interface of the application. If software components are added or changed, through the semantic mediation a new solution model can be deployed. The change control board has the significant but nontrivial task to decided which software components shall be reused. This is sometimes, from a management perspective, a high level decision which is influenced by political and company-wide strategies and not only based on technology and standards. Following elements have to be provided by the software components model:

- **Services**: Each related component is associated to one or more services. A service can be seen as OBSD component when it is encapsulated enough to allow a reuse.
- **Software Components**: Each component within one or more designated domain areas needs to be specified. This does not necessarily mean that each class within the component is reused, but for reasoning a reusable component, the semantic mediation requires to have knowledge of the complete product/application portfolio.
- **Payload**: The physical content provided via a service and process by a software component including their attributes. Often described within a standardized data model.

Subject	***Predicate***	***Object***
Aircraft	*executes*	*Flight*
Departure	*begins*	*Flight*
Arrival	*ends*	*Flight*
Departure	*preceeds*	*Arrival*
Arrival	*follows*	*Departure*
Arrival	*occursAt*	*Aerodrome*
Aerodrom	*isLocationOf*	*Arrival*
Departure	*occursAt*	*Aerodrome*
Aerodrome	*isLocationOf*	*Departure*

Table 4.3: RDF/OWL statements of an imported AIRM class .

The software components model is designed in a domain independent way, supporting a usage across various domains. This persuasive mechanism provides an agile way for the reuse of code domain independent. The reusable software ontologies are merged together with the OBSD methodology into the so called semantic description (cf. section 4.5). The elements in the software components model are described in an conceptual way to enable the probability of the use of the reusable software ontologies of the model in different domains (e.g., an instance of a IWXXM/WXXM data mapper component can either be used in a weather related or an aeronautical project). Since the software components model contains a considerable number of concepts, it is useful to group these concepts into functional elements. The following subsections describe the functional segments for reusable component concepts, general service and data concepts, software classes, and data elements in a generic way. The partition of the software components model allows better visualization of the components. Each concept in the reusable software ontology is defined using a URI. This URI is used for accessing, querying or filtering a specific component by a rule-set. To improve human legibility of the ontology, the URIs semantically identify the components they are part of, using a human-readable notation. The concepts of this segment of the software components model describe the individual reusable components of the OBSD-based approach.

4.4.2.1 Software Component Ontologies

A software component denotes a software package, a module, or complex service which in itself can have multiple components that encapsulates a set of related data elements or functions cf. [Hamza, 2009] and [AL-Badareen et al., 2011]. For example a service to convert or map data elements, a GIS module, or an aerodrome map package consist of objects which are an instance of a class. Each service has one or more components (e.g. AMDB, DigitalBriefing), but at least one single component is part of the service, even the service itself is one. The ontology-based description of the case study follows the SOA and SWIM principals, whereby a service can be a specific component and subsequently acquire additional characteristics beyond that of a common component.

The advantage of reusable software components is, that they can be replaced as long as the requirements are fulfilled. Another component may use other algorithms/functions or add additional features but as long as the payload exchanged via the interfaces is the same as of the initial component it can be replaced. This allows updating components to support newer standards or using alternative versions if the successor component does not fulfill the needs. This is supported as different ontology versions are possible, find a detailed description in subsection 4.4.3. Software components also refer to one or more specific domain concepts. A domain concept is a semantic representation of the knowledge of a certain domain described in the domain information model. Each software component has to be mapped to one or more appropriate domain concept, in order to allow automatic identification of possible reuse and further use in an IDE. It captures if any legacy applications are required by a specific software component. A legacy application is an isolated application or system that has to be integrated to run the software component. These applications may already communicate in various aspects with other entities out of the scope of the OBSD components, but are identified to provide the full potential of integration. By using OBSD as integration platform, the full potential of the software components

can be unleashed. Every component has a unique name and can provide a service. A logical domain contains a number of components. This domain is used for modeling common physical or logical relations of components. Every logical domain has a unique name and contains at least one component. For more details on the mediation process see section 4.2.3.2.

As described in subsection 3.4.5 IWXXM/WXXM as part of the AIRM was used as baseline for the meteorological ontology of the domain information model. On the counterpart the software components model contains a weather message mapper between different versions of IWXXM/WXXM. This is one of the cases were a software component has a fixed constraint to a domain. Figure 4.5 shows an example of the weather message mapper component which leads to a more complex ontology schema describing a short and long format of weather data. While the short format only contains the weather data as text summary (`Payload_1` includes the class `Segment_Summary`), the long format additionally contains a satellite picture (`Payload_2` includes the classes `Segment_Summary` and `Segment_Photo`). This particular weather information element is a good example how easy overlapping parts can be mixed up. Through an upgrade of the related weather exchange model version the container-file was changed but not the structure of the information itself. The only difference was to store the same information in a XML-file that is standardized. With the OBSD methodology this element can still be identified as reusable with some integration effort needed, in this particular case to convert the message into a XML-file.

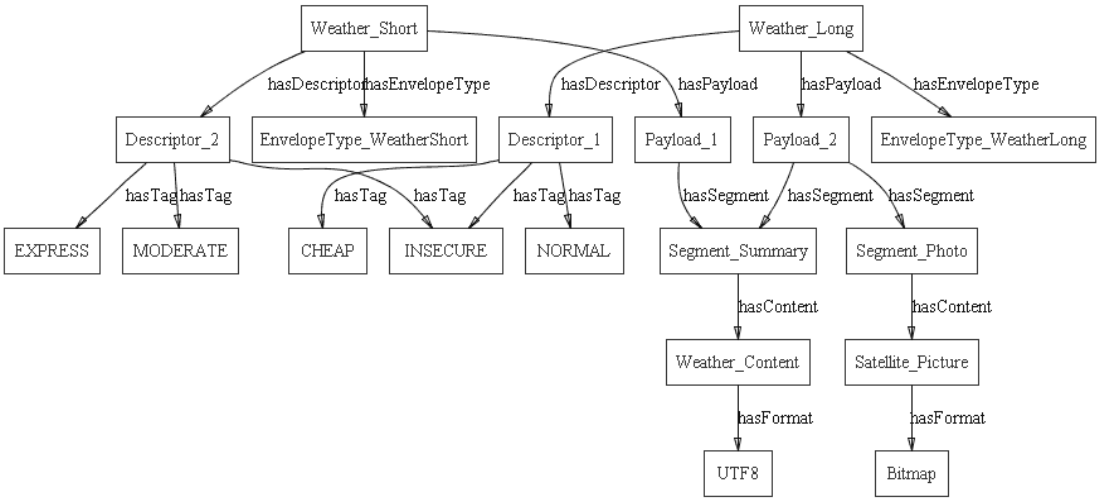

Figure 4.5: Weather data description using ontologies.

4.4.2.2 Services Ontology

The ontology concepts of services described in the software components model inherit the taxonomy of services in a service-oriented architecture manner. The concepts describe the services and also their relations to other ontologies of the software components model. For the case study the description follows the rules that are used to model the ISRM (cf. chapter 3). The concepts for classes and payload are described in the next section. A service is most of the time the communication part and consists of software components, either providing or consuming information elements. Each service is connected to one or many components in the OBSD ontology, and a number of different services may be connected to the same software components. A service provides or consumes a specific type of a message. Additional requirements for the collaboration with a certain service can be defined using a service contract consisting of obligatory attributes, for example the secure transportation of a message type. Furthermore, the type of a service is modeled. In the OBSD ontology four basic types of services are modeled (this can be extended at any time). It is either a provider or a consumer service. The basic types can be modeled as synchronous services (e.g. a service for publication, or a service for subscription). The service description contains a clear and comprehensive definition including a technology agnostic model derived form the ISRM. The ISRM contains a set of SWIM enabled services developed during SESAR which fit perfectly into the service ontology of the software components model. The service definition represents the semantic description which assists the progress of service mediation and consistency checking of an enterprise architecture. In addition the service description covers non-functional requirements like accessibility, availability, compliance, performance, duration, etc. The service specification further consists of a link to the different domain information models to identify what payload the service processes and which information it references that is owned by other services. This information is necessary to select the right reusable components by querying with the specific requirement rules.

4.4.2.3 Payload Ontology

Reusable software components encapsulate data elements and algorithms processing these elements. As objects are created as an instance of the class by the constructor, classes and there instances including the specific object variables and implementations of behavior (attributes, member functions, methods) are modeled. As the idea of reusable components build up on the idea of SOA and related software design patterns, it claims that software components can ultimately be made interchangeable and reliable. The physical payload is important to be described to either link the payload elements to a specific exchange standard (e.g. FIXM, IWXXM/WXXM, AIXM, etc.) when modeled or to state the related domain. The AIXM standard UML metamodel was loaded into Protégé including the model description as UML XMI to support various aeronautical services. For example, service messages are indicated through their operation parameters and the ontologies capture their structure and actual information content. Specifically the defined data type in the various ATM exchange models is a projection of an AIRM class to the single attribute designator. The related AIRM entity is an added part of the domain information model to allow back-tracing. The merge of software components and domain information concepts is one of the most important features that the OBSD methodology offers.

4.4.3 Ontology Versioning

This subsection describes the versioning and refinement of the OBSD ontology models. Since information models and software components evolve over time it is improvement to support the life-cycle of a software product but also various versions of standardized exchange models. Therefore OBSD allows to adapt and refine the modeled ontologies. They are versioned and allow backward compatibility with older versions of OBSD ontologies. This makes maintenance and refinement possible without touching previous models. There are nice side effects, like the possibility for the developers to query for backward compatibility software components which is a not foreseen feature of OBSD (e.g. mapper or parser after a data model update). The example in subsection 4.4.2.1 shows the need of multiple ontology versions for one component. But also other domain information or software components ontologies should exist to support the different needs of customer requirements. Therefore different versions of the knowledge described should be captured by the semantic models. In order to support backward compatibility, the old version of the domain ontology has to be stored. Multiple versions of either the software components model or the domain information model can be consolidated and some may not be compatible with others. A new version may capture statements which are in conflict with old ones. To check the interoperability of the new version all subsets of a single ontology have to be processed manually, which is a time consuming task. It can happen that two versions of an ontology evolve in contrary directions and therefore cannot be consolidated. For the production of the solution model the semantic sub-description is used by the semantic mediation, whereas the versioning effects the OBSD during design-time only (cf. figure 4.1). Inside an `owl:Ontology` element the `owl:versionInfo` tag defines the version of a specific ontology. It is possible to keep different versions for classes, properties, and even for individuals. The tags `owl:incompatibleWith` and `owl:backwardCompatibleWith` show if an ontology is backward compatible or incompatible with another ontology version. It can also be referenced to a prior version through the `owl:priorVersion` tag. For more details have a look into the OWL guide by [Smith et al., 2004]. See an example for OWL versioning below:

```
<owl:Ontology rdf:about="">
  <owl:versionInfo>3.0</owl:versionInfo>
  <rdfs:comment>Weather Domain Ontology</rdfs:comment>
  <owl:priorVersion rdf:resource="OBSD/DomainModel/Weather\_2.3"/>
  <owl:backwardCompatibleWith rdf:resource="OBSD/DomainModel/Weather\_2.0"/>
  <owl:incompatibleWith rdf:resource="OBSD/DomainModel/Weather\_1.9"/>
</owl:Ontology>
```

Algorithm 4.1: Example for OWL versioning.

One important feature of versioning is to deprecate a class, a property, or a concept by using the affiliated tags `owl:DeprecatedClass`, `owl:DeprecatedProperty`, or the `owl:DeprecatedConcept` tag. This is needed when new or updated elements are modeled and the related or old ones have to exist further to allow backward compatibility. A domain expert or software architect can then identify the need of mapping individuals of the existing concepts to the new ones for upcoming release.

With Protégé or any other semantic editors the compatibility of various versions can be

```
<owl:Ontology rdf:about="">
  <owl:versionInfo>1.3</owl:versionInfo>
  <rdfs:comment>Airport Ontology</rdfs:comment>
  <owl:backwardCompatibleWith rdf:resource="OBSD/DomainModel/airport\_1.2"/>
  <owl:priorVersion rdf:resource="OBSD/DomainModel/airport\_1.2"/>
</owl:Ontology>

<owl:DeprecatedClass rdf:ID="Airport">
  <rdfs:comment>Changed to Aerodrome consisting an Airport & Heliport</rdfs:comment>
  <owl:equivalentClass rdf:resource="#Aerodrome"/>
</owl:DeprecatedClass>

<owl:Class rdf:ID="Aerodrome"/>

<owl:DeprecatedProperty rdf:ID="Runways">
  <rdfs:comment>inverse property hasRunway is now preferred</rdfs:comment>
  <owl:inverseOf rdf:resource="#hasRunway"/>
</owl:DeprecatedProperty>

<owl:ObjectProperty rdf:ID="hasRunways"/>
```

Algorithm 4.2: Example for deprecation in OWL.

checked during design-time. The sub-description with the current ontology version is loaded and replaced with the new version in the editor. Protégé shows any issues or problems which may occur in case of nonexistent concepts, mismatched individuals, or other incompatibilities. If this is the case the responsible expert shall update the model accordingly to assure compatibility, either between the old and the new version in case the old one is used to trace the backward compatibility, or to release a new major version.

4.4.4 Ontology Refinement

The deletion of redundancies or other anomalies is the main goal of ontology refinement. The aim is to reduce deficiencies without changing the semantics, which means that the captured knowledge shall be the same but the design should be better. This task takes place at OBSD design-time and is performed if a new version of an ontology is introduced. Improvements and better design can be discovered with semi-automatic processes executed in the semantic editor, in review sessions of the expert board, or during reviews of the ontologies by the corresponding expert. A refined ontology gets versioned even though it is just a change of the syntax. Unused classes are the best example of elements that are adjusted during the ontology refinement. Such classes occur after merging ontologies that may capture unused elements cf. [Baumeister and Seipel, 2010]. If no rule, policy, etc. uses this class and it has no inheritances, it may is unused. This can also happen when either a more specific definition or, on the other hand a more generic concepts, is merged. To identify if the specific class is the leaf in the hierarchy, Protégé is used. If the class has no individuals, it can be deleted as such within the semantic editor. Since unused classes come to light after a merge of ontologies, the refinement task takes place after the semantic sub-description. The elimination of such classes should be proceeded with proper responsibility of the other ontologies, which are aligned with the ontology to be refined.

As outcome of the ontology integration task, isolated and disarranged classes and other

anomalies can appear cf. [Baumeister and Seipel, 2010]. Such classes also appear after manual adjustment of ontologies (i.e. displacing the class into another stream without the consecutive rework of the disjoint relations), especially if the class itself is not disjoint with any of its siblings, but has disarranged relations to a set of classes that are common relatives in another trees of the semantic description. Like unused classes, isolated and disarranged classes can be discovered with Protégé in a manual way. To resolve the problem, the disarranged property shall be removed from the isolated class.

Overdefined properties are another issue that occurs during the ontology refinement task. The data modeler or corresponding expert may define a weather related value in the property of a class too specific (i.e. `WindSpeed = {0,1,2,3,4,5,6,7,8,9,10,11,12}` according to the Beaufort scale). While creating the semantic description it turns out that it is not practicable and a more flexible way would be `WindSpeed = {speed, speedVariesFrom, speedVariesTo}` as specified by WXXM. Global rules can be defined to identify properties containing such design values and highlight them. Generic value groups which define value ranges, can be designed to be reused in other concepts. Another way to prevent such misdesigns is to use the common value classes defined in the AIRM which already delivers well structured generic values, which then can be reused.

Repetitious or unnecessary relations arise when overlapping parts are integrated which have redundant information. If a segregation is only possible by the definition name itself, a class or rule is identical. These issues are either human made or be part of a merging process of ontologies. The refinement task has specific rules implemented to highlight them for manual inspection. Anomalous inheritances may nest in a series of degenerated concepts which are just referencing themselves but are not used elsewhere $C_1\ is{-}a\ C_2\ is{-}a\ ...\ is{-}a\ C_n$ cf. [Baumeister and Seipel, 2010]. If this is the case, those hierarchical series most of the time can be deleted by decrease the intermediate concepts, which are not needed.

4.4.5 Ontology Consistency

During the ontology modeling phase several checks regarding completeness and consistency are performed. Most of the classification and consistency tests are accomplished with the Protégé plug-in Hermit (cf. subsection 2.4.3.4). As shown in section 4.3 the consistency is tested after a specific ontology is built and after the calculation of the semantic sub-description. The feedback of the checks performed is either message that everything is okay, or an error message about minor problems in form of a warning or a problem state if serious failures occurred (i.e. redundant definitions of software components, missing labels, etc.). Those have to be fixed otherwise the ontology mediation process and selection of possible reusable software components will not work. The ontology consistency task shall be performed after creation of the domain information model and the software components model by the corresponding owners, before and after the refinement task, and after the semantic sub-description is generated. Structural errors like, invalid cardinalities, invalid references, erratum and validation of the ontology are performed by the semantic editor automatically. There are also logical checks in place to analyze if all software component instances are mapped to a domain concept and all data elements are described. Those checks are performed by the various experts manually in the semantic editor. The logical checks are also applied during the initialization phase of the semantic interface (cf. 4.5.3).

This ensures a logical persistent of the semantic sub-description and allows accessing it via the semantic interface. After the refinement task, the captured meaning is checked before the old and new ontology are tested. Both ontologies are considered as equal in meaning if the accessed information via the semantic interface is the same. Moreover the consistency tests integrated in the semantic interface can be extended at any time if needed.

4.5 Ontology Mediation

Reasoning, semantic mapping and semantic description transformation are the main parts of the ontology mediation process. Since the domain information model is based on the AIRM and the payload described in the software component model is primarily based on standardized ATM exchange models, the mediation process between those two knowledge basements is a reachable task. UML class models, in the AIRM or one of the ATM exchange models, follow the closed world assumption which inherits that all statements have not mentioned explicitly are false. Ontologies follow the opposite way of the open world assumption where these statements are undecidable cf. [Baclawski et al., 2002]. These different semantic strategies bring up the need to add multiple restrictions to the semantic description during the mediation process to keep the authentic semantics of the respective model. A often claimed issue with RDF/OWL is that both are not able to express the described data mediation rules cf. [Kai and Steele, 2009] and [Bleiholder and Naumann, 2008]. For example there is an issue when two divergent feature type properties are matched to another target feature type property. During the semantic match the original properties might need special mediation rules to convert them correctly into the new property.

```
Source Property 1              ->             Target Property 3
(Pressure in Pascal) (Conversion Factor A)   (Pressure in Bar)
Source Property 2              ->             Target Property 4
(Pressure in Torr)   (Conversion Factor B)   (Pressure in Bar)
```

Non trivial mappings are conditional `if-then-else` statements. The important question is how much such be automatically transformed by the semantic mediator and how much should be developed as a rule to define an appropriate transformation. There are several rule languages available, some of them described in subsection 2.4.1 like SWRL or SQWRL. Semantic mediation can be achieved by multiple ways. Within this thesis ontology mapping and transformation rules were considered. Ontology mapping is a semantic information process were the knowledge base is used to map the ontology models in the domain information model to another ontology model in the software components model. The ontology mediation task transforms the information encapsulated in the knowledge base cf. [Bruijn et al., 2006] and [Ehrig, 2010] to create a selected semantic sub-description. If there is a semantic match between classes the ontology mediation task assigns an opportune transformation based on the semantics and the ontological definition (cf. OWLs `owl:equivalentClass`, `owl:equivalentProperty` or `owl:sameAs`). Since mapping languages insufficiently express all imaginable transformations a lot of manual work needs to be done to cover everything needed cf. [Ehrig, 2010].

In the mediation task the OBSD models are merged into the semantic description by reasoner tools. A serious problem with ontologies is to develop different types, which in the end cannot be merged because of semantic and structural incompatibility. A workaround is to insert links between the types as an indicator of incompatibility. To avoid this possible risk, existing semantic methodologies in the OBSD environment were further evolved. OBSD is structured in different phases to implement the goals described above (cf. subsection 4.2.3). These areas reflect the usual approach of an integration project by first describing requirements, afterward developing a solution model that meets the demands. Figure 4.1 gives an overview of the main components of OBSD and shows the split into three different steps. The relationships between the semantic models are merged together in the design phase [Bleiholder and Naumann, 2008], [Kai and Steele, 2009]. Based on the match of different ontologies, a semantic description is developed. In Step 2 tools for the production of the semantic description are used, such as reasoners, alignment tools or visual representations. The fact that relationships in OWL are formally defined, offers the possibility to use a reasoner. One main service that such a reasoning system can provide is to test whether or not one class is a subclass of another class such as shown in figure 3.6. `:AeronauticalInformationEvent` has the subclass `:Notam`. This relationship is called a necessary implication. So we can conclude that because `:RunwayClosure` is some sort of `:Notam`, and all types of :Notam are :AeronauticalInformationEvent, then a :RunwayClosure is also a type of `:AeronauticalInformationEvent`. A reasoner can show that the class of NOTAM is a valid subclass of aeronautical information events, 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. Without any instance you can properly conclude that a class is inconsistent. Protégé enables to take advantage of different OWL reasoners as a plug-in. This all sounds great in theory, 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. Within the OBSD project the capabilities of some state-of-the-art reasoners, which support OWL were also evaluated in chapter 2. In addition, FACT++ and Pellet are selected for reasoning. The decision was based on the fact that both are compatible with OWL and are well supported. For the developer it is also necessary to select visualization tools, which support OWL. Visualization often deals with abstract data and offers a bundle of techniques to represent hierarchical or semi-structured data. There are several studies where different ontology visualization tools are compared [Catenazzi et al., 2009], [Lanzenberger et al., 2009b]. 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 was picked as the semantic editor, the visualization tools OWLViz in combination with OWLdiff, Matrix and Cloud View complete the list of tools, which are used within the OBSD.

4.5.1 Semantic Sub-Description

Software products built with the OBSD methodology typically use a number of data elements conducted from different domains offered by a composition of several organizations. As described in section 3.3, System Wide Information Management (SWIM) enabled software products are mostly non trivial integration projects. First of all, the data provided from different

domains need to have a fundamental baseline where overlapping parts are matched and elements are linked to each other. OBSD therefore uses the SESAR AIRM (cf. subsection 3.4.1) as baseline of harmonized data elements covering various domains like weather, aeronautical, flight or environmental related data. In addition, OBSD needs another semantic base for common software components that are identified to be reusable. To prepare the now available knowledge, a set of rules have to be identified, which can than be used by the semantic mediation to produce a so called semantic description. After processing a consolidated list of information exchange requirements coming from the operational and business context, those rules can be modeled. The created joint knowledge base is called semantic sub-description, as it represents a part of the overall semantic description defined by the participating software components, data domains and the operational requirements. The semantic sub-description contains all ontology concepts from the reusable software model, domain model, queried by the operational requirement rules, which are needed to specify the OBSD-based semantic solution. For the creation of a semantic sub-description the needed processes and methods are filtering by the requirements. Figure 4.6 presents a simplified example of the OBSD semantic sub-description.

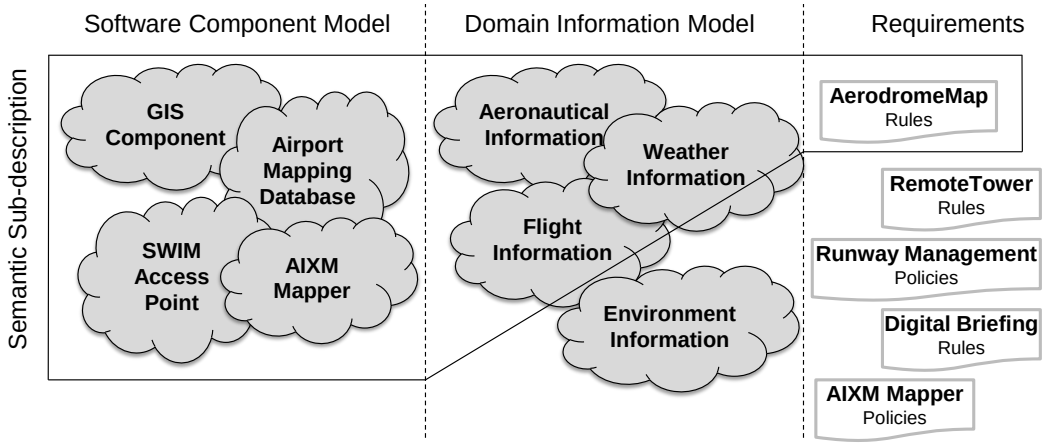

Figure 4.6: Definition of a semantic sub-description in OBSD.

For the creation of the semantic sub-description of an aerodrome map product, the particular ontologies, a set of the reusable software ontology and at least one domain ontology are chosen and aligned in order to form the semantic sub-description. A number of operational and business rules is then querying to filter the right components for software reuse. If more than one domain ontology is needed, all included domain ontologies need to be aligned a priori. As a first step, the domain and the reusable software model are aligned, forming a knowledge base for the OBSD-based development for a specific product. Aligning ontologies is defined as the integration of the contained concepts and individuals. It is a trivial case if there is no concept or individual contained in more than one ontology, Than it is just a simple extension of all concepts and individuals. A nontrivial case for example, is the circumstance if there exist either semantically or textually identical concepts or individuals in the ontologies to be aligned. This needs human intervention for successfully solving this specific alignment problem. In this case a domain

expert and a software architect, responsible for the specific components, need either to specify the semantic context of a concept or individual (e.g., in one ontology a concept “plane” exists, in another ontology the same concept is named “aircraft”) or to rename the concept or individual to be added. Using the AIRM as a baseline for the domain ontologies minimizes the effort at least for the intra-alignment of the domain models. Within SESAR a lot of effort was spent to harmonize the domain models and form the ATM Information Reference Model. Then, the operational rules query the concepts and individuals to successively build the semantic sub-description, resulting in the OBSD semantic sub-description for an AMDB application. Since the necessary adaptation of an ontology for an AMDB semantic sub-description may have an impact on semantic sub-descriptions of other selections, a new version of the adapted ontology is created. The OBSD expert is responsible for the specific components, and the domain expert, owner of the domain ontology, need to perform additional consistency checks after the alignment of every single ontology. In case of inconsistent or incompatible concepts in any ontology, the particular ontologies need to be adapted to achieve a consistent semantic sub-description for the consortium.

4.5.2 Rules and Policies

The ontology mediation uses the instructions defined in rules and policies to form the solution model. A policy formulates restrictions for the semantic description or sub-description, in order to find the best results. Global policies contain specific property value settings (e.g. components which are just using open source libraries, maximum transformation time of a mapper, etc.) whereas optimization policies apply when multiple components are found during the ontology mediation task to provide the software developer a ranking of the components (e.g. sorted by integration time, requirements that are covered by the component, etc.). Global policies are implemented by the domain expert and are part of the domain information model. During the matching process those global policies are taken into account. More specific policies can focus just on a semantic sub-description or just specific components and are defined by the software component expert (e.g. aeronautical service which support AIXM 5.1). Since all the software capabilities shall be modeled in the software component model, the customers have to define their policies as a set of requirements for future products. After the matching process, the reusable components are selected based on the set of customer policies. For each component, the semantic interface proves consistency of the customer policies during the ontology matching. For the matching process, specific methods are implemented by the semantic interface (cf. subsection 4.5.3). New methods can be implemented at any time in order to support the matching process. New policies are easy to add since OBSD defines policies only as textual data in form of name/value pairs and the fulfilling condition. As a result it is possible to break down more complex requirements into a set of textual name/value pairs and the condition the value has to satisfy. They also have to follow the structure of the software components model (cf. subsection 4.4.2). The domain expert together with the software expert select the semantic sub-description of which the solution model is calculated by the ontology mediation process. During the generation of the list of reusable software components, it is controlled if the component capabilities and the component elements meet the requirements of the customer.

Rules specify logical dependencies for reasoning in the captured knowledge of the semantic models. Within this thesis, besides the ontologies defined in the models and the policies described above, a set of rules are used in the transformation process to form the solution model. A rule is defined by a header and a body. Following the first order logic, an atom is an atomic statement in form of a triple. By mapping those atoms "new" knowledge can be gained. If the entity `Runway` is part of an entity `Aerodrome` and has a service called `RunwayManagement` it can be assumed that `Aerodrome` also has a service `RunwayManagement`:

```
(Runway isPartOf Aerodrome
 Runway hasService RunwayManagement)
(Aerodrome hasService RunwayManagement)
```

Since the OBSD ontologies are written in OWL and store the knowledge as triples, rule languages are used to add value to the captured semantics to infer new explicit facts. Upfront the rules need to be defined for the domain information model and the software components model to derive new fact statements. The OBSD semantic interface supports SWRL as rule language and allows to define SQWRL queries (cf. subsection 2.4.1.5 and 2.4.1.6). At a later stage SPARQL 1.1 was added, which offers direct OWL support while using SPARQL queries (cf. subsection 2.4.1.7). Both rule languages share the basic split between body and header but are different when it comes to the expressiveness and structure of their syntax. Within Protégé SWRL is handled with the SWRLTab and SQWRL queries can be defined, edited and executed in the SQWRLQueryTab. With Snap-SPARQL there is also a plug-in available for Protégé to process SPARQL. HermiT was used to apply the rules. For more details about different reasoners have a look at subsection 2.4.3. The reasoning process can cover the following functionalities, some of them are implemented by the semantic interface, but could be covered by reasoning techniques as well (cf. subsection 4.5.3):

- Analyze ontologies regarding structural errors.
- Detect redundant information.
- Check logical and functional integrity by processing the customer policies.
- Clarify data integrity, same message types processed within a component as specified by a customer (linked by the domain information model).
- Prove if components are not used by any service.
- Identify possible software components for reuse. This process can lead to non trivial and complex rules and is therefore part of the semantic interface implementation (i.e. the reusability check often needs various objectives for the valid identification).
- Search for errors in the solution model. This process is covered during the initialize phase of the semantic interface by the consistency constraints whereas checks are defined as XML files.

- Discover missing information in the semantic description through individual consistency checks and appropriate rules (e.g. missing services/components/elements/definitions, etc).

As described in subsection 4.3.4, the domain information expert implements the semantic concepts during design-time of the respective domain as ontology of the domain information model. Possible reusable components are described in the software components model as ontologies. At the same time customer requirements are captured as policies and rules to identify the recyclable artifacts. Each element of every component is mapped to an ontology of the domain information model. The whole process is supervised by the domain information expert to reveal inconsistencies. The mapping between the domain knowledge and the implemented standards formalized by international information exchange models allow the selection of semantically indistinguishable information. Former issues like different naming conventions or structural differences of the elements are cleared up, if they match together to the domain information model. This functionality allows the mapping of product requirements from other domains which use the same data (which may have different definition names). Most notably, this feature supports the domain independent identification of recyclable components, as the same aeronautical, meteorological, and flight information is used in different domains in various ways.

4.5.3 Semantic Interface

The semantic interface grants access to the semantic description where the mapped information is stored to ensure that the solution model is decoupled from the low level semantic syntax. It parses the semantic concepts of the ontology and provides the queried information as objects to the solution model. The semantic interface is used by the semantic mediation process for querying information about the reusable components to calculate the solution model for a certain OBSD project. In addition the semantic interface supports this process with information about the components, elements, policies, and global rules. The selected semantic sub-description can than be transformed by the mediation process to form the solution model. The semantic interface connects the knowledge contained in the semantic sub-description and the solution model which can be used within the development process to reuse the selected components. As input for the semantic interface a set of configurations, defining the ontologies of the domain information model and the software components model including the rules of the selected semantic sub-description, is needed for a certain OBSD-based development project. The interface than maps the selected ontologies to form the semantic sub-description. The consistency checks already mentioned are performed during the initialization process of the semantic interface.

4.5.3.1 Semantic Interface Architecture

The OBSD semantic interface architecture is split into two phases. During the initial phase, the semantic sub-description is generated and structured. In the second phase the solution model can be used to retrieve the stored knowledge to support the run-time deployment phase as foreseen by the OBSD methodology. Therefore the semantic interface architecture provides methods, which can be utilized to obtain the semantic information, later on transformed into the solution model.

First of all the semantic sub-description is defined with a set of configurations. The configuration of the semantic sub-description is done via a simple XML-file. Each of the selected ontologies are matched together into the semantic sub-description. After all ontologies are loaded without any errors the semantic sub-description represents the selection which can now be queried. The rules and policies are specified as a set of SPARQL rules (cf. subsection 2.4.1.7).

```
<?xml version="1.1"?>
<configuration>
  <domainInformationModel>
    <domainOntology>aeronautics.owl</domainOntology>
    <domainOntology>meteorology.owl</domainOntology>
    <domainOntology>flight.owl</domainOntology>
  </domainInformationModel>
  <softwareComponentsModel>
    <softwareOntology>digitalBriefing.owl</softwareOntology>
    <softwareOntology>runwayManagement.owl</softwareOntology>
    <softwareOntology>aerodromeMap.owl</softwareOntology>
    <softwareOntology>aeronauticalServices.owl</softwareOntology>
  </softwareComponentsModel>
  <rules>
    <rule>globalRules.txt</rule>
    <rule>aeronauticalRules.txt</rule>
  </rules>
  <policies>
  <policy>globalPolicies.txt</rule>
    <policy>aerodromeMap_ACG.txt</rule>
  </policies>
  <consistencyChecks>
    <consistencyCheck>commonChecks.xml</consistencyChecks>
<consistencyCheck>aeronauticalChecks.xml</consistencyChecks>
  </consistencyChecks>
</configuration>
```

Algorithm 4.3: Configuration of the semantic sub-description

The next step during the initialization phase is to perform consistency checks which are defined in the configuration file. Supplementary consistency checks can be added and defined as external XML files if needed to be executed while initiating the semantic interface (cf. subsection 4.5.2). Separate sets of mandatory consistency checks can be defined in different configuration files. In case a consistency check fails either a warning or error message is displayed and logged. The semantic sub-description is still available via the semantic interface if a failed consistency check runs into a warning, just an appropriate message is thrown. For safety reasons, if an error exception is shown, the specified semantic sub-description cannot be constructed using the semantic interface. This safety procedure prevents the calculation of an incomplete or invalid solution model. The information captured in the semantic sub-description concepts is than loaded into the objects which are used by the semantic interface. Since the relationship between the concepts is non trivial, the order of loading the sequence is important to reduce inessential information.

Finally the calculation process of possible software components that could be reused is

started. The semantic interface is ready to use, after the entire knowledge from the semantic sub-description has been loaded. During run-time, the semantic interface offers read-only methods for accessing the solution model. The dynamic calculated information, like reusable software components or services, is too case specific to be saved for further instances of the semantic interface. Other things like domain or sub-domain groupings are stored beyond the lifetime of an individual case and do not need to be calculated for every semantic interface instance. The developed methods of the semantic interface are separated in ones related to the software components (incl. services and component elements) and ones dealing with the domain information, standards, and operational concepts. The methods are used to retrieve information about the sub-description knowledge, the most important ones are listed below:

- `List<Component> getAllComponents()` returns all existing components. An equivalent method exists for services `List<Service> getAllServices()`, to retrieve the components of one `List<Component> getServiceComponents()` can be used or list the attributes of them with `List<Attribute> getAttributes()`.
- `List<Interface> getServiceInterfaces(Service)` to call for a service interfaces. Specific methods like `List<Service> getAllConsumerServices()` or `List<Service> getAllProviderServices()` provide lists of all services that require input (consumer services) or produce information (provider services) or both.
- The method `List<Component> getResuableComponents(Components)` and `List<Service> getResuableServices(Service)` return all possible components that match the needs of the given requirements, fulfill the policies and rules applied, and are part of the semantic sub-description.
- `List<Domain> getAllDomains()` returns a list of all domains used by a component or service. The same method, `List<Sub-Domain> getSubDomains()`, exists to extract the used sub-domains.
- `getAllComponents4Domain(Domain)` to receive all components which belong to a specific domain this method can be called.
- The method `List<Concept> getOperationalConcepts()` lists all used operational concepts, whereas `List<Standards> getStandards()` shows the list of standards used.

Object descriptions are filled with the information coming from the semantic sub-description to form the solution model. They are grouped in the same way as the semantic interface methods. For example the `component` object is defined by the following methods:

- `String getID()`: provides the ID of the component.
- `String getName()`: returns the name of the component.
- `Product getProduct()`: returns the application/product the component is part of.

- `Map<String,Object> getAttributes()`: returns a map of the attributes of the component. An attribute is defined as a non-Boolean characteristic of a component (e.g. the delay of the component). These attributes are used during matching for the identification of possible reusability.

- `List<Standard> getStandards()`: returns a list of the standards the component supports.

- `List<Domain> getDomains()`: returns a list of all domains of the component.

- `List<SubDomain> getSubDomains(Domain)`: returns the actual sub-domain of the component for a particular domain the component is part of.

A `service` object is defined through the following methods:

- `String getID()`: provides the ID of the service.

- `String getName()`: returns the name of the service.

- `String getType()`: returns the type of the service. A service can either be a provider service, a consumer service, or both.

- `Product getProduct()`: returns the application/product the service is part of.

- `Map<String,Object> getAttributes()`: returns a map of the attributes the service has. An attribute is defined as a non-Boolean characteristic of a service (e.g. the delay of the service). These attributes are used during the matching for the identification of possible reusability.

- `Message getMessage()`: returns the message either provided or consumed by the service.

- `Boolean isRequestOnDemand()`: returns true, if the service uses request/reply style communication. Otherwise, it returns false.

- `Boolean isPubSub()`: returns true, if the service uses the publish-subscribe pattern. Otherwise, it returns false.

- `List<Component> getAllComponents()`: returns a list of all components that initialize this service.

The following list describes the methods defined for a `message` object:

- `String getID()`: returns the ID of the message.
- `String getName()`: returns the name of the message.
- `List<Element> getElements()`: returns a list of all element of the message.

The bullet-points listed below describe the methods defined for a `payload` object:

- `String getID()`: returns the ID of the payload.
- `String getName()`: returns the name of the payload.

The `domain` and `subdomain` object is defined by the following methods:

- `String getID()`: returns the ID of the (sub)domain.
- `List<Component> getComponents()`: returns a list of all components that are part of the (sub)domain.
- `List<Service> getServices()`: returns a list of all services that are part of the (sub)domain.

Below the methods are listed, which define the operational `concept` object:

- `String getID()`: returns the ID of the concept.
- `String getName()`: returns the name of the concept.
- `List<Component> getComponents()`: returns a list of all components that are part of the concept.
- `List<Service> getServices()`: returns a list of all services that are part of the concept.

4.5.3.2 Semantic Interface Querying

In this subsection the identification of reusable components using the semantic interface is described. For every requested operational scenario, a set of possible components and services providing the requirements is queried. The list, selected from the software component model, contains all the artifacts that are calculated to be recyclable within a (new) software project. Within the OBSD life-cycle the QA process allows the OBSD modeler and architect to review the calculations for improvement. In a stable setup the developer can use the solution model to choose from the reusable component repository, which is plugged in into the IDE. The whole querying process is implemented as a heuristic algorithm to improve the quality of the results. To minor the selection of the sub-description the identification process needs to narrow down the list of possible reusable candidates. This can be applied by the following rules:

- Find all components that match all requirements, with the component attributes consistent with the applied rules and policies.

- If a component is mapped to the same domain and fits the requirements except the output format, check whether there is a data mapper available.
- If the component differs regarding the domain concept of the requirements, check if there exists a component mapped to the same domain concept as requested and suggest a data mediator to be reused if possible.
- Check if there are existing components within a service that are mandatory.
- Check if there are existing services which could be used instead of multiple components to meet the requirements.
- If all the rules mentioned above are successfully applied to a set of one or more services and components, the particular set is suggested as recyclable candidates.
- Define the elements and attributes of each selected component. For each component ID of every component of every single service, the mapped domain concept and properties are queried using the semantic interface.
- The same is done for each service selected as reusable.
- The selected mapper and mediator components have special dependencies with other components (i.e. some of them won't work without them).

During the initialization phase of the semantic interface, the relationship between all components in the model and the ones in the repository, which are the actual code artifacts, are checked. The calculated and selected components needs to be accessible via the repository to allow working with them at run-time. To ensure this the links between the software components model and the reusable software component repository are checked upfront. The chosen code artifacts are then provided via the IDE plug-in.

4.5.3.3 Monitoring

The monitoring process captures how the results of the solution model are actually used for both the simulation and the real software development integration. It measures which components are used and how much effort was needed for the integration. The collected feedback from the OBSD monitoring process is precious for the OBSD modeler and architect to improving the quality of the entire OBSD life-cycle (e.g the component documentation, bugs, technical performance, etc.). During run-time the software development system is monitored and the experts in charge take into account this feedback to implement enhancements and adjustments into the semantic description. The assumptions made in the semantic description are compared with the recordings of the monitoring process to change, based on those findings, the configuration of the mediation process. In addition an improved solution model can be deployed for the next iteration of simulation and deployment to the real environment. The processing of those results through the experts are not only input for the semantic description but evolve and improve the whole OBSD life-cycle.

4.6 Conclusion

In the traditional development process, software developers have just a partial overview of the entire system and may optimize their applications locally (e.g. recycling of components is restricted to a department, etc.). This can lead to redundant implementations, higher integration efforts and separated developments which in the end cannot talk to each other even they use the same standards. This chapter introduced the OBSD design-time approach, which offers the possibility to get a complete overview of all software components available and improves the collaboration between the different development teams. This includes optimizations of the entire development life-cycle of a company. The OBSD life-cycle is designed in a way that allows the installation of several QA checkpoints, the usage of monitoring components and making improvements in the model descriptions by using quality feedback methods offered by OBSD. Therefore OBSD allows concentrating on the technical benefits with a few experts only to keep the whole integration effort as low as possible (sometimes political issues will play a role as well). This becomes a particular advantage with an increasing number of components captured by the OBSD methodology. Quality and effort improvements that result from multiple usage affect not just the quality of OBSD system itself, but also improve OBSD based products and applications. In addition the methodology can also be used during new offers to estimate the development effort of integration projects.

Through appropriate verification checks, the issues and errors captured in the semantic models can be kept to a minimum. The role-oriented abstraction is an advantage that OBSD models support and allows domain and software experts to understand the models more easily. This also leads to simplified error correction. At present, the semantic mediation can handle semantic policies and rules that are currently known. A solution option for implementing policies will grant the input of policies directly from the IDE plug-in. In future implementations of the OBSD cycle it is foreseen that these policies can be implemented via a specific interface and would therefore allow an easier integration into the QA steps. Another benefit is the quality improvement of the components themselves. With the option to keep component versions and the OBSD goal to reuse components as much as possible, code quality improves. To better understand possible trade-offs, evaluations comparing the classic development processes and the OBSD methodology are analyzed in the case study. In addition it is planned that the IDE integration through the semantic interface will be further developed to provide more features and make it even easier to use for software developers. For the case study, presented in the next chapter, it was important to have a monitoring task in place to capture feedback and draw assumptions about the effects of the OBSD methodology.

CHAPTER 5

Case Study Evaluation and Analysis

"Semiotics is in principle the discipline studying everything which can be used in order to lie. If something cannot be used to tell a lie, conversely it cannot be used to tell the truth: it cannot in fact be used 'to tell' at all."

[Eco, 1957]

The main goal of the semantic web was and still is to handle huge amounts of information and to create a web of data. The popularity brought up new techniques how to handle knowledge and information more efficiently in the World Wide Web. The processing of information through ontologies was adopted from the semantic web to many other domains including the software development domain. This led to the idea of Ontology-Based Software Development. As stated in the prologue, the motivation of this thesis is to show the benefits an ontology-based life-cycle can offer in a real business context. The crucial factor for such a methodology in order to be used successfully is, how much effort in terms of integration and development is saved. This chapter presents the evaluation and analysis of the case study results of two industrial software development projects using OBSD. In the introduction the related work is explained. In the following section the case study is explained in detail. The study is conducted within the agile development of two products while underlying standards and data exchange models have changed. The Extreme Programming Evaluation Framework (XP-EF) was used for comparing software development processes. For this purpose the old and the new release, using OBSD, were evaluated. Careful and meticulous planning of the case study is essential to gain scientific relevant and meaningful data. Therefore the context factors and qualitative and quantitative adherence metrics information was captured twice. The outcome metrics gave a detailed comparison about positive and negative effects between the old process and the OBSD concept.

5.1 Introduction

Of utmost importance for the higher management is the question about the benefits of the OBSD concept. The analysis of the case study shall provide output measurements of the OBSD lifecycle to evaluate the effects. This chapter focuses on the results of two projects evaluated with the Extreme Programming Evaluation Framework (XP-EF), which was introduced in subsection 1.5.2. The XP-EF allows analyzing any agile software development process. A comprehensive evaluation was accomplished within the case study. The provided benchmark metrics of the framework are split into the context factors, the adherence metrics, and the outcome metrics cf. [Layman et al., 2004b]. The development projects chosen are both new implementations of existing legacy products and tailored to the needs and the environment of the customer (cf. section 3.3). The first project selected, is the Aerodrome Map Information Service (AMIS) dealing with aeronautical data in the AIXM 5.1 format and the prior version 4.5. The existing legacy product is just able to process AIXM 4.5 which was used as reference baseline for the evaluation. The new implemented AMIS is a SWIM service and is as such more than predestined for the purpose of reuse. It houses components such as the Airport Mapping DataBase (AMDB), a GIS database or the AIXM exchange model mediator, an AIXM 5.1 to AIXM 4.5 mapper. It has to be noted that this project was pilot project using the OBSD methodology for the first time. For this reason, problems have occurred which were avoided during the follow-up projects. The AMIS project was developed in an agile way and has proven to be very suitable for the evaluation.

The second evaluation was done in the context of an integrated briefing system. More precisely, the prior briefing product used non SWIM compliant data models for weather, aeronautical and flight information. The new generation of this product follows the SOA principles of SWIM and is called Integrated Digital Briefing Service (IDBS). This application enables users to access aggregated data in form of an integrated briefing which is calculated from multiple sources. This non trivial circumstance offers the possibility to reuse different components to please the requirements. The agile software development process of IDBS was evaluated according to the metrics contained in the XP-EF. This project was chosen to be part of the case study in a later stage and as a result the OBSD processes were already more rehearsed than within the first evaluation.

Both projects followed the SWIM concept defined in section 3.3 and are SWIM service implementations within the European Air Traffic Management Enterprise Architecture (EAEA). This is essential to understand the needs and requirements for components that are possible candidates for recycle. The Information Service Reference Model (ISRM) is substantial as input requirement for the deployment of the services (cf. subsection 3.4.2). The model captures the logical shared information via an ATM service and can be used as instruction guide during the implementation of a SWIM related service. The following subsections explain the related work.

5.2 Aerodrome Map Evaluation

In the Air Traffic Management domain, surface routing in the context of aerodrome mapping is a challenging and difficult task. Although the scientific community as well as the industry recognizes the usage of common data models and approaches to be of utmost importance in terms of interoperability. The effort needed in automated data processing, at the time being no global unified approach exists when it comes to aerodrome mapping. In particular, the United States of America and Europe have different approaches. For this reason, an important step towards improved interoperability between air navigation service providers, airspace users, and airports would be the global alignment with respect to surface routing. Under the umbrella of European's SESAR and American's NextGen initiative, there is an attempt in harmonizing the various concepts used. Taking advantage of the service-oriented architecture infrastructure developed in both initiatives over the past few years, the intention is to consolidate and align this topic by means of using common data models as well as standards-based interfaces and technology and create a global service-based implementation of an AMIS. This section describes the developed AMIS, underlying exchange models, identified operations of the service and the verification and validation process. The developed approach clearly proved the harmonization of existing concepts and ensures interoperability resulting in reduced development effort.

Operations at large aerodromes have become a complex combination of many activities being performed by many individuals such as pilots, air traffic controllers, apron controllers, surface vehicle operators, construction and maintenance, emergency and security, commercial and cargo airline, and general and business aviation operations personnel. All of these individuals must work collaboratively to ensure safe and efficient flight operations at the aerodrome. Furthermore, many of these actors require some shared knowledge and awareness of the aerodrome layout. This can be provided through standardized digital aerodrome mapping. More and more aircraft are equipped with on-board systems using airport mapping information in order to display static airport maps in the cockpit or to present airport maps layered with aircraft position information. Plans are now emerging to develop advanced avionics able to support more critical operations related to surface movement such as traffic display, runway alerts and runway incursion prevention, taxi routing and taxi guidance. The use of accurate aerodrome mapping data implies an interchange between data providers and users based on common agreed information standards and services. However, the definitions of aerodrome map services are not part of current industry standards. The standardization of these services is the main goal in order to achieve a common level of service provision and standardization in alignment with the service-oriented approach of NextGen [Luckenbaugh et al., 2007], [FAA, 2010], and SESAR [Gringinger et al., 2011]. It will fulfill the requirements and intended functions related to the collaborative use of aerodrome map data. Consequently, aircraft manufacturers are engaging development and industrialization of systems using aerodrome map databases (e.g.: providing an aircraft's position overlaid on the airport map helps to ease taxi operations for an aircraft). Starting from 2006, the on-board airport navigation system was fully integrated in the cockpit of the new Airbus A380 using aerodrome databases according the Aeronautical Radio, Incorporated (ARINC) 816 [Pschierer and Schiefele, 2007] airborne airport database format derived from EUROCAE ED-119B [EUROCAE, 2011a]. This airborne standard defines a single open encoding format

for airport databases to be loaded into airborne systems.

The next subsection describes the new product, its relation to operational focus areas and the service implementation process. Furthermore, the evaluation results of the AMIS project are presented. The feedback from the evaluation helped to identify issues and they were fixed within the OBSD development process and the underlying OBSD rules were adopted. In order to prove the validity and benefits of the OBSD life-cycle, a second OBSD development is presented that was conducted subsequently.

5.2.1 Related Work

The implemented Aerodrome Map Information Service (AMIS), supports pilots, air traffic controllers, airport personnel, as well as pre-flight planning personnel. AMIS has four sub services:

- AccessFullAMDB Service based on Information Exchange Requirements (IER) of accessing a complete AMDB for a certain airport. This is described as a package of a complete AMDB including all available FeatureTypes (layers). The data payload per FeatureType is XML-encoded accompanied with the meta-data, whilst the package itself is in compressed format (e.g. zip).

- AccessAMDBFeatures Service based on IER of accessing specific features of an AMDB based on multiple filtering capabilities. The service model defined here should enable technical implementation in accordance with OGC standard WFS version 1.1 [OGC, 2002a] and 2.0 [OGC, 2010].

- AccessAMDBGraphics Service based on IER of accessing pre-styled geo-referenced AMDB graphics (bitmap images). The service model defined here should enable technical implementation in accordance with OGC standard Web Map Service (WMS) version 1.1.1 [OGC, 2002b] and 1.3.0 [OGC, 2006].

- Publish/Subscribe Service based on a generic SESAR service, in support of the other three main services.

The product aims to supply aerodrome map information as described by EUROCAE industry standard ED-99C [EUROCAE, 2011b] and ED-119B [EUROCAE, 2011a] to consumers as shown in figure 5.1. The content consists of 40 FeatureTypes covering geospatial information of an airport layout. AMDB data is primarily intended to be used as map layers within various applications. Only the `AccessAMDBFeatures` service was implemented in the first release delivered in March 2014, whilst the `AccessFullAMDB` and `AccessAMDBGraphics` were developed within release version 1.1 in mid 2014.

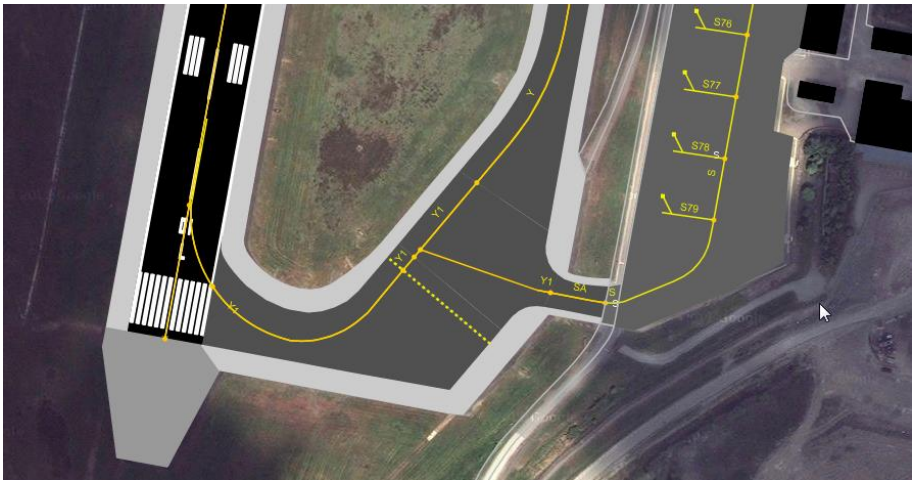

Figure 5.1: AMIS displaying AMDB data over Stockholm Arlanda airport.

For example the `AccessAMDBFeatures` is a service that exposes the individual features within a specific AMDB `FeatureType`. Multiple filter arguments could be used such as aerodrome identifier (ICAO-code), `FeatureTypeName`, identifier, and temporality. The single capability aims to hold the three main types of services.

5.2.1.1 Service Interface Specifications

The AMIS service interface is defined as two interfaces, one of them is exposed as a public interface called `AccessAMDBFeatures` and shown in figure 5.2. The lower interface in the diagram holds the underlying logic for the interface not necessarily of the consumers benefit or interest. This service design aims to simplify the interface for the consumer.

5.2.1.2 Service Dynamic Behavior

The service behavior in this first iteration of the product is basically based upon a request/reply process (cf. figure 5.2). It is foreseen in the diagram below that related replies are transmitted by the service interface. The behavior of the service evolved in a more complex process when additional service interfaces were developed for the service (AccessFullAMDB, AccessAMDB-Graphics).

5.2.1.3 Message Types and Payload

The message types provide requirements for the definition of services as defined in ISRM and EUROCAE ED-99C [EUROCAE, 2011b], additionally a message type is a representation of one or more communications exchanged between originator and addressee [Pola and Solberg, 2013], where the payload is the body of a message that holds the content. As mentioned previously in this section, the mechanism by which a service communicates is through the system's interfaces (Figure 5.2), it means that an interface is a contract between providers and consumers

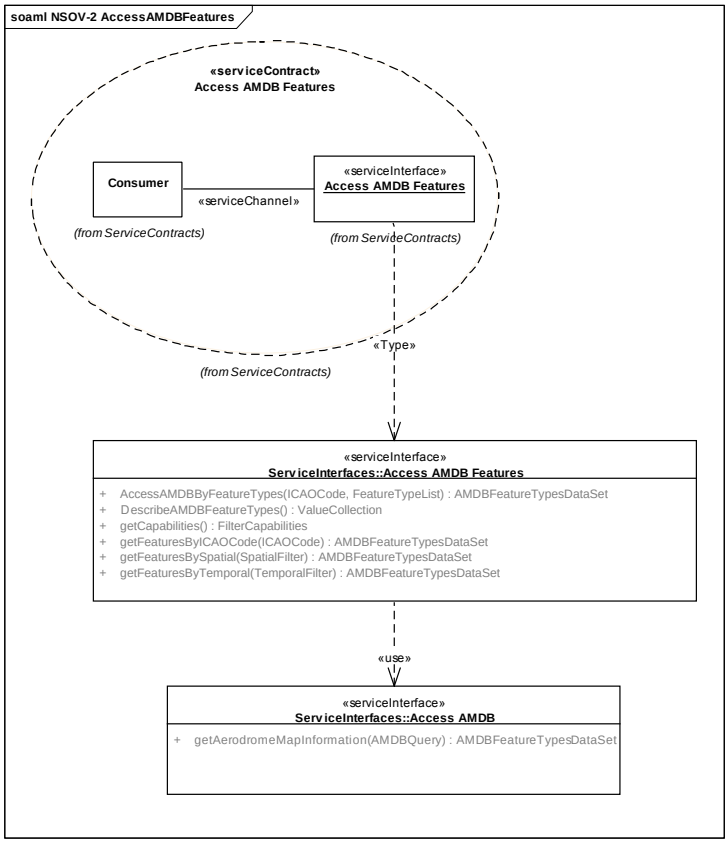

Figure 5.2: Aerodrome map information service public interface.

of services by exchanging information through the message type. The data payload which specifies the access and consolidates the exchange requirements in the AccessAMDBFeature Service is detailed and explicitly defined by EUROCACE ED-119B [EUROCAE, 2011a]. Besides, data payload for functional requirements was identified and consolidated by the SESAR project dealing with aeronautical information ATM services. Those are listed as IER, and some of them are mentioned below:

- Requests in Airport Map with graphical NOTAM.
- Common Aerodrome Map Data which it will be able to have access to consistent Aerodrome Map Information thus enabling situational awareness.
- Display the current status of aerodrome (e.g. a runway closure).
- Access to temporality information for Aerodrome map data (e.g. time filtering variables).
- Indicate Alert information providing information about all runways, description of airport's runway and taxiways, etc.

5.2.1.4 Quality of Service

To be able to implement the product in accordance with the operational requirements it is of major importance that the non-functional requirements are well understood and defined. This is a work and process that needs to be undertaken in collaboration between the software developers, system developers and operational concept developers. Within the common QoS areas defined by confidentiality, integrity, availability, non-repudiation, and accountability the overall security of the service and its payload needs to be treated with special focus. The level of impacts if failure occurs are analyzed in categories of personnel, capacity, performance, economic, branding, regulatory and environment.

5.2.1.5 Verification and Validation

Verification and validation of an ISRM service like AMIS is based on rules provided by the ISRM foundation rulebook [Pola and Solberg, 2013]. It provides rules and information service modeling standards to be applied, in order to facilitate the development and maintenance of the service. The use and conformance to the standards ensures a consistent and high qualitative of the service and supports effective consolidation, validation and verification, conformance, and quality check processes.

5.2.1.6 Aerodrome Mapping Database

One of the largest components reused is the AMDB, a GIS database describing the spatial layout of an airport including the geometry of features (e.g.: runways, taxiways, buildings) such as points, lines, polygons, and further information characterizing the features and their functions which are stored as attributes (e.g.: surface type, name/object identifier, runway slope). The developed AMIS can access and filter the data provided by such a component. Fast information distribution and retrieval are key elements according to EA and SOA principles. In addition to the NextGen and SESAR related requirements, industry standardization activities within the EUROCAE and RTCA throughout over the past 10 years have defined user requirements as well as exchange standards for aeronautical information data, namely the EUROCAE ED-99C [EUROCAE, 2011b] and RTCA DO-272C [RTCA, 2011a] documents which mention user requirements for aerodrome mapping information and the EUROCAE ED-119B [EUROCAE, 2011a] and RTCA DO-291B [RTCA, 2011b] documents which describe the interchange standards for terrain, obstacle, and aerodrome mapping data. In contrast to current approaches, SESARs AIRM approach covers this by extending AIXM 5.1 [Eurocontrol, 2010] to cover the ED-99C and ED-119B standards. Identified operations of the service aim to fulfill IERs by accessing a complete AMDB over a certain airport, specific features of an AMDB, based on multiple filtering capabilities, pre-styled geo-referenced AMDB graphics (bitmap images) and a generic publish/subscribe service.

5.2.2 Context Factors

During the evaluation phase the XP-EF metrics collected data for comparing the old legacy system with the new AMIS product. In the following subsection the term "*old product*" always refers to the existing product and "*new product*" stands for the latest release of the completely new developed AMIS using the OBSD life-cycle. For both of them, detailed information about the context factors, the adherence metrics, and the outcome metrics were collected. The context factors are split up into six different categories, *software classification*, *sociological*, *geographical*, *project-specific*, *technological*, *ergonomic*, and *developmental factors* cf. [Layman et al., 2004a].

Software Classification XP-EF distinguishes between six different types of software projects:

1. *Information systems* dealing with business information,
2. *Technical systems* used to control physical devices,
3. *End user projects* developing applications for personal and private use only,
4. *Commercial projects* leased or marketed to external clients,
5. *Outsourced projects* which are developed under contract, and
6. *Military projects* located in the defense domain.

The AMIS product is a classic commercial project, developed for a launching customer, an ANSP. A former scientific research project funded by the European Union and Eurocontrol was used as baseline. The service is provided by an AMDB data provider, in this special case an AIS provider. It is foreseen that this product will be marketed for other ANSPs, commercial data house providers, and others. The consumer of the service can be an airport system, displaying the information for air-side ground controllers, vehicle display, aircraft taxiing operations, and ATC controllers. It may also be consumed by other airport management systems, airline operations centers, pilot briefing, and other airport stakeholders consuming airport layout maps. It can also be used for commercial data integrators or data houses as input to refine proprietary formats such as ARINC 816 [Pschierer and Schiefele, 2007].

Sociological Factors One of the most critical factors in software development is the staff itself [McLeod and MacDonell, 2011]. Therefore the analysis of the team composition is essential. The XP-EF sociological factors capture the team constitution for the old and the new product. They sum up the development experience of the team members as well as their domain knowledge of the target area. The factors of both products are compared against each other. As shown in table 5.1, sociological factor include team size, level of education and experience, domain and technical expertise, experience of the project manager and involved specialists, change of personnel, and morale factors. The team size consists of the sum of the members of the development team (e.g. developer, architects, tester, etc.) which are full-time or at least half-time engaged (count as 0.5). Both teams were rather small development teams who have worked

Sociological Factor	Old Product	New Product
Development Team Size	7	5
Level of Education	BSc: 5 MSc: 1 PhD: 1	BSc: 3 MSc: 1 PhD: 1
Level of Experience	< 5 years: 3 < 10 years: 3	< 5 years: 2 < 10 years: 2
Domain Expertise	Medium	Medium
Technical Expertise	High	High
Experience of Project Manager	High	Medium
Assigned Specialists	Senior Architect	Senior Lead Architect
Change of Personnel	15.4 %	10 %
Morale Factors	Director replaced	First SWIM Service

Table 5.1: AMIS sociological factors.

quickly and efficiently. The level of education is self explaining, whereas the mix of seniors and juniors was balanced. The first beta test was accomplished when AMIS was sill part of a scientific research project. This allowed the whole team to play with different technologies and investigate possible reusable components upfront. Due to the circumstance that the initial project was a research project acquiescing, additional planning and implementation time compared to a regular development project was available. Furthermore the team was thus already well established. At the beginning of the latest release most members of the development team were already familiar with OBSD and some of them gained domain experience from the former scientific research project. The domain knowledge was gained from the old legacy system. The change of personnel indicated in table 5.1 shows a significant rate of 15,4% during the development of the old product against 10% during the new one. Even though the 10% include no changes of the development team, as this number also includes roles like product manager, sales people, etc. The change of personnel is calculated by the sum of people who left or joined, divided by the team size at the end of the release. The morale factors categorize irregular factors, changes like the replacement of the director of the department during the development of the old product, which was accompanied by a great uncertainty. On the other hand during the development of AMIS, the team members were motivated to bring the research prototype to the level of a real product. In the addition the developer team was inspired to be implementing one of the first SWIM services.

Project-Specific Factors Project scope, duration, and size are extremely influential factors that can determine the success and failure of a project. Project-specific factors record this important context information about a project (cf. table 5.2). Since most software development projects follow some kind of release cycle, the project-specific factors concentrate on the entire development phase from start to delivery to the customer. Additional releases were rolled out later but not captured for the evaluation. Even though the development team was not that small the person months spent were lower than expected because for both, the new and the old product,

some of the team members were just engaged half-time. Table 5.2 also reveals the numbers of reused components and the total sum of them. The factors were collected for classes and methods, whereas the Kilo Lines of Executable Code (KLOEC) are split up between components and lines for the whole product. More than 12.2 % of the overall KLOEC within the new product were filled by reused components.

Project-specific Factor	**Old Product**	**New Product**
Nature of Project	New Release	New Development
Domain	Aeronautical Information Management	
Person Months	36	40.5
Duration in Months	6	9
Relative Feature Complexity	Medium	Medium
Product Age	4 years	11 Months
Number of Use Cases	12	25
Constraints	Safety, Scope and Resource Constrained	
Reused Components Total Components	N/A 7	28 37
Reused Classes Total Classes	N/A 671	411 3015
Reused Methods Total Methods	N/A 6891	3071 10342
Component KLOEC	Approx. N/A	21.4
Product KLOEC	132.5	175.4

Table 5.2: AMIS project-specific factors.

Ergonomic Factors The local environment can influence the work-flow and work-culture. Analyzing those ergonomic factors and collecting feedback is important for the evaluation and for further improvements [Layman et al., 2004b]. XP-EF compares the old and new product physical working space as summarized in table 5.3. During the development of the old product, the whole team was physically located in one open office, within one department. While during the implementation of the new product, the team was split up between separate offices between two different departments, since one part of the time was working on the research project before. As in-between talks no longer could take place, more regular meetings had to be agreed on (e.g. conversation on the corridor, etc.). The fact that the offices have a lot of glass elements, similar to an open space office, simplifies the communication (e.g. it is possible to recognize if someone is available or not). The work environment offered enough individual desks and company organization tools were available in both cases.

Ergonomic Factor	Old Product	New Product
Physical Layout	One Open Office Semi-private	Separate Offices Semi-private
Distraction Level of Office Space	Low	Medium
Customer Communication	On-site, Telephone Conference	

Table 5.3: AMIS ergonomic factors.

Technological Factors The technological factors, such as software development methodology, tools and processes, have a dramatic impact on the whole project and therefore are evaluated by the XP-EF framework. Defect prevention and removal practices are established to reduce costs and enhance the code quality cf. [Layman et al., 2004a]. These technological influences of AMIS are represented in table 5.4. The old product followed the waterfall software development methodology and used Microsoft's Project for planning purposes. The new implementation used a mix of agile and plan-driven software development methodology supported by the OBSD processes. Both projects worked with use-cases to estimate tasks and forecast release points and iterations. A look at the used software languages gives insight about the aged software baseline the old product was built on. Within the years more and more features were added on top of it. Small parts were developed in C, whereas the main product was implemented using C++. The new development of AMIS was built from scratch in Java with the help of the gained knowledge from the prior research project. Defect prevention and removal practices differ between the two products due to the different software development methodologies. For both products, the teams had software testers. Instead of code reviews, pair programming was performed during the implementation of the new product. In addition ad hoc customer tests were used for the first time.

Technological Factor	Old Product	New Product
Software Development Methodology	Waterfall	Mix of Agile and Plan-Driven OBSD
Project Management	Microsoft Project	Internal PM-Tool
Language	C, C++	Java
Reusable Materials	Existing Product 3rd Party Libraries	Research Prototypes OBSD Components 3rd Party Libraries Unit Test Suites
Defect Prevention and Removal Practices	Code Review QA Ad Hoc Testing	Pair Programming Unit Tests & QA Continuous Integration Collective Code Ownership Customer Tests

Table 5.4: AMIS technological factors.

Geographic Factors In a world where development teams are split all over the place while working together, geographic factors are important for this evaluation. Remote locations and different time-zones influence the software development life-cycle cf. [Layman et al., 2004b]. Table 5.5 summarizes the geographical factors. Both teams were working in the same time-zone. The development of the new product was split up between two departments. Most of the time both teams worked remotely. There were some on-site tests performed and installations led by the respective project manager. Both products had one launching customer, whereas the number of customers reflects the growth of the market.

Geographic Factor	**Old Product**	**New Product**
Team Location	One Location	Two Departments One Time-Zone
Number of Customers	Approx. 15	Approx. 40
Customer Location	Remote and On-Site Multi-National Various Time-Zones	
Supplier Location	None	

Table 5.5: AMIS geographic factors.

Developmental Factors [Layman et al., 2004b] used the developmental factors to identify if a project is more in the range of the agile or plan-driven development methodology. This factor was adopted for this thesis to identify if a project should use OBSD or not. In other words the meaningfulness of the use of component reuse in a specific project. [Happel and Seedorf, 2006] acknowledge ontology-based methodologies in the context of software engineering and introduced classification scheme (cf. figure 1.2). OBSD is combining two of them to form the new invented methodology (cf. chapter 4). The four existing developmental factors *team size*, *criticality*, *dynamism*, and *culture* were extended with the fifth factor *component reuse* to identify OBSD should be used for a software project or not. Figure 5.3 shows old and new developmental factors plotted on a polar chart with five axes. The points that are closer to the periphery suggest that the project perfectly fits into the concept of reusing existing components. The more to the center they come, the less useful is it to recycle components. More varied shapes indicate an OBSD usage to a moderate extent. Since most of the ATM related projects and products have a very high criticality, the aerodrome map is placed on a rather low security level risking essential funds but not human lives. The personnel factor ranks the technical skills from level 1B, implement procedural methods, to level 2 and 3, the ability to revise a method in an unprecedented situation cf. [Layman et al., 2004b]. The factor time size, indicates the number of personnel. Dynamism measures the percentage of changed requirements per month, whereas component reuse captures the amount of recycled artifacts. Opposed to the old one, the new product fits more to the OBSD method (cf. figure 5.3).

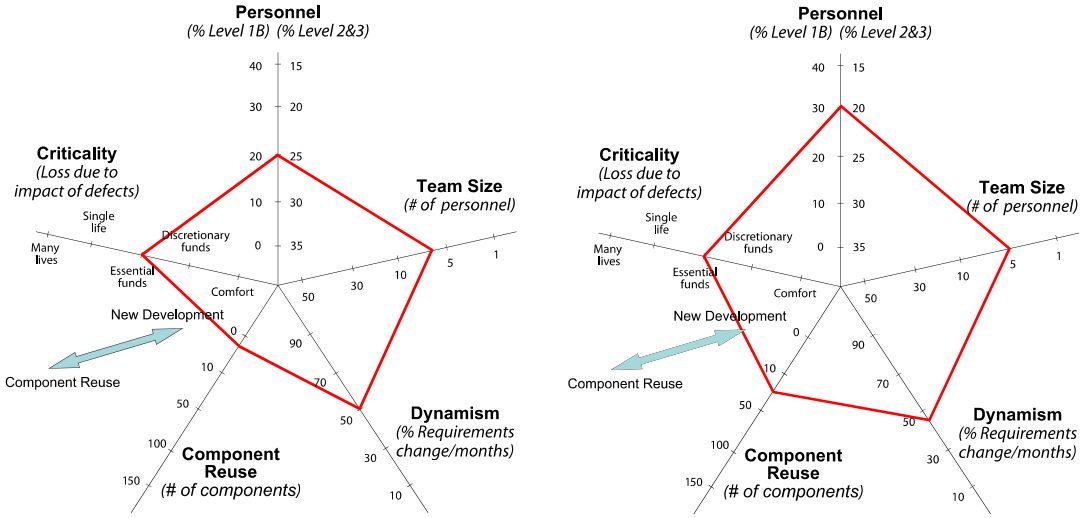

Figure 5.3: AMIS developmental factors.

5.2.3 Adherence Metrics

The adherence metrics examine three different aspects: *planning*, *coding*, and *testing*. XP-EF uses a slick designed survey to gather subjective information from the development team members cf. [Layman et al., 2004a]. Each category captures objective factors and subjective factors and compares the old against the new ones. The main goal of the survey is to measure quantifiable adherence of software development processes and practices by calculating the mean of the respondents. The higher the percentage number is the more often it was used. Especially the adherence metrics coding and testing provide essential information on the effectiveness of OBSD. The survey took place in a lessons learned workshop with members of both development teams.

Planning Adherence Metrics Table 5.6 shows the planning adherence metrics differences between the old development cycle and the new one. The shorter release length of the new AMIS implementation was adjusted to fit the agile development needs. The major difference was the automated build mechanism, which allowed to run small iteration cycles. The reuse of components was facilitated by stand up meetings held every second day. The stand up meetings were essential for the integration of the OBSD life-cycle since feedback from the team was collected and used to improve the OBSD processes constantly. Due to the different time-zones the customer testing turned out to be more difficult than imagined. Since the software development methodology was a different one, most of the factors are not available for the old implementation.

Planning Metric	Old Product	New Product
Objective Metrics		
Release Length	6 Months	3 Months
Iteration Length	None	2 Weeks
Requirements Dynamism	50 %	45 %
Subjective Metrics (Survey)	**Mean**	
Stand Up Meetings	N/A	56 %
Short Releases	N/A	74 %
Customer Access / On-Site Customer	N/A	15 %

Table 5.6: AMIS planning adherence metrics.

Coding Adherence Metrics The coding adherence metrics of the old and new product are illustrated in table 5.7. The factor component reuse depicts how many of the queried components of OBSD were selected and integrated in the development of AMIS. As a result the feedback for the OBSD team was to work on sharpening the group of selected components, to improve this factor. Most of the time OBSD is used during the beginning of a development cycle, the improvements are visible in subsequent OBSD projects (cf. section 5.3). But also during the old development cycle, code was recycled but no numbers were available. The pairing frequency, another objective adherence measurement, was never used for the old product, but 25% of the time for the new AMIS. The integration of components was accomplished most of the time in pairs to get the developer quickly familiar with the components. Pair programming was highly discussed and also reflects the outcome of the survey, whereas the team member slightly overrated the objective metrics according to the survey (55 % versus 45%). Classical code inspection was performed during the old release. This was not the case during the new one.

Coding Metric	Old Product	New Product
Objective Metrics		
Component Reuse	N/A	25 %
Pairing Frequency	0 %	45 %
Inspection Frequency	60 %	0 %
Subjective Metrics (Survey)	**Mean**	
Component Integration	N/A	55 %
Refactoring	N/A	60 %
Pair Programming	N/A	55 %
Simple Design	N/A	74 %
Collective Ownership	N/A	86 %
Coding Standards	N/A	76 %
Sustainable Pace	N/A	70 %
Metaphor	N/A	68 %

Table 5.7: AMIS coding adherence metrics.

The subjective metrics reflect the outcome of the survey. The integration of components was higher than expected and during the interviews some developers stated that OBSD motivated them to refactor or even extend existing components in the OBSD repository. Refactoring of code was high in general. The baseline prototype used for AMIS was developed within a research project which also delivered components for OBSD. The concept of simple design was rated high too by the respondents. This may have been due to the use of the OBSD methodology. Collective ownership was definitely strengthened, 86 % was the mean of all interviewed personnel. It seems the OBSD methodology breaks up with the old thinking of code as their own property. Company coding standards were applied to an extent of 76 %. The sustainable pace suffered from the unusual overload at the beginning, as AMIS was the first project using OBSD. Therefore the development had to deal with issues and a lot of internal feedback. But in the end it didn't have as much impact as feared. The metaphor was relatively good since most of the requirements were already discovered during the research project, but of course some of them changed during the actual AMIS implementation.

Testing Adherence Metrics Table 5.8 presents the testing adherence metrics. The test coverage is averaged over the whole product and nearly touched 90%. One of the reasons were the automated test that were available for the OBSD components but also new classes were covered nearly by 45%. Unfortunately for the old implementation only inadequate data was available. The coverage of test classes of the reused classes was higher due to the fact that existing test classes were reused. The test run frequency value within AMIS was 2.0, which indicates that each team member started the automatic tests twice per day. It is conspicuous that the ratio between test Lines Of Code (LOC) and source LOC are significant higher for the new product. Test code was used more often (either created or reused) which was also effected through the high test coverage of the recycled parts.

Testing Metric	**Old Product**	**New Product**
Objective Metrics		
Test Coverage	N/A	87.3 %
Test Run Frequency	None	2.0
New Classes with Test Class	4 %	45 %
Reused Classes with Test Class	N/A	80 %
Test LOC / Source LOC	0.061	0.152
Subjective Metrics (Survey)	**Mean**	
Test First Design	N/A	48 %
Automated Unit Tests	N/A	61 %
Customer Acceptance Test	N/A	63 %

Table 5.8: AMIS testing adherence metrics.

The customer acceptance test metric was not as good as expected (63 %), due to some late changes of the customer requirements and the lacking of a high quality acceptance tests. The circumstance that the technology stack used was new for the customer and the fact that there

were some problems with the automated unit tests of AMIS led to this rather of low result. Both other tests also left room for improvements but this had less to do with OBSD as stated by the interviewed personnel.

5.2.4 Outcome Metrics

Figure 5.9 illustrates the outcome metrics which comprised measurements regarding productivity and code quality, as well as values about customer satisfaction and morale captured by the survey. Since the company had their own QA metrics in place, both developments software defects for both were tracked. For easier comparison and to protect proprietary information a relative scale with the old product at 1.0 is used in the figure. The interviews examined the extent to which OBSD proved useful. Through the survey one can conclude that a benefit was the exchange of knowledge between different departments. For example the fact, that specific components already existed in the company, fulfilling the requirements needed. This led to a lot of discussion with developers outside of the team and helped to solve problems before they even occur. Also technical knowledge and experience was exchanged. It was stated that more components should be included into the OBSD repository. Furthermore the functionality of the interface should be extended to offer more features. Over time, more and more recyclable artifacts will be collected and made accessible through the OBSD methodology. In the mean time this was confirmed by the second case study, more artifacts were available and the quality of them improved (cf. subsection 5.3.3). According to the feedback from some of the team members, in the beginning, the query results of OBSD provided inaccurate results. This feedback has helped to improve and narrow down the search results. For some components the documentation was poor and the lack of support of the originator of the components was mentioned. This led to the circumstance that for every component a leader was chosen to cover perceived inconsistencies. It also seemed that the use of OBSD opened the mind of some developers regarding collective ownership in a positive way.

Outcome Metrics	**Old Product**	**New Product**
Objective Metrics		
Internal-Visible Quality (Test Defects/KLOEC)	1.0	0.65
External-Visible Quality (Released Defects/KLOEC)	1.0	0.61
Productivity (KLOEC/Person Month)	1.0	1.11
Subjective Metrics (Survey)	**Mean**	
Customer Satisfaction	High	High
Morale	70 %	71 %

Table 5.9: AMIS outcome metrics.

The testing metric, *internally-visible quality*, showed defect density improvements of 35% with respect to the prior implementation. This is an answer to one of the sub-questions of the main research question defined in section 1.5. Improving the quality of the code before the actual roll-out, is one way to save software development costs through ontology-based technologies. Of course the direct comparison between the old and the new product was also influenced by other factors. The quality of the supporting tools within the company increased and all the other difference listed in the subsection of the context factor had influence as well. Following the feedback encountered during the interview, defects in general were lower with reused components, which underpins the figures of the internally-visible quality.

Externally-visible quality metric shows the defect density after the release was even better. The number of defects in the rolled-out product has improved by approximately 39%. The reuse of already tested components brougth an advantage in terms of software quality and stability. This also correlates with the results of the QA metrics. Some errors and issues happened due to weak requirements which were not identified upfront. The outcome measure of the old product should be interpreted with caution, since according to the developer the value could be even worse. However, the new solution will be sold to much more customers, which multiples the positive effect of less software defects.

Productivity is a very important outcome metric for decision makers to answer the cost benefit question. Of course productivity can not be equated one to one with less development costs, but it is a major hint. And there is also the circumstance that more and more standardized models helped to improve the teams efficiency within this safety critical domain. The survey reinforced the hypothesis that OBSD practices led to a productivity increase of 11%. Of course also other factors had an influence on this result, but still this approves the improvements. According to the developers, the recycling brought a decrease of complexity. On the other hand the integration effort increased but still not as much to influence the productivity in a bad way. The QA data regarding the old product did not record much details (e.g. the amount of effort spent on non-production activities), otherwise a more comprehensive comparison would be possible. Another reason for the increase is the fact, that the team members had time to get familiar with the domain and the requirements were investigated during the previous research project.

The long-standing relationship with the customer led to a high level of *satisfaction* in both cases. Most of the problems and issues could be solved directly by the experts, without involving (higher) management. OBSD was used during offers as unique selling point and as a marketing tool to make a difference to other competitors. During the lessons learned workshop it was mentioned by the customer that the usage of OBSD was not noticed, which was interpreted as a positive feedback on all sides.

During the lessons learned workshop the members of the development team were asked to estimate the *morale* of the team over the whole project. In detail they were asked about their opinion about OBSD and a lot of feedback was received. Of course not only in a positive way, but the fact that almost all interviewed personnel gave feedback on how to improve and make better use of the life-cycle was an overwhelming success. The feedback also influenced the management to implement a bonus system for the developers in case they fix bugs in reused components and check them in so everybody benefits.

5.3 Integrated Digital Briefing Evaluation

The evaluation from textual briefing to digital briefing was an important step within the Aeronautical Information Management domain. This was made possible by the foundation established through the joint forces of research programs like Eurocontrol's SESAR and FAA's NextGen. This section describes the evaluation of the development of the prior developed briefing product and the new briefing application, the Integrated Digital Briefing Service (IDBS). IDBS is enabled by the availability of digital aeronautical services, meteorological services, and flight information services to distribute a digitally enhanced briefing. The IDBS allows consumers to request briefing information based on various filtering criteria as well as to subscribe for being informed about updates in the briefing information (e.g. in-flight updates). Briefing information includes meteorological data (e.g., METAR, TAF, SIGMET) as well as aeronautical information (e.g., DNOTAM). The Evaluation not only analysis the new service development IDBS, which is also called the "new product", but also the previous product, which is referred to in the following chapter as the "old product". While current briefings often require manual steps for the creation, IDBS is an integrating service, providing added value by combining information that is already available via other services, such as meteorological services (e.g., METAR, TAF services), aeronautical information services (e.g., AeronauticalInformationFeature service), mapping services (e.g., AerodromeMapInformation, AeronauticalInformationMap services). The old product, used for the comparison, didn't utilize DNOTAMs instead NOTAMs in the previous format defined by AIXM 4.5 were used for publishing a briefing. Another big difference is the support of the in-flight briefing use case, while the old product just covers the pre-flight briefing use-case. IDBS uses existing services to aggregate new information needed for creation of the briefing shown in figure 5.4. A lot of preliminary work was accomplished in a previous SESAR research project. For example, customer requirements from various ANSPs were collected and a briefing prototype was developed which was used as foreground for the new product.

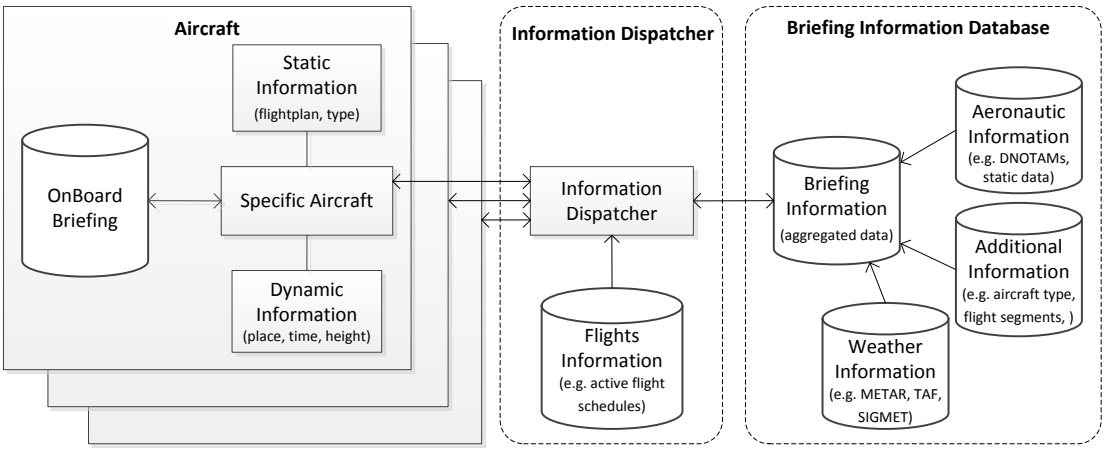

Figure 5.4: Integrated digital briefing overview.

5.3.1 Context Factors

As already stated in subsection 5.2.2 the evaluation framework XP-EF makes use of six separate context factors. The outcome of the evaluation and the determined information is presented in this subsection. The work on IDBS started beginning of 2015 and lasted for 12 months for the first release submitted to the customer. The last release of the old product was finished two years ago and the evaluation data was collected from the company's QA team based on the last release. During the beginning of the new product development a workshop together with the development team was held to introduce OBSD. The lessons learned from previous projects using OBSD were presented and everyone had the chance to get familiar with the OBSD feedback loop.

Software Classification XP-EF classifies software projects into six different groups: *information systems*, *technical systems*, *end user products*, *commercial projects*, *outsourced projects*, or *military projects* a more detailed description can be found in subsection 5.2.2. Since both products were built for a launching customer with follower customers in mind, both projects are classified as commercial ones.

Sociological Factors Table 5.10 presents the sociological factors. The team sizes were slightly bigger in comparison with the other case study in section 5.2. The level of experience for both development teams was high. Four developers of the new product worked for the AMIS project before. This was a great benefit for the new developed IDBS. Both project managers had a medium experience level but worked in the domain before. From the beginning the requirements were stable, through the research work accomplished before. The change of personnel was quite low with two persons during the development of the old product and one person during the new one. The moral factor is an important factor to be looked at. The interviewed team members mentioned routine development, while working on the old product. In contrast to this, according to survey statements, the new development was under increased time pressure.

Sociological Factor	**Old Product**	**New Product**
Development Team Size	10	13
Level of Education	BSc: 8 MSc: 1 PhD: 1	BSc: 9 MSc: 3 PhD: 1
Level of Experience	> 5 Years: 3 > 10 Years: 3	> 5 Years: 2 > 10 Years: 2
Domain Expertise	Moderate	
Technical Expertise	High	
Experience of Project Manager	Medium	
Assigned Specialists	Software Architect	
Change of Personnel	20 %	13 %
Morale Factors	Routine Development	Delivery Pressure

Table 5.10: IDBS sociological factors.

Project-Specific Factors Table 5.11 compares the project-specific factors for both products. Even though the last release of the old product was two years ago, some parts of the product are much older. The duration of the last release was relatively short with 5 months compared with the duration of the 12 months of development for the new product. The complexity increased from medium to high since IDBS now supports more complex use-cases such as in-flight briefing. The increase of functionality is also reflacted by the total numbers of classes and components. More than 15% of the overall KLOEC (new product) was produced by reused components.

Project-specific Factors	**Old Product**	**New Product**
Nature of Project	New Release	New Development
Domain	Aeronautical Information Management	
Person Months	34	103
Duration in Months	5	12
Relative Feature Complexity	Medium	High
Product Age	4 Years	3 Months
Number of Use Cases	8	27
Constraints	Safety, Scope and Resource Constrained	
Reused Components Total Components	N/A 12	28 45
Reused Classes Total Classes	N/A 728	812 2812
Reused Methods Total Methods	N/A 6513	4018 12541
Component KLOEC	N/A	30.5
System KLOEC	125.1	205.7

Table 5.11: IDBS project-specific factors.

Ergonomic Factors The ergonomic factors for both products were identical and are described in table 5.12. It is worth mentioning that the semi-private offices can hold between four or eight persons and are ideal for agile development even though it is not the classical open space office environment. Stand-up meetings were easy to schedule during the development of IDBS since a meeting-point room was near by. The communication with the customer within both developments was done via e-mail, telephone conferences, and via face-to-face meetings on-site.

Ergonomic Factor	**Old Product**	**New Product**
Physical Layout	Semi-Private	
Distraction Level of Office Space	Low	
Customer Communication	On-Site, Telephone Conference, E-Mail	

Table 5.12: IDBS ergonomic factors.

Technological Factors Technological factors for both the new and the old product, are quite the same as they were for the AMIS development. During the old product the waterfall was used as software development methodology, while the new IDBS implementation used a mix of agile and plan-driven software development methodology supported by OBSD. For the project management the company own tools were used which were standard at the time. Both projects worked with use-cases to estimate tasks and forecast release points and iterations. For both products Java was used as software language for the development. The initial release of the old product was developed years before the last release, as a result the core of the product underwent a lot of refactoring over the years. The new development of IDBS was built from scratch in Java based on the gained knowledge from the previous research project. Because of the different software development methodologies used, the defect prevention and removal practices differ between the two products. Software testers were available within both teams. Instead of code reviews, pair programming was performed during the implementation of the new product. In addition ad hoc customer tests were used as it was done for AMIS. In addition the feedback loop was improved allowing the developers to use the OBSD IDE plug-in to directly submit their feedback.

Technological Factor	**Old Product**	**New Product**
Software Development Methodology	Waterfall	Mix of Agile and Plan-Driven OBSD
Project Management	Microsoft Project	Internal PM-Tool
Language	Java	
Reusable Materials	Existing Product 3rd Party Libraries	Research Prototypes OBSD Components 3rd Party Libraries Unit Test Suites
Defect Prevention and Removal Practices	Code Review QA Ad Hoc Testing	Pair Programming Unit Tests & QA Continuous Integration Collective Code Ownership Customer Tests

Table 5.13: IDBS technological factors.

Geographic Factors The geographical factors shown in table 5.14 cover the team location, number of customers, and location of the customer. Both product developments took place in one department within one time-zone. This simplified the communication between the team members. The location of the customer was in both cases multi-national. Most of the time telephone conference were used as communication form. During the beginning of the IDBS development an introduction face-to-face meeting was arranged to get to know all the involved participants, also on the costumer side. This improved the relation to the customer, as every individual name now had a corresponding face, which usually facilitates the whole collaboration.

Geographic Factor	Old Product	New Product
Team Location	One Location	One Department One Time-Zone
Number of Customers	Approx. 15	Approx. 50
Customer Location	Remote and On-Site Multi-National Various Time-Zones	
Supplier Location	None	

Table 5.14: IDBS geographic factors.

Developmental Factors A detailed explanation about the five developmental factors was given in subsection 5.2.2. The developmental factors for the old and the new product, collected during this evaluation, are presented in figure 5.5. Even though most factors are the same, a difference of the dynamism occurred due to the previous research work accomplished. The launching customer defined the requirements quite well and the prototyping upfront improved the quality of the requirements before the actual development of the new product. The number of reused components during development was another big change between the old and the new developmental factors. A lot of new features were added to IDBS compared to the previous product which led to this huge difference. In addition it was quite helpful that between this case study and the first one, a couple of other projects were using OBSD which improved the OBSD life-cycle significantly. This together led to an improved reuse of components.

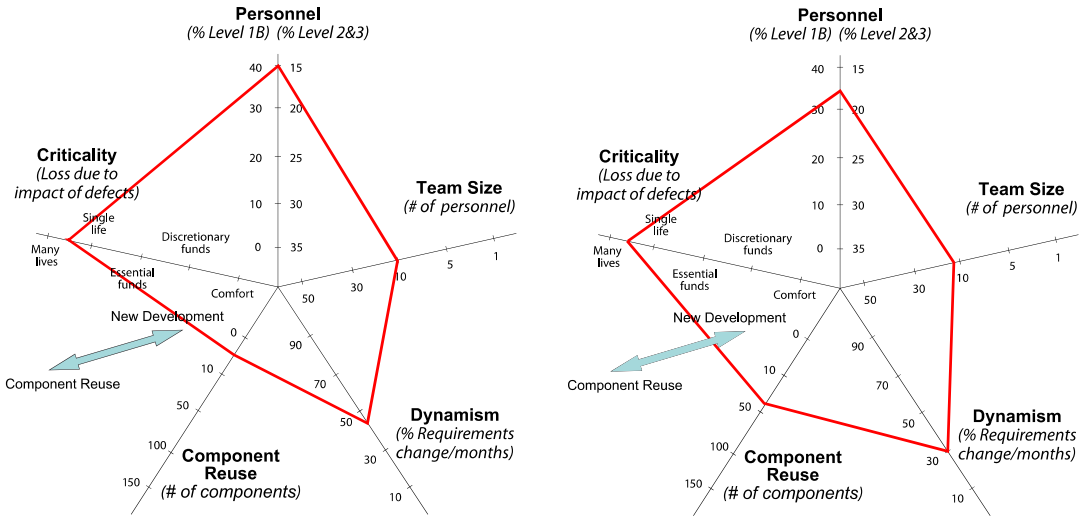

Figure 5.5: IDBS developmental factors.

5.3.2 Adherence Metrics

The results of the adherence metrics of the new and the old product are presented in this subsection. A description of the different metrics used, can be found in subsection 5.2.3. Within a safety critical environment, special practices and processes were established to master the specific challenges in context of such a domain. The OBSD software development life-cycle was built to fit the needs of such a diverse and demanding domain. The main goal of the survey is to measure quantifiable adherence of software development processes and practices by calculating the mean of the respondents. The higher the percentage number is, the more often it was used. Especially the adherence metrics about coding and testing provide essential information on the effectiveness of OBSD. The survey took place in a lessons learned workshop with members of the development teams.

Planning Adherence Metrics The planning adherence metrics are condensed in table 5.15. The new product used a release length of three months with an iteration length of two weeks. The release length of the old product was the same as the project duration which was five months for the last release. The objective planning metrics also captured the requirements dynamism which is an important factor. The amount of changed requirements allows some conclusions. Due to the outstanding work of the previous research projects the defined requirements were identified as quite stable only 25% were affected by changes afterward. Around 45% of the requirements were changed during the development phase of the old product. As highlighted during the AMIS lessons learned workshop, the assigned component leaders were invited to standup meetings if issues were raised and feedback was required.

Planning Metric	**Old Product**	**New Product**
Objective Metrics		
Release Length	5 Months	3 Months
Iteration Length	None	2 Weeks
Requirements Dynamism	45 %	30 %
Subjective Metrics (Survey)	**Mean**	
Stand Up Meetings	20 %	56 %
Short Releases	N/A	71 %
Customer Access / On-Site Customer	8 %	9 %

Table 5.15: IDSB planning adherence metrics.

Coding Adherence Metrics The coding adherence metrics of the old and new IDBS product are shown in table 5.16. The factor component reuse depicts how many of the queried components of OBSD were selected and integrated in the development of IDBS. The feedback from the previous case study led to an improvement of the queried components, which allows the conclusion that the OBSD process has improved in terms of preciseness and accuracy. The subjective metric for integration of components reflects the objective metric and was accomplished most of

the time in pairs to get the developer quickly familiar with the components. This was an already proven concept taken on board from the AMIS case study. Unfortunately for some metrics no data was available for the old product. The subjective metric for collective ownership for the new IDBS was even higher compared to the value shown in the coding adherence metrics of AMIS (cf. section 5.2.3).

Coding Metric	**Old Product**	**New Product**
Objective Metrics		
Component Reuse	N/A	21.4 %
Pairing Frequency	0 %	25 %
Inspection Frequency	57 %	0 %
Subjective Metrics (Survey)	**Mean**	
Component Integration	N/A	45 %
Refactoring	N/A	58 %
Pair Programming	N/A	35 %
Simple Design	N/A	65 %
Collective Ownership	N/A	82 %
Coding Standards	N/A	80 %
Sustainable Pace	N/A	72 %
Metaphor	N/A	67 %

Table 5.16: IDBS coding adherence metrics.

The subjective metrics reflects the outcome of the survey. Component integration, with a value of 45%, was a bit lower than compared to evaluation metrics of AMIS. Since the baseline prototype of IDBS was developed within a research project the refactoring of code was high. The hypothesis, that the OBSD methodology breaks up with the old thinking of code as own proprietary, can be again undermined with a metric value over 82%. As shown in the previous evaluation the developers mentioned that OBSD motivated them to use existing components from the OBSD repository and improve them. The pairing frequency was lower (25%) than the 45% rate measured during the development of the new AMIS (cf. section 5.2.3). The concept of simple design was rated almost as high as during the development of AMIS by the respondents. Company coding standards were applied to an extend of 80 %. The sustainable pace was fine with a value of 72%. Due to the circumstance that enough foreground was developed during the research project led to an acceptable metaphor of 67%. The general feedback about OBSD was good, the IDE integration improved since the first case study and brought a better user acceptance. The developers, which already worked with the new methodology during the development of the AMIS also stated that some of the issues regarding OBSD were solved. Improvements like, enhanced component description, extensions to the IDE, more reliable components for reuse, a faster repository, and a direct feedback loop to ameliorate the OBSD knowledge were implemented.

Testing Adherence Metrics Table 5.17 illustrates the testing adherence metrics of the IDBS. The test coverage was more than 80%. One of the reasons therefore are the automated tests which were available for the OBSD components. In addition new classes were covered nearly 35% by automated testing. The test run frequency value was lower than the case study brought to light during the evaluation of the AMIS (1.0 versus 2.0). It is conspicuous that the ratio between test LOC and source LOC significantly higher for IDBS compared to AMIS (cf. subsection 5.2.3). During the interview developers mentioned an improvement of the quality of the components themselves. According to the numbers of the survey capturing the subjective metrics, 55% was covered by the test first design due to functional tests. Automated unit tests and customer acceptance test metrics reached 60% and more. This can be attributed to the clear requirements coming from the customer and the gained knowledge of the development team during the research project up front.

Testing Metric	**Old Product**	**New Product**
Objective Metrics		
Test Coverage	N/A	81.2 %
Test Run Frequency	None	1.0
New Classes with Test Class	N/A	33 %
Reused Classes with Test Class	N/A	85 %
Test LOC / Source LOC	N/A	0.211
Subjective Metrics (Survey)	**Mean**	
Test First Design	N/A	55 %
Automated Unit Tests	N/A	59 %
Customer Acceptance Test	N/A	65 %

Table 5.17: IDBS testing adherence metrics.

5.3.3 Outcome Metrics

Figure 5.18 depicts the outcome metrics using objective and subjective metrics, like internal and external visible quality, productivity, customer satisfaction, or morale. The companies QA principles were used to trace software defects. As done for the outcome metrics of AMIS, the relative scale was set to 1.0 for the prior product. During the lessons learned workshop the feedback was overall positive. The developers worked with OBSD before, mentioned an improved quality of the components. This was shown mainly by an improved description, stability and documentation of the code. Due to the customer requirements the component repository was used to identify which components can be reused or extended. It was even used for offers to estimated the needed effort. The success of the exchange of knowledge (cf. subsection 5.2.4) within the own company formed an own group dealing with common components which is also responsible for the OBSD life-cycle. As discovered during the comparison of AMIS, the number of available artifacts increased and will rise in the future.

Outcome Metrics	Old Product	New Product
Objective Metrics		
Internal-Visible Quality (Test Defects/KLOEC)	1.0	0.62
External-Visible Quality (Released Defects/KLOEC)	1.0	0.59
Productivity (KLOEC/Person Month)	1.0	1.23
Subjective Metrics (Survey)	**Mean**	
Customer Satisfaction	Medium	High
Morale	60 %	68 %

Table 5.18: IDBS outcome metrics.

The *internally-visible quality* correlates with the defect density. This value improved by 38% compared to the prior implementation. The quality improvement of the components is one result obtained by OBSD. The more often components are reused the better the stability gets. Defects in those components are significant lower than in new or even refactored artifacts. This pattern was also reflected during the survey's feedback from the developers. The usability of the IDE plug-in still needs to better support the feedback of identified errors and submission to the repository. Further improvements are planned to deliver a smoother user experience based on the received feedback.

Externally-visible quality measures software defects found after the first roll-out to the customer. 41% less errors were found in the first released version. Improved software development processes, a better tool support, and the reuse of field proven software were mentioned by the developers as major aspects for this improvement. This outcome value is even better than the one achieved during the AMIS development. The close relation ship with the customer and the foreground done in the related research project led to stable and high quality requirements.

Productivity is an economic indicator, which is categorized by the management as extremely important. It also correlates directly with the cost-benefit question. Increased productivity is usually accompanied by a reduction of development costs. It is measured by the number of KLOEC per person month. With the reuse of components this value has increased by 23% compared to the previous product. IT was helpful that most of the developers had worked on the research project and with OBSD before.

The high customer *satisfaction* was achieved due to a very close relationship with the customer. The great commitment from the customer, from the beginning of the research project till the end of the development of the product itself, certainly helped. According to the feedback of the developers the use of stable components allowed a very early and smooth integration compared to the integration necessary during the new AMIS development.

The *morale* metric was slightly higher than during the prior development. It seems that due to the huge amount of new features the pressure on the team had effects on the morality compared to relaxed development during the old product. The feedback also suggests that the costs of integrating components increased and also had a negative impact on the morale, although this

improved in contrast to the past development. The use of OBSD should therefore be handled carefully and always focus on the support of the developer. A potentially too ambitious goal should be avoided (e.g. to reuse too many components). Otherwise, the OBSD advantages can quickly turn into disadvantages.

5.4 Conclusion

This chapter presented two projects utilizing the idea of Ontology-Based Software Development evaluated within this case study. For the purpose of comparing the old development process with the methodology introduced within this thesis, two new product developments were chosen to be evaluated and analyzed against the old development using the adopted XP-EF evaluation framework (cf. subsection 1.5.2). Context factors, adherence metrics data, and outcome metrics were recorded for each of the processes. Quantitative and qualitative measurements were captured during a lessons learned workshop from the development teams. Those findings have then been reviewed and analyzed during the OBSD feedback loop to improve the OBSD life-cycle. Within the context of this chapter, the final results of the investigation and analyses were shown. It can be concluded that the knowledge about already existing components is better distributed within the company and that the reusability increased. Reusability, in a security-critical environment, is worth more, since software is often used over a longer time horizon than in other domains. Interestingly the survey also covered the fact that OBSD is used for superior offer supply, since requirements can be used to estimate more accurate. The findings in both case studies support the hypothesis that OBSD can to some extend close the semantic interoperability gap, which is one of the main research questions stated in section 1.5.

The new implementations of the AMIS and the IDBS showed that the OBSD life-cycle and its underling processes are capable of improving the reusability of components. From a pure financial aspect it is not easy to gain scientific numbers regarding the integration effort of reused components, which were not refactored or extended. This is also bounded due to the limitations the case studies and their evaluation and analyzes have. Both outcome metrics showed that the quality of code improved before the roll-out, which is one way to save software development costs through ontology-based technologies. The outcome metrics also revealed a significant positive effect of OBSD on the external-visible quality (cf. subsections 5.2.4 and 5.3.3). Another interesting finding of the case studies is the fact that developers mentioned the motivation to improve existing components from the OBSD repository. The findings suggest that the OBSD practices can improve the productivity and can lead to the reuse of components, which in the end improves the product quality. To some extend the main research question and the derived subquestions could be answered. The interpretation of all findings should be taken with care since both products evolved significantly in terms of feature richness between two compared releases for both evaluations. In addition it should also be mentioned, that these results are based on case study in one particular domain. The general outcome of the case study is, that under specific circumstances OBSD solves the reusability challenge to reuse software components across various domains. Of course more research and improvements will be necessary to enhance the proposed solution. Future steps include the improvement of the quality of existing components in the repository and the increase of reusable components which are available.

CHAPTER 6

Epilogue

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."

[Churchill, 1942]

In this chapter a critical reflection about the achieved results is drawn by answering the design science research checklist outlined in the prologue 1.5. In addition a general conclusion summarizes the outcome of the thesis. In the end open problems and interesting topics for future research are presented.

6.1 Results and Critical Reflection

As described in section 1.5.1 [Hevner et al., 2004] defined seven guidelines to assist researchers and readers for conducting top quality design science research. Based on those guidelines a more specific checklist of questions to evaluate a design research project was developed in 2013 [Gregor and Hevner, 2013]. The checklist was used to ensure that this thesis addresses the key aspects of design science research. The list below accompanies an explanation of how the thesis answers these questions.

1. **What is the research question?**

   Following the design science research pattern, design requirements define the research question. The main requirement related to this thesis is to improve software development in terms of cost-efficiency. The main research question of this thesis was defined in chapter 1, section 1.5:

   Is it possible to solve the reusability challenge to reuse software components across various domains?

The main goal of OBSD is to develop an ontology-based solution to this important business problem. This main research question is further divided into two sub questions: “Can software development costs be lowered with the help of ontology-based technologies?” and “Which strategy closes the semantic interoperability gap?”.

2. **What is the artifact? How is the artifact represented?**

   Design, as defined by [Gregor and Hevner, 2013], must produce a viable artifact in the form of a construct, a model, a method, or an instantiation. During the work on this thesis a couple of artifacts were developed. The OBSD methodology presented in section 4.2 outlines a life-cycle of how to use ontologies to support the reuse of software components for different domains. After establishing the OBSD methodology two industrial case studies were chosen to answer the research questions described above. The results were then evaluated against the old process of development with a corresponding evaluation method (cf. chapter 5).

3. **What design processes are used to build the artifact?**

   A computable model of design processes [Takeda et al., 1990] was used to build the artifacts described above. It was assumed that the design process changes iteratively and each state holds the description of the current design, the properties of it, and the actual knowledge. It is possible to build a computable model by interpreting the cognitive model in the logical framework discussed by [Takeda et al., 1990].

4. **How are the artifact and the design processes grounded by the knowledge base?**

   The relation between design process and artifacts is quite obvious. The artifacts were grounded on state-of-the-art technology, deep domain knowledge, scenario-based design approaches and well settled techniques for an empirical prof of concept. This all was made possible by bringing very diverse topics like technical, operational or domain know-how together. The OBSD methodology unites these diverse topics to gain value through the aggregation of the existing knowledge.

5. **What evaluations are performed during the internal design cycles?**

   The OBSD methodology changed after each iteration of the design cycle as it was not clear in a first step which ontologies should be built and matched. Following [Gregor and Hevner, 2013], design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact. A couple of iterations later the OBSD methodology was evolved enough to be used for a first case study. Evaluation tests have demonstrated that the artifacts iteratively improved and in the end have significant advantages in terms of efficiency and effectiveness with respect to existing approaches. Moreover, as a result of the evaluation comparison preformed on the two case studies the OBSD methodology was adopted over time.

6. **How is the artifact introduced into the application environment and how is it field tested?**

   The OBSD methodology was tested within two case studies by developing two different products using this ontology-based technique. Furthermore, additional validation was performed to show the advantage in terms of effectiveness and efficiency of the new artifacts in respect to the current solutions. Evaluation metrics and surveys provided a lot of useful feedback to improve the solution sustainable.

7. **What new knowledge is added to the knowledge base and in what form?**

   As design science research must be presented effectively to both technology-oriented and management-oriented audiences several peer-reviewed papers were submitted in the context of this thesis (cf. section 1.6). The idea of the thesis was honored with the innovation award[1]. In addition the idea of the OBSD methodology was presented during conferences world wide and semantic artifacts were submitted to be included for the upcoming release of AIRM and ISRM.

8. **Has the research question been satisfactorily addressed?**

   The utility, quality, and efficacy of a design artifact was rigorously demonstrated via well-executed evaluation methods. Ontology-Based Software Development can have a significant impact on software development, if it is used for the development of different applications or products with overlapping payload. OBSD increases productive code usage and reduces the effort for software development of existing components, if a governance structure is in place and necessary roles are defined. It focuses on improving efficiency and gaining effort by code reusability, thus contributing to reduction of deployment costs of such solutions by improving the quality of the implemented solutions. The performed evaluation shows the benefits of an ontology-based approach to reuse software components.

6.2 General Conclusions

It is most important for higher management whether or not the OBSD life-cycle and its underlying processes are capable of solving the reusability challenge to reuse software components across various domains by saving the software development costs and closing the semantic interoperability gap through ontology-based technologies. As stated in the prologue, the motivation of this thesis is to show the benefits an ontology-based life-cycle can offer in a real business context. The crucial factor for such methodology in order to be used and successful is how much effort in terms of integration and development is saved. The thesis addressed significant problems like loss of knowledge, single point of knowledge, reinvention of the wheel, domain specific development, and lack of feedback cycles as described in Prologue 1.

[1] http://i-cns.org/2010/student-paper-award-winners-announced-for-icns-2010/, accessed: 2016-10-23

State-of-the-art information management engineering tools and languages were examined in chapter 2. In order to address the research question accordingly a set of tools, languages, and technologies was chosen. It also became clear that a lot of additional work on the tool side will be necessary to implement the concept of OBSD.

In chapter 3, the operational context in which the case study evaluation took place was described. The introduction of the Air Traffic Management domain gave necessary insight about the special requirements a safety critical environment is accompanied by. It highlighted the great benefits of reusing components in this domain, in which products have a very long life cycle.

The OBSD methodology, life-cycle, and processes were discussed in chapter 4. The knowledge models which form the semantic description were introduced. The key principles used to design the OBSD ontologies and ontology mediation techniques were outlined. Semantic description origination, mediation and solution model deployment were described in detail. OBSD roles and responsibilities needed during the OBSD life-cycle required by the different processes were mentioned. In the end of this chapter the semantic interface was discussed, which grants access to the semantic description to calculate the solution model.

The OBSD patterns were validated and analyzed in an industrial case study and are outlined in chapter 5. The evaluation was conducted out of two new developed products which were compared with their predecessors with an adopted version of the Extreme Programming Evaluation Framework. Both outcome metric results showed a significantly positive effect on the quality aspects of the code due to the reuse of components. Based on the results of the investigation, it can be concluded that the knowledge about components is better distributed within the company and therefore also the reusability increases. In a security-critical environment, this is especially efficient because software is often used over a longer time horizon than in other areas. The findings also support the hypothesis that OBSD can to some extent close the semantic interoperability gap. The exchange of knowledge beyond company division boundaries, and the reuse of components domain independent improved the quality of the delivered products. OBSD was also used for offers to estimate the effort needed. The benefit of the exchange of knowledge formed an own group within the company dealing with common components which is also responsible for the OBSD life-cycle and the underlying processes. The number of available reusable artifacts increased and will continue to rise in the future.

6.3 Open Problems and Future Perspectives

One important observation is that although a lot of effort was spent in the information management engineering domain through the rise of the the Semantic Web, an all-in-one semantic framework for industrial use is still not available. This has numerous reasons, on the one hand, all existing languages, tools, and frameworks have their main focus on the semantic web and due to their inherent complexity are often not adopted for other domains. On the other hand, industrial software development is still lacking support in the awareness of the importance of semantic technologies and therefore the need of such techniques is not pushed by the industry.

For the purpose of this thesis it was necessary to implement extensions to better integrate the ontology-based tool chain and future ontology-based tools and frameworks should be studied to identify potential new solution that increase the productiveness of the OBSD methodology. An open problem for future research is the improvement of semantic tools in order to simplify the knowledge processing.

The OBSD life-cycle showed its capabilities during the performed case study but still offers room for improvement. The list of products (AMIS, IDBS) which used the OBSD methodology will be expanded from the Remote Tower product [Eier et al., 2016] to other products not related to the ATM domain. OBSD influenced also other areas, like the public safety domain [Flachberger et al., 2015], [Flachberger and Gringinger, 2016] or led to other usages of semantic knowledge in projects like the FFG sponsored project SemNOTAM[2] or the just recently started BEST[3], a Horizon 2020 SESAR Exploratory Research project focusing on new SWIM data classification methodologies.

[2]http://semnotam.frequentis.com, accessed: 2016-12-16
[3]http://best.futurado.hu, accessed: 2016-12-16

APPENDIX A

List of Abbreviations

“學而不思則罔，思而不學則殆。”

[Confucius, 500]

ABox ..... Assertion Box
AGPL ..... Affero General Public License
AIAA ..... American Institute of Aeronautics and Astronautics
AIDX ..... Aviation Information Data Exchange
AIM ..... Aeronautical Information Management
AIRM ..... ATM Information Reference Model
AIS ..... Aeronautical Information Services
AIS-AIMSG .. Aeronautical Information Services-Aeronautical Information Management Study Group
AIXM ..... Aeronautical Information Exchange Model
AMDB ..... Airport Mapping Database
AMIS ..... Aerodrome Map Information Service
ANSP ..... Air Navigation Service Provider
API ..... Application Programming Interface
ARINC ..... Aeronautical Radio, Incorporated
ASCII ..... American Standard Code for Information Interchange
ATC ..... Air Traffic Control
ATM ..... Air Traffic Management
CANSO ..... Civil Air Navigation Services Organisation
CARATS ..... Collaborative Actions for Renovation of Air Traffic Systems
CDM ..... Collaborative Decision Making
COTS ..... Commercial-Off-The-Shelf
DAML ..... DARPA Agent Markup Language
DIG ..... DL Implementation Group
DL ..... Description Logic

DNOTAM . . . . . Digital NOtices To AirMen
EA . . . . . Enterprise Architecture
EAD . . . . . European AIS Database
EAEA . . . . . European Air Traffic Management Enterprise Architecture
EL . . . . . Expression Language
ESB . . . . . Enterprise Service Bus
EUROCAE . . . . . European Organisation for Civil Aviation Equipment
Eurocontrol . . . . . European Organisation for the Safety of Air Navigation
FAA . . . . . Federal Aviation Administration
FFG . . . . . Austrian Research Promotion Agency
FIXM . . . . . Flight Information Exchange Model
F-Logic . . . . . Frame Logic
GIS . . . . . Geographic Information System
GNU . . . . . GNU's Not Unix!
GML . . . . . Geography Markup Language
HMI . . . . . Human-Machine Interface
HTTP . . . . . Hyper Text Transfer Protocol
IATA . . . . . International Air Transport Association
ICAO . . . . . International Civil Aviation Organization
IDE . . . . . Integrated Development Environment
IEEE . . . . . Institute of Electrical and Electronics Engineers
IER . . . . . Information Exchange Requirements
ISRM . . . . . Information Service Reference Model
ISO . . . . . International Organization for Standardization
IRI . . . . . Internationalized Resource Identifier
IWXXM . . . . . ICAO Weather Information Exchange Model
JAR . . . . . Java ARchive
JPDO . . . . . Joint Planning and Development Office
JPL . . . . . Jet Propulsion Laboratory
KIF . . . . . Knowledge Interchange Format
KLOEC . . . . . Kilo Lines of Executable Code
LGPL . . . . . Lesser General Public License
LOC . . . . . Lines Of Code
METAR . . . . . Meteorological Terminal Air Report
MOF . . . . . Meta-Object Facility
N3 . . . . . Notation 3
NAF . . . . . NATO Architecture Framework
NASA . . . . . National Aeronautics and Space Administration
NATO . . . . . North Atlantic Treaty Organization
netCDF . . . . . network Common Data Form
NextGen . . . . . Next Generation Air Transportation System
NNEW . . . . . NextGen Network Enabled Weather
NOTAM . . . . . NOtices To AirMen

NOV ..... NATO Operational View
nRQL ..... Racer Query Language
NSOV ..... Service-Oriented Views
NSV ..... NATO System Views
OASIS ..... Organization for the Advancement of Structured Information Standards
OBSD ..... Ontology-Based Software Development
OFA ..... Operational Focus Area
OGC ..... Open Geospatial Consortium
OIL ..... Ontology Inference Layer
OMG ..... Object Management Group
OWL ..... Web Ontology Language
PEP ..... Policy Enforcement Points
QA ..... Quality Assurance
QoS ..... Quality of Service
RDF ..... Resource Description Framework
RDF(S) ..... RDF Schema
RIF ..... Rule Interchange Format
RTCA ..... Radio Technical Commission for Aeronautics
SE ..... Software Engineering
SESAR ..... Single European Sky ATM Research program
SHOE ..... Simple HTML Ontology Extensions
SIGMET ..... Significant Meteorological Information
SJU ..... SESAR Joint Undertaking
SOA ..... Service Oriented Architecture
SoaML ..... Service oriented architecture Modeling Language
SOAP ..... Simple Object Access Protocol
SPARQL ..... SPARQL Protocol and RDF Query Language
SQWRL ..... Semantic Query – Enhanced Web Rule Language
SWEET ..... Semantic Web for Earth and Environmental Technology
SWIM ..... System Wide Information Management
SWRL ..... Semantic Web Rule Language
T-Map ..... Transformation-Map
TAF ..... Terminal Aerodrome Forecast
TBox ..... Terminological Box
TCP/IP ..... Transmission Control Protocol / Internet Protocol
TRACON ..... Terminal Radar Approach Control
TRREE ..... Triple Reasoning and Rule Entailment Engines
Turtle ..... Terse RDF Triple Language
UML ..... Unified Modeling Language
URI ..... Uniform Resource Identifiers
UTF ..... UCS Transformation Format
W3C ..... World Wide Web Consortium
WMO ..... World Meteorological Organization

WP ..... Work Package
WSDL ..... Web Service Description Language
WSN ..... Web Services Notification
WXXM ..... Weather Information Exchange Model
XACML ..... eXtensible Access Control Markup Language
XHTML ..... Extensible HyperText Markup Language
XML ..... Extensible Markup Language
XP-EF ..... Extreme Programming Evaluation Framework

List of Figures

List of Tables

List of Algorithms

Bibliography

[Adida et al., 2008] Adida, B., Birbeck, M., McCarron, S., and Pemberton, S. (2008). *RDFa in XHTML: Syntax and Processing*. World Wide Web Consortium (W3C). http://www.w3.org/TR/rdfa-syntax/, accessed: 2016-11-13.

[AL-Badareen et al., 2011] AL-Badareen, A., Selamat, M., Jabar, M., Din, J., and Turaev, S. (2011). An evaluation model for software reuse processes. In Zain, J., Wan Mohd, W., and El-Qawasmeh, E., editors, *Software Engineering and Computer Systems*, volume 181 of *Communications in Computer and Information Science*, pages 586–599. Springer Berlin Heidelberg.

[Allemang and Hendler, 2008] Allemang, D. and Hendler, J. (2008). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann.

[Angles and Gutierrez, 2008] Angles, R. and Gutierrez, C. (2008). The expressive power of sparql. In Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., and Thirunarayan, K., editors, *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin Heidelberg.

[Aristotle, 2006] Aristotle (2006). *Aristotle: Metaphysics Theta: Translated with an Introduction and Commentary*. Oxford University Press.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F. (2003). *The description logic handbook: theory, implementation, and applications*. Cambridge University Press.

[Baclawski et al., 2002] Baclawski, K., Kokar, M. K., Kogut, P. A., Hart, L., Smith, J., Letkowski, J., and Emery, P. (2002). Extending the unified modeling language for ontology development. *Software and Systems Modeling*, 1(2):142–156.

[Basili, 1992] Basili, V. R. (1992). *Software modeling and measurement: the Goal/Question/Metric paradigm*. University of Maryland at College Park.

[Baumeister and Seipel, 2010] Baumeister, J. and Seipel, D. (2010). Anomalies in ontologies with rules. *Journal of Web Semantics*, 8(1):55–68.

[Beckett and Berners-Lee, 2008] Beckett, D. and Berners-Lee, T. (2008). *Turtle - Terse RDF Triple Language*. World Wide Web Consortium (W3C). http://www.w3.org/TeamSubmission/turtle/, accessed: 2016-11-13.

[Beckett and McBride, 2004] Beckett, D. and McBride, B. (2004). *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium (W3C). http://www.w3.org/TR/rdf-syntax-grammar/, accessed: 2016-11-13.

[Benjamin et al., 1996] Benjamin, J., Borst, P., Akkermans, H., and Wielinga, B. J. (1996). Ontology construction for technical domains. In *Proceedings of the 9th European Knowledge Acquisition Workshop on Advances in Knowledge Acquisition*. Springer-Verlag.

[Bleiholder and Naumann, 2008] Bleiholder, J. and Naumann, F. (2008). Data fusion. *ACM Computing Surveys (CSUR)*, 41(1):1–41. 1456651.

[Blomqvist et al., 2012] Blomqvist, P., van der Stricht, S., Helleblad, L., Pola, T., Wilson, S., Häggström, N., and Månström, M. (2012). *WP8, Internal Workflow including Modeling Artifacts, Edition 00.00.15*. Single European Sky ATM Research (SESAR) Program.

[Bray et al., 2008] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. World Wide Web Consortium (W3C). http://www.w3.org/TR/2008/REC-xml-20081126/, accessed: 2016-11-13.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium (W3C). http://www.w3.org/TR/PR-rdf-schema, accessed: 2016-11-27.

[Bruijn et al., 2006] Bruijn, J. d., Ehrig, M., Feier, C., Martíns-Recuerda, F., Scharffe, F., and Weiten, M. (2006). *Ontology Mediation, Merging, and Aligning*, pages 95–113. John Wiley and Sons, Ltd.

[Buhl et al., 2013] Buhl, H. U., Röglinger, M., Moser, F., and Heidemann, J. (2013). Big data. *Wirtschaftsinformatik*, 55(2):63–68.

[Burgstaller et al., 2016] Burgstaller, F., Steiner, D., Neumayr, B., Schrefl, M., and Gringinger, E. (2016). Using a model-driven, knowledge-based approach to cope with complexity in filtering of notices to airmen. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '16, pages 46:1–46:10, New York, NY, USA. ACM.

[Burgstaller et al., 2015] Burgstaller, F., Steiner, D., Schrefl, M., Gringinger, E., Wilson, S., and van der Stricht, S. (2015). Airm-based, fine-grained semantic filtering of notices to airmen. In *Integrated Communication, Navigation and Surveillance Conference (ICNS)*, pages D3–1–D3–13. Institute of Electrical and Electronics Engineers (IEEE).

[Caldiera and Basili, 1991] Caldiera, G. and Basili, V. R. (1991). Identifying and qualifying reusable software components. *Computer*, 24(2):61–70.

[Calero et al., 2006] Calero, C., Ruiz, F., and Piattini, M. (2006). *Ontologies for Software Engineering and Software Technology*. Springer-Verlag.

[CANSO, 2013] CANSO (2013). *The Transition from AIS to AIM*. Civil Air Navigation Services Organisation (CANSO). https://en.wikiquote.org/w/index.php?title=Winston_Churchill&oldid=2196223, accessed: 2016-11-24.

[Catenazzi et al., 2009] Catenazzi, N., Sommaruga, L., and Mazza, R. (2009). User-friendly ontology editing and visualization tools: The owleasyviz approach. In *Information Visualisation, 2009 13th International Conference*, pages 283–288.

[Chandrasekaran et al., 1999] Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999). What are ontologies, and why do we need them? *Intelligent Systems and their Applications, IEEE*, 14(1):20–26.

[Churchill, 1942] Churchill, W. (1942). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Winston_Churchill&oldid=2196223.

[Clarke, 1973] Clarke, A. C. (1973). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Arthur_C._Clarke&oldid=2192351.

[Confucius, 500] Confucius (BC 500). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Confucius&oldid=2180119.

[Cruellas and Roelants, 2013] Cruellas, P. and Roelants, E. (2013). *08.01.01, SWIM Concept of Operations, Edition 00.03.03*. Single European Sky ATM Research (SESAR) Program.

[Di Crescenzo et al., 2010] Di Crescenzo, D., Strano, A., and Trausmuth, G. (2010). System wide information management: The swim-suit prototype. In *Integrated Communications Navigation and Surveillance Conference (ICNS), 2010*, pages C2–1–C2–13.

[Dillon et al., 2008] Dillon, T., Chang, E., and Wongthongtham, P. (2008). Ontology-based software engineering- software engineering 2.0. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 13–23.

[Eco, 1957] Eco, U. (1957). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Umberto_Eco&oldid=2192739.

[Edison, 1903] Edison, T. (1903). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Thomas_Edison&oldid=2194506.

[Ehrig, 2010] Ehrig, M. (2010). *Ontology Alignment: Bridging the Semantic Gap*. Springer Publishing Company, Incorporated, 1st edition.

[Eier et al., 2016] Eier, D., Gringinger, E., and Klopf, M. (2016). Semantic information management in a SWIM enabled remote tower environment. In *Integrated Communications Navigation and Surveillance (ICNS)*, pages 1–18. Institute of Electrical and Electronics Engineers (IEEE).

[Einstein, 1955] Einstein, A. (1955). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Albert_Einstein&oldid=2191410.

[EUROCAE, 2011a] EUROCAE (2011a). *ED-119B - Interchange Standards for Terrain, Obstacle, And Aerodrome Mapping Data, Edition final*. European Organisation for Civil Aviation Equipment (EUROCAE).

[EUROCAE, 2011b] EUROCAE (2011b). *ED-99C - User Requirements for Aerodrome Mapping Information, Edition final*. European Organisation for Civil Aviation Equipment (EUROCAE).

[Eurocontrol, 2009] Eurocontrol (2009). *Strategic Guidance in Support of the Execution of the European ATM Master Plan, Edition 1*. Eurocontrol. http://www.eurocontrol.int/sites/default/files/publication/files/surveillance-strategic-guidance-support-of-the-execution-european-atm-master-plan-20090513.pdf, accessed: 2016-11-17.

[Eurocontrol, 2010] Eurocontrol (2010). *Aeronautical Information Exchange Model (AIXM), Version 5.1, Edition final*. Eurocontrol.

[FAA, 2010] FAA (2010). *Web Service Description Document - Flight Plan Service (FPS), Revision A*. Federal Aviation Administration (FAA).

[Fensel et al., 2001] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., and Patel-Schneider, P. F. (2001). Oil: an ontology infrastructure for the semantic web. *Intelligent Systems, IEEE*, 16(2):38–45.

[Flachberger and Gringinger, 2016] Flachberger, C. and Gringinger, E. (2016). Decision support for networked crisis & disaster management – a comparison with the air traffic management domain. In Tapia, A., Antunes, P., Bañuls, V. A., Moore, K., and Porto, J., editors, *ISCRAM 2016 Conference Proceedings – 13th International Conference on Information Systems for Crisis Response and Management*. Federal University of Rio de Janeiro, Federal University of Rio de Janeiro.

[Flachberger et al., 2015] Flachberger, C., Gringinger, E., and Obritzhauser, T. (2015). Collaboration in crisis management – learning from the transportation domain. In *10th Security Research Conference »Future Security«*.

[Gašević et al., 2009] Gašević, D., Kaviani, N., and Milanović, M. (2009). Ontologies and software engineering. In Staab, S. and Studer, R., editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 593–615. Springer Berlin Heidelberg.

[Gartner, 2012] Gartner (2012). *The Importance of 'Big Data': A Definition*. https://www.gartner.com/doc/2057415/importance-big-data-definition, accessed: 2016-11-17.

[Genesereth et al., 1992] Genesereth, M., Fikes, R. E., Brachman, R., Gruber, T., Hayes, P., Letsinger, R., Lifschitz, V., Macgregor, R., Mccarthy, J., Norvig, P., and Patil, R. (1992). *Knowledge Interchange Format Version 3.0 Reference Manual*. Stanford Logic Group.

[Glimm et al., 2013] Glimm, B., Ogbuji, C., Hawke, S., Herman, I., Parsia, B., Polleres, A., and Seaborne, A. (2013). *SPARQL 1.1 Entailment Regimes*. World Wide Web Consortium (W3C). https://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/, accessed: 2016-11-13.

[Gomez-Perez and Corcho, 2002] Gomez-Perez, A. and Corcho, O. (2002). Ontology languages for the semantic web. *Intelligent Systems, IEEE*, 17(1):54–60.

[Grau et al., 2008] Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., and Sattler, U. (2008). Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322.

[Gregor and Hevner, 2013] Gregor, S. and Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact. *Management Information Systems (MIS) Quarterly*, 37(2):337–356.

[Gringinger, 2014] Gringinger, E. (2014). Ontology-based representation and semantic querying of digital notices to airmen. In *Integrated Communications, Navigation and Surveillance Conference (ICNS) Conference Proceedings*, pages 1–23. Institute of Electrical and Electronics Engineers (IEEE).

[Gringinger, 2017] Gringinger, E. (2017). *Ontology-based Software Development in the ATM domain*. Single European Sky ATM Research Joint Undertaking (SJU).

[Gringinger et al., 2010a] Gringinger, E., Eier, D., and Merkl, D. (2010a). A concept for semantic-based information management for control room development. In *International Council of the Aeronautical Sciences (ICAS)*.

[Gringinger et al., 2010b] Gringinger, E., Eier, D., and Merkl, D. (2010b). Ontology-based cns software development. In *Integrated Communications Navigation and Surveillance Conference (ICNS)*, pages C3–1–C3–13. Institute of Electrical and Electronics Engineers (IEEE).

[Gringinger et al., 2011] Gringinger, E., Eier, D., and Merkl, D. (2011). Nextgen and sesar moving towards ontology-based software development. In *Integrated Communications, Navigation and Surveilance Conference (ICNS)*, pages H3–1–H3–10. Institute of Electrical and Electronics Engineers (IEEE).

[Gringinger et al., 2012a] Gringinger, E., Merkl, D., and Graf, G. (2012a). How semantic technologies enrich aeronautical information management for ontology-based software development. In *International Council of the Aeronautical Sciences (ICAS)*.

[Gringinger et al., 2013] Gringinger, E., Milchrahm, H., Andersson, M., and Guerrero, E. (2013). Verification, validation, and demonstration of an aerodrome map information service. In *Integrated Communications, Navigation and Surveillance Conference (ICNS)*, pages 1–16. Institute of Electrical and Electronics Engineers (IEEE).

[Gringinger et al., 2012b] Gringinger, E., Trausmuth, G., Balaban, A., Jahn, J., and Milchrahm, H. (2012b). Experience report on successful demonstration of swim by three industry partners. In *Integrated Communications, Navigation and Surveillance Conference*, pages G6–1–G6–8. Institute of Electrical and Electronics Engineers (IEEE).

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220.

[Gruber, 1995] Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928.

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

[Guohua et al., 2007] Guohua, S., Zhiqiu, H., Xiaodong, Z., Lei, W., and Gaoyou, X. (2007). Using description logics reasoner for ontology matching. In *Intelligent Information Technology Application, Workshop on*, pages 30–33.

[Haag et al., 1997] Haag, S., Cummings, M., and Dawkins, J. (1997). *Management Information Systems for the Information Age*. McGraw-Hill, Inc.

[Haarslev et al., 2004] Haarslev, V., Möller, R., and Wessel, M. (2004). Querying the semantic web with racer + nrql. In *Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04)*.

[Halevy, 2005] Halevy, A. (2005). Why your data won't mix. *Queue*, 3(8):50–58.

[Hamlet, 2010] Hamlet, D. (2010). *Composing Software Components - A Software-testing Perspective*. Springer-Verlag New York, Inc., New York, NY, USA.

[Hamza, 2009] Hamza, H. (2009). A framework for identifying reusable software components using formal concept analysis. In *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on*, pages 813–818.

[Happel and Seedorf, 2006] Happel, H. J. and Seedorf, H. (2006). Applications of ontologies in software engineering. In *Proc. of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006)*.

[Harris et al., 2013] Harris, S., Seaborne, A., and Prud'hommeaux, E. (2013). *SPARQL 1.1 Query Language*. World Wide Web Consortium (W3C). https://www.w3.org/TR/2013/REC-sparql11-query-20130321/, accessed: 2016-11-13.

[Hayes, 1990] Hayes, P. J. (1990). *The second naive physics manifesto*, pages 46–63. Morgan Kaufmann Publishers Inc.

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

[Hendler and McGuinness, 2000] Hendler, J. and McGuinness, D. (2000). The darpa agent markup language. *IEEE Intelligent Systems*, 15(6):67–73.

[Hesse, 2008] Hesse, W. (2008). *Engineers Discovering the "Real World" — From Model-Driven to Ontology-Based Software Engineering*, pages 136–147. Springer-Verlag.

[Hevner and Chatterjee, 2010] Hevner, A. R. and Chatterjee, S. (2010). *Design Science Research in Information Systems*, volume 22 of *Integrated Series in Information Systems*, chapter 2, pages 9–22. Springer US.

[Hevner et al., 2004] Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Q.*, 28(1):75–105.

[Hitzler et al., 2012] Hitzler, P., Parsia, B., Patel-schneider, P. F., and Rudolph, S. (2012). *OWL2 Web Ontology Language Primer*. World Wide Web Consortium (W3C). http://www.w3.org/2012/pdf/REC-owl2-primer-20121211.pdf, accessed: 2016-08-13.

[Horridge and Musen, 2016] Horridge, M. and Musen, M. (2016). Snap-sparql: A java framework for working with sparql and owl. In *Revised Selected Papers of the 12th International Experiences and Directions Workshop on Ontology Engineering - Volume 9557*, pages 154–165, New York, NY, USA. Springer-Verlag New York, Inc.

[Horrocks et al., 2006] Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible sroiq. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67. AAAI Press.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). 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(1):7–26.

[ICAO, 2005] ICAO (2005). *Global Air Traffic Management Operational Concept - Doc 9854 AN/458*. International Civil Aviation Organization (ICAO).

[ICAO, 2010a] ICAO (2010a). *Aeronautical Information Services-Aeronautical Information Management Study Group (AIS-AIMSG)/3-SN No. 6 - Appendix A - AIM Definitions*. International Civil Aviation Organization (ICAO). http://www.icao.int/safety/ais-aimsg/AISAIM%20Meeting%20MetaData/AIS-AIMSG%203/SN%206%20AppA%20rev.doc, accessed: 2016-10-27.

[ICAO, 2010b] ICAO (2010b). *Annex 15 - Aeronautical Information Services, Edition 13*. International Civil Aviation Organization (ICAO).

[Jie-ning et al., 2009] Jie-ning, W., RongRong, B., Xiaohao, X., and Xinsheng, Y. (2009). Ontology-based parameterized aerodrome modelling. In *Fifth International Conference on Semantics, Knowledge and Grid*, pages 440–441.

[JPDO, 2011] JPDO (2011). *Concept of Operations for the Next Generation Air Transportation System, Version 3.2*. Joint Planning and Development Office (JPDO).

[Kai and Steele, 2009] Kai, Y. and Steele, R. (2009). Ontology mapping based on concept classification. In *Digital Ecosystems and Technologies, 2009. DEST '09. 3rd IEEE International Conference on*, pages 656–661.

[Keller, 2016] Keller, R. M. (2016). Ontologies for aviation data management. In *IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–9.

[Kifer and Lausen, 1997] Kifer, M. and Lausen, G. (1997). F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*. ACM.

[Kifer et al., 1995] Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843.

[Kiryakov et al., 2005] Kiryakov, A., Ognyanov, D., and Manov, D. (2005). Owlim - a pragmatic semantic repository for owl. In Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., and Sheng, Q., editors, *Web Information Systems Engineering - WISE 2005 Workshops*, volume 3807 of *Lecture Notes in Computer Science*, pages 182–192. Springer Berlin Heidelberg.

[Klyne and Carroll, 2004] Klyne, G. and Carroll, J. (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium (W3C). http://www.w3.org/TR/rdf-concepts/, accessed: 2016-11-22.

[Konev et al., 2008] Konev, B., Lutz, C., Walther, D., and Wolter, F. (2008). Logical difference and module extraction with cex and mex. In Baader, F., Lutz, C., and Motik, B., editors, *Description Logics*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Lanzenberger et al., 2009a] Lanzenberger, M., Sampson, J., and Rester, M. (2009a). Visualization in ontology tools. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, pages 705–711.

[Lanzenberger et al., 2009b] Lanzenberger, M., Sampson, J., and Rester, M. (2009b). Visualization in ontology tools. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, pages 705–711.

[Layman et al., 2004a] Layman, L., Williams, L., and Cunningham, L. (2004a). Exploring extreme programming in context: An industrial case study. In *Proceedings of the Agile Development Conference*, ADC '04, pages 32–41, Washington, DC, USA. IEEE Computer Society.

[Layman et al., 2004b] Layman, L., Williams, L., and Cunningham, L. (2004b). Motivations and measurements in an agile case study. In *Proceedings of the 2004 Workshop on Quantitative Techniques for Software Agile Process*, QUTE-SWAP '04, pages 14–24, New York, NY, USA. ACM.

[Lee and Connolly, 2008] Lee, B. and Connolly, D. (2008). *Notation 3 (N3) A readable RDF syntax*. World Wide Web Consortium (W3C). http://www.w3.org/TeamSubmission/n3/, accessed: 2016-11-13.

[Luckenbaugh et al., 2007] Luckenbaugh, G., Dehn, J., Rudolph, S., and Landriau, S. (2007). Service oriented architecture for the next generation air transportation system. In *Integrated Communications, Navigation and Surveillance Conference (ICNS)*, pages 1–9.

[Lung et al., 2007] Lung, C.-H., Urban, J. E., and Mackulak, G. T. (2007). Analogy-based domain analysis approach to software reuse. *Requirements Engineering*, 12(1):1–22.

[McCarthy, 1987] McCarthy, J. (1987). *Circumscription, a form of non-monotonic reasoning*, pages 145–152. Morgan Kaufmann Publishers Inc.

[McLeod and MacDonell, 2011] McLeod, L. and MacDonell, S. G. (2011). Factors that affect software systems development project outcomes: A survey of research. *ACM Computing Surveys*, 43(4):24:1–24:56.

[Motik et al., 2009a] Motik, B., Cuenca, G. B., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. (2009a). *OWL 2 Web Ontology Language Profiles*. World Wide Web Consortium (W3C). http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/, accessed: 2016-11-13.

[Motik et al., 2009b] Motik, B., Patel-Schneider, P. F., and Parsia, B. (2009b). *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax*. World Wide Web Consortium (W3C). http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/, accessed: 2016-10-27.

[Motik et al., 2007] Motik, B., Shearer, R., and Horrocks, I. (2007). Optimized reasoning in description logics using hypertableaux. In Pfenning, F., editor, *Automated Deduction - CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 67–83. Springer Berlin Heidelberg.

[NATO, 2010] NATO (2010). *NATO Architecture Framework 3.1*. North Atlantic Treaty Organization (NATO).

[O'Connor and Das, 2011] O'Connor, M. and Das, A. (2011). *A Method for Representing and Querying Temporal Information in OWL*, volume 127 of *Communications in Computer and Information Science*, chapter 8, pages 97–110. Springer Berlin Heidelberg.

[O'Connor et al., 2005] O'Connor, M., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W., and Musen, M. (2005). Supporting rule system interoperability on the semantic web with swrl. In Gil, Y., Motta, E., Benjamins, V., and Musen, M., editors, *The Semantic Web - ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 974–986. Springer Berlin Heidelberg.

[O'Connor and Das, 2008] O'Connor, M. J. and Das, A. K. (2008). Sqwrl: A query language for owl. In Hoekstra, R. and Patel-Schneider, P. F., editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[OGC, 2002a] OGC (2002a). *OpenGIS Web Feature Service Implementation Specification 1.1*. Open Geospatial Consortium (OGC). https://portal.opengeospatial.org/files/?artifact_id=8339, accessed: 2016-11-27.

[OGC, 2002b] OGC (2002b). *Web Map Service Implementation Specification 1.1.1*. Open Geospatial Consortium (OGC). https://portal.opengeospatial.org/files/?artifact_id=39967, accessed: 2016-11-27.

[OGC, 2006] OGC (2006). *OpenGIS Web Map Server Implementation Specification 1.3.0*. Open Geospatial Consortium (OGC). http://portal.opengeospatial.org/files/?artifact_id=14416, accessed: 2016-11-27.

[OGC, 2007] OGC (2007). *OpenGIS Geography Markup Language (GML) Encoding Standard, version 3.2.1*. Open Geospatial Consortium (OGC). http://portal.opengeospatial.org/files/?artifact_id=20509, accessed: 2016-11-27.

[OGC, 2010] OGC (2010). *OpenGIS Web Feature Service 2.0 Interface Standard*. Open Geospatial Consortium (OGC). https://portal.opengeospatial.org/files/?artifact_id=39967, accessed: 2016-11-27.

[OMG, 2012] OMG (2012). *Service oriented architecture Modeling Language (SoaML) 1.0.1*. Object Management Group (OMG). www.omg.org/spec/SoaML/1.0.1/, accessed: 2016-11-27.

[Oppy, 2007] Oppy, G. (2007). *Ontological Arguments*, pages 145–152. Zalta, E. N.

[Papazoglou and Heuvel, 2007] Papazoglou, M. P. and Heuvel, W.-J. (2007). Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415.

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). *OWL Web Ontology Language Semantics and Abstract Syntax*. World Wide Web Consortium (W3C). http://www.w3.org/TR/2004/REC-owl-semantics-20040210/, accessed: 2016-12-13.

[Patel-Schneider et al., 2008] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2008). *OWL 1.1 web ontology language structural specification and functional-style syntax*. World Wide Web Consortium (W3C). http://www.w3.org/TR/2008/WD-owl11-syntax-20080108/ organization, accessed: 2016-08-13.

[Peffers et al., 2006] Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., and Bragge, J. (2006). The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. In *1st International Conference on Design Science in Information Systems and Technology (DESRIST)*, pages 83–106.

[Pola and Solberg, 2013] Pola, T. and Solberg, B. (2013). *08.03.10, ISRM Foundation Rulebook, Edition 00.02.00*. Single European Sky ATM Research (SESAR) Program.

[Pérez et al., 2006] Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and complexity of sparql. In *International Semantic Web Conference (ISWC)*, page 30–43.

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). *SPARQL - Query Language for RDF*. World Wide Web Consortium (W3C). http://www.w3.org/TR/rdf-sparql-query/, accessed: 2016-11-13.

[Pschierer and Schiefele, 2007] Pschierer, C. and Schiefele, J. (2007). Open standards for airport databases - arinc 816. In *Digital Avionics Systems Conference (DASC) IEEE/AIAA 26th*, pages 2.B.6–1–2.B.6–8.

[Reiss et al., 2006] Reiss, M., Moal, M., Barnard, Y., and Froger, A. (2006). Using ontologies to conceptualize the aeronautic domain. In *International Conference on Human-Computer Interaction in Aeronautics*, pages 55–63. Cépaduès-Editions, Toulouse, France.

[RTCA, 2011a] RTCA (2011a). *DO-272C - User Requirements for Aerodrome Mapping Information, Edition final*. Radio Technical Commission for Aeronautics (RTCA).

[RTCA, 2011b] RTCA (2011b). *DO-291B - Minimum Interchange Standards for Terrain, Obstacle and Aerodrome Mapping Data, Edition final*. Radio Technical Commission for Aeronautics (RTCA).

[Rus and Lindvall, 2002] Rus, I. and Lindvall, M. (2002). Knowledge management in software engineering. *Software, IEEE*, 19(3):26–38.

[Sametinger, 1997] Sametinger, J. (1997). *Software Engineering with Reusable Components*. Springer-Verlag New York, Inc., New York, NY, USA.

[Seifert et al., 2008] Seifert, C., Kump, B., Kienreich, W., Granitzer, G., and Granitzer, M. (2008). On the beauty and usability of tag clouds. In *Information Visualisation, 2008. IV '08. 12th International Conference*, pages 17–25.

[Seneca, 65] Seneca, L. A. (65). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Seneca_the_Younger&oldid=2191743.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55.

[Siricharoen, 2007] Siricharoen, W. V. (2007). Ontologies and software engineering. In Shi, Y., Albada, G. D., Dongarra, J., and Sloot, P. M., editors, *Computational Science - ICCS 2007*, volume 4488 of *Lecture Notes in Computer Science*, pages 1155–1161. Springer Berlin Heidelberg.

[Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics*, 5(2):51–53.

[SJU, 2012] SJU (2012). *Program Management Plan, Edition 02.00.00*. Single European Sky ATM Research Joint Undertaking (SJU).

[SJU, 2015] SJU (2015). *European Air Traffic Management Master Plan, 2015 Edition*. Single European Sky ATM Research Joint Undertaking (SJU). https://www.atmmasterplan.eu/, accessed: 2016-11-17.

[Smith et al., 2004] Smith, M. K., Welty, C., and McGuinness, D. L. (2004). *OWL Web Ontology Language Guide*. World Wide Web Consortium (W3C). http://www.w3.org/TR/2004/REC-owl-guide-20040210/, accessed: 2016-11-24.

[Spencer and Hill, 1981] Spencer, B. and Hill, T. (1981). accessed: 2016-11-27. http://www.imdb.com/title/tt0085327/.

[Steiner et al., 2016a] Steiner, D., Burgstaller, F., Gringinger, E., Schrefl, M., and Kovacis, I. (2016a). In-flight provisioning and distribution of atm information. In *International Council of the Aeronautical Sciences (ICAS)*.

[Steiner et al., 2016b] Steiner, D., Kovacic, I., Burgstaller, F., Schrefl, M., Friesacher, T., and Gringinger, E. (2016b). Semantic enrichment of dnotams to reduce information overload in pilot briefings. In *Integrated Communications Navigation and Surveillance (ICNS)*, pages 6B2–1–6B2–13. Institute of Electrical and Electronics Engineers (IEEE).

[Stoilos et al., 2010] Stoilos, G., Grau, B. C., and Horrocks, I. (2010). How incomplete is your semantic web reasoner? In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 10)*.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

[Takeda et al., 1990] Takeda, H., Veerkamp, P., Tomiyama, T., and Yoshikawa, H. (1990). Modeling design processes. *AI Magazine*, 11(4):37–48.

[Tetlow et al., 2006] Tetlow, P., Pan, J. Z., Oberle, D., Wallace, E., Uschold, M., and Kendall, E. (2006). *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering*. World Wide Web Consortium (W3C). http://www.w3.org/2001/sw/BestPractices/SE/ODA/, accessed: 2016-11-17.

[Trausmuth and Klopf, 2010] Trausmuth, G. and Klopf, M. (2010). Evolutionary adaptation of atm systems for swim. In *Integrated Communications, Navigation and Surveilance Conference (ICNS), 2011*, pages E4–1–E4–9.

[Tsarkov and Horrocks, 2006] Tsarkov, D. and Horrocks, I. (2006). Fact++ description logic reasoner: System description. *Lecture Notes in Artificial Intelligence*, 4130:292–297.

[Ulfbratt et al., 2008] Ulfbratt, E., Systems, S., McConville, J., and May, C. (2008). *Comparison of the SESAR and NextGen - Concepts of Operations*. NCOIC Aviation IPT.

[Vaishnavi and Kuechler, 2013] Vaishnavi, V. and Kuechler, B. (2013). *Design Science Research in Information Systems*. http://desrist.org/design-research-in-information-systems/, accessed: 2013-12-06.

[van Harmelen et al., 2001] van Harmelen, F., Schneider, P., and Horrocks, I. (2001). *Reference description of the DAML+OIL ontology markup language*. http://www.daml.org/2001/03/reference.html, accessed: 2016-12-16.

[van Meenen et al., 2013] van Meenen, J., Balaban, A., Baz, D. M., Javier, F., and Souami, H. (2013). *14.01.03, SWIM Architectural Definition for Iteration 2.0, edition 00.01.00*. Single European Sky ATM Research (SESAR) Program.

[Vaudrey, 2013] Vaudrey, T. (2013). *WPB04.01, EAEA Guidance Material, Edition 00.00.10*. Single European Sky ATM Research (SESAR) Program.

[von Braun, 1957] von Braun, W. (1957). accessed: 2016-11-27. https://en.wikiquote.org/w/index.php?title=Wernher_von_Braun&oldid=2085988.

[Wongthongtham et al., 2009] Wongthongtham, P., Chang, E., Dillon, T., and Sommerville, I. (2009). Development of a software engineering ontology for multisite software development. *Knowledge and Data Engineering, IEEE Transactions on*, 21(8):1205–1217.

[Wongthongtham et al., 2008] Wongthongtham, P., Dillon, D., Dillon, T., and Chang, E. (2008). Use of uml 2.1 to model multi-agent systems based on a goal-driven software engineering ontology. In *Semantics, Knowledge and Grid, 2008. SKG '08. Fourth International Conference on*, pages 428–432.

[Zdun et al., 2007] Zdun, U., Hentrich, C., and Dustdar, S. (2007). Modeling process-driven and service-oriented architectures using patterns and pattern primitives. *ACM Transactions on the Web (TWEB)*, 1(3).

APPENDIX B

Curriculum Vitæ

"Sciant quae optima sunt esse communia."

[Seneca, 65]

Education

*with distinction

since 2009	Ph.D. student in Computer Science Vienna University of Technology, Austria
2006/07 - 2008/03	Master of Computer Science Management, M.Soc.Ec.Sc.* Vienna University of Technology, Austria
2006/07 - 2007/10	Master of Computer Science, M.Sc.* Vienna University of Technology, Austria
2001/10 - 2006/06	Bachelor of Media and Computer Science, B.Sc.* Vienna University of Technology, Austria
1993/09 - 2001/07	Graduation (Matura) Sport BRG Wallererstraße, Wels, Austria

Work Experience

since 2016/02	Member of EUROCAE WG-104 - SWIM Services The European Organisation for Civil Aviation Equipment
since 2014/05	Expert for European Commission and its Executive Agencies/Bodies Horizon 2020 - The EU Framework Programme for Research and Innovation
since 2014/02	Member of EUROCAE WG-76 - AIS/MET Datalink Applications The European Organisation for Civil Aviation Equipment
since 2013/08	Task Team on Aviation XML (AvXML/IWXXM) World Meteorological Organization
since 2011/01	Senior Expert & Scientist Corporate Research, Frequentis AG
since 2011/01	Lecturer, Basic Principles of Control Center Solutions The Vienna University of Technology
2000/01 - 2016/05	IT Consultant, Freelancer Eduard Gringinger Ges.m.b.H. und Co.KG
2008/03 - 2011/01	Executive Management Assistant to the CEO Frequentis AG
2005/04 - 2006/06	Software Engineer, Self-Employee AKH Wien, Medical University of Vienna
2002/03 - 2005/03	Press Guide and Media Activities Iriepathie, Irievibrations Label

Research Grants

- SESAR 2020 Pj 19 - ATM systems and services (SESAR 2020, 2016-2020)
- BEST - Achieving the BEnefits of SWIM by making smart use of Semantic Technologies (Horizon 2020, Exploratory Research Call, 2016-2018)
- SemanticNOTAMs - Ontology-based representation and semantic querying of Digital Notices to Airmen (FFG, TakeOff-Call, 2013-2017)
- SESAR Project 08.01.03 - ATM Information Reference Model (SESAR 1, 2013-2016)
- SESAR Project 08.03.10 - Information Service Reference Model (SESAR 1, 2013-2016)
- SESAR Project 08.01.04 - Aeronautical Information (SESAR 1, 2010-2013)
- SESAR Project 08.01.06 - Meteorological Information (SESAR 1, 2010-2013)
- SESAR Project 08.01.08 - Environment Information (SESAR 1, 2010-2013)
- SESAR Project 08.01.10 - Airport Information (SESAR 1, 2010-2013)
- SESAR Project 08.03.03 - Aeronautical Information Services (SESAR 1, 2010-2013)

Publications

*best paper award, **innovative idea award

- Gringinger E, Ontology-based Software Development in the ATM domain. Proceedings of the 7^{th} SESAR Innovation Days, 2017. *Under review.*

- Steiner D, Burgstaller F, Gringinger E, Schrefl M, Kovacis I. In-Flight Provisioning and Distribution of ATM Information. International Council of Aeronautical Sciences (ICAS) Conference, Daejeon, South Korea, 2016.

- Flachberger C, Gringinger E. Decision Support for Networked Crisis & Disaster Management – a Comparison with the Air Traffic Management Domain. Intelligent Decision Support in the Networked Society Proceedings of the ISCRAM 2016 Conference, Rio de Janeiro, Brazil, 2016.

- Steiner D, Kovacis I, Burgstaller F, Schrefl M, Friesacher T, Gringinger E. Semantic Enrichment of DNOTAMs to Reduce Information Overload in Pilot Briefings. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2016.

- Eier D, Gringinger E, Klopf M. Semantic Information Management in a SWIM enabled Remote Tower Environment. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2016.

- Steiner D, Kovacis I, Burgstaller F, Schrefl M, Friesacher T, Gringinger E. Coping with Complexity in SemNOTAM - A Model-driven Approach to Knowledge-based Filtering of Notices to Airmen. Asia-Pacific Conference on Conceptual Modelling (APCCM) Conference, Canberra, Australia, 2016.

- Flachberger C, Gringinger E, Obritzhauser T. Collaboration in Crisis Management – Learning from the Transportation Domain. Future Security Conference, Berlin, Germany, 2015.

* Burgstaller F, Steiner D, Gringinger E, Schrefl M, Wilson S, van der Stricht S. AIRM-based fine-grained semantic filtering of Notices to Airmen. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2015.

- Gringinger E. Ontology-based representation and semantic querying of Digital Notices to Airmen. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2014.

- Gringinger E, Anderson M, Guerrero E, Milchrahm H. Verification, Validation and Demonstration of an Aerodrome Map Information Service. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2013.

- Gringinger E, Merkl D, Graf G. How Semantic Technologies enrich Aeronautical Information Management for Ontology Based Software Development. International Council of the Aeronautical Sciences (ICAS), Brisbane, Australia, 2012.

- Gringinger E, Trausmuth G, Balaban A, Jahn J, Milchrahm H. Experience report on successful demonstration of SWIM by three industry partners. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2012.
- Gringinger E, Eier D, Merkl D. NextGen and SESAR moving towards ontology-based software development. Integrated Communications Navigation and Surveillance (ICNS) Conference, Washington D.C, USA, 2011.
- Gringinger E, Merkl D, Eier D. A Concept for Semantic-based Information Management for Control Room Development. International Council of the Aeronautical Sciences (ICAS), Nice, France, 2010.

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

- Gringinger E., Trieb N. Clinical Trials Moving Towards Online Solutions. VDM Verlag Dr. Müller, ISBN 3836480530, 2008.

Patents

- Method for Displaying Relevant Information in an Aircraft. Europe, Patent/Application Number EP 3118839 A1, 2016.