ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Grigorian** Jméno: **Artem** Osobní číslo: **452981**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Sémantické služby pro úkoly v doméně plánování údržby letadel

Název diplomové práce anglicky:

Semantic services for tasks in aircraft maintenance planning domain

Pokyny pro vypracování:

AMOS [1] is complex software for management of processes within organization CSAT [2]. It solves many generic tasks, however has serious limitations with domain specific ones.
The goal of this thesis is to create a set of services that would easily integrate with AMOS like systems and provide services for domain specific tasks. Input of such services would come from lightweight and simple integration with the system and domain expert knowledge represented in spreadsheets. The thesis addresses issues with poor-quality non-structured data that often occurs in such systems. Based on possibly denormalized data of the system it will provide extraction of events within processes of the organization.
It is assumed that the tool SPipes [3] will be used for data processing and services will be based on semantic web technologies.
Instructions:
1) get acquainted with the technologies of semantic web for knowledge representation (RDF, RDFS, OWL, JSONLD), querying (SPARQL) and persistence (triplestores)
2) analyze relevant tools for transforming and mapping relational data to RDF
3) analyze existing datasets and related processes within the organization
4) define requirements and scenarios for the services, design the system
5) implement a prototype
6) test prototype on defined scenarios with domain experts

Seznam doporučené literatury:

[1] AMOS, Swiss Aviation Software (https://www.swiss-as.com/amos-mro)
[2] Czech Airlines Technics a.s. (https://www.csatechnics.com/)
[3] SPipes, Semantic pipeline language and engine (https://github.com/kbss-cvut/s-pipes)
[4] Guizzardi, Giancarlo. "Ontological foundations for structural conceptual models." (2005).
[5] Hert, Matthias, Gerald Reif, and Harald C. Gall. "A comparison of RDB-to-RDF mapping languages." Proceedings of the 7th international conference on semantic systems. 2011.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Miroslav Blaško, Ph.D., skupina znalostních softwarových systémů

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **27.07.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce: **19.02.2023**

Mgr. Miroslav Blaško, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

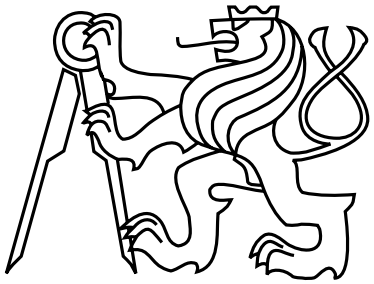
III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Semantic services for tasks in aircraft maintenance planning domain

Bc. Artem Grigorian

Supervisor: Mgr. Miroslav Blaško, Ph.D.

Study Programme: Open Informatics, Master

Field of Study: Software Engineering

January 4, 2022

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

Prague, date ....

Abstract

The goal of this thesis is to design and develop a set of ontology-based services that are capable of providing domain-specific functionalities and that can be easily integrated with MRO software systems, particularly AMOS.

Keywords: Web Application, JavaScript, TTL, Ontology, Maintenance, AMOS, Semantic Web

Abstrakt

Cílem této práce je navrhnout a vyvinout sadu služeb založených na ontologii, které jsou schopny poskytovat funkce specifické pro danou oblast a které lze snadno integrovat se softwarovými systémy MRO, zejména se systémem AMOS.

Klíčová slova: Webová aplikace, JavaScript, TTL, Ontologie, Údržba, AMOS, Sémantický web

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Currently, ontologies are becoming more prominent on the Internet. They are being used more and more frequently as an accessible tool for the exchange and expression of information by specialists in their fields, thus ceasing to be a purely academic endeavor. Many disciplines are now developing both general-purpose ontologies and standardized ontologies to formulate and express knowledge in their fields. One of their core principles is the definition of a common vocabulary for the exchange of information, including the specification of the basic concepts of a given domain and the relationships between them.

In specialized domains, ontologies are developed to share the understanding of information structure, to provide reuse of domain knowledge, and to separate domain and operational knowledge. Furthermore, the ontology-based approach for domain knowledge specification can significantly simplify the process of establishing a communication process with domain experts. [1]

A great example of a field where ontologies can be used is an aircraft maintenance domain since it has a relatively low digitalization level. Moreover, the generalized knowledge can be applied to most aircraft maintenance organizations. [2] One of such organizations is Czech Airlines Technics (CSAT), a subsidiary of Prague Airport which is mainly focused on aircraft repair and maintenance, and offers a wide extent of services on Maintenance, Repair, and Operations (MRO) market. [3]

AMOS is a comprehensive MRO software solution designated for solving a great variety of clients' maintenance, engineering, and logistics needs. CSAT uses AMOS for the management of their MRO-related processes to ensure the conditions for the provision of airworthiness services to the high standard required by national aviation authorities, operators, and aircraft owners. Despite the fact that AMOS is designed to provide the full range of MRO-related functionality needs, it does not cover entirely all processes within CSAT. More importantly, in the majority of the domain-specific tasks within maintenance planning and execution, it only provides support to persist important information for those tasks. It is left to domain experts to process stored information manually and optimize those tasks using their expert knowledge. On the other hand, ontology-based systems are well suited to incorporate expert knowledge for the optimization of such tasks. For example, while planning or executing maintenance tasks, the knowledge of how parts of the aircraft depend on each other is critical to optimizing tasks.

The goal of this thesis is to design and develop a set of ontology-based services that are capable of providing domain-specific functionalities and that can be easily integrated with MRO software systems, particularly AMOS.

The first part of the thesis (2) is dedicated to the introduction of the most important and essential aspects of the topic. It outlines brief information about the fundamentals of the Semantic Web, as well as the main characteristics and limitations of the AMOS system. The latter part of the thesis (3) discusses the information gathering approaches used to specify the requirements. It also examines the techniques of mapping tabular data to RDF, provides the analysis of the source data, and an overview of the charting libraries required for prototype implementation. Chapter 4 summarizes the specified functional and non-functional requirements. Afterward, chapter 5 demonstrates the system design of the application services. It is followed by chapter 6, describing the most noteworthy decisions made during the implementation phase. Chapter 7 presents the methods and results of the performed testing activities. The last chapter 8 of the thesis concludes and summarises the result of the work.

Chapter 2

Background

This chapter of the thesis is dedicated to the introduction of the most important and essential aspects of the topic. It outlines brief information about the fundamentals of the Semantic Web, as well as the main characteristics and limitations of the AMOS system.

2.1 AMOS

This section provides a brief general overview of the AMOS system to give a reader an idea of its purpose and what problems it solved. Additionally, AMOS is used by Czech Airlines Technics organization as a principal MRO solution and serves as the primary source of data for the project.

2.1.1 Description

AMOS is one of the many MRO (Maintenance, Repair and Operations) software solutions, which is intended to manage clients' maintenance, engineering, and logistics needs. AMOS was developed by Swiss Aviation Software AG and is used by more than 200 customers worldwide [4] [5], including Czech Airlines Technics (CSAT).

AMOS has a large set of functionalities and features. In view of the fact that AMOS appears to be a classic example of the enterprise-level software application, it follows the modular architecture design principles and consists of several different modules. It comprises the following core modules [6]:

- Material Management - for managing complex logistic tasks, parts availability and component utilization, warranty control, inventory management, and vendor rating [7]
- Engineering - for managing maintenance tasks in accordance with the standards of a Continuing Airworthiness Management Organisation (CAMO) [8]
- Planning - for managing the preparation of scheduled and unscheduled, short- and long-term maintenance events [9]
- Component Maintenance - for managing extensive MRO activities in a shop environment [10]

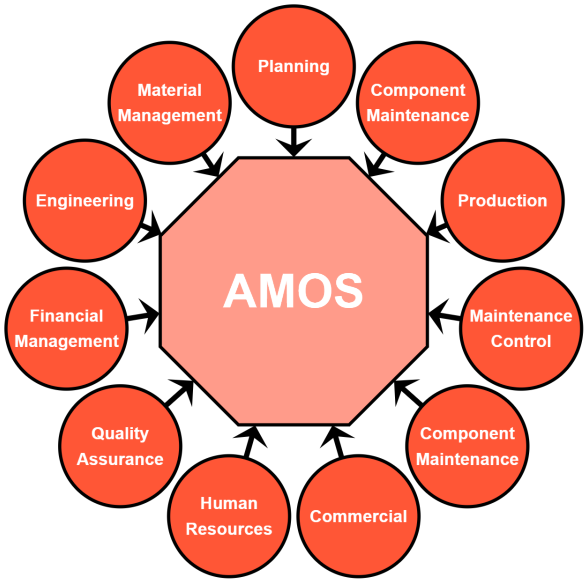


Figure 2.1: AMOS core modules

- Production - for managing troubleshooting, the actual execution of Line and Base Maintenance and ensuring a handover of work packages from the Maintenance Planning to the Production Department [11]
- Maintenance Control - this module is the main tool to control and improve the overall reliability of the technical organisation and to ensure uninterrupted airworthiness of the operating fleet [12]
- Commercial - for enabling AMOS users to monitor their relationships with all their customers and tracking the ongoing pre-sales activities with potential and existing customers [13]
- Human Resources - for allocating and deploying human resources in accordance with man-hours, employee's skills / licences and workload [14]
- Quality Assurance - for guaranteeing that agreed-upon quality requirements are respected [15]
- Financial Management - for performing such tasks as cost and warranty control, invoice generation, stock value determination and integration into the customer's financial accounting system [16]

2.1.2 Limitations

Despite the fact that AMOS software solution is designed to encompass the full range of functionality needed by MRO customers, it can not cover all the customer-specific needs in

advance. Therefore it was designed to make the integration of AMOS into the customers' current system landscape as easy, smooth, and convenient as possible.

Since AMOS is an open system based on advanced technology, it can be easily linked with other corporate information systems using different generic interfaces, such as flat files over direct database access, XML-based interfaces, and others. This flexibility allows customers to have many different ways on how to export and import data. [17]

2.2 Semantic Web

The World Wide Web (WWW) refers to a core retrieval system of the internet which is also called simply Web. Nowadays three types of the web are distinguished: Web 1.0, Web 2.0, and Web 3.0. The original Web 1.0 stands for the earliest version of the web and is typically associated with the 1990s and early 2000s. It is often referred to as a read-only (RO) web since the content of the websites was created by their owner [18] [19]. On the other hand, Web 2.0 refers to the current version of the web internet, while Web 3.0 represents its next phase. The creation and shaping of the content in Web 2.0 also involve the visitors and users of the website. Therefore it is usually referred to as read-write (RW) web. Unlike Web 2.0 where data is possessed by the owner of the website, the idea of Web 3.0 lies in decentralization. This implies that the data is open and no permission is needed to post the information. [19]

The term Semantic Web [1] is often referred to as Web 3.0 since Web 3.0 is based on the same idea. It is an extension of the WWW that follows the standards specified by the World Wide Web Consortium (W3C). The objectives of the Semantic Web are aimed at establishing the machine-readability of the web. [20] It demands the use of standardized semantic technologies and components that make up the architecture of the Semantic Web.

The figure 2.2 illustrates the architecture of the Semantic Web, also termed Semantic Web Stack, Semantic Web Cake, or Semantic Web Layer Cake [21]. The stack builds on the following technologies: Resource Description Framework (RDF), RDF Schema (RDFS), Web Ontology Language (OWL), SPARQL, and others. The most relevant for this thesis semantic technologies will be outlined in this chapter.

2.2.1 RDF

Resource Description Framework is a standard model for data interchange on the Web. It is the core technology of the Semantic Web which defines a way of representing knowledge in a decentralized world. It is intended for enabling the computer programs to make use of all the structured information distributed over the Web. RDF specifies a universal way of breaking down the knowledge into small pieces and sets certain rules regarding the meaning of those pieces. [22]

There are plenty of formats that can be used to represent RDF data. The most common

[1] More information about Semantic Web available at <https://www.jstor.org/stable/26059207>

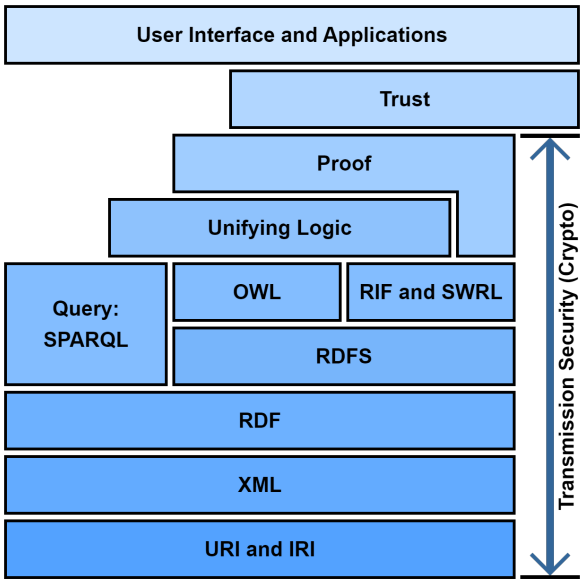


Figure 2.2: Semantic Web Stack [21]

and standardized ones are: N-Triples [2], Turtle [3], JSON-LD [4] and RDF/XML [5]. The following RDF example represents a simple object described using Turtle syntax:

```
@prefix cm: <http://onto.fel.cvut.cz/ontologies/csat-maintenance/> .

cm:aircraft-AIR-1 a cm:aircraft;
cm:age 19.728767;
cm:model "32A";
cm:registration "OE-LBX" .
```

The same example in form of the RDF graph is depicted in figure 2.3:

2.2.2 JSON-LD

JSON-LD is one of the standardized [6] RDF formats. It is based on the existing JSON format with minimal changes made so that the data can be interpreted as Linked Data. It is worth noting, that the data in JSON-LD format are fully compatible with JSON, therefore the majority of JSON tools available today can be reused. [23]

The following example describes the same object from the the previous example, but in the JSON-LD format:

[2] Specification is available at <https://www.w3.org/TR/n-triples/>
[3] Specification is available at <https://www.w3.org/TR/turtle/>
[4] Specification is available at <https://www.w3.org/TR/json-ld/>
[5] Specification is available at <https://www.w3.org/TR/rdf-syntax-grammar/>
[6] Specification is available at <https://www.w3.org/TR/sparql11-overview/>

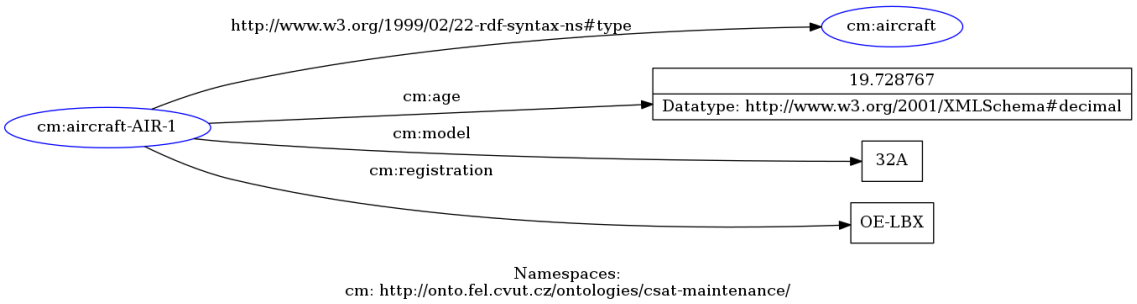


Figure 2.3: RDF data visualized as a graph

```
{
  "@context": {
    "cm": "http://onto.fel.cvut.cz/ontologies/csat-maintenance/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "owl": "http://www.w3.org/2002/07/owl#"
  },
  "@id": "cm:aircraft-AIR-1",
  "@type": [
    "cm:aircraft",
    "owl:NamedIndividual"
  ],
  "cm:age": {
    "@type": "xsd:decimal",
    "@value": "19.728767"
  },
  "cm:model": "32A",
  "cm:registration": "OE-LBX"
}
```

JSON-LD is designed to satisfy the following objectives:

- Simplicity - no additional tools are necessary to use JSON-LD in its most basic form
- Compatibility - the documents in JSON-LD format are consistently valid JSON documents
- Expressiveness - the JSON-LD syntax serializes labeled directed graphs to ensure that the majority of real world data models can be expressed
- Terseness - the syntax is human readable and concise
- Zero Edits, most of the time - it ensures a straightforward transition from existing JSON-based systems
- Usable as RDF - it can be used as plain JSON, so there is no need to understand RDF. [23]

2.2.3 RDFS

RDFS stands for Resource Description Framework Schema (also variously abbreviated as RDF(S), RDF-S, or RDF/S). It is the most fundamental schema language in the Semantic Web technology stack which is intended to explain how nodes of a graph relate. It is designed to be lightweight, intelligible, and straightforward. [24] [25]

The RDFS vocabulary introduces the concept of classes and class hierarchy by allowing defining superclasses and subclasses of the nodes. [24]

2.2.4 OWL

The Web Ontology Language (OWL) is a Semantic Web language specified by World Wide Web Consortium (W3C) that is designed to represent complex knowledge about things and the relationships between them. Since OWL belongs to the Semantic Web and is part of the Semantic Web Stack, the knowledge given in OWL can be utilized by computer programs. [26]

In contrast to the RDFS, OWL provides a more expressive and wider vocabulary and allows one to express much more about the data model. [27]

2.2.5 SPARQL

SPARQL is the standardized [7] language and protocols to query and manipulate RDF data. It is used as a common query language in RDF triple stores and, similarly to SQL, allows users to perform CRUD operations on the RDF data.

SPARQL queries are commonly constructed using the following fundamental clauses:

- SELECT - basic query command which is used to find and return the data matching certain patterns
- CONSTRUCT - serves for creating and transforming the data
- ASK - used for checking if certain patterns exist in the data
- DESCRIBE - command for viewing the RDF graph that describes a particular resource.

Additionally, there exist optional statements such as PREFIX, FROM and WHERE, and also several query result modifiers: ORDER BY, OFFSET, LIMIT, GROUP BY, and HAVING. [28]

2.2.6 RDF Triple Stores

An RDF triple store is a graph database that is designed to store and retrieve the entries from the collections of strings. These entries are constructed from semantic triples (also known as RDF triples or just triples) which represent a sequence of three entities in the form of subject-predicate-object relationships. [29] An example of such triple entry could

[7] Specification is available at <https://www.w3.org/TR/sparql11-overview/>

be: “aircraft-AIR-1 model 32A”, where “aircraft-AIR-1” is a subject, “model” is a predicate and "32A" is an object.

There are a plethora of different implementations of RDF triple stores. Some of them are based on the database engines that have been built de novo, while others have been constructed on the basis of existing ones. The following list outlines some of the most popular triplestores in use [30]:

- AllegroGraph
- Apache Jena
- BlazeGraph
- MarkLogic
- Eclipse RDF4J
- GraphDB
- Virtuoso
- 3Store

Since triple store databases are non-relational (i.e. do not work with relational data) and do not require a fixed schema, they are considered to be NoSQL [8]. Therefore, they aren't favorable for transactional applications since there is a lack of ACID [9] transactions. [30] The data in RDF triple stores are queried by SPARQL query language.

[8] More information about NoSQL is available at <https://www.couchbase.com/resources/why-nosql>

[9] More information about ACID is available at <https://database.guide/what-is-acid-in-databases/>

Chapter 3

Analysis

This chapter is dedicated to the analysis of the source data. The first section of the chapter outlines the information gathering process. The following section describes the techniques used to transform and map the source tabular data into the ontology of the OWL domain model. Afterward, the analysis of the charting libraries required for further implementation is described.

3.1 Analysis of domain of interest

As can be seen from the requirements formulated in the chapter 4.2, the main objectives of the project are aimed at defining and calculating certain metrics. These metrics have to be calculated from the provided data spreadsheets. The problem is that in this case the data are provided in a denormalized form, and what is more important, have poor quality. Therefore, in order to achieve the stated goals, it is necessary to thoroughly analyze the data, distinguish relevant and irrelevant parts and finally transform them into meaningful and practical form. This is the main objective of this section of the thesis.

3.1.1 Information gathering

Information gathering activities are necessary for identification and definition of stakeholder requirements. During the analysis phase of this project, information gathering was the most time-consuming part, since it involved regular meetings with the domain experts. These meetings were needed to build a knowledge base from the provided source data since the data encompassed no information about the meaning of its contents. No information about the data origins, data life cycle, or original data flows was given, therefore the source system was treated as a black-box.

One of the objectives was to extract knowledge from the provided data sets. This process of building a knowledge base comprised the following activities: defining the domain data model by expressing it using OWL and specifying the mapping of the source data to the data model. Another important part of the information gathering process was the correct and precise definition of the dashboard metrics since they build the foundation for the software requirements fulfillment.

3.1.2 Formalization of dashboard statistics

One of the goals of the meetings with domain experts was the formalization of dashboard statistics and data constraints. In order to achieve that it was necessary to prepare in advance a prototype of the dashboard with mock-up data. Every data metric was specified by defining formulas referring to the columns and rows of the source CSV files since the domain experts are familiar with their structure and content. For example, the following formula was defined to calculate the number of closed task cards per work package wp based on the data from wo-tc-ref (WOTA) file:

$$C_{TC}(wp) = |\{row \in WOTA \| row.WP = wp, row.type = "TC", row.state = "C"\}|$$

However, these formulas are hardly applicable for the actual metrics calculation since they refer to the CSV files and not to the domain model of the application. The prepared OWL ontology for the domain model is formulated and described in the chapter 5.1. The process and result of formulas definition concerning the domain model is described in chapter 5.

Another point worth mentioning is that the CSV data had to be transformed to RDF in order to perform mapping of the data to the ontology. The complete process of transforming and mapping was handled by the SPipes application [1] and is outlined in the chapter 3.2.

3.1.3 Maintenance, Repair and Operations

The term "Maintenance, Repair and Operations" is also referred to as "Maintenance, Repair and Overhaul" or MRO. In the aviation industry, the meaning of this term comprises a group of activities that are aimed at the upkeep of an aircraft or its components. The principal objective of the MRO is to ensure the continuing airworthiness of the aircraft during its service life. This section provides a brief overview of MRO activities, focusing on the most significant aspects within the scope of this thesis.

3.1.3.1 Aircraft Maintenance

Aircraft maintenance is an essential part of the MRO. It is a fundamental process of ensuring the airworthiness of the aircraft and satisfying the conditions to guarantee the safe operation of the vehicle. [58]

Due to the fact that the aviation industry is highly regulated, airline companies are obliged to provide continuous inspection programs established by aviation authorities. It is crucial to perform the aircraft maintenance checks continually as they keep it airworthy and reliable. These checks include routine and non-routine maintenance activities such as replacing aircraft components, repairing previously discovered defects, performing scheduled repairs, etc. [31] The fundamental types of the maintenance checks are outlined in the following sections.

Base Maintenance

Base maintenance, also known as heavy or depth maintenance, is one of the most crucial phases of aircraft maintenance. It involves the detailed, long-lasting, and extensive inspection

[1] GitHub repository is available at <https://github.com/kbss-cvut/s-pipes>

Check type	Check frequency*	Check duration*
A check	400-600 flight hours/200–300 flights	10-100 man-hours
B check	6-8 months	160-180 man-hours/1-3 days
C check	2 years	10000-30000 man-hours/1-4 weeks
D check	6-10 years	30000-50000 man-hours/2 months

** provided values are approximate and depend on numerous factors, such as aircraft type, operational environment, etc.*

Table 3.1: Maintenance checks

of an aircraft and always requires its downtime during the check process. Typically it puts the aircraft out of service for 1-4 weeks with intervals of 2-6 years. The operational time between base maintenance inspections depends on the requirements specified by the manufacturer and may be adapted based on the operational environment of the aircraft. [32] This type of maintenance is usually referred to as "checks" of different levels, classified based on the intensity and magnitude of the inspections and overhaul. There are 4 types of checks [31] [33]:

- A check - this is a lighter check that is typically performed at a hangar and takes tens of working hours to be finished. The A checks commonly include general inspections of the interior and the aircraft hull for evidence of damage, deformation, corrosion, and missing parts, as well as the service, engine, and function checks.
- B check - these checks are often performed along with the A checks. For the sake of resource efficiency and downtime reduction, the airlines often merge B check tasks into A checks. A typical example of a task completed during this phase is inspecting the wheel well hydraulic tubing for corrosion and leakage.
- C check - this type of check is significantly more comprehensive and extensive than the A and B checks. It involves a thorough inspection of a majority of the aircraft's components. During this check, the maintenance technicians typically perform complex servicing tasks such as examination of structures for corrosion and damage, in-depth lubrication of fittings and cables, and others.
- D check - this is the most embracive check type, otherwise known as "heavy maintenance visit" (HMV), that includes deep inspections and repairs of the entire aircraft. This can involve dismantling the entire vehicle so that it can be examined and inspected for damage and corrosion. Because of the nature and the cost of the D checks, the majority of airlines plan them years in advance. In certain cases (typically after two or three D checks) the cost of repair may even exceed the actual cost of the vehicle.

The table below aggregates the rough information about how often each of the checks happens and how long it typically takes [31] [33]:

Line Maintenance

This is one of the phases of aircraft maintenance that is performed on the apron, during the turnaround time (TAT), i.e. the time between the landing and the take-off for a new flight. [34] Line maintenance includes routine inspections, daily checks, and troubleshooting.

This includes the maintenance activities that are performed on an aircraft while it is fully operational. Basically, line maintenance can be characterized as a set of minor servicing, inspection, and repair tasks, which do not require vehicle downtime due to disassembly and can be carried out in a simple way. [33] The typical examples of such activities are changing a tire or checking aircraft navigation lights for their functioning. [35]

3.1.3.2 Work packages

In MRO systems, a work package is a group of related maintenance tasks that are performed on an aircraft during a maintenance period. The work package includes information about the components, materials, inspections, and repair locations on which the maintenance work has to be performed. Every work package is generally associated with a customer on the basis of a customer agreement. [36]

At CSAT, the work package encompasses all the maintenance tasks assigned to the aircraft of the work package, including:

- Task Cards (TC), which specify the detailed information about the maintenance work on aircraft or its components to be done, including the job card information that is copied to work orders, when the tasks become due [37]
- Non-Routine Cards (NRC), also referred to as Findings. These are the tasks that occurred due to additional maintenance needs detected during the aircraft maintenance checks, but not included in the scheduled task requirements or maintenance plan. [38]
- Work Orders (WO), which contain information about the work that must be performed. The WO also outlines a process for completing the task, and typically includes the details on the scope, assigned technician, and what is expected to be done [39] [40]. At CSAT, the work orders are divided into two types: the unscheduled Maintenance Work Orders (MWO), which basically represent the Findings (NRC), and the Scheduled Work Orders (SWO).

3.1.3.3 Maintenance line

At CSAT, the term Maintenance line (also referred to simply as Line) has a specific meaning. It might be confused with Line maintenance from MRO, but it's not the same thing. The term "Maintenance Line" applies to maintenance technicians (also known as mechanics) as well as to work packages. The term refers to the group of employees to which these mechanics belong. Each of these groups corresponds to a specific station in the MRO hangar. These groups can also be assigned to work packages, thus representing a responsible (main) maintenance line. A maximum of one Maintenance Technician can be assigned to a work package. Similarly, one maintenance technician can be assigned to a maximum of one maintenance line at a time.

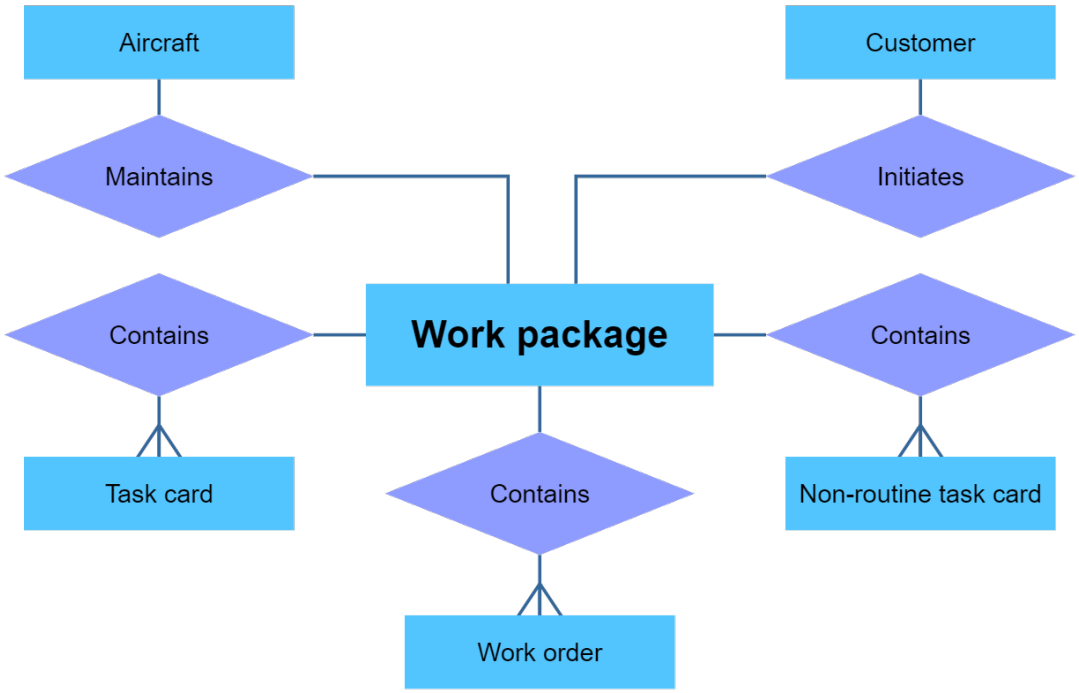


Figure 3.1: Work package elements

3.1.4 Provided source data

All source files are regularly exported by two CSAT systems: AMOS and Carded Personnel Attendance Control System (CPACS). AMOS provides MRO data, such as work log within the current work packages, information about corresponding task cards and work orders, time estimation of execution steps, information about work packages, etc. CPACS, on the other hand, provides data about the time and attendance of employees, such as the arrivals and departures of maintenance technicians (mechanics).

All provided files are exported to an external SFTP server in nonunified comma-separated values format (CSV). Although CSV is very common and has been widely used for data tabular exchange for quite some time, it has never been formally specified, i.e. it is not fully standardized. [41] Likewise, the provided CSV files do not follow any common format. Therefore it has been evaluated with respect to the format that seems to be followed by most implementations (stated in Section 2 of RFC4180 [2]). The evaluation result is depicted in the table below:

There are five source files presented: time-analysis (TA), time-estimates (TE), wo-tc-ref (WOTA), wp-catalog (WPC) and presence (P). Together they provide plenty of useful information, sufficient to fulfill described requirements. However, some of them contain a certain amount of irrelevant information, that is, irrelevant fields. Furthermore, some of

[2]RFC4180 is available at <https://datatracker.ietf.org/doc/html/rfc4180>

Source file	Source	Delimiter	Header	Cells contain line breaks	Cells contain delimiter	Ending line break
Time-analysis	AMOS	Tab	Yes	No	No	Yes
Time-estimates	AMOS	Tab	Yes	No	No	Yes
Wo-tc-ref	AMOS	Tab	Yes	No	No	Yes
Wp-catalog	AMOS	Tab	Yes	No	No	Yes
Presence	CPACS	Semicolon	No	No	No	Yes

Table 3.2: Source data format

the column names are quite vague and do not clearly indicate the meaning. The following sections define the most relevant data fields needed to fulfill the requirements. A complete list of fields with descriptions can be found in Appendix B.

3.1.4.1 Time-analysis

Time-analysis (TA) file contains records of activities that employees perform during their work hours, i.e. work log. It consists of the following columns:

1. "Employee No" - unique identifier of employee
2. "Start Date" - start date of the working activity
3. "Start Time" - start time of the working activity
4. "End Date" - end date of the working activity
5. "End Time" - end time of the working activity
6. "Duration Full" - overall time spent on the task in hours
7. "Scope" - the work orientation of the employee, i.e. the specific aircraft maintenance activity they perform (also referred to as maintenance group)
8. "Shift Group" - identifier of a group/shift, to which the worker is assigned
9. "Type" - type of the task (e.g. “M” is MWO, “S” is SWO and “TC” is TC)
10. "WO/TC" - identifier of the WO (for MWO and SWO tasks) or identifier of the TC (for TC tasks)
11. "Workpackage" - identifier of the WP
12. "A/C model" - model of the aircraft, to which the task is assigned
13. "A/C age" - age of the aircraft in years, to which the task is assigned
14. "WP Start date - scheduled" - planned start date of the work package
15. "WP End date - scheduled" - planned end date of the work package

16. "WP Start date - real" - actual start date of the work package
17. "WP End date - real" - actual end date of the work package
18. "Line" - identifier of the responsible (main) line, to which the work package is assigned
19. "AC registration" - unique code, representing an aircraft registration number
20. "operator" - code of the customer, i.e. aircraft owner

3.1.4.2 Time-estimates

Time-estimates (TE) file contains time estimates for the specific tasks and consists of the following columns:

1. "AC registration" - unique code, representing an aircraft registration number
2. "operator" - code of the customer, i.e. aircraft owner
3. "WP" - identifier of the WP
4. "WO" - identifier of the WO (for MWO and SWO tasks)
5. "TC" - identifier of the TC (for TC tasks)
6. "sequence" - sequential number of the task (also known as workstep) within the TC or WO that has one or more worksteps
7. "scope" - the work orientation of the employee, i.e. the specific aircraft maintenance activity they perform (also referred to as maintenance group)
8. "est-min" - time estimate for the task in hours

3.1.4.3 Wo-tc-ref

Wo-tc-ref (WOTA) contains WO texts and WO actions records, which are given as individual task steps (action sequence). Every task within the work package is split into these particular sequences and each sequence action has its order number defining the order in task execution. This WOTA file consists of the following columns:

1. "AC" - unique code, representing an aircraft registration number
2. "A/C age" - age of the aircraft in years, to which the task is assigned
3. "WP" - identifier of the WP
4. "CSAT WO/TC" - identifier of the WO (for MWO and SWO tasks) or identifier of the TC (for TC tasks)
5. "type" - type of the task (e.g. “M” is MWO, “S” is SWO and “TC” is TC)
6. "state" - state of the task step (e.g. “O” for open and “C” for close)

7. "TC reference" - identifier of the TC (for TC tasks)
8. "issue date" - date when the task was issued
9. "closing date" - date when the task was closed
10. "sequence" - order number of the task step

3.1.4.4 Wp-catalog

Wp-catalog (WPC) is a work package catalog file. It contains overall information about the work package, such as details about time estimation from strategic planning prior agreement between customer and maintenance provider. It only includes information related to current (active) work packages, work packages that are expected to be finished soon or that have been closed in the last few days. CSV file consists of the following columns:

1. "AC registration" - unique code, representing an aircraft registration number
2. "Workpackage" - identifier of the WP
3. "Start date - scheduled" - planned start date of the work package
4. "End date - scheduled" - planned end date of the work package
5. "Start date - real" - actual start date of the work package
6. "End date - real" - actual end date of the work package

3.1.4.5 Presence

Presence (P) file contains information about the work time and attendance of employees, such as arrival and departure time of mechanics. It consists of the following columns:

1. "Employee No" - unique identifier of employee
2. "status-1" - status of the employee (e.g. “A” for “available” and “N” for “not available”).
3. "status-2" - status of the employee (e.g. “1” for “available” and “2” for “not available”).
4. "date" - date of the status change (i.e. employee arrival or departure)
5. "time" - time of the status change (i.e. employee arrival or departure)

Source file	Primary key	Time period	Constraints
Time-analysis	"bookingno_i"	Work log from the last 24 hours	CR1
Time-estimates	("WP" + "WO" + "TC" + "sequence" + "scope")*	All estimates related to the relevant (current, ongoing, latest) work packages	CR2
Wo-tc-ref	("WP" + "CSAT WO/TC" + "sequence")	All relevant (current, ongoing, latest) work packages	CR3
Wp-catalog	"Workpackage"	All relevant (current, ongoing, latest) work packages	none
Presence	("Employee No" + status_1" + "status_2" + "date" + "time")	Attendance log from the last 30 days	CR4

** there might occur multiple rows with the same primary key, but different "est-min" values. The possible reasons for that could be human error (wrongly entered data), the way AMOS (at CSAT) implements data modifications (e.g. increase of "est-min" value yields an additional record) or specifics of exporting SQL. It was decided to solve these inconsistencies by summing "est-min" values with the same primary key.*

Table 3.3: Source data details

3.1.5 Completeness of data and data constraints

Based on the file content analysis described in the previous chapter, it appears that the source tabular data are provided in denormalized form. Moreover, after a more thorough examination, it seems that there also is a high degree of redundancy and repetition. As an example, all four TA, TE, WOTA, and WPC files contain columns referring to the same entities. Examples of such columns are "AC registration" for TA, TE, and WPC and "AC" for WOTA, all of which hold a unique code value, representing an aircraft registration number, e.g. "PH-HZD". In order to properly and correctly process the input data, define a mapping, and construct domain model instances, the comprehensiveness and wholeness of the data need to be examined. Furthermore, it is evident that the data from each source file refers to a certain time period. Therefore, in order to ensure the completeness of the data per work package, it was necessary to identify time windows for every data set. In the following table, the main detected issues related to the data completeness are described. In addition, the principal data constraints are indicated. It is worth noting, that only the relevant data fields are considered:

The table contains the references to the following constraints:

CR1 Time-analysis constrains

- "Start Date" is either equal to "End Date" or "End Date" minus 1 day
- "Start Time" = "End Time" => "Duration Full" is 0.00
- "End Date" is null <=>"End Time" = 0:00 and "Duration Full" = 0.00
- "Type" is "ID-JOB" or "ARID-JOB" => "Workpackage",
  "A/C model", "A/C age", "WP Start date - scheduled", "WP End date - scheduled", "WP Start date - real", "WP End date - real", "Line" are null

CR2 Time-estimates constraints

- "WO" is null <=> "TC" is not null
- "WO" is not null <=> "TC" is null

CR3 Wo-tc-ref constraints

- state = "O" => "closing date" is null
- "Customer ref" contains "(finding)" => "type" = "M"

CR4 Presence constraints

- "status-1" = "A" <=> "status-2" is "1" or "28"
- "status-1" = "N" <=> "status-2" is "2" or "29"

3.2 Mapping of Relational Data to RDF

The tabular form is one of the simplest, oldest, and most prevalent structured data forms. The data elements of these forms represent a set of data cells arranged in columns (vertical) and rows (horizontal). [42] Tabular and relational databases use this structure to keep and manage the internal data. In tabular databases, each record (i.e. row) always has the same set of properties and the same set of columns. In case one of the rows is missing a value for a particular column, that cell is associated with a special value representing the missing data. In this thesis, it was of interest to investigate the process of mapping relational data to RDF to ensure and confirm data traceability, i.e. the ability to follow the data all the way back to its original source. Another motivation for becoming familiar with the mapping technique was to understand the process that most converters follow, which will be important in the design and implementation phases.

3.2.1 Generating RDF from Tabular Data

The "CSV2RDF" W3C specification describes the procedure of converting the annotated tabular data model into RDF data representation. It expresses the strategy of creating an RDF from the tabular data with associated metadata [43]. These metadata provide information about the cells, rows, columns, tables, and groups of tables with which they are associated. [44] There are plenty of various extensions, applications, and tools that are intended to convert the data from tabular-text formats (CSV-like, e.g. TSV) into RDF such as:

- RDF123 - an application that works with simple spreadsheets and converts them into an RDF graph. [45]
- XLWrap - typically used as an RDF spreadsheets wrapper that can transform and convert CSV files to arbitrary RDF graphs using the provided mapping. It supports plenty of tabular data formats, including CSV-like ones. [46]
- Tarql - an application capable of converting the CSV data to RDF based on the provided mapping in the form of standard SPARQL 1.1 language. It uses the CONSTRUCT queries for bindings the particular fields and generating the output data records. [47]

The "CSV2RDF" specification distinguishes two modes of transform operation: Standard and Minimal. It also states that the conversion applications must provide at least two of them. In Standard mode, the conversion processes the obtained information in frames and takes into account the details of the rows, tables, and a group of tables (containing the source data). On the contrary, the Minimal mode implies the conversion that includes only the information obtained from the cells of the tabular data.

The tabular data below presented as a CSV file is used to demonstrate a simple example of the data transformation. This table contains information attributes about airlines, including country name, IATA code, ICAO code, and call sign:

```
code_IATA,code_ICAO,call_sign,country_name
DV,ACK,''ACK AIR'',''United States''
OK,CSA,CSA,''Czech Republic''
EO,ALX,ALLCONGO,''Democratic Republic of the Congo''
```

In view of the fact that the conversion procedure described in CSV2RDF operates on the annotated tabular data model and does not include the specification of the process of converting CSV-encoded data into tabular data form, the annotation of the presented table needs to be considered. There are several annotation levels: table, columns, rows, and cells. The annotation of table A from the example mentioned above is the following:

- id: A
- URL: *http://example.org/airlines.csv*
- rows: B1, B2, B3
- columns: C1, C2, C3, C4

Annotation of the rest levels (columns, rows, and cells) can be found in Appendix A. The input CSV file contains a header line and is located at *http://example.org/airlines.csv*. The data transformation in Minimal mode, adhering to the specification mentioned above, results in the following RDF file presented in "Turtle" notation:

```
http://example.org/airlines-minimal.ttl

@base <http://example.org/airlines.csv> .

_:771750a0-7193-4677-94d3-63f3a126e7e4
  <#code_IATA> "DV" ;
  <#code_ICAO> "ACK" ;
  <#call_sign> "ACK AIR" ;
  <#country_name> "United States" .

_:bfff7f8a-0603-456c-9575-ef592a7efae0
  <#code_IATA> "OK" ;
  <#code_ICAO> "CSA" ;
  <#call_sign> "CSA" ;
  <#country_name> "Czech Republic" .

_:26929753-418e-4d4f-83ab-280ca8697d3b
  <#code_IATA> "EO" ;
  <#code_ICAO> "ALX" ;
  <#call_sign> "ALLCONGO" ;
  <#country_name> "Democratic Republic of the Congo" .
```

In addition, the transformation result of the same *http://example.org/airlines.csv* file (in Standard mode) is presented below (also in "Turtle" notation):

```
http://example.org/airlines-standard.ttl

@base <http://example.org/airlines.csv> .
@prefix csvw: <http://www.w3.org/ns/csvw#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:3869b505-e8d7-4f00-bacb-ee79bb72e8cc a csvw:TableGroup ;
  csvw:table [ a csvw:Table ;
    csvw:url <http://example.org/airlines.csv> ;
    csvw:row [ a csvw:Row ;
      csvw:rownum "1"^^xsd:integer ;
      csvw:url <#row=2> ;
      csvw:describes _:771750a0-7193-4677-94d3-63f3a126e7e4
    ], [ a csvw:Row ;
      csvw:rownum "2"^^xsd:integer ;
      csvw:url <#row=3> ;
      csvw:describes _:bfff7f8a-0603-456c-9575-ef592a7efae0
    ], [ a csvw:Row ;
      csvw:rownum "3"^^xsd:integer ;
      csvw:url <#row=4> ;
      csvw:describes _:26929753-418e-4d4f-83ab-280ca8697d3b
```

```
    ]
  ] .


_:771750a0-7193-4677-94d3-63f3a126e7e4
  <#code_IATA> "DV" ;
  <#code_ICAO> "ACK" ;
  <#call_sign> "ACK AIR" ;
  <#country_name> "United States" .

_:bfff7f8a-0603-456c-9575-ef592a7efae0
  <#code_IATA> "OK" ;
  <#code_ICAO> "CSA" ;
  <#call_sign> "CSA" ;
  <#country_name> "Czech Republic" .

_:26929753-418e-4d4f-83ab-280ca8697d3b
  <#code_IATA> "EO" ;
  <#code_ICAO> "ALX" ;
  <#call_sign> "ALLCONGO" ;
  <#country_name> "Democratic Republic of the Congo" .
```

As is evident from the provided example, the conversion in Standard mode has yielded a more complete and comprehensive result, that additionally contains extra metadata such as positions of the rows amongst the rows of the source table.

3.2.2 RDB to RDF Mapping

In the previous section, the strategies of generating RDF from CSV-like files were considered. Since the concept of mappings from relational databases to RDF datasets is also applicable in the context of this thesis, it will be revealed in this section.

There is a good deal of mapping languages that provide the ability to represent the existing relational data in the RDF data model. [48] An "RDB-Direct-Mapping" W3C specification describes the approach for RDB to RDF transformation and provides a framework for defining and comparing more sophisticated conversions. [49] There is also a complement W3C specification, that characterizes the R2RML mapping language. This language is a relaxed variant of the Direct Mapping and is intended as a default mapping for further customization. [50]

The most straightforward way to describe the idea of R2RML mapping language technique is by providing examples. The following sample demonstrates the RDB to RDF transformation on a simple database that includes multiple data types, primary keys, and a single reference (foreign key). All transformation artifacts, inputs, and outputs are presented below.

Consider having the following input database consisting of two tables:

Both presented tables are in normalized form and have multiple data columns. The first table contains a single record of the aircraft with a serial number (primary key), registration

serial_number:INT, PK	**registration:TEXT**	**type:TEXT**	**airline_id:INT, FK airline(id)**
50020	N101DU	CS100	3712

Table 3.4: Aircraft table

id:INT, PK	**call_sign:TEXT**	**country_code:TEXT**
3712	Delta Air Lines	US

Table 3.5: Airline table

number, aircraft type, and a column that references the record from the latter table. The latter table also has a single entity, representing a specific airline and contains the information about the call sign of the airline and the country code.

In order to transform the RDB to RDF using the R2RML mapping language, it is necessary to construct the mapping RDF file. This mapping does not essentially have to map all fields of the input tables and can be partial. Since the database has multiple tables, it is necessary to define a mapping for each of them. A possible mapping of the airline table could be the following:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "airline" ];
    rr:subjectMap [
        rr:template "http://data.example.com/airline/{id}";
        rr:class ex:Airline;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:has_call_sign;
        rr:objectMap [ rr:column "call_sign" ];
    ].
    rr:predicateObjectMap [
        rr:predicate ex:has_country_code;
        rr:objectMap [ rr:column "country_code" ];
    ].
```

This mapping generates the following output triples:

```
<http://data.example.com/airline/3712> rdf:type ex:Airline.
<http://data.example.com/airline/3712> ex:has_call_sign
"Delta Air Lines".
<http://data.example.com/airline/3712> ex:has_country_code "US".
```

At this point, a complete mapping of the airline table is defined. The only missing part is the aircraft table which additionally has a column referencing the records in the airline

table. The following mapping demonstrates how to define a linking between two different mappings:

```
<#TriplesMap2>
rr:logicalTable [ rr:tableName "aircraft" ];
rr:subjectMap [
    rr:template "http://data.example.com/aircraft/{serial_number}";
    rr:class ex:Aircraft;
];
rr:predicateObjectMap [
    rr:predicate ex:has_registration;
    rr:objectMap [ rr:column "registration" ];
].
rr:predicateObjectMap [
    rr:predicate ex:belongs_to_airline;
    rr:objectMap [
        rr:parentTriplesMap <#TriplesMap1>;
        rr:joinCondition [
            rr:child "airline_id";
            rr:parent "id";
        ];
    ];
].
```

It is worth noting that the stated mapping is partial and does not include column "type". The result of the mapping is displayed below:

```
<http://data.example.com/aircraft/50020> rdf:type ex:Aircraft.
<http://data.example.com/aircraft/50020> ex:has_registration "N101DU".
<http://data.example.com/aircraft/50020> ex:belongs_to_airline
<http://data.example.com/airline/3712>.
```

3.2.3 Mapping using SPipes

In order to represent the provided source tabular data according to the OWL data model, it was necessary to first convert them into RDF. For this purpose, an implementation of the SPipes [3] scripting language named SPipes Engine was used. An individual SPipes transformation script was prepared for each source data file. Each such transformation script consists of two stages: CSV to RDF conversion along with the validation of constraints and subsequent mapping of the result to the OWL data model. It should be emphasized that all RDF transformation scripts were created by Mgr. Miroslav Blaško, Ph.D., Knowledge Based Software Systems Group (KBSS) and lie outside the scope of the thesis.

[3] GitHub repository is available at <https://github.com/kbss-cvut/s-pipes>

3.2.3.1 CSV to RDF conversion

The conversion of source CSV files to RDF is an initial stage of the data transformation process. It is achieved using a tabular data to RDF convertor module for SPipes called Tabular. Tabular was created by KBSS and is intended for converting tabular data (e.g. CSV or TSV) to RDF. Its implementation loosely follows the CSV2RDF W3C Recommendation discussed in the previous section. It should be pointed out that this stage also involves the validation of the constraints defined for the corresponding source CSV file.

3.2.3.2 RDF data mapping

The result of the CSV to RDF conversion serves as input for the latter stage of the data transformation process. This stage involves the construction of the OWL ontology with instance data which is compliant with the OWL ontology defining the data model (the domain model as well as the prefixes used are described in chapter 5.1). The construction is defined using CONSTRUCT statements of SPARQL. For instance, the following excerpt from the RDF mapping script (slightly modified for readability) has been used to construct the *cm:workpackage* (work package) and *cm:aircraft* (aircraft) entities based on the result of the CSV to RDF conversion of WOTA file:

```
:construct-csat-maintenance-data
    a sml:ApplyConstruct ;
    sm:next :transform ;
    sml:constructQuery [
        a sp:Construct ;
        sp:text """# 1 - extract ontology
CONSTRUCT {
    cm:wo-tc-ref-34567 a owl:Ontology ;
        owl:imports
        <http://onto.fel.cvut.cz/ontologies/csat-maintenance>
    .

    ?workpackageIRI a cm:workpackage, owl:NamedIndividual ;
        cm:id ?workpackageId ;
        cm:is-repair-of ?aircraftIRI ;
    .

    ?aircraftIRI a cm:aircraft, owl:NamedIndividual;
        cm:registration ?acRegistration ;
        cm:age ?acAge ;
    .
} WHERE {
    ?rd :AC ?acRegistration .
    ?rd :A_C_age ?A_C_age .
    ?rd :WP ?workpackageId .
    BIND(dl:instance(cm:workpackage, ?workpackageId) as ?workpackageIRI)
```

```
        BIND(?A_C_age as ?acAge)
        BIND(dl:instance(cm:aircraft, ?workpackageId) as ?aircraftIRI)
    }""" ;
        ] ;
      sml:replace true ;
      rdfs:label "construct-csat-maintenance-data" ;
```

The possible result of the mentioned script could be the following (represented using Turtle syntax):

```
cm:wo-tc-ref-34567 a owl:Ontology ;
        owl:imports <http://onto.fel.cvut.cz/ontologies/csat-maintenance>
.
cm:workpackage--PH-XRY-H-21-HMV9-AWL-LDG a cm:workpackage,
owl:NamedIndividual;
cm:has-client cm:client--TAV;
cm:id "PH-XRY/H-21 HMV9+AWL+LDG";
   cm:is-repair-of cm:aircraft--PH-XRY-H-21-HMV9-AWL-LDG;
.
cm:aircraft--PH-XRY-H-21-HMV9-AWL-LDG a cm:aircraft, owl:NamedIndividual;
cm:age "18.783561";
   cm:registration "PH-XRY"
.
```

3.3 Analysis of charting libraries

The implementation of the application involves the creation of the graphical user interface (GUI). The system design of the application separates the presentation layer from the rest of the system. Since the application is built on the web technology stack, the communication with the controllers is done through asynchronous JavaScript HTTP requests (AJAX). The user interface of the application will primarily consist of the data charts, i.e. tools for data metric visualization, in which the values are represented by graphical elements. Therefore its implementation will require a choice of the JavaScript library that can create the data charts of necessary types. The main requirements for the charting library are the following:

R1 it has to be open-source and actively maintained

R2 it has to work with HTML5 elements (Canvas [4] or SVG [5])

R3 it has to be lightweight and fast

R4 it has to be responsive [6]

[4] More information about Canvas available at <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas>

[5] More information about SVG available at <https://developer.mozilla.org/en-US/docs/Web/SVG/Element/svg>

[6] More information about Responsive Design available at <https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design>

R5 it has to work in modern browsers defined in NFR1 of non-functional requirements (4.3)

R6 it has to support multiple X and Y axes

R7 it has to have minimal dependencies

R8 it has to have sufficiently good documentation

R9 it has to be able to visualize the following chart types*: bar, stacked bar,line, area, stacked area, line column, pie

** the mentioned chart types are referring to the definitions from the ThoughtSpot Software Documentation* [7]

The following section describes five libraries that were considered as graph visualization solutions. Every library was checked for compliance with the stated requirements and subsequently compared with the others.

3.3.1 CHARTIST.JS

CHARTIST.JS [8] is a charting library for creating simple responsive charts with the aim of providing customization capabilities. The main benefits of the library are the following:

1. based on SVG, i.e. using vector graphics
2. open-source and free for all uses
3. lightweight and fast
4. supports animations
5. responsive charts of all required types
6. no dependencies
7. compatible with the majority of the modern browsers
8. highly customizable

According to the documentation of the library, it supports only three types of charts: line, bar, and pie. However, because of its high customization abilities, these basic chart types can be used to build more complex ones. For instance, a simple line chart can be used as a basis for building the stacked area chart with multiple series:

[7] ThoughtSpot Software Documentation is available at <https://docs.thoughtspot.com/software/6.2/about-charts>

[8] CHARTIST.JS homepage available at <http://gionkunz.github.io/chartist-js/>

```
new Chartist.Line('.ct-chart', {
  labels: [1, 2, 3, 4, 5, 6, 7, 8],
  series: [
    [447, 681, 572, 409, 449, 332, 642, 670, 672],
    [666, 641, 540, 434, 343, 545, 396, 323, 443],
    [652, 548, 302, 405, 621, 361, 505, 592, 612],
    [356, 404, 492, 590, 699, 300, 420, 395, 413]
  ]
}, {
  low: 0,
  showArea: true
});
```

The result of the code is depicted in the figure 3.2:

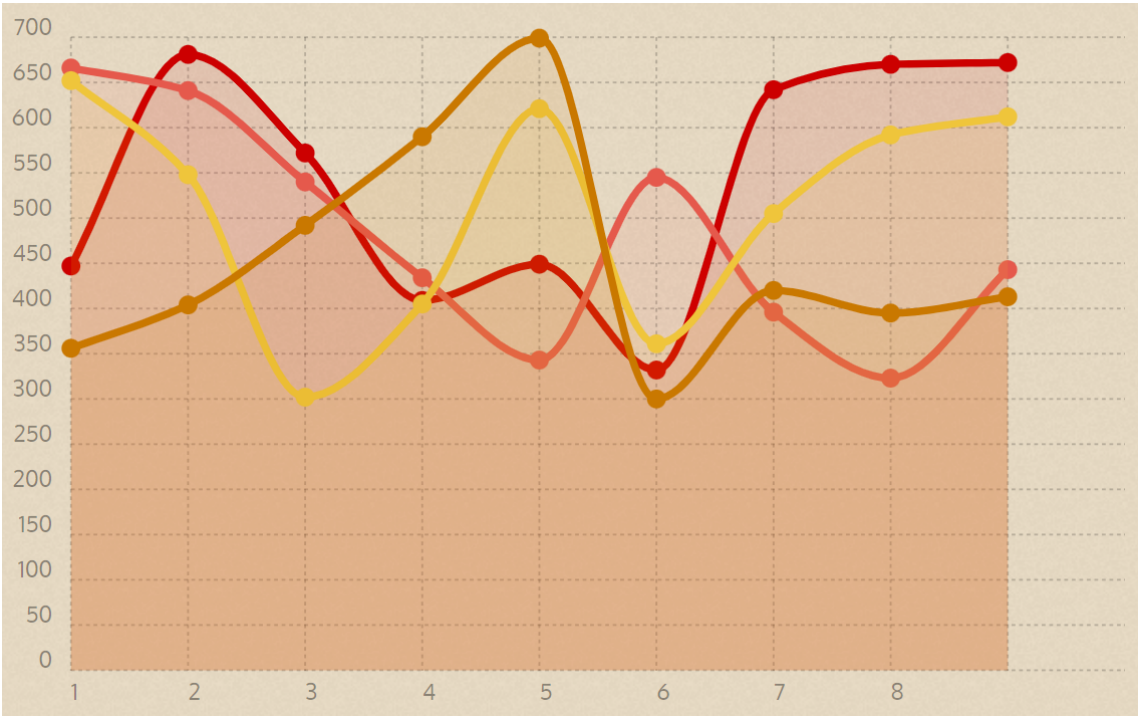


Figure 3.2: CHARTIST.JS - stacked area chart with multiple series

The library fulfills the majority of the stated requirements. However, there are several significant disadvantages, such as relatively weak documentation and quite a humble feature set. Furthermore, it does not support multiple X and Y axes out of the box, and, most importantly, it is not being actively maintained.

3.3.2 Chart.js

Chart.js [9] is a simple yet flexible JavaScript HTML5 based charting library for creating interactive and customizable charts. Its API is quite simple, decently organized, and well-documented. It has the following advantages:

1. open-source and free for all uses
2. lightweight and fast
3. responsive animated charts of required types
4. no dependencies
5. custom plugins support
6. compatible with the majority of the modern browsers
7. highly customizable and supports multiple axes
8. clear documentation

The Sample page [10] of the library demonstrates the examples of the various chart types, including:

- bar charts
  - horizontal and vertical bar charts
  - stacked bar chart
  - floating bars
- line chart
- area chart
- bubble chart
- scatter chart
- doughnut chart
- pie chart
- polar and radar charts

This set is absolutely sufficient and contains all chart types from the stated requirements. The example of the diagram with multiple chart types (stacked bar/line) is depicted in the image below.

[9] Chart.js homepage available at <http://www.chartjs.org/>
[10] Sample page is available at <https://www.chartjs.org/docs/latest/samples/information.html>

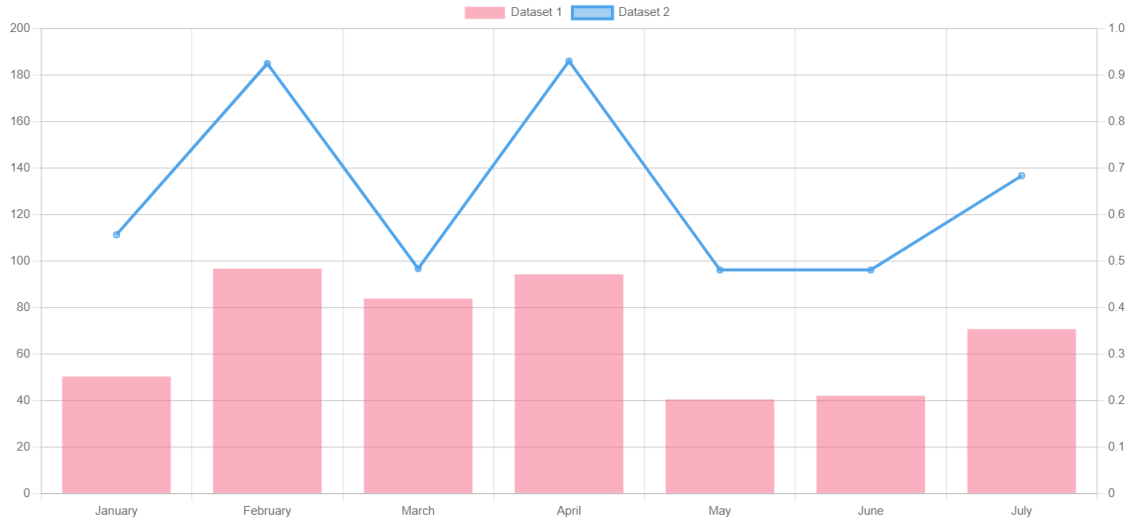


Figure 3.3: Chart.js - stacked bar/line chart with multiple Y axes

One of the most important advantages of Chart.js is the support of customized plugins. It provides the ability to modify the existing graphical elements of the charts, as well as to create personalized ones. In addition, there exist a good deal of useful plugins created by the community [11].

In fact, the only drawback of this library is that it is based on the raster graphics (i.e. it uses a Canvas API of the HTML5), meaning they cannot scale up losslessly. Nevertheless, it is still responsive and meets all the stated requirements.

3.3.3 Highcharts

Highcharts [12] is a charting library that comes with plenty of tools for creating reliable and secure data visualizations. It is built on JavaScript and TypeScript and works with multiple back-end technologies. Highcharts library also comes with the wrappers for the programming languages, such as .Net, PHP, Python, R, Java. The mobile platforms are also supported (iOS and Android) as well as the most popular front-end frameworks like Angular, Vue, and React. [51]

The following advantages are offered by the Highcharts library:

- based on SVG and can be saved as raster image
- supports many different charts with many options
- provides good documentation and plenty of examples
- excellent browser support

[11] List of plugins available at <https://github.com/chartjs/awesome>
[12] Highcharts homepage available at <https://www.highcharts.com/>

- has a large and active community
- supports multiple Y axes

As can be seen from the demo pages [13], the library includes all essential chart types, such as:

- line charts
- area charts
- column and bar charts
- pie charts
- scatter and bubble charts
- and many others, including event 3D versions of them

The library is extremely customizable and can combine basically unlimited numbers of charts together. A good example of a chart showing the capabilities of the library is portrayed in the image below:

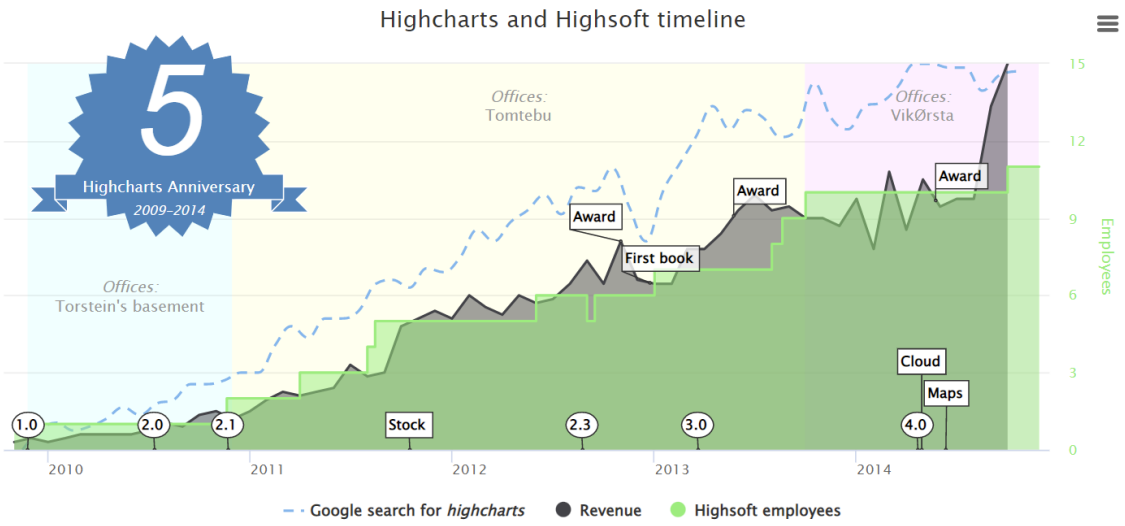


Figure 3.4: Highcharts - Highcharts and Highsoft timeline [52]

This is an advanced example showing a combination of various features, including flags and plot bands. It comprises multiple charts of different types with several notes, labels, and graphical elements. The library fulfills the majority of defined requirements, although it may appear to be quite complex and not lightweight due to the number of functionalities it provides. Moreover, the library has a proprietary license, meaning it is free for personal and non-commercial uses but paid for commercial applications.

[13] Demo page available at <https://www.highcharts.com/demo>

3.3.4 Google Charts

Google Charts [14] is a charting library developed by Google. It allows users to create simple charts without any complex interactions [53] and provides a variety of interactive and zoomable visualization elements based on pure HTML5/SVG technology. The following strengths can be highlighted in the library:

- cross-browser compatibility (including older IE versions)
- cross-platform portability to iOS and Android
- no dependencies, no plugins are needed
- good documentation, plenty of guides and samples
- easy to use
- an extensive number of different chart types, such as:
  - area charts
  - bar charts
  - bubble charts
  - calendar charts
  - candlestick charts
  - Gantt charts
  - line charts
  - and many others, including word trees and 3D charts

The following image demonstrates the capabilities of the library on the example of a moderately complex chart, combining two different chart types with multiple series of the data:

Despite the listed advantages, the library has a fundamental flaw. In order to use the interactive features of Google Charts, it requires the users' computers to have access to the gstatic server [15] (where the JS library script is hosted). The reason is that the visualization libraries are loaded dynamically in advance. Moreover, the terms of service of the library do not allow downloading the particular code. This makes it impossible to use on an intranet, in offline mode, or to be hosted locally. [55] Therefore the library might not be the best option for enterprise systems or systems with sensitive data.

3.3.5 D3.js

D3.js is a JavaScript charting library for using HTML, SVG, and CSS. Its emphasis on web standards provides the full capabilities of modern browsers and combines powerful visualization components for data representation. [56]

[14] Google Charts homepage available at <https://developers.google.com/chart/>
[15] Server gstatic available at <https://www.gstatic.com/charts/loader.js>

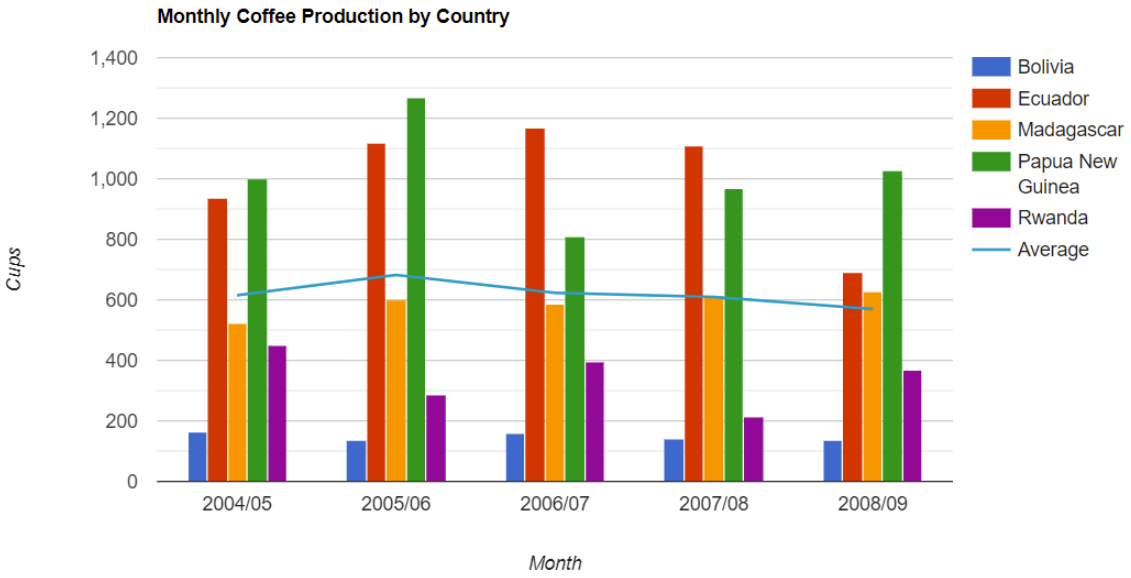


Figure 3.5: Google Charts - combo chart (bar/line) with multiple series of data [54]

The charts created with this library are based on SVG elements. There are a large number of different charts of all types, including various bar, line, area, radial, pie, donut charts, and others. The home web pages of the library provide a huge amount of examples showing its capabilities. It also provides decent documentation in form of a GitHub repository wiki [16].

The following figure 3.6 depicts the example of a relatively complex chart with multiple data series and negative values:

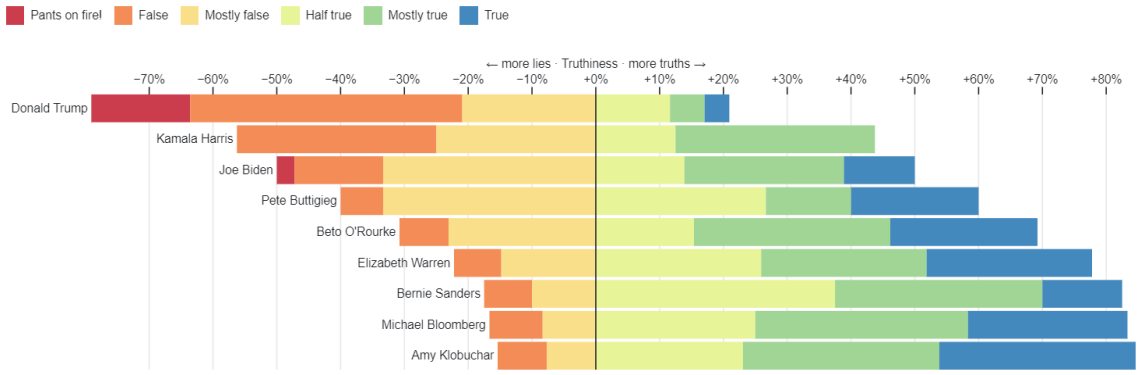


Figure 3.6: D3.js - horizontal stacked bar chart multiple series of data [57]

Amongst the strengths of D3.js is certainly the absence of dependencies, its lightweightness, and featurefulness. However, to some, it may seem that the library is overwhelmed to some degree and has a steep learning curve.

[16] D3.js GitHub repository wiki available at <https://github.com/d3/d3/wiki>

Library	R1	R2	R3	R4	R5	R6	R7	R8	R9
CHARTIST.JS	+	+	+	+	+	-	+	+	±*
Chart.js	+	+	+	+	+	+	+	+	+
Highcharts	-	+	-	+	+	+	-	+	+
Google Charts	±**	+	+	±***	+	-	+	+	+
D3.js	+	+	+	+	+	+	+	+	+

** basic chart types can be used to build the more complex ones; ** not open-source, but free for all uses; *** requires additional coding and configuration*

Table 3.6: Charting libraries - requirement coverage

3.3.6 Summary

The following table 3.6 summarizes the overview and comparison results with respect to the defined requirements. Each considered library is assigned a certain symbol in the column corresponding to one of the requirements, indicating the degree of compliance with the requirement.

The results show that both Chart.js and D3.js libraries best meet the stated requirements. However, when choosing a library, it should be taken into account the fact that D3.js has a steeper learning curve compared to Chart.js. It is also worth noting that Chart.js library is based on raster graphics, while the latter uses vector graphics. This should be considered since it might influence to a certain extent the related development process.

Chapter 4

Requirement analysis

The requirement analysis is the process of determining user expectations for a new or modified product. [59] It is an essential step in the early stages of software development. The software requirements are intended to provide a vision of the functionalities and capabilities of the software. It enables the developer team to prioritize tasks and goals during the rest of the development process.

In this chapter of the thesis, the software requirements are defined. They have been gathered from the regular meetings with a domain expert from CSAT throughout the development process of the project. The specified software requirements are divided into two groups: functional and non-functional. The functional requirements describe a particular behavior of functions of the system when certain conditions are met. On the other hand, the non-functional requirements are intended to describe how a system should behave and the limits on its functionality. [60]

4.1 Prioritization of requirements

Requirement prioritization helps to define the relative importance and urgency of different requirements to cope with the limited resources of projects. Its purpose is to ensure that the most critical requirements are addressed immediately in case time runs out. [61]

MoSCoW prioritization method is one of the widely-used prioritization techniques which consists of assigning a category to each requirement in order to indicate the degree of its importance. The acronym MoSCoW is formed from the following categories [62]:

- Must have initiatives represent non-negotiable needs for the project, product, or release in question
- Should have initiatives are essential to the product, project, or release, but they are not vital
- Could have initiatives are not necessary to the core function of the product
- Will not have (this time) initiatives are not a priority for this specific time frame but might be prioritized in the future.

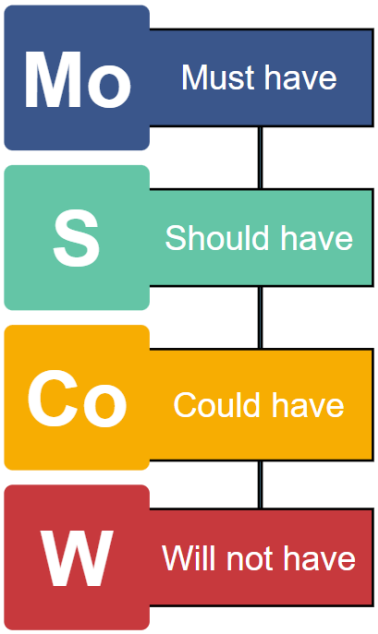


Figure 4.1: MoSCoW priorities

The following sections outline the functional and non-functional requirements. Each of the stated requirements is assigned with one of the MoSCoW priorities using the following marks [M] for "Must have", [S] for "Should have", [C] for "Could have", and [W] for "Will not have".

4.2 Functional requirements

FR1 Work package list page

FR1.1 [M] Users are able to view the list of the ongoing work packages

FR1.2 [S] Users are able to see the status of every listed WP

FR1.3 [C] Users are able to filter listed work packages by name using regular expression

FR1.4 [C] Users are able to filter listed work packages by end date

FR2 Data management

FR2.1 [M] Users are able to initiate the data update process

FR2.2 [S] Users are able to get information about the latest performed update

FR2.3 [S] Users are able to download a source dataset from the latest update

FR2.4 [M] System is able to consume data exported by AMOS system*

FR2.5 [M] System is able to consume personnel attendance data exported by CPACS*

**Both these systems are providing their data through the external dedicated secure FTP server in the form of multiple CSV files. A more detailed specification of CSV files is stated in section 3.1.4.*

FR3 Work package execution progress (Dashboard 1.0)

FR3.1 System is able to display the current execution progress of the selected WP by:

FR3.1.1 [M] number of remaining days until the end of the WP

FR3.1.2 [M] number of issued and closed task cards

FR3.1.3 [M] number of issued and closed scheduled work orders

FR3.1.4 [M] number of issued and closed non-routine cards

FR3.2 System is able to visualize the current resource utilization of the selected WP as a chart consisting of:

FR3.2.1 [M] resource utilization goal of WP and its responsible line

FR3.2.2 [M] overall average resource utilization of WP*

FR3.2.3 [M] overall average resource utilization of WP responsible line*

FR3.2.4 [M] amount of productive man-hours performed by maintenance technicians on each day of WP execution*

FR3.2.5 [M] amount of cost hours performed by maintenance technicians on each day of WP execution*

FR3.2.6 [M] resource utilization rate of maintenance technicians on each day of WP execution*

**None of these metrics should take into account the "support" maintenance groups: Administrative works group, Washer supervision group, Planner group*

FR3.3 [M] System is able to visualize the progress of NRC tasks (NRC progress) of the selected WP as a chart

FR3.4 System is able to visualize the current WO findings statistics of the selected WP as a chart consisting of:

FR3.4.1 [M] amount of estimated man-hours for each main heavy-maintenance group*

FR3.4.2 [M] amount of used man-hours for each main heavy-maintenance group*

**The set of main heavy-maintenance groups consist of: Avionics mechanic group, Sheet metal support group, Landing gear mechanic group, Engine mechanic group, Interior mechanic group, Exterior mechanic group*

FR3.5 System is able to display current information about the selected WP such as:

FR3.5.1 [M] WP name

FR3.5.2 [M] customer name

FR3.5.3 [M] aircraft model

FR3.5.4 [M] WP execution status

FR3.5.5 [M] WP start date and end date

FR4 Work package execution progress details (Dashboard 2.0)

FR4.1 System is able to display current detailed information about the selected WP such as:

FR4.1.1 [S] total amount of productive man-hours performed by maintenance technicians during WP execution*

FR4.1.2 [S] total amount of cost man-hours performed by maintenance technicians during WP execution*

FR4.1.3 [S] overall average resource utilization of WP*

**None of these metrics should take into account the "support" maintenance groups*

FR4.2 System is able to display current detailed information about the resource utilization related to the selected WP such as:

FR4.2.1 [W] planned amount of man-hours to be spent on a WP within agreed fixed price based on time estimation for inspections

FR4.2.2 [W] planned amount of man-hours to be spent on a WP above agreed fixed price based on time estimation for unplanned work orders

FR4.3 [W] System is able to display current detailed information about the resource utilization related to the responsible maintenance line of the selected WP

FR4.4 System is able to display detailed information about the resource estimate for the following maintenance groups of the selected WP:

FR4.4.1 [W] Avionics mechanic group

FR4.4.2 [W] Sheet metal support group

FR4.4.3 [W] Landing gear mechanic group

FR4.4.4 [W] Engine mechanic group

FR4.4.5 [W] Interior mechanic group

FR4.4.6 [W] Exterior mechanic group

It should be emphasized that the majority of the Dashboard 2.0 functional requirements (all except FR4.1) have [W] priority, which means that they will neither be implemented nor comprehensively specified in this thesis.

The following use case diagram (figure 4.2) depicts the interactions between the system and its users. It comprises the activities based on the functional requirements with higher priority, i.e. the ones that will be implemented in the software prototype.

There are two main system roles are distinguished: Plan Execution Manager, which requires additional privileges, and Maintenance Technician, which represents a basic user. The basic users are only interested in the main dashboard statistics and do not need any interaction with the system. On the other hand, the manager wants to be able to utilize extra functionalities which are not visible to the ordinary employee.

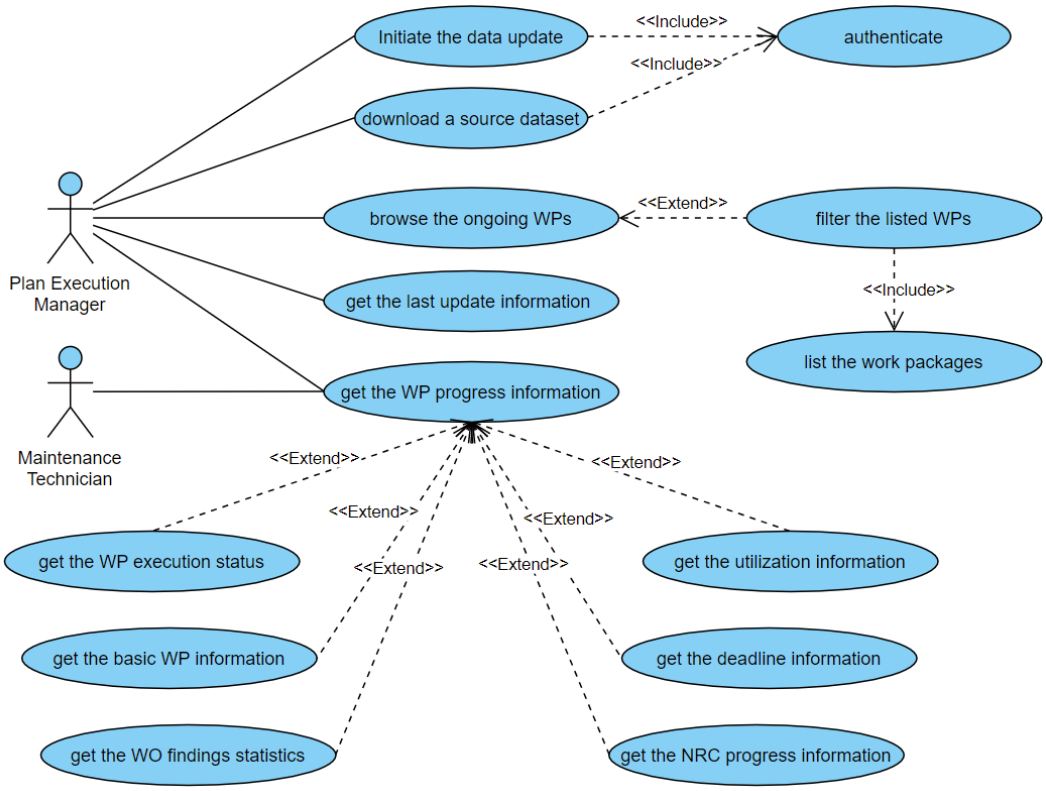


Figure 4.2: Use case diagram: functional requirements

4.3 Non-functional requirements

NFR1 System is accessible using modern internet desktop web browsers:

- NFR1.1 [M] Google Chrome version 96.0.4664 or higher
- NFR1.2 [M] Firefox 91.0 version or higher
- NFR1.3 [M] Safari version 14.0 or higher

NFR2 [M] System user interface is in English

NFR3 [M] System is designed to be extensible to enable integrability with other systems

NFR4 [S] System is developed using a modern and widely used technology stack

NFR5 [M] Only authenticated users are able to access the system

NFR6 [S] The charts for metric visualization are created using the charting library that meets all the requirements stated in chapter 3.3.

Chapter 5

System Design

This chapter is intended to describe general system design decisions. It includes information about system components, services, and modules, as well as the definition and specification of the dashboard metrics required to meet the requirements.

5.1 Domain model

In order to define object-relational mapping (ORM), it is needed to introduce the domain model of the application.In order to define object-relational mapping (ORM), it is needed to introduce the domain model of the application. This process began with an examination of the already existing "CSAT-maintenance" ontology. This process was accompanied by consultations and discussions with domain experts and ontology engineers. During the development process, an extension ontology "CSAT-maintenance-spec" was created, in which the concepts not suitable for the target system were changed and expanded. The diagram below reflects a "CSAT-maintenance-spec" domain model of the aircraft maintenance execution process as a collection of OWL classes, relations, and properties. For readability, the following prefixes are used instead of full URIs (presented using Turtle syntax):

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix cm:   <http://onto.fel.cvut.cz/ontologies/csat-maintenance/> .
```

Every relation presented on the diagram is either a type inference (*rdfs:subClassOf*), object property (*owl:ObjectProperty*) linking source class to target class, or a data property (*owl:DatatypeProperty*) linking class to an attribute. Each particular instance also implicitly includes an attribute, representing its identifier (cm:id).

One of the core entities of the domain model is a work package (*cm:workpackage*). This entity reflects a group of related tasks within a maintenance plan execution and is always associated with a single client (*cm:client* by *cm:has-client* relation) and an aircraft (*cm:aircraft* by *cm:is-repair-of relation*). The work package entity also includes a set of attributes: *cm:start-date-scheduled*, *cm:end-date-scheduled*, *cm:start-date*, *cm:end-date*, providing the following information respectively: work package scheduled start date, scheduled end date, actual start date, and actual end date. The aircraft entity, in turn, has three attributes:

cm:model, showing a model code of the aircraft, *cm:age*, reflecting the age of the aircraft, and a unique aircraft registration number, represented by *cm:registration* attribute.

The work package can be assigned to a single responsible (or main) maintenance line (*cm:maintenance-line*). It is reflected using relation *cm:has-responsible-maintenance-line*. Each work package embraces a collection of maintenance tasks (*cm:maintenance-task*), which provide details about maintenance, repair, or operations work, such as replacing a part, returning an asset to operating condition, or performing an inspection of aircraft components, etc.

The maintenance tasks indicate which action an assigned maintenance technician (or simply mechanic *cm:mechanic*) is supposed to complete. It comprises two data properties: issue time (*cm:issue-time*), indicating when the task was issued, and end time (*cm:end-time*), indicating the end time in case the task is finished. In addition, there are two major categories of tasks - planned tasks (*cm:planned-task*) and non-routine tasks, i.e. work orders (*cm:work-order*). The planned tasks are the task orders issued by a client or organization, usually for a service technician. Every non-routine task is either the scheduled task (*cm:scheduled-work-order*) or the failure-finding (represented by *cm:maintenance-work-order*). Failure-findings, i.e. non-routine cards (NRC) used to reveal hidden or potential failures of components. It should be noted that every scheduled task is also a planned task. Planned tasks comprise two different types - client extra order task (*cm:client-extra-order-task*) or a task card (*cm:task-card*).

Every work order consists of one or more execution steps (*cm:scoped-task-step*). Each step is interconnected with its work order through the corresponding work order step (*cm:work-order-step*) and belongs to a single maintenance group (*cm:maintenance-group*), namely a scope. Furthermore, the majority of execution steps have a time estimate represented by *cm:time-estimate-in-hours* attribute. On the other hand, the work order steps (*cm:work-order-step*) contain two additional attributes: *cm:work-order-text*, indicating the description of the corresponding task, and *cm:work-order-action-text*, reflecting a description of the aircraft maintenance activity performed.

Service technicians (mechanics) perform maintenance, operations, diagnostic testing, and repairs to complete assigned maintenance tasks. Their activity is represented by the *cm:work-session* entities (through *cm:performs* relation), containing information about the performed task and the amount of spent hours. Work activity time rarely matches the actual workplace attendance, therefore an additional class *cm:presence* is introduced to represent the period of time when employees are at work. Work sessions, as well as maintenance tasks, always belong to a certain maintenance group (*cm:maintenance-group*). Every maintenance technician (mechanic) is assigned to a single maintenance group (*cm:maintenance-group*), also referred to as a shift group. Moreover, both *cm:work-session* and *cm:presence* classes are sub-classes of the event class (*cm:event*). Such events contain five attributes: *cm:duration-in-hours*, reflecting the duration of the event (in hours), *cm:start-date* with *cm:start-time*, indicating when the event started, and *cm:end-date* with *cm:end-time*, representing an instant when the event ended. Additionally, in case the start date and end date of the event differ, the event can be divided into multiple instances of *cm:day-bound-event*, which are basically the events that started and ended within one day. Then, *cm:day-bound-event* the must refer to the original event through *cm:is-day-bound-part-of* relation.

5.2 Navigation

The navigation part of the application is designed in accordance with the described functional requirements (4.2), namely the ones defined under the FR1 group. It is intended to allow the users to navigate between different views of the application to access its functions. The navigation part includes a work package explorer and the navigation menu.

The work package explorer is done in the form of a separate web page. It is responsible for listing the existing work packages and performing filtering based on the name and end date of the work package. Each listed item is assumed to have a navigation element which can be used to open the dashboard page of a corresponding work package. Moreover, a schedule status of the WP is assumed to be presented for each listed item.

Additionally, the navigation menu is designed to be accessible from every page of the application. It is supposed to provide the navigation links to access different parts of the application, such as Dashboard 1.0, Dashboard 2.0, work package explorer, etc.

5.3 Data management

According to the requirements stated in (4), the data management is one of the objectives of the application. It includes system processes designed to keep the application data up-to-date, as well as communication and data exchange with the source.

The whole data update process can be initiated by the user manually or by an internal system event (such as a timer) and comprises the following steps:

- download of an exposed source dataset, provided by AMOS and CPACS through secured FTP server
- transformation of the fetched tabular data into domain model compliant RDF (as entities)
- processing of the incoming entities
- propagation of the entities changes to the persistent database storage.

Additionally, the system should provide users with the ability to access (download) the source dataset that was utilized in the most recent successful update. It is assumed that the graphical user interface of the application will contain the control elements such as a button to initiate the update or the links to download the source data from the latest successful update. Furthermore, the interface should also provide information about the date and time of the last successful update, as well as a status message reflecting the progress of the ongoing data update process.

5.4 Dashboard 1.0

This section outlines the purpose and advantages of Dashboard 1.0 with respect to the original solution used by CSAT. In addition, the prototype of the dashboard page is demonstrated, as well as the compulsory metrics, required to fill the charts with the data.

5.4.1 Data representation

According to the requirements stated in (4), Dashboard 1.0 should visualize metrics related to the execution progress of the selected work package. It is also supposed to facilitate the monitoring of work performance and effectiveness of resource utilization. Initially, CSAT used a simple visualization of these metrics done in the form of spreadsheet charts (depicted in figure 5.2).

Each update was time-consuming since it was done manually, and therefore could lead to potential errors and inconsistencies. Moreover, the metric values contained in the original dashboard lack accuracy and precision and therefore could not be used for more demanding use cases.

The Dashboard 1.0 is designed on the basis of the structure of the original dashboard with certain improvements in UI and component layout.

Dashboard structure comprises a set of charts and metrics and consists of multiple separate segments: work package details, work package execution status, NRC progress, utilization, and work order findings. A brief description of these segments is provided in the following sections. For each mentioned metric, a function will be assigned to calculate its value.

Work package details

This section of the dashboard provides the information about the selected work package wp, including work package identifier $WP_{id}(wp)$, client (customer) label $WP_{cl}(wp)$, and a model of aircraft $WP_{am}(wp)$.

Work package execution status

This part of the dashboard discloses the maintenance plan execution status and consists of the following elements:

- schedule status $WP_{ss}(wp)$
- number of issued $TC_{issued}(wp)$ and closed TC $TC_{closed}(wp)$
- number of issued $SWO_{issued}(wp)$ and closed SWO $SWO_{closed}(wp)$
- number of issued $MWO_{issued}(wp)$ and closed NRC $MWO_{closed}(wp)$
- overall information about the schedule, i.e. WP scheduled start date $WP_{ssd}(wp)$, scheduled end date $WP_{sed}(wp)$ and actual end date $WP_{aed}(wp)$.

NRC progress

This section of the dashboard consists of a single chart that shows the number of closed NRC $MWO_{closed}(d, wp)$ and a number of issued NRC $MWO_{issued}(d, wp)$ on each day d throughout the work package wp execution.

Utilization

The utilization chart of the dashboard is intended to depict the relation between productive hours $H_{prod}(d, wp)$ and cost hours $H_{cost}(d, wp)$ spent by employees on each day d within the work package wp. This relation is also depicted on the chart and shows the productivity (utilization) $UT(d, wp)$ per day d. In addition, the average resource utilization of the responsible (main) maintenance line $UT_{rline}(wp)$ of the work package wp is portrayed.

Work order findings

This section of the dashboard consists of a single chart displaying a relation between the total time estimate of maintenance tasks $H_{est}(sc, wp)$ per each scope sc and the amount of the productive hours $H_{prod}(sc, wp)$ spent by employees on the corresponding maintenance tasks.

5.4.2 Metric functions

In this section the formulas for dashboard metric calculation are provided. The stated formula definitions are primarily describing the sets by enumerating their elements or stating the properties that its members must satisfy. The ones that use predicates are given in a set-builder notation [1].

There are multiple types of predicate presented in the set definitions:

- object predicates, e.g. $tc.is\text{-}part\text{-}of\text{-}workpackage = wp$, where tc and wp are variables basically means that there must exist the following triple:

```
?tc cm:is-part-of-workpackage ?wp .
```

- property predicate, for instance:
  - $mg.has\text{-}abbreviation \in \{"WASH", "ADM", "PLA"\}$, where mg is variable basically means that there must not exist the following triples:

```
?mg cm:has-abbreviation ‘‘WASH’’ .
?mg cm:has-abbreviation ‘‘ADM’’ .
?mg cm:has-abbreviation ‘‘PLA’’ .
```

  - $tc.end\text{-}date = \emptyset$, where tc is variable basically means that there must not exist the following triple:

```
?tc cm:end-date ?anything .
```

  - $tc.end\text{-}date \neq \emptyset$, where tc is variable basically means that there must exist the following triple:

```
?tc cm:end-date ?anything .
```

There are also several getter functions presented that always return a single property value or none. To exemplify $F(x) = x.has\text{-}y.has\text{-}name$ where x is a variable will return the result of the following SPARQL expression:

[1] More information about set-builder notation available at <https://www.cuemath.com/algebra/set-builder-notation/>

```
SELECT ?z WHERE {
 ?x cm:has-y ?y .
 ?y cm:has-name ?z .
}
```

The list of getter functions relevant for the dashboard is presented below:

- get identifier of the responsible (main) maintenance line of the work package *wp*: $RL(wp) = wp.has\text{-}responsible\text{-}maintenance\text{-}line$
- get identifier of the work package *wp*: $WP_{id}(wp) = wp.id$
- get label of the client (customer) of the work package *wp*: $WP_{cl}(wp) = wp.has\text{-}client$
- get model of the aircraft of the work package *wp*: $WP_{am}(wp) = wp.is\text{-}repair\text{-}of.model$
- get scheduled start date of the WP *wp*:
  $WP_{ssd}(wp) = wp.workpackage\text{-}scheduled\text{-}start\text{-}time$
- get scheduled end date of the WP *wp*:
  $WP_{sed}(wp) = wp.workpackage\text{-}scheduled\text{-}end\text{-}time$
- get actual end date of the work package *wp*: $WP_{aed}(wp) = wp.workpackage\text{-}end\text{-}time$
- get schedule status of the work package :

$$\begin{cases} \text{"}Behind\ schedule\text{"},\ if\ now() < WP_{sed}(wp)\ \|\ WP_{sed}(wp) < WP_{aed}(wp) \\ \text{"}Ahead\ of\ schedule\text{"},\ if\ now() \geq WP_{sed}(wp)\ \&\ WP_{sed}(wp) \geq WP_{aed}(wp) \\ \text{"}On\ schedule\text{"},\ otherwise \end{cases} \quad (5.1)$$

The following sets are presented in metric definitions:

- TC = set of all individuals of type *cm:task-card*
- WP = set of all individuals of type *cm:workpackage*
- WS = set of all individuals of type *cm:work-session*
- DBE = set of all individuals of type *cm:day-bound-event*
- MG = set of all individuals of type *cm:maintenance-group*
- SWO = set of all individuals of type *cm:scheduled-work-order*
- MWO = set of all individuals of type *cm:maintenance-work-order*
- P = set of all individuals of type *cm:presence*
- M = set of all individuals of type *cm:mechanic*
- C = set of all individuals of type *cm:client*

- A = set of all individuals of type *cm:aircraft*
- STS = set of all individuals of type *cm:scoped-task-step*
- $MG_{support} = \{mg \in MG \| mg.has\text{-}abbreviation \in \{"WASH","ADM","PLA"\}\}$ defines a set of irrelevant support maintenance groups, i.e. Administrative works group, Washer supervision group, and Planner group

For example, the TC set can be obtained by executing the following SPARQL expression:

```
SELECT DISTINCT ?tc WHERE {
    ?tc a cm:task-card .
}
```

In order to dramatically reduce the complexity and size of the formulas, the following assumption was introduced: every *cm:event* (i.e. work sessions (*cm:work-session*) and attendances (*cm:presence*)) starts and ends within the same day. In other words, *cm:start-date* and *cm:end-date* of such entities are assumed to be the same. Nevertheless, in practice, this is not always the case, especially when it comes to night shifts. Therefore, to solve this problem, every event for which the assumption does not apply is split into multiple *cm:day-bound-event* entities.

Metric 1: number of closed TC in work package $wp \rightarrow |TC_{closed}(wp)|$
wp = identifier of the work package

$$TC_{closed}(wp) = \{tc \in TC \| tc.is\text{-}part\text{-}of\text{-}workpackage = wp, tc.end\text{-}date \neq \emptyset\} \tag{5.2}$$

Metric 2: number of issued TC in work package $wp \rightarrow |TC_{issued}(wp)|$

$$TC_{issued}(wp) = \{tc \in TC \| tc.is\text{-}part\text{-}of\text{-}workpackage = wp\} \tag{5.3}$$

Metric 3: number of closed SWO in work package $wp \rightarrow |SWO_{closed}(wp)|$

$$SWO_{closed}(wp) = \{swo \in SWO \| swo.is\text{-}part\text{-}of\text{-}workpackage = wp, swo.end\text{-}date \neq \emptyset\} \tag{5.4}$$

Metric 4: number of issued SWO in work package $wp \rightarrow |SWO_{issued}(wp)|$

$$SWO_{closed}(wp) = \{swo \in SWO \| swo.is\text{-}part\text{-}of\text{-}workpackage = wp\} \tag{5.5}$$

Metric 5: number of closed MWO in work package $wp \rightarrow |MWO_{closed}(wp)|$

$$MWO_{closed}(wp) = \{mwo \in MWO \| mwo.is\text{-}part\text{-}of\text{-}workpackage = wp, mwo.end\text{-}date \neq \emptyset\} \tag{5.6}$$

Metric 6: number of issued MWO in work package $wp \rightarrow |MWO_{issued}(wp)|$

$$MWO_{closed}(wp) = \{mwo \in MWO \| mwo.is\text{-}part\text{-}of\text{-}workpackage = wp\} \tag{5.7}$$

Metric 7: number of closed MWO in work package wp on the day $d \rightarrow |MWO_{closed}(d, wp)|$

$$MWO_{closed}(d, wp) = \{mwo \in MWO \| mwo.is\text{-}part\text{-}of\text{-}workpackage = wp, \\ mwo.end\text{-}date < d\} \tag{5.8}$$

Metric 8: number of issued MWO in work package wp on the day $d \rightarrow |MWO_{issued}(d, wp)|$

$$MWO_{issued}(d, wp) = \{mwo \in MWO \| mwo.is\text{-}part\text{-}of\text{-}workpackage = wp, \\ mwo.issue\text{-}date < d\} \tag{5.9}$$

Metric 9: total duration of all attendances during the day d within work package wp besides the irrelevant support maintenance groups $\rightarrow H_{cost}(d, wp)$

$$H_{cost}(d, wp) = \sum_{m \in M_{dwp}(d,wp)} H_{cost}(d, m, wp) \tag{5.10}$$

$M_{dwp}(d, wp)$ defines a set of mechanics worked at work package wp during the day d

$$M_{dwp}(d, wp) = \{m \in M \| m.performs.ipomt.ipow = wp, m.performs.start\text{-}date = d, \\ m.performs.ipomt.has\text{-}scope \notin MG_{support}\} \tag{5.11}$$

where *ipomt* is a shortcut for *is-part-of-maintenance-task* and *ipow* is a shortcut for *is-part-of-workpackage*.

$H_{cost}(d, m, wp)$ defines a total duration of all attendances of mechanic m during the day d within work package wp besides the irrelevant support maintenance groups

$$H_{cost}(d, m, wp) = H_{cost}(d, m) * coeff(d, m, wp) \tag{5.12}$$

$$H_{cost}(d, m) = \sum_{p \in P_{dm}(d,m)} p.duration\text{-}in\text{-}hours \tag{5.13}$$

$$P_{dm}(d, m) = \{p \in P \| p.start\text{-}date = d, p.has\text{-}participant = m\} \tag{5.14}$$

Since *cm:presence* entities do not have any property that directly connects it to a work package, a multiplier value has been introduced. This value helps to approximate the real attendance duration with respect to the work package. The value calculation is based on the work sessions of the corresponding mechanic m on the corresponding day d within the required work package wp and is defined by the $coeff(d, m, wp)$.

$$coeff(d, e, wp) = H_{prod}(d, m, wp)/H_{prod}(d, m) \tag{5.15}$$

$H_{prod}(d, m, wp)$ defines a total duration of all work sessions performed by mechanic m during the day d within work package wp besides the irrelevant support maintenance groups

$$H_{prod}(d, m, wp) = \sum_{ws \in WS_{dmwp}(d,m,wp)} ws.duration\text{-}in\text{-}hours \tag{5.16}$$

$WS_{dmwp}(d, m, wp)$ defines a set of work sessions performed by mechanic m during the day d within work package wp besides the irrelevant support maintenance groups

$$\begin{aligned} WS_{dmwp}(d, m, wp) = \{ws \in WS \| ws.ipomt.ipow = wp, m.performs = ws, \\ ws.has\text{-}scope \notin MG_{support}, ws.start\text{-}date = d\} \end{aligned} \tag{5.17}$$

$H_{prod}(d, m)$ defines a total duration of all work sessions performed by mechanic m during the day d across all work packages

$$H_{prod}(d, m) = \sum_{wp \in WP} H_{prod}(d, m, wp) \tag{5.18}$$

It must be noted that there are cases when some mechanic m on some date d has zero presence time, i.e. $H_{cost}(d, m, wp) = 0$, despite the fact that the same mechanic m on the same date has some work records, i.e. $H_{prod}(d, m, wp) > 0$. This is an indication of inconsistencies in source data, probably caused by situations when employees forget to clock in or out. As a solution for such cases the artificial attendance values $H'_{cost}(d, m, wp)$ will be used for computation instead of $H_{cost}(d, m, wp)$, given as $H'_{cost}(d, m, wp) = H_{prod}(d, m, wp)$.

Metric 10: total duration of all work sessions performed during the day d within work package wp besides the irrelevant support maintenance groups $\rightarrow H_{prod}(d, wp)$

$$H_{prod}(d, wp) = \sum_{m \in M} H_{prod}(d, m, wp) \tag{5.19}$$

Metric 11: utilization (productivity) on day d within work package $wp \rightarrow UT(d, wp)$

$$UT(d, wp) = H_{prod}(d, wp)/H_{cost}(d, wp) \tag{5.20}$$

Metric 12: average utilization of the responsible (main) maintenance line of work package $wp \rightarrow UT_{rline}(wp)$

$UT_{rline}(wp) = UT_{line}(line, t1, t2)$, where $line = RL(wp)$, $t1 = wp.scheduled\text{-}start\text{-}time$ and $t2 = now()$, i.e. today's date

$UT_{line}(line, t1, t2)$ defines the utilization of line line during interval $< t1, t2)$

$$UT_{line}(line, t1, t2) = \frac{\sum_{d \in DATES(t1,t2)} \sum_{m \in M_{line}(line)} H_{prod}(d, m)}{\sum_{d \in DATES(t1,t2)} \sum_{m \in M_{line}(line)} H_{cost}(d, m)} \tag{5.21}$$

$DATES(t1, t2)$ defines a set of every date in interval $< t1, t2)$, for instance, the function $DATES(2042\text{-}01\text{-}01, 2042\text{-}01\text{-}04)$ yields the following set of dates: *{ 2042-01-01, 2042-01-02, 2042-01-03 }*

$M_{line}(line)$ defines a set of mechanics having shift group related to line $line$

$M_{line}(line) = \{m \in M \| m.ipoml = line\}$

Metric 13: total time estimate of maintenance work orders per scope sc within work package $wp \rightarrow H_{est}(sc, wp)$

$$H_{est}(sc, wp) = \sum_{sts \in STS_{scwp}(sc,wp)} sts.time\text{-}estimate\text{-}in\text{-}hours \tag{5.22}$$

$STS_{scwp}(sc, wp)$ defines set of all scoped task steps per scope sc within work package wp

$$\begin{aligned} STS_{scwp}(sc, wp) = \{sts \in STS \| sts.is\text{-}part\text{-}of \in MWO, \\ sts.is\text{-}part\text{-}of.ipow = wp, sts.has\text{-}scope = sc\} \end{aligned} \tag{5.23}$$

Metric 14: total duration of all work sessions performed within work package wp and scope $sc \rightarrow H_{prod}(sc, wp)$

$$H_{prod}(sc, wp) = \sum_{ws \in WS_{scwp}(sc,wp)} ws.duration\text{-}in\text{-}hours \tag{5.24}$$

$WS_{scwp}(sc, wp)$ defines a set of work sessions performed within work package wp and scope sc

$$WS_{scwp}(sc, wp) = \{ws \in WS \| ws.ipomt.ipow = wp, m.performs = ws, ws.has\text{-}scope = sc\} \tag{5.25}$$

5.5 Dashboard 2.0

This section partially outlines the purpose and functionalities of Dashboard 2.0. It should be stressed that the scope of this thesis covers neither the implementation of the dashboard prototype nor the comprehensive definition of the dashboard metrics.

5.5.1 Data representation

The dashboard is intended for monitoring operational metrics of the work package, such as aircraft maintenance progress of individual revisions and maintenance lines.

In accordance with the functional requirements, Dashboard 2.0 is intended to visualize the work package execution progress in a more detailed way than in Dashboard 1.0.

A partial realization of the dashboard should portray the following information:

- total amount of productive man-hours performed by mechanics during WP execution
- total amount of cost man-hours performed by mechanics during WP execution

- overall average resource utilization of WP

It is worth noting that as in the Dashboard 1.0 none of the mentioned metrics should take into account the values associated with the "support" maintenance groups. These maintenance groups are defined by the $MG_{support}$ set in the Metric functions section of Dashboard 1.0.

5.5.2 Metric functions

In this section, the formulas for dashboard metric calculation are provided. The stated formula definitions follow the same notation and share the sets and formulas from the corresponding section of Dashboard 1.0 description.

Metric 1: total duration of all attendances within the work package wp besides the irrelevant support maintenance groups $\rightarrow H_{cost}(wp)$

$Hcost(wp) = H_{cost}(t1, t2, wp)$, where $t1 = wp.scheduled\text{-}start\text{-}time$ and $t2 = now()$, i.e. today's date

$$H_{cost}(t1, t2, wp) = \sum_{d \in DATES(t1,t2)} H_{cost}(d, wp) \tag{5.26}$$

Metric 2: total duration of all work sessions performed within the work package wp besides the irrelevant support maintenance groups $\rightarrow H_{cost}(wp)$

$Hprod(wp) = H_{prod}(t1, t2, wp)$, where $t1 = wp.scheduled\text{-}start\text{-}time$ and $t2 = now()$, i.e. today's date

$$H_{prod}(t1, t2, wp) = \sum_{d \in DATES(t1,t2)} H_{prod}(d, wp) \tag{5.27}$$

Metric 3: overall utilization (productivity) within the work package $wp \rightarrow UT(wp)$

$$UT(wp) = H_{prod}(wp)/H_{cost}(wp) \tag{5.28}$$

5.6 System modules

In order to back the extensibility and integrability of the application the system is designed to consist of several modules. Each module is designed to be responsible for a specific set of functionalities it provides. This section of the thesis provides details about the particular components and services of the system.

5.6.1 API gateway

The API gateway is a module that serves as the "front door" of the application. The system design assumes it to be the only publicly available component of a system, exposing the HTTP RESTful API. Using such a gateway API service in the system architecture allows to lower the coupling degree and eliminate its dependence on internal subsystems and components, as well as to unify the API for external clients. This is made possible because the

API gateway does not communicate with internal subsystems and components directly, but rather through a message service. Moreover, such an approach backs the extensibility and also integrability of a system, which is part of the non-functional requirements.

5.6.2 Plan Execution Viewer module

Since both Dashboard 1.0 and Dashboard 2.0 share similar functionality and are intended to be used exclusively for visualization purposes, i.e. perform read-only operations, it was decided to compose them together to implement a Plan Execution Viewer service. The main purpose of the service is to provide a graphical interface to the user that is capable of displaying the plan execution progress of a concrete work package. The implemented functionalities of the service are supposed to cover the functional requirements FR1, FR3, and FR4 (4.2)

5.6.3 Updater module

The Updater is a service that is meant to solve the tasks related to data management. The main purpose of this module is to maintain the application data up to date. The whole update process is depicted in the figure 5.3.

It consists of multiple steps and involves interaction with an external CSAT SFTP server, containing the source data, as well as with the SPipes module, which is responsible for data transformation. The Updater module is assumed to be the only part of the system that performs write operations on the application database.

5.6.4 SPipes for data transformation

This part of the system is intended to provide the functionalities described in section 3.2.3. It is presented as a separate module of the application that exposes an HTTP RESTful API for communication with other services. It is worth noting, that this module does not assume the use of a Messaging service for communication. Furthermore, the system design implies that this module will only communicate with the Updater module, therefore it can be isolated event more.

5.6.5 Messaging service

In every system consisting of multiple interconnected services, there must be some sort of messaging channel. For instance, one of the traditional approaches for communicating between services in a microservice architecture is through REST APIs. However, as the system grows in a number of services, communication becomes more complex and leads to tight coupling, which is considered an anti-pattern and slows down further development process. One possible way to overcome this design flaw is to use an asynchronous messaging platform. [63] The design of the application implies strong isolation of the system modules from each other. Communication between them is supposed to be done through the dedicated messaging platform service.

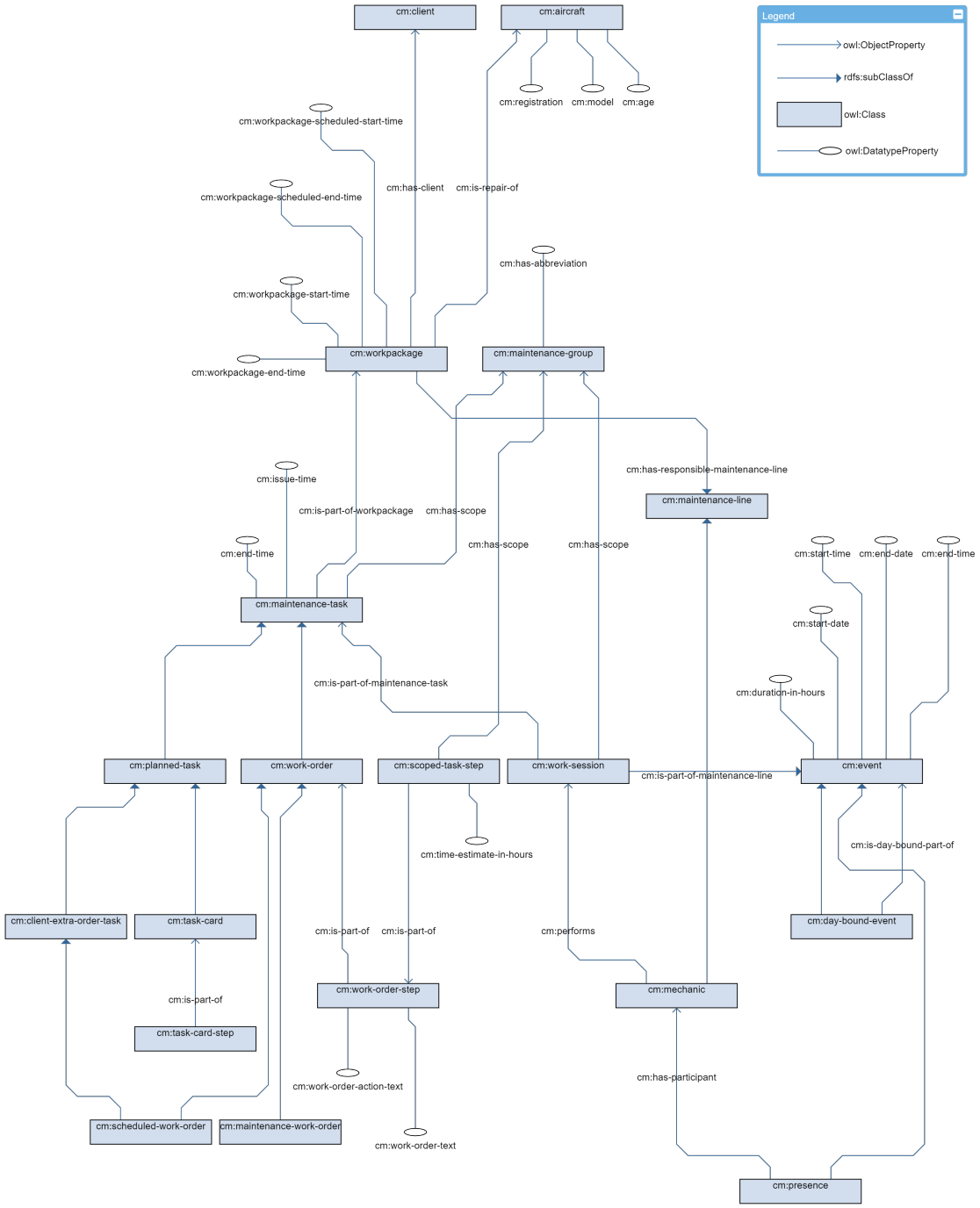


Figure 5.1: “CSAT-maintenance-spec” domain model

Figure 5.2: CSAT original dashboard

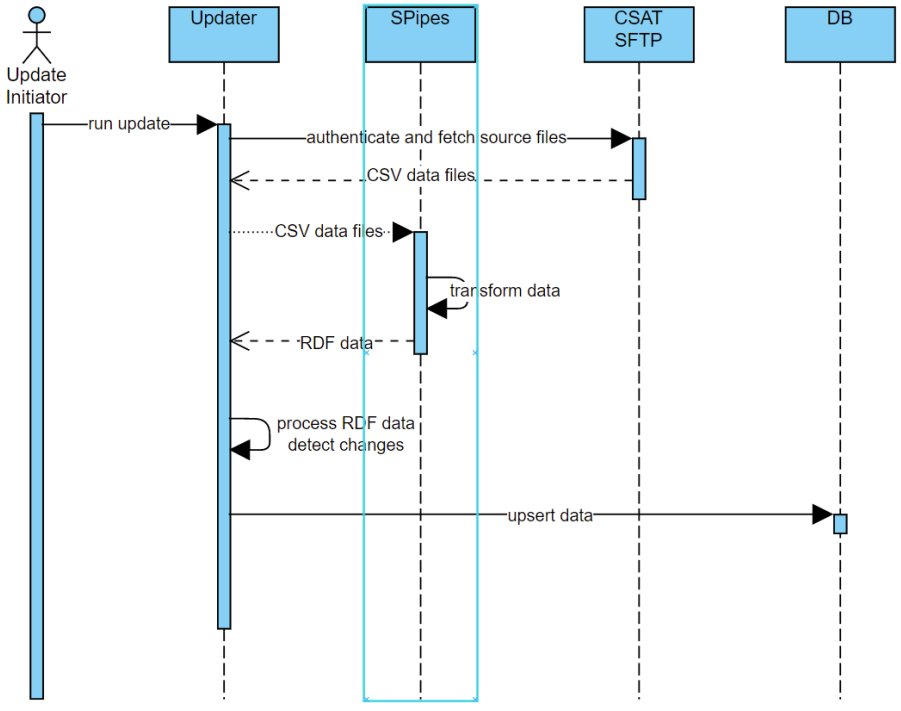


Figure 5.3: Sequence diagram: data update process

Chapter 6

Implementation

This chapter contains a brief description of the technologies and methods used in the implementation phase of the project. The first section describes the decisions made regarding the architecture of the application. The next section provides basic information about the technology stack of the application. In the third section, the most significant approaches in the development process are discussed.

6.1 Application architecture

The application architecture combines the principles of Service Oriented Architecture (SOA) and Microservices Architecture (MA). It uses the most appropriate ideas and features of both architectural styles to best fit the conditions and requirements of the project. In contrast to the original approaches, to avoid the high resource demand and additional limitations of the database transaction models caused by the Database-per-service pattern [1], it was decided to use the Shared-database-per-service pattern [2], which is considered an anti-pattern in MA. On the other hand, a lightweight Message Queue (MQ) was chosen instead of the traditional SOA heavyweight and complex Enterprise service bus (ESB). Furthermore, to avoid excessive formality and complexity of the service interface caused by SOAP and XML Schema, the more lightweight solutions (such as REST and JSON) were chosen.

Additionally, the implementation of the application has been following the methodology called Twelve-Factor App [3]. This methodology comprises the best practices for developing applications with high portability and sustainability.

6.2 Technology stack

This section provides an overview of the technology stack used for the implementation of the application. It is composed of a combination of used frameworks, libraries, programming

[1] More information about Database-per-service pattern available at <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/database-per-service.html>

[2] More information about Shared-database-per-service pattern available at <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/shared-database.html>

[3] More information about Twelve-Factor App methodology available at <https://12factor.net/>

languages, and technologies.

6.2.1 User interface

The user interface of the application is has been made using the classic WEB UI technologies:

- HTML5
- CSS3
- JavaScript (JS)

To ensure lightweightness and reduce the number of dependencies, the frontend part is done using a plain JS without any JS framework. It uses Asynchronous JavaScript and XML (AJAX) to communicate with the backend part of the system and retrieve the data. The charts of the dashboards are created using Chart.js charting library. This library has been chosen based on the analysis results presented in section 3.3.

6.2.2 Persistence layer

The persistence layer has been built on top of the JOPA library, providing Java OWL Persistence API to manage and access the database model in external triplestore storage. This layer is prepared for using any RDF4J compatible database, e.g. GraphDB. As it was described in section 5.1, the domain model is defined in and generated from an OWL ontology file.

All tabular source data consumed by application modules are exposed through an external secured FTP (STFP) server and eventually transformed to RDF-compliant format by S-Pipes service utilizing its exposed RESTful API endpoints.

6.2.3 Programming language and software framework

The Java programming language has been chosen as a programming language for implementation due to its significant advantages over other languages in terms of this project. The main advantages are that it is platform-independent, robust, object-oriented, and has a number of available libraries and frameworks, including Spring and Spring Boot.

Spring Boot is a Java framework extending a Spring platform and providing a radically faster and widely accessible getting started experience for all Spring development. Due to Its security, flexibility, and robustness, it was favored to be used as the main application framework to create Spring-powered services with absolute minimum difficulties.

6.2.4 Reverse proxy

Reverse proxies are applications that reside between a web server and the server's clients [64]. The reverse proxy can also serve as an additional layer to the web application in order to improve its security and is also considered a best practice.

NGINX web server is one of the most popular web servers [65] and therefore has been used as a reverse proxy server to route incoming requests and also to enhance the security of the application.

6.2.5 Containerization

Containerization (OS-level virtualization) improves portability and efficiently delivers higher utilization of computing resources (compared to VMs). It also allows developers to integrate with existing environments and augments security by isolating applications from the host system and from each other. Containerization in this project has been managed by the Docker engine and defined using the Docker-compose tool to ease the configuration of applications' services.

The logging layer consists of several services, represented by the EFK stack - Elasticsearch as persistent log storage, Fluentd for log message routing, and Kibana for log data accessing, filtering, and visualization of metrics.

The fundamental technology stack can be shortened in the following list:

1. Java, Spring Boot
2. JOPA, RDF4J/GraphDB
3. SFTP, CSV
4. S-Pipes, RDF
5. NGINX, HTTP
6. EFK stack: Elasticsearch, Fluentd, Kibana
7. Docker, Docker-compose
8. HTML, CSS, JavaScript + Chart.js

6.3 Development process

This section outlines the key aspects of the software development process. It describes the approaches used to facilitate and automate the integration and deployment of the application components and services in different environments.

6.3.1 CI/CD using GitHub

GitHub is a cloud-based web service that provides software developers with many features such as collaborative coding, automation, project management, team administration, bug tracking, version control, issue tracking, and many others. It is a highly useful tool that fulfills all the major requirements for software development flow including agile development, primarily consisting of:

1. features for managing and tracking the project tasks, including participant notifications
2. features for tracking and managing changes to software code and project artifacts
3. tools for DevOps-related processes, such as continuous integration and continuous deployment

The entire application code, along with the related configuration files, OWL ontologies, deployment manifest files, and scripts have been stored in the private GitHub repository, belonging to the KBSS.

Since the project time was quite limited and the thorough analysis could take the majority of available time, it was considered to follow the hybrid agile-like project management approach. This approach has been focused on sustainable development with automated testing to preserve quality and continuous feedback retrieving for more precise requirements specification. Following this strategy, the software code has been continuously built and tested by the GitHub Actions services, reporting any failures caused by code changes.

6.3.2 Environments and deployment

To facilitate the deployment process during development, the services were defined as containers. The containerization has been implemented using Docker and Docker-compose tools, which eases the automation of the deployment and management of applications in containerized environments. It allows composing an entire application with all its dependencies into an OS-level virtualized environment that can be deployed to a target system.

All environment-relevant deployment configuration files and scripts are stored in a separate directory 'deploy'. Two environments are distinguished during software development: testing environment and production environment. The particular compose configuration has been prepared for each of them:

1. *docker-compose.yaml* is made for a production environment and defines the essential services, required for a correct and stable running of the application
2. *docker-compose-dev.yaml* is intended for a testing environment and defines some extra services for debugging and tuning purposes.

Along with the continuous testing the continuous deployment was performed in a testing environment. It has been done by using GitHub Actions, Container registry, and Webhooks. The whole process is depicted in the following diagram:

Every commit to the repository triggers a predefined deployment workflow (located at *.github/workflows/docker-image.yml*). This workflow represents a set of activities divided into multiple stages. It starts with the application build to create the software binaries and supplementary artifacts to be used in the subsequent stages. In case of failure, the author of the commit gets notified (e.g. by email) and the workflow terminates. Otherwise, the second stage of the workflow takes place and builds the application Docker image (i.e. set of instructions for running docker containers) containing all relevant files required to successfully run the application. The created image is pushed to the GitHub Container Registry in the final stage of the workflow which produces a specific Packages event. Thereafter the event is consumed by the GitHub Webhook service that afterward sends the HTTP message to the Nginx web server on the target environment. This message is received by the FastCGI module for Nginx that runs the deployment Bash script (located at *deploy/nginx/cgi-bin/deploy.sh*) for pulling the updated Docker image and restarting the application container.

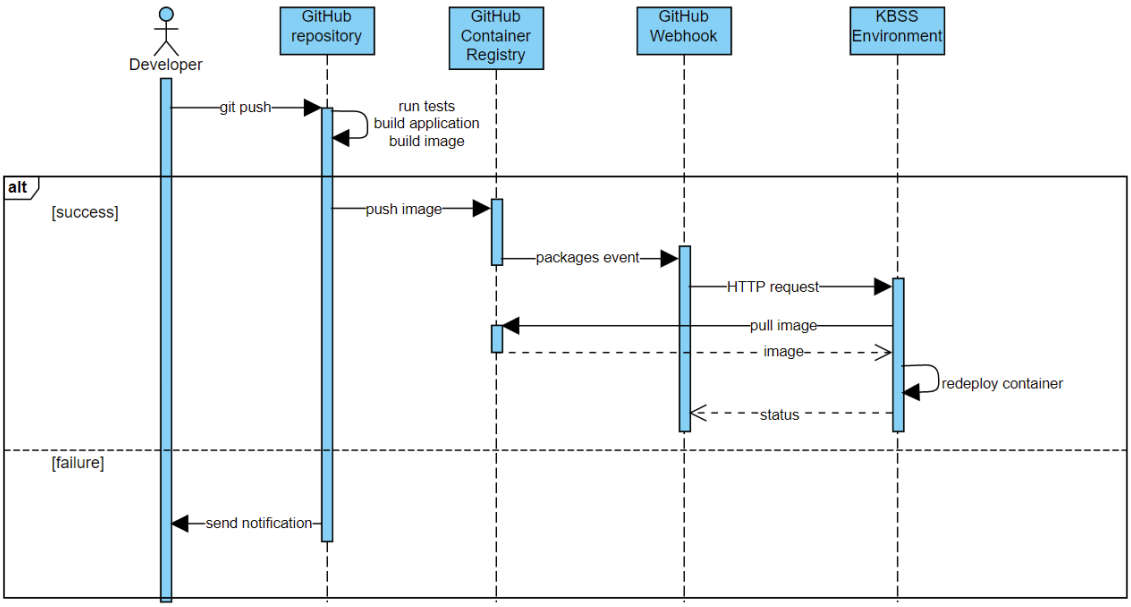


Figure 6.1: Sequence diagram: CI/CD

Chapter 7

Testing

Testing is an essential process of software development. Its main goal is to ensure that the software meets the defined project requirements. In this chapter, the main testing phases are outlined. Additionally, the process of user testing is described and further evaluation results are provided.

7.1 Unit and integration testing

Unit testing is the earliest stage of software testing. Unit tests are intended to evaluate the functionality of components of the system. Integration testing, on the other hand, is more abstract and does not require knowledge of the software implementation details. During the implementations phase, the code of the application has been continuously tested by the CI pipelines. It provided quick feedback on the quality and ability of the system to perform its functions. Since the majority of the core modules are implemented in Java using Spring Boot framework, they were tested with the related testing platforms and libraries, including JUnit Jupiter (JUnit5), Mockito, embedded Kafka, etc.

7.2 User testing

The user testing was performed by several domain experts. For this purpose, the shared online document was prepared. This document was intended to test the user experience of the graphical user interface part of the application. Two test scenarios testing different functionalities of the application have been described:

- Scenario 1 - focuses on the navigation tests i.e. how users navigate through the application
- Scenario 2 - focuses on the retrieving of information about the work package

These scenarios contained a set of tasks and steps that users had to perform. Some of the tasks required users to measure the time spent on them. In addition, each scenario contained a post-test questionnaire that was designed to elicit feedback from users and improve the evaluation of the application. The following sections contain the testing scenarios, as well as the post-test questionnaire.

7.2.1 Scenario 1: Navigation

Open the web application <https://kbss.felk.cvut.cz/csat/dashboards.html> in the web browser Find a WP with the name "PH-HZE/H-21 HMV12" in a work package list

1. Open the main dashboard of the found work package
2. Locate and open the side navigation menu
3. Navigate to the additional (secondary) dashboard with the WP details
4. Navigate back to the main dashboard of the WP using the side navigation menu
5. Using the side navigation menu, open the list of work packages with the name starting with "OH."

Post-test questionnaire

Please answer the following questions:

1. What part of the test was the most difficult and why?
2. Describe any problems you had in understanding the task.
3. Describe any problems you had in navigating through the application.
4. Do you have any suggestions for user interface improvements (e.g. rearrange the items in the navigation menu)?
5. Other comments (e.g. how did you like the user interface).

7.3 Scenario 2: Retrieving work package information

In some test steps, you will be asked to measure the time spent on the task. Please use the stopwatch on your mobile phone and note the time spent in seconds. Some of the tasks will ask you to determine certain values based on the provided description. Please note the found values along with the units and send them with the report.

Open the dashboard of the work package "PH-XXX/H-21 XXX12" in the web browser using the link Please measure the duration of the next task (3)

1. Take a look at the dashboard and try to obtain the following information:
   (a) what is the model of the maintained aircraft
   (b) what is the name of the customer
   (c) when is the inspection deadline (related to NRC progress)
   (d) how many task cards (TC) are in the WP and how many of them have been closed
2. Locate and open the side navigation menu

3. Navigate to the additional (secondary) dashboard with the WP details
4. Try to obtain the following information:
   (a) how many productive hours have been spent during the WP execution
5. Navigate back to the main dashboard of the WP using the side navigation menu
6. Please measure the duration of the next task (9)
7. Try to obtain the following information:
   (a) what is the total time estimate for the tasks from the exterior mechanic group (MECH-EXT)
   (b) how many productive hours have been spent on the fifth day of the WP execution (December 2, 2021)
   (c) what was the utilization rate on the fifth day of the WP execution (December 2, 2021)
   (d) what is the average utilization rate of the maintenance line of the WP
   (e) what is the average utilization rate of the WP

Post-test questionnaire

Please answer the following questions:

1. What part of the test was the most difficult and why?
2. How much time total (in seconds) did you spend on task 3?
3. In task 3 what metric values you did/did not manage to determine?
4. How much time total (in seconds) did you spend on task 9?
5. In task 9 what metric values you did/did not manage to determine?
6. Describe any problems you had in understanding the task.
7. Describe any problems you had in determining the values of the metrics.
8. Do you have any suggestions for user interface improvements (e.g. add tooltips to certain metric values)?
9. Other comments (e.g. how did you like the user interface).

7.4 Evaluation

Three domain experts participated in the test process. Each of them went through both prepared testing scenarios and gave thorough feedback during the answering the post-testing questions.

As a result of the testing, several shortcomings were identified, which were mainly related to user experience and user interface of the application. No critical flaws were identified. However, in spite of this, the project is still under development. There is a plenty of thing that have to be improved, including all the found flaws that are planned to be fixed in the next releases.

Chapter 8

Conclusion

The main goal of the thesis work was to design and develop a set of ontology-based services. These services had to be integrated into the AMOS software solution, which is designed to handle a large number of maintenance and customer design tasks and is used by CSAT to manage their MRO-related processes. However, it has serious limitations that can be addressed with an ontology-based approach. Developing an extension of the ontology was an important objective of the thesis.

As a result, a solution was developed and a prototype was created to automate day-to-day problem solving in the organization, as well as increased the accuracy of the overall maintenance calculations within the planning and monitoring processes. This was made possible by the ontology-based approach, which was implemented with domain experts.

Bibliography

[1] Noy, N. and Mcguinness, Deborah. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Knowledge Systems Laboratory. 32.

[2] Verhagen, Wim and Curran, R.. (2013). An ontology-based approach for aircraft maintenance task support. 20th ISPE International Conference on Concurrent Engineering, CE 2013 - Proceedings. 494-506. 10.3233/978-1-61499-302-5-494.

[3] Czech Airlines Technics. (n.d.). Czech Airlines Technics. Linkedin - Czech Airlines Technics. Retrieved June 5, 2021, from <https://cz.linkedin.com/company/czech-airlines-technics>

[4] Swiss-AS. (n.d.). AMOS | Swiss-AS. AMOS | Swiss-AS. Retrieved May 17, 2021, from <https://www.swiss-as.com/amos-mro>

[5] Swiss-AS. (n.d.). Customers | Swiss-AS. AMOS | Swiss-AS. Retrieved December 1, 2021, from <https://www.swiss-as.com/customers>

[6] Swiss-AS. (n.d.). Modules | Swiss-AS. AMOS | Swiss-AS. Retrieved June 19, 2021, from <https://www.swiss-as.com/modules>

[7] Swiss-AS. (n.d.). Material Management | Swiss-AS. Material Management | Swiss-AS. Retrieved July 22, 2021, from <https://www.swiss-as.com/amos-mro/modules/material-management>

[8] Swiss-AS. (n.d.). Engineering | Swiss-AS. Engineering | Swiss-AS. Retrieved July 20, 2021, from <https://www.swiss-as.com/amos-mro/modules/engineering>

[9] <https://www.swiss-as.com/amos-mro/modules/planning> Swiss-AS. (n.d.). Planning | Swiss-AS. Planning | Swiss-AS. Retrieved July 25, 2021, from <https://www.swiss-as.com/amos-mro/modules/planning>

[10] Swiss-AS. (n.d.). Component Maintenance | Swiss-AS. Component Maintenance | Swiss-AS. Retrieved January 4, 2021, from <https://www.swiss-as.com/amos-mro/modules/component-maintenance>

[11] Swiss-AS. (n.d.). Production | Swiss-AS. Production | Swiss-AS. Retrieved June 14, 2021, from <https://www.swiss-as.com/amos-mro/modules/production>

[12] Swiss-AS. (n.d.). Maintenance Control | Swiss-AS. Maintenance Control | Swiss-AS. Retrieved June 14, 2021, from <https://www.swiss-as.com/amos-mro/modules/maintenance-control>

[13] Swiss-AS. (n.d.). Commercial | Swiss-AS. Commercial | Swiss-AS. Retrieved December 10, 2021, from <https://www.swiss-as.com/amos-mro/modules/commercial>

[14] Swiss-AS. (n.d.). Human Resources | Swiss-AS. Human Resources | Swiss-AS. Retrieved October 22, 2021, from <https://www.swiss-as.com/amos-mro/modules/human-resources>

[15] Swiss-AS. (n.d.). Quality Assurance | Swiss-AS. Quality Assurance | Swiss-AS. Retrieved October 22, 2021, from <https://www.swiss-as.com/amos-mro/modules/quality-assurance>

[16] Swiss-AS. (n.d.). Financial Management | Swiss-AS. Financial Management | Swiss-AS. Retrieved October 1, 2021, from <https://www.swiss-as.com/amos-mro/modules/financial-management>

[17] Swiss-AS. (n.d.). Interfaces | Swiss-AS. Interfaces | Swiss-AS. Retrieved October 22, 2021, from <https://www.swiss-as.com/amos-mro/interfaces>

[18] THE INVESTOPEDIA TEAM. (2021, November 15). Web 2.0 and Web 3.0 Definitions. Investopedia. Retrieved December 10, 2021, from <https://www.investopedia.com/web-20-web-30-5208698>

[19] Narottam04, N. (2021, December 24). Web 1.0, Web 2.0 and Web3 Explained. DEV Community. Retrieved December 25, 2021, from <https://dev.to/narottam04/web-10-web-20-web-30-explained-591n>

[20] MATTR. (n.d.). Semantic Web Machine Readable Data. MATTR | Learn. Retrieved October 13, 2021, from <https://learn.mattr.global/docs/concepts/semantic-web>

[21] Admin. (2011). MyLabBook.org. MyLabBook. Retrieved October 22, 2021, from <https://www.mylabbook.org/blog-drupal-semantic-web.php>

[22] Tauberer. (2005). RDF. XMLHack. Retrieved October 1, 2021, from <http://xmlhack.ru/texts/06/rdf-quickintro/rdf-quickintro.html>

[23] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, Niklas Lindström. (2020, July 16). JSON-LD 1.1. W3C. Retrieved October 7, 2021, from <https://www.w3.org/TR/json-ld/>

[24] RDFS - W3C Wiki. (2005). W3C. Retrieved December 10, 2021, from <https://www.w3.org/wiki/RDFS>

[25] Semantic University. (2021, July 23). Learn OWL and RDFS. Cambridge Semantics. Retrieved August 14, 2021, from <https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/>

[26] OWL Working Group. (2012). OWL - Semantic Web Standards. OWL. Retrieved October 7, 2021, from <https://www.w3.org/OWL/>

[27] Cambridge Semantics. (2021, August 24). RDFS vs. Owl. Retrieved October 7, 2021, from <https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/>

[28] Cambridge Semantics. (2021a). SPARQL Query and Command Clauses. Retrieved October 22, 2021, from <https://docs.cambridgesemantics.com/anzograph/v2.4/userdoc/sparql-queries.htm>

[29] Jack Rusher, Radar Networks, J. (2001). Rhetorical Device: Triple Store. Triple Store. Retrieved October 22, 2021, from <https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html>

[30] Gilbert, E. (2016, September 29). Triplestores 101: Storing Data for Efficient Inferencing. DATAVERSITY. Retrieved June 14, 2021, from <https://www.dataversity.net/triplestores-101-storing-data-efficient-inferencing/>

[31] Douglas, S. (2021, October 21). The Different Types of Aviation Maintenance Checks. National Aviation Academy. Retrieved December 10, 2021, from <https://www.naa.edu/types-of-aviation-maintenance-checks/>

[32] Forde-Hyde, M. (2021). What are some examples of line and base maintenance work in an aircraft maintenance facility? Quora. Retrieved October 13, 2021, from <https://www.quora.com/What-are-some-examples-of-line-and-base-maintenance-work-in-an-aircraft-maintenance-facility>

[33] Department for Business, Innovation and Skills. (2016). UK Aerospace Maintenance, Repair, Overhaul and Logistics Industry Analysis. <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/502588/bis-16-132-uk-mrol-analysis.pdf>

[34] Dviation. (2018, December 19). The Difference Between Line, Base and Component Maintenance. Retrieved October 13, 2021, from <https://blog.dviation.com/2018/11/14/the-difference-between-line-base-and-component-maintenance/>

[35] What does "line maintenance" on aircraft entail, and how many hours does line maintenance take? (2019). Quora. Retrieved October 7, 2021, from <https://www.quora.com/What-does-line-maintenance-on-aircraft-entail-and-how-many-hours-does-line-maintenance-take>

[36] IBM. (2015). Work packages. IBM Documentation. Retrieved October 7, 2021, from <https://www.ibm.com/docs/en/maximo-for-aviation/7.6.1?topic=packages-work>

[37] IBM. (2016). Configuring task cards and master task cards. IBM Corporation. Retrieved December 10, 2021, from <https://www.ibm.com/docs/en/maximo-for-aviation/7.6.3?topic=management-configuring-task-cards-master-task-cards>

[38] Aungst, Josanne and Johnson, Mary and Soo, Sung and Denver, Lee and Williams, Lopp. (2009). Planning of Non-Routine Work for Aircraft Scheduled Maintenance. the Technology Interface Journal/Winter Special Issue Aungst.

[39] IBM. (2017). Creating work orders. IBM Corporation. Retrieved October 7, 2021, from <https://www.ibm.com/docs/en/maximo-for-aviation/7.6.6?topic=orders-creating-work>

[40] Cousineau, M. (2021, November 23). Mastering the fundamentals: Maintenance work orders. Fiix. Retrieved November 29, 2021, from <https://www.fiixsoftware.com/blog/work-order/>

[41] Shafranovich, Y. (2005, October). rfc4180. Ietf. Retrieved June 14, 2021, from <https://datatracker.ietf.org/doc/html/rfc4180>

[42] Techopedia. (2018, December 6). Tabular Database. Techopedia.Com. Retrieved June 14, 2021, from <https://www.techopedia.com/definition/26181/tabular-database>

[43] Jeremy Tandy, Met Office Ivan Herman, W3C Gregg Kellogg, Kellogg Associates. (2015, December 17). Generating RDF from Tabular Data on the Web. W3C. Retrieved October 13, 2021, from <https://www.w3.org/TR/csv2rdf/>

[44] Jeni Tennison, Open Data Institute Gregg Kellogg, Kellogg Associates Ivan Herman, W3C. (2015, December 17). Model for Tabular Data and Metadata on the Web. W3C. Retrieved December 10, 2021, from <https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/>

[45] Ebiquity, U. (2007). RDF123. UMBC Ebiquity. Retrieved October 7, 2021, from <https://ebiquity.umbc.edu/project/html/id/82/RDF123>

[46] Langegger, A. (2009). XLWrap – Spreadsheet-to-RDF Wrapper. Sourceforge. Retrieved October 7, 2021, from <https://xlwrap.sourceforge.io/>

[47] Tarql. (2019). Tarql: SPARQL for Tables – Tarql – SPARQL for Tables: Turn CSV into RDF using SPARQL syntax. Github. Retrieved December 10, 2021, from <https://tarql.github.io/>

[48] Hert, Matthias and Reif, Gerald and Gall, Harald. (2011). A comparison of RDB-to-RDF mapping languages. 25-32. 10.1145/2063518.2063522.

[49] Marcelo Arenas, W3C, Juan Sequeda. (2012). A Direct Mapping of Relational Data to RDF. W3C. Retrieved October 1, 2021, from <https://www.w3.org/TR/rdb-direct-mapping/>

[50] Souripriya Das, Seema Sundara, Richard Cyganiak. (2012). R2RML: RDB to RDF Mapping Language. W3C. Retrieved June 14, 2021, from <https://www.w3.org/TR/r2rml/>

[51] Highcharts. (2021). Interactive javascript charts library. Highcharts.Js. Retrieved June 14, 2021, from <https://www.highcharts.com/>

[52] Highcharts. (2021a). Advanced timeline | Highcharts.com. Highcharts.Js. Retrieved June 14, 2021, from <https://www.highcharts.com/demo/combo-timeline>

[53] Fusioncharts. (n.d.). Google Charts vs Chart.js. Fusioncharts.Com. Retrieved October 1, 2021, from <https://www.fusioncharts.com/javascript-charting-comparison/google-charts-vs-chartjs>

[54] Google. (2021). Visualization: Combo Chart | Charts. Google Developers. Retrieved October 22, 2021, from <https://developers.google.com/chart/interactive/docs/gallery/combochart>

[55] Google. (2021). Frequently Asked Questions | Charts. Google Developers. Retrieved October 22, 2021, from <https://developers.google.com/chart/interactive/faq>

[56] Bostock, M. (2021). D3.js - Data-Driven Documents. D3 - Data-Driven Documents. Retrieved October 13, 2021, from <https://d3js.org/>

[57] Bostock, M. (2020). Stacked Bar Chart, Diverging. ObservableHQ. Retrieved October 22, 2021, from <https://observablehq.com/@d3/diverging-stacked-bar-chart>

[58] Reportlinker. (2018, June 28). Top 20 Commercial Aircraft Maintenance, Repair and Overhaul (MRO) Companies 2015: Leaders in Engine, Component, HMV and Line Maintenance. PR Newswire. Retrieved August 12, 2021, from <https://www.prnewswire.com/news-releases/top-20-commercial-aircraft-maintenance-repair--overhaul-mro-companies-2015-leaders-in-engine-component-hmv--line-maintenance-300052039.html>

[59] Contributor, T. (2007, March 31). Requirements analysis (Requirements engineering). SearchSoftwareQuality. Retrieved October 1, 2021, from <https://searchsoftwarequality.techtarget.com/definition/requirements-analysis>

[60] ReQtest. (2012, April 5). Functional vs Non-Functional Requirements - Understand the Difference. Retrieved June 14, 2021, from <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

[61] Schedlbauer, M. (2011, February 23). Requirements Prioritization Strategies. Project Management Articles, Webinars, Templates and Jobs. Retrieved October 1, 2021, from <https://www.projecttimes.com/articles/requirements-prioritization-strategies/>

[62] ProductPlan. (2021, September 11). MoSCoW Prioritization. Retrieved October 22, 2021, from <https://www.productplan.com/glossary/moscow-prioritization/>

[63] Garbarino, J. (2020, January 22). Communicate Between Microservices with Apache Kafka. Okta Developer. Retrieved October 13, 2021, from <https://developer.okta.com/blog/2020/01/22/kafka-microservices>

[64] Tracy, Miles and Jansen, Wayne and Scarfone, Karen and Winograd, Theodore. (2007). NIST Special Publication 800-44 Version 2, Guidelines on Securing Public Web Servers.

[65] Mauro, T. (2021, June 10). Now the World's 1 Web Server, NGINX Looks Forward to an Even Brighter Future. NGINX. Retrieved June 14, 2021, from <https://www.nginx.com/blog/now-worlds-1-web-server-nginx-looks-forward-to-even-brighter-future>

Appendix A

Annotated table airlines.csv

id	table	position	source position	cells
B1	A	1	2	D11, D12, D13, D14
B2	A	2	3	D21, D22, D23, D24
B3	A	3	4	D31, D32, D33, D34

Table A.1: Annotated rows of the table http://example.org/airlines.csv from the example

id	table	position	source position	cells	name	title
C1	A	1	2	D12, D21, D31	code_IATA	code_IATA
C2	A	2	3	D12, D22, D32	code_ICAO	code_ICAO
C3	A	3	4	D13, D23, D33	call_sign	call_sign
C4	A	4	4	D14, D24, D34	country_name	country_name

Table A.2: Annotated columns of the table http://example.org/airlines.csv from the example

id	table	column	row	string value	value	property URL
D11	A	C1	B1	“DV”	“DV”	null
D12	A	C2	B1	“ACK”	“ACK”	null
D13	A	C3	B1	”ACK AIR”	”ACK AIR”	null
D14	A	C4	B1	”United States”	”United States”	null
D21	A	C1	B2	“OK”	“OK”	null
D22	A	C2	B2	“CSA”	“CSA”	null
D23	A	C3	B2	“CSA”	“CSA”	null
D24	A	C4	B2	”Czech Republic”	”Czech Republic”	null
D31	A	C1	B3	“EO”	“EO”	null
D32	A	C2	B3	“ALX”	“ALX”	null
D33	A	C3	B3	“ALLCONGO”	“ALLCONGO”	null
D34	A	C4	B3	”Democratic Republic of the Congo”	”Democratic Republic of the Congo”	null

Table A.3: Annotated cells of the table http://example.org/airlines.csv from the example

Appendix B

Columns of source CSV files

List of source file fields with short descriptions (the relevant ones are marked by the star symbol): **Time-analysis:**

- "bookingno-i" - unique identifier of the action within the AMOS (internal value, irrelevant)
- "Employee No" - unique identifier of employee
- "user-sign" - text value of employee made up from forename and surname
- "Start Date" - start date of the working activity
- "Start Time" - start time of the working activity
- "End Date" - end date of the working activity
- "End Time" - end time of the working activity
- "Duration Full" - overall time spent on the task in hours
- "Scope" - the work orientation of the employee, i.e. the specific aircraft maintenance activity they perform (also referred to as maintenance group)
- "Shift Group" - identifier of a group/shift, to which the worker is assigned
- "Type" - type of the task (e.g. “M” is MWO, “S” is SWO and “TC” is TC)
- "WO/TC" - identifier of the WO (for MWO and SWO tasks) or identifier of the TC (for TC tasks)
- "TC reference" - reference to the TC in which the fault was detected
- "Closing date" - closing date of the task, That is, whether the task has already been closed.
- "Description" - description of the task
- "Workpackage" - identifier of the WP

- "A/C model" - model of the aircraft, to which the task is assigned
- "A/C age" - age of the aircraft in years, to which the task is assigned
- "WP Start date - scheduled" - planned start date of the work package
- "WP End date - scheduled" - planned end date of the work package
- "WP Start date - real" - actual start date of the work package
- "WP End date - real" - actual end date of the work package
- "Line" - identifier of the responsible (main) line, to which the work package is assigned
- "Real date of booking entry" - date when the action record was created within the AMOS (internal value, irrelevant)
- "AC registration" - unique code, representing an aircraft registration number
- "operator" - code of the customer, i.e. aircraft owner

Time-estimates:

- "AC registration" - unique code, representing an aircraft registration number
- "operator" - code of the customer, i.e. aircraft owner
- "WP" - identifier of the WP
- "WO" - identifier of the WO (for MWO and SWO tasks)
- "TC" - identifier of the TC (for TC tasks)
- "sequence" - sequential number of the task (also known as workstep) within the TC or WO that has one or more worksteps
- "scope" - the work orientation of the employee, i.e. the specific aircraft maintenance activity they perform (also referred to as maintenance group)
- "est-min" - time estimate for the task in hours

Wo-tc-ref:

- "AC" - unique code, representing an aircraft registration number
- "A/C age" - age of the aircraft in years, to which the task is assigned
- "WP" - identifier of the WP
- "CSAT WO/TC" - identifier of the WO (for MWO and SWO tasks) or identifier of the TC (for TC tasks)
- "type" - type of the task (e.g. “M” is MWO, “S” is SWO and “TC” is TC)

- "state" - state of the task step (e.g. “O” for open and “C” for close)
- "ATA" - ATA 100, reference to the ATA numbering system which is a common referencing standard for commercial aircraft documentation
- "Customer ref" - reference to the identifier of the work according to the customer (it may vary according to the aircraft operator, therefore it is not unified for all)
- "TC reference" - identifier of the TC (for TC tasks)
- "issue date" - date when the task was issued
- "closing date" - date when the task was closed
- "sequence" - order number of the task step
- "WO text" - description of task
- "WO action" - description of the aircraft maintenance activity performed

Wp-catalog:

- "AC owner" - code of the customer, i.e. aircraft owner
- "AC registration" - unique code, representing an aircraft registration number
- "Workpackage" - identifier of the WP
- "Start date - scheduled" - planned start date of the work package
- "End date - scheduled" - planned end date of the work package
- "TAT scheduled [days]" - scheduled turnaround time (TAT) of the work package, i.e. duration of inspection in days
- "Start date - real" - actual start date of the work package
- "End date - real" - actual end date of the work package
- "TAT real [days]" - actual TAT of the work package, i.e. duration of inspection in days
- "TAT difference [days]" - difference between scheduled TAT and actual TAT, i.e. delay in days
- "Delay [hours]" - difference between scheduled TAT and actual TAT, i.e. delay in hours
- "Delay reasons" - reasons that caused delay
- "Line" - identifier of the responsible (main) line, to which the work package is assigned
- "Delay reason" - incorrect unidentified column, irrelevant and should be ignored
- "WP status" - status of the work package (open/close)

- "MHrs in FIX" - planned amount of man-hours to be spent on a work package (within agreed fixed price, based on time estimation for inspections)
- "MHrs above FIX" - planned amount of man-hours to be spent on a work package (above agreed fixed price, based on time estimation for unplanned work orders)
- "Routine (customized included) work" - estimated man-hours for routine tasks
- "Findings Up To Limit" - Faults, estimated hours spent on faults that the customer has already paid for
- "Additional Work, Mods, Sbs, Ads Etc." - hours spent on the additional work, service bulletins and modifications
- "Findings Over Limit - Estimation" - hours estimated for troubleshooting beyond the fixed price
- "Customer Requests" - hours spent on resolving customer requirements or extra work done after the contract is signed
- "Findings MECH-EXT" - estimated hours for troubleshooting within the work package for mechanics in “MECH-EXT” maintenance group
- "Findings MECH-INT" - estimated hours for troubleshooting within the work package for mechanics in “MECH-INT” maintenance group
- "Findings MECH-ENG" - estimated hours for troubleshooting within the work package for mechanics in “MECH-ENG” maintenance group
- "Findings MECH-LDG" - estimated hours for troubleshooting within the work package for mechanics in “MECH-LDG” maintenance group
- "Findings AVIO" - estimated hours for troubleshooting within the work package for mechanics in “AVIO” maintenance group
- "Findings SHM" - estimated hours for troubleshooting within the work package for mechanics in “SHM” maintenance group

Presence:

- "Employee No" - unique identifier of employee
- "status-1" - status of the employee (e.g. “A” for “available” and “N” for “not available”).
- "status-2" - status of the employee (e.g. “1” for “available” and “2” for “not available”).
- "date" - date of the status change (i.e. employee arrival or departure)
- "time" - time of the status change (i.e. employee arrival or departure)

Appendix C

The Contents of the Enclosed CD

- Readme.md - installation guide
- application.zip - archive with an application
- AG.pdf - thesis