F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Formal Representation of Quality Inspection Process

Petr Jůza
Open Informatics, Artificial Intelligence and Computer Science

January 2023
Supervisor: Ing. Martin Macaš, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Jůza Petr** Personal ID number: **492172**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Formal Representation of Quality Inspection Process

Bachelor’s thesis title in Czech:

Formální reprezentace procesu kontroly kvality

Guidelines:

The main objective is to formally represent a quality control process and show how the representation can be utilized. The use-case is quality control of a machining process. A natural part of the thesis is interaction with Škoda Auto quality department, which will provide necessary knowledge to be formally represented.
1. Conduct a survey of approaches to formal representation of production processes and quality control processes.
2. Study and analyze machining use-case provided by thesis supervisor and take into account the purposes of formal representation (e.g. reduction of quality costs, numerical simulation of production and inspection processes, or diagnostics).
3. Propose a suitable formal representation of quality control for the given use-case.
4. Implement the proposed formal representation of the selected production process quality control.
5. Demonstrate how to use the formal representation.

Bibliography / sources:

[1] Chen, Xiaojun, Shengbin Jia, and Yang Xiang. "A review: Knowledge reasoning over knowledge graph." Expert Systems with Applications 141 (2020): 112948.
[2] Montgomery, Douglas C. Introduction to statistical quality control. John Wiley & Sons, 2020.
[3] Bonatti, Piero Andrea, et al. "Knowledge graphs: New directions for knowledge representation on the semantic web (dagstuhl seminar 18371)." Dagstuhl reports. Vol. 8. No. 9. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Name and workplace of bachelor’s thesis supervisor:

Ing. Martin Macaš, Ph.D. Cognitive Neurosciences CIIRC

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **26.08.2022** Deadline for bachelor thesis submission: **10.01.2023**

Assignment valid until: **19.02.2024**

Ing. Martin Macaš, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgement / Declaration

I would like to thank my supervisor Ing. Martin Macaš, Ph.D. for his support and guidance. Also, I would like to thank Ing. Václav Jirkovský, Ph.D. for his advice in the field of semantic web technologies and Ing. Ondřej Košťák from Škoda Auto a.s. for all consultations.

Last but not least, I would like to thank my family and friends for their support during my studies.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, January 10, 2023

.....

Abstrakt / Abstract

Inspekce kvality je nedílnou součástí výrobního procesu. Průmysl 4.0 přináší nové výzvy v oblasti monitoringu kvality a jejího zlepšování. Vhodná formální reprezentace výrobních znalostí umožňuje zlepšení inspekce kvality, například redukce nákladů na kvalitu nebo diagnostika.

Představujeme ontologii pro reprezentaci výrobního a měřícího procesu. Jako modelový příklad jsme vytvořili znalostní bázi reprezentující znalosti týkající se výroby a měření kvality motorových hlav ve Škoda Auto. Implementovali jsme rozhraní pro přístup do naší báze znalostí, která mohou být využita mnoha aplikacemi cílenými na inspekci kvality a její kontrolu.

Klíčová slova: Reprezentace znalostí, Ontologie, OWL, Výroba

Překlad titulu: Formální reprezentace procesu kontroly kvality

Quality inspection is a necessary part of the manufacturing process. Industry 4.0 brings new challenges when it comes to quality monitoring and quality improvement. Having a proper formal representation of manufacturing knowledge enables improvements in quality inspection, such as reduction of quality costs or diagnostics.

We present an ontology for the representation of manufacturing and measuring processes. As a use case, we developed a knowledge base representing knowledge about manufacturing and measuring processes of engine head manufacturing in the Škoda Auto. We implemented an interface for accessing our knowledge base that can be utilized by various applications regarding quality inspection and control.

Keywords: Knowledge representation, Ontology, OWL, Manufacturing

Contents /

1 Introduction **1**
1.1 Goals 1
1.2 Motivation 1
1.3 Thesis structure 2

2 Quality inspection use case **3**
2.1 Quality control loop 3
2.1.1 Process 4
2.1.2 Measurement 4
2.1.3 Out of control state detection 4
2.1.4 Assignable cause finding . . . 4
2.1.5 Selection of control action . . 4
2.2 Engine head manufacturing and inspection 5
2.2.1 Machining Part 5
2.2.2 Measuring Part 6

3 Knowledge representation and reasoning **8**
3.1 Knowledge representation 8
3.1.1 Semantic networks 8
3.1.2 Rules 9
3.1.3 Logic 9
3.2 Reasoning 10
3.3 Description logics 11
3.3.1 TBox 11
3.3.2 ABox 11
3.4 Ontology 12
3.5 Knowledge representation for manufacturing domain . . . 13

4 Proposal of ontology **16**
4.1 Modeling the ontology 16
4.1.1 First stage 17
4.1.2 Second stage 17
4.1.3 Third stage 18

5 Implementation **20**
5.1 Semantic web 20
5.2 Semantic web stack 20
5.2.1 RDF 20
5.2.2 RDFS 21
5.2.3 OWL 21
5.2.4 SPARQL 21
5.3 Ontology APIs 22
5.3.1 OWL API 22
5.3.2 Owlready2 23
5.4 Architecture of the application 23
5.4.1 Defining the ontology . . . 23
5.4.2 Defining individuals 25

6 Usage and results **27**
6.1 Reduction of quality costs . . . 27
6.2 Numerical simulation 28
6.3 Diagnostics 30

7 Conclusion **32**
7.1 Discussion 32
7.2 Future work 33

References **34**

A Attached files **37**

Tables / Figures

4.1 Enumeration of all important terms 16

2.1 Quality control loop 3
2.2 3D model of an engine head 5
3.1 Example of semantic network ... 9
3.2 Knowledge representation system based on Description Logics 11
3.3 Example of geographical ontology 12
3.4 Product, process, and resource design 13
4.1 Core part of machining process 17
4.2 Relation between characteristic, element, reference, and object 17
4.3 Construction and measure settings with a measuring machine 18
4.4 Parameters of measure and construction settings 18
4.5 Tool, nest, and machine tool .. 18
4.6 Sensor and sensor carrier 18
4.7 Overview of the ontology 19
5.1 Architecture of semantic web .. 21
5.2 Class diagram 23
5.3 Class diagram 24
5.4 Simplified structure of the table 25
6.1 Penalization of related individuals 30

Chapter 1
Introduction

The expansion of the fourth industrial revolution in recent years brings new challenges regarding artificial intelligence and automation. Besides the fields of the internet of things, digital twins, augmented reality or blockchains, a proper representation of domain knowledge is required to enable interconnected data in manufacturing and quality inspection.

This thesis focuses on the formal representation of quality inspection system. It is a partial output of the National Competence Center — Cybernetics and Artificial Intelligence[1] (NCK KUI) project aiming at research in the field of Industry 4.0, smart cities, intelligent transport systems and cybersecurity. NCK KUI is supported by the Technology Agency of the Czech Republic.

1.1 Goals

The goal of the thesis is to study formal representation approaches and select the best approach. The main goal is to design and implement a formal representation of the quality control process. The designed representation will be tested and utilized by the machining use case provided by Škoda Auto. The necessary part of the thesis is to study the machining domain.

1.2 Motivation

The formal representation of process structure is a key method for representing and working with process knowledge not only in automotive but everywhere in Industry 4.0. Semantic web principles and technologies can serve as semantic interoperability providers [1–2].

The general concept that this work is part of is creating a "smart measure plan" for manufacturing, i.e., exploiting artificial intelligence in quality inspection in terms of measuring characteristics as the key quality index.

The main motivation behind building a formal representation of the process and measurement is its possible utilization indeed. Moreover, if we know the usage it is easier to design and test the formal representation. The utilization goes hand in hand with cooperation with Škoda Auto — the use case provider, and Diribet — the supplier of software for data analysis for Škoda Auto. A wide range of applications unveils, for illustration, a comparison of different processes, general interoperability among processes, or selection of control action. We present three significant proposals of utilization that are subjects of further study.

[1] https://starfos.tacr.cz/en/project/TN01000024

1.3 Thesis structure

The thesis is divided into seven chapters. The first, current, chapter is a brief introduction to the goals and motivation of this work. Chapter 2 presents the quality control loop and describes the use-case domain. Chapter 3 is a brief introduction to knowledge representation formalisms. A survey of formal representations in the manufacturing domain is made. Chapter 4 proposes an ontology representing the provided use case. Chapter 5 presents technologies used for the proposed ontology and describes the implementation of the ontology together with populating a knowledge base. Chapter 6 demonstrates how to access the knowledge base knowledge. The last, chapter 7 concludes the thesis and suggests future work.

Chapter 2
Quality inspection use case

In this chapter, we analyze the provided use case, describe the given domain and present possible utilization. Section 2.1 presents a quality control loop which is a significant part of quality management, describes its steps and briefly suggests possible utilization. Section 2.2 presents the provided use case focusing on both the manufacturing and quality inspection parts in detail.

2.1 Quality control loop

Before we describe the quality control loop, a quality inspection needs to be introduced. Quality inspection can be characterized as a process of reviewing final product characteristics and checking for conformance to required standards. In contrast to quality control, quality control employs results obtained by quality inspection to find the cause of problems.

The quality control loop (Fig. 2.1) captures the manufacturing process as a set of inputs transformed into a set of outputs, along with quality inspection (measurement) and process control. Quality inspection is carried out according to the sampling plan. The sampling plan defines which and when characteristics should be acquired. The resulting characteristics are then subjected to out-of-control state detection. The positive detection triggers an assignable cause finding procedure (diagnostics) and selection of an appropriate control action. The knowledge representation (KR) described in this work directly covers the process (machining) and measure (quality inspection) activity of a particular engine head type. The following subsections analyze each step of the control loop [3].

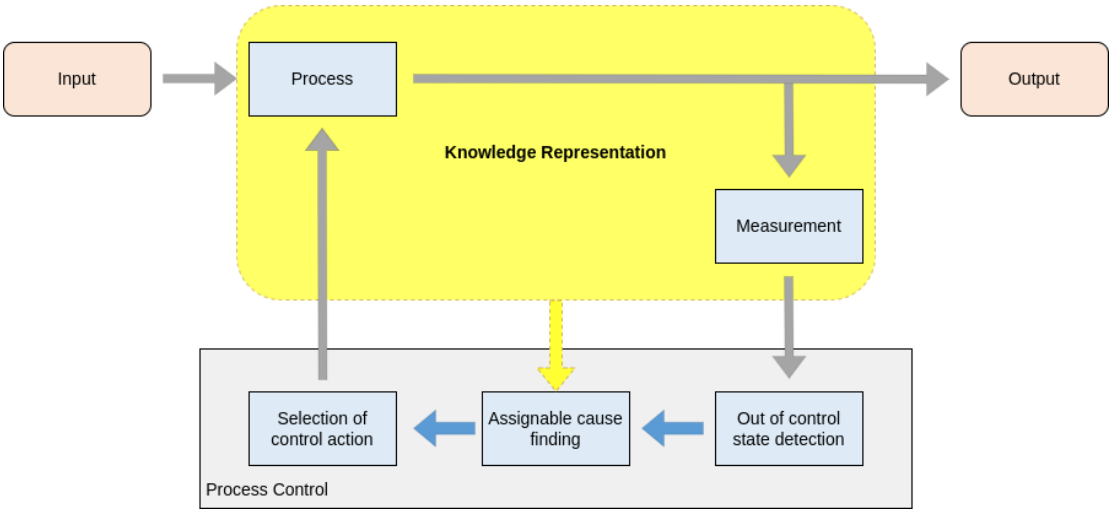

Figure 2.1. Quality control loop with the intended application of KR, adopted from [3], modified

2.1.1 Process

Process, in the context of Škoda Auto use case, is machining, where a raw material is cut into the desired shape, i.e., a machine tool processes a raw piece of engine head by creating new objects on the engine head. The formal representation covers not only individual steps of machining but also objects that are the results of machining activities, tools, null points settings, machine tools, etc. Subsection 2.2.1 presents the process representation in detail.

2.1.2 Measurement

In our use case, measurement is a quality inspection activity that collects data from a machined product in the form of characteristics. The measurement itself is performed via measuring systems — coordinate measuring machines (CMM) or handheld metrology tools. The system can be based on computer vision techniques in other applications. According to the measurement strategy [4], a CMM scans the engine head, constructs abstract elements representing machined objects and computes the final characteristic through elements. E.g., the CMM scans a circular path inside a machined hole then the CMM constructs a circle from the scanned data and computes the diameter of the circle. The diameter represents the diameter of the hole. Another example is two surfaces that make an angle. The CMM scans the surfaces, constructs two planes, and computes an angle between them. Subsection 2.2.2 focuses on the measuring section of the use case.

2.1.3 Out of control state detection

First, we clarify an out-of-control process concept. Montgomery [5] defines it as follows: "A process that is operating in the presence of assignable causes is said to be an out-of-control (OOC) process." Although this could lead to out-of-control state detection by finding the assignable cause, this is not our case. Montgomery describes tools for OOC state detection involving measured characteristics as mentioned in the previous subsection 2.1.2. Two sufficient methods are presented: the Shewhart control chart and its multivariate extension - the Hotelling T^2 control chart.

A paper [3] applies One-Class Support Vector Machine (OSVM) [6] as a supervised machine learning-based method for OOC state detection. A comparison of the Shewhart, Hotelling T^2, and OSVM was made and OSVM outperformed the other methods.

2.1.4 Assignable cause finding

Positive detection of the OOC state launches an assignable cause-finding procedure [3]. An objective is to find an underlying cause that raised an alarm. Currently, it is up to a machine tool operator to analyze a product and carry out an assignable cause. For automation of this procedure, a proper formal representation has to be available to allow an automated potential cause finding. Further thoughts regarding this task are presented in subsection 6.3.

2.1.5 Selection of control action

Selection of control action is a consecutive procedure that follows after the assignable cause is found. Currently, a machine tool operator performs a corrective action manually, e.g., a visual inspection of a product.

2.2 Engine head manufacturing and inspection

The following subsections describe machining and measuring parts of the production process as applied in the provided Škoda Auto use case. Although the production process is quite complex for a noninvolved person, we will try to describe it as close as it happens in reality, although with some simplifications. There were several consultation meetings between colleagues from the Škoda Auto Measurement Laboratories and us to profoundly understand the manufacturing and measuring domain.

Every concept presented below might be a potential cause of out of control state detection, therefore it is essential to have a formal representation of both machining and measuring processes to detect a possible cause of defect, to have the ability to perform a process simulation, or to minimize the costs of measurement.

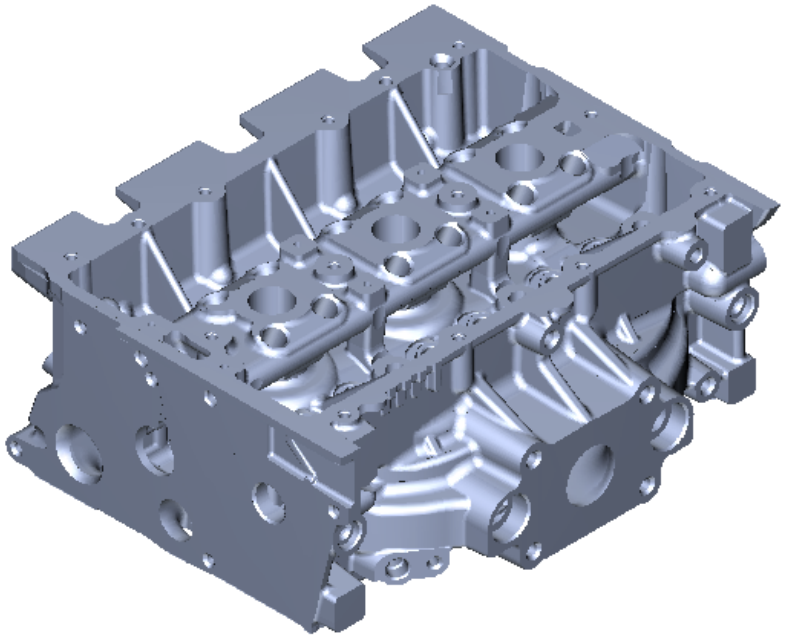

Figure 2.2. 3D model of an engine head with manufactured holes and surfaces

2.2.1 Machining Part

At this time, the process handles the manufacturing of eleven engine head types. Fig. 2.2 shows one of the types of engine heads. A machine tool manufactures engine head type according to a program. It follows that there are eleven programs.

A program is a setting of NPVs (NPV is described in the paragraph below) to a tool. It is defined by a machine tool operator and executed by a machine tool. The actual values of NPVs are determined by a ZOS (ZOS is described in the paragraph below) that is used by the program. The program has set exactly one ZOS.

The provided use case focuses on one specific operation from a sequence of operations, specifically operation 30. Previous operations may have prepared the manufactured product in order to allow processing defined by the current operation. Following operations may need the current operation in favor to complete the product manufacturing.

The manufacturing of the whole engine head is a pipeline of production operations and the quality control loop depicted in Fig. 2.1) covers the control of one specific operation.

Null Point Shift (abbreviated NPV, from german Nullpunktverschiebung) can be seen, for simplification, as a coordinate in a rectangular cuboid defined by a machine tool. In fact, it is a so-called g-function (from a g-code[1]) that sets a default position of the NPV. Particular coordinates are bound by ZOS. coordinates. A machine tool operator sets (via a program) one or more NPVs to any number of tools equipped on a machine. It is necessary to move together all tools that are linked with the same NPV. This situation typically occurs whenever an object is drilled at first, then milled and finally ground; i.e. three different tools operate on this procedure.

A mapping table (abbreviated ZOS, from german Zuordnungstabelle) is a setting of NPVs in a machine tool. Operation 30 makes use of four ZOS tables, each of ZOS used by one or more programs.

A machine tool is a machine that transforms an input product into an output froduct according to a program. Currently, five machine tools operate in a manufacturing process. Each of the machine tools has two fixtures and spindles, we call a pair of fixture and spindle a nest. Thus, a machine tool has two nests. Each nest has a worktable and a set of tools to manufacture the input. One machine tool handles the manufacturing of two engine heads and a whole workshop of ten engine heads at a time.

Tools are mounted to a machine tool nest and are necessary to complete the current operation. There are several tool types. Tool quality is a key parameter to delivering optimal quality of a manufactured product. Every tool gets worn over time and is extremely important to monitor the tool's state and replace it if needed.

Objects are entities that are created by a tool. Two types of objects are being created — holes and surfaces (as visible in Fig. 2.2). An object is usually machined by multiple tools. The quality of an object is measured in form of characteristics related to a given object.

2.2.2 Measuring Part

A measuring of characteristics of manufactured engine head is accomplished by either one of the commercial coordinate measure machines Zeiss Prismo[2] or Zeiss DuraMax[3], or by handheld metrology tools.

Elements are abstract geometric entities that are computed by a coordinate measuring machine (CMM) from the measured data. Those entities are cones, cylinders, planes, circles, points, etc. The CMM measures an element with a certain sensor, according to element type and its position. The CMM measures elements according to a measurement plan, the element is measured only if it is needed for a characteristics computation, i.e. no elements are measured twice. A measurement plan defines a measurement strategy — parameters set by a CMM operator for optimal element measurement. We call these parameters as measurement and construction settings. Last but not least, some elements are essential to derive other elements (e.g., a point derived from a plane), and some elements compose a reference (e.g., a reference plane composed of several planes).

Measurement settings are sets of parameters determining the measurement strategy. It represents parameters such as measurement method (scanning or tactile), path type

[1] https://en.wikipedia.org/wiki/G-code

[2] https://www.zeiss.com/metrology/products/systems/coordinate-measuring-machines/bridge-type-cmms/prismo.html

[3] https://www.zeiss.com/metrology/products/systems/coordinate-measuring-machines/production-cmms/duramax.html

(polyline, circular path, single points), speed and step (for polylines and circular paths), desired point count, actual point count, and angle range (for circular paths). We can easily compute a duration of measurement for an element component by multiplying the actual point count by the step, divided by the speed.

Construction settings are sets of parameters determining filtering and construction parameters for the measured data. It represents parameters such as filtering method, filter type, filter kind, additional filter parameters, elimination of outliers condition, and construction method.

Two types of sensor carriers can be mounted on the CMM. Type K130 has five sensors, named by numbers from one to five. Type K60 has two sensors, named two and four. Although K130 and K60 have sensors two and four labeled identically, they are different physical objects.

References are of two subtypes, either a coordinate system or a group of elements (even a single-element group). In most cases, the CMM computes characteristics by referencing the standard coordinate system, in some cases, there can be a coordinate system defined by one or two objects. An example of a reference consisting of elements is a plane, constructed by other planes.

Characteristics are the key factors of quality. Characteristics are computed by accessing elements or by referring to a reference. A computation is up to the CMM and can be influenced by adjusting a measurement strategy. Characteristics such as diameter, position, angle, distance, concentricity, flatness, parallelism, or tilt are then subjected to an out-of-control detector.

Chapter 3
Knowledge representation and reasoning

Knowledge representation and reasoning is a sub-field of Artificial Intelligence that deals with representation and reasoning of the real world in a machine-interpretable way. Knowledge-based systems keep an abstraction of the real-world domain of interest in the form of a computational model. Real entities, such as physical objects, events, relationships, etc., are substituted for symbols and can cover any part of the real world or any hypothetical system. Besides this concept, known as knowledge base, a knowledge-based system allows inferring new knowledge via reasoning [7].

In the following sections, we describe knowledge representation (section 3.1) and reasoning (section 3.2), description logics as underlying theories of ontologies (section 3.3), ontology (section 3.4) as our selected formal representation. In section 3.5, we conducted a survey of formal representations in the manufacturing domain.

3.1 Knowledge representation

A survey [8] collects statistics about papers citing knowledge representation (KR) in machining process planning. The survey includes predicate logic-base, rule-based, semantic network-based, frame-based, script-based, Petri-net-based, object-oriented-based, ontology-based, and neural network-based formalisms. The ontological approach appears as the most cited formalism with the ability to share and reuse, a wide range of applications, and suitability for both easy and complex systems as advantages. However, ontologies are not suitable for their encoding complexity.

Due to the recent popularity of ontologies, we decided to use them as our formal knowledge representation. Section 3.4 presents them in detail. Other formal representations related to ontologies are presented in the following subsections.

3.1.1 Semantic networks

Semantic networks can generally be viewed as some form of a graph where nodes represent concepts and edges represent relations between these concepts. An example of semantic network is in Fig. 3.1. The binary relation employee that is employed at a company represents two concepts, i.e., *Employee* and *Company* (denoted by a rectangle), participating in a relation *isEmployedAt* (denoted by an arrow). The general concept *Employee* participates also in a relation *isA MisterX*, signifying that *MisterX* is an "instance" of *Employee*. Other important relations are e.g. *kindOf* denoting subsumption, or *partOf* denoting part-whole relationship [7].

The semantic network approach is beneficial for categorizing domain concepts into taxonomies and capturing general statements about the domain. However, capturing individuals and data values (numbers, strings) is problematic in semantic networks [7].

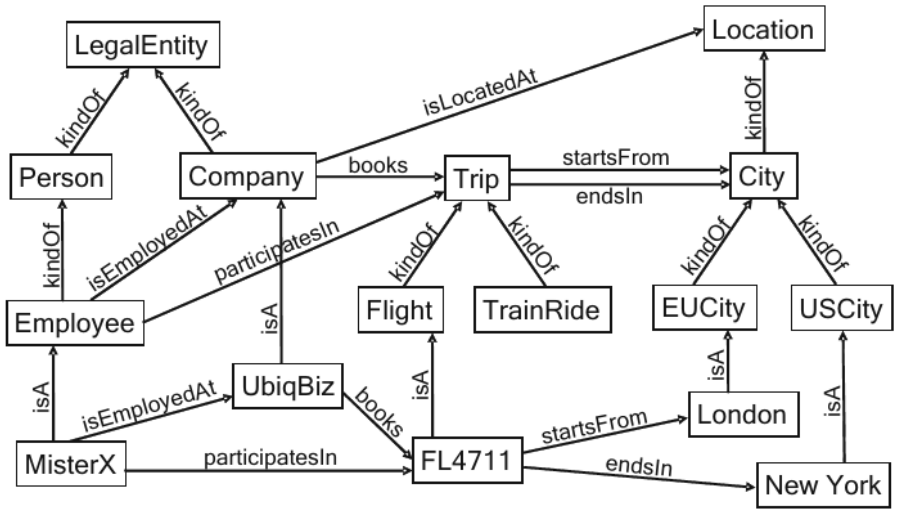

Figure 3.1. Semantic network capturing B2B travelling use-case [7]

3.1.2 Rules

Another knowledge representation formalism known from logic programming[1] are rules forming if-then statements. An example of such rules illustrating the semantic network in Fig. 3.1 is expressed as e.g. if some city is European then it is a city, or if a company books a trip that starts and ends in the same city then the trip is by train. Such rules are formalized in a machine-readable language written in the example 3.1 [7].

Example 3.1.

```
1. City(?c) :- EUCity(?c)
2. TrainRide(?t) :- Company(?x) ∧ books(?x, ?t) ∧ Trip(?t)
   ∧ starts(?t, ?c) ∧ ends(?t, ?c) ∧ City(?c)
```

The rule consists of two parts separated by the ":-" symbol. The first part is called the head (the consequence in the if-then notation) and the second part is called the body (the condition in the if-then notation). Note that the machine-readable formalism has an inverse notation in opposite to the standard if-then [7].

Rules are suitable forms for reasoning about individuals. They are not suitable for more complicated consequences [7].

3.1.3 Logic

Since previous formalisms do not allow a representation of some facts, logic formalism, specifically *first-order predicate calculus* needs to be introduced. First-order logic (FOL) comes with predicates, functions, variables, and logical connectives, all of them constructing formulas around objects [7]. For illustration, some relations from the semantic network depicted in Fig. 3.1 are listed in examples 3.2, 3.3, and 3.4.

Example 3.2. *kindOf* is an implication in FOL:

```
∀x: (EUCity(x) → City(x))
```

Example 3.3. The first formula states that *startsFrom* is a relation between city and trip respectively, while the second formula states that every trip must have a starting city:

[1] https://en.wikipedia.org/wiki/Logic_programming

```
1. ∀x, y: (startsFrom(x, y) → Trip(x) ∧ City(y))
2. ∀x: ∃y: (Trip(x) → City(y) ∧ startsFrom(x, y))
```

Example 3.4. The second logic rule from example 3.1 rewritten to FOL:

```
∀x, t, c: (Company(x) ∧ books(x, t) ∧ starts(t, c)
          ∧ ends(t, c) ∧ City(c) → TrainRide(t))
```

Semantic networks, rules, and logics play a huge role in many knowledge representation languages described in section 5.2. In the next section, we will briefly introduce an application of FOL in the reasoning about knowledge.

3.2 Reasoning

The second major concept in knowledge representation is reasoning. To draw a parallel, computer reasoning is a simulation of human thinking. Like human reasoning, a computer deduces new conclusions from the existing knowledge base (KB)[9].

FOL enables asserting and inferring knowledge to/from the knowledge base using *tell* and *ask* operations. The tell operation adds a new sentence to the knowledge base by explicitly stating a new fact about the domain [10]. To illustrate that, let's use the semantic network (3.1) again. Example 3.5 asserts that *MisterX* is an *Employee* employed at *UbiqBiz*, and *Person* differs from *Company*.

Example 3.5.

```
Tell(KB, Employee(MisterX))
Tell(KB, ∀x, y: (isEmployedAt(x, y) → Employee(x) ∧ Company(y)))
Tell(KB, isEmployedAt(MisterX, UbiqBiz))
Tell(KB, ∀x: (Person(x) → ¬Company(x)))
```

Now, we can ask the knowledge base whether *UbiqBiz* is a *Company* by query in example 3.6. The answer should be true, as expected.

Example 3.6.

```
Ask(KB, Company(UbiqBiz))
```

An essential property of a knowledge base is satisfiability, i.e., the absence of contradictory statements [7]. Example 3.7 shows adding an inconsistent fact that *UbiqBiz* is a *Person*. Since the KB is now inconsistent, we should avoid these situations.

Example 3.7.

```
Tell(KB, Person(UbiqBiz))
```

The tell operation adds explicit knowledge, whereas a knowledge base derives implicit knowledge from explicitly told statements. Statements are called entailed if they are logical consequences of knowledge stored in the KB. An inference process that operates over the KB should have two properties — soundness and completeness. Soundness is ensured by deriving only entailed statements, while completeness is guaranteed by deriving any entailed statements [7].

For now, first-order logic was the most powerful tool of the three presented however its deduction system is semi-decidable meaning that the positive result of every entailed statement terminates in a finite time, but the negative result of the non-entailed statement may not be acquired in a finite time [10]. Nevertheless, there exist some logic theories capable of decidability. The next section deals with such theories.

3.3 Description logics

Description logics (DLs) are the following knowledge representation formalisms. Compared to previous formalisms, DLs are capable of representing a specific domain of interest with formal knowledge-based semantics focused on reasoning with full decidability, although, some fragments of DLs are undecidable. The family of DLs includes many variants, e.g. Attributive langue ($\mathcal{AL}$), Frame-based description language ($\mathcal{FL}$), or Existential language ($\mathcal{EL}$). Similar to Semantic networks as discussed in subsection 3.1.1, DLs capture the field of interest by distinguishing between concepts and individuals. DLs split the representation into so-called TBox (terminology) and ABox (assertion) as in Fig. 3.2. We describe these components below [11].

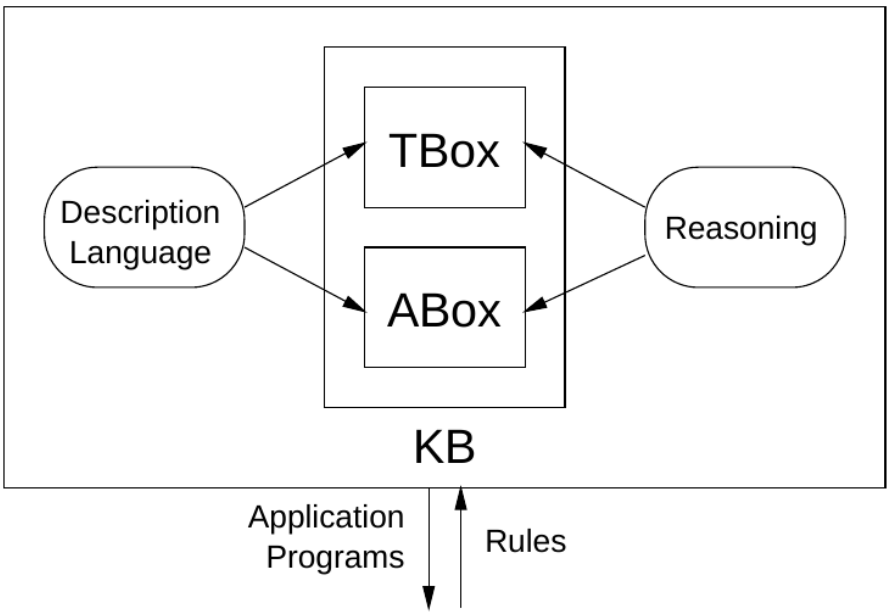

Figure 3.2. Knowledge representation system based on Description Logics [11]

3.3.1 TBox

The TBox defines a vocabulary of a selected domain by constituting concepts and roles between them, even complex. Terminological axioms are statements relating concepts and roles together. The basic and most important terminological axioms are inclusions and equalities. Equalities can further extend the logic by making definitions [11]. Let's show definitions and inclusions of some relation regarding the Semantic network example 3.1 again.

Example 3.8. Definition of *Employee*: *Person* that works for at least one *Company*.

$$\texttt{Employee} \equiv \texttt{Person} \sqcap \geq 1\ \texttt{isEmployedAt.Company}$$

3.3.2 ABox

The ABox is a part of the knowledge base describing individuals and their properties. One can assert any individual the concept or the role, the ABox is then a finite set of such assertions. Individuals may also appear in the TBox in a set constructor as nominals, i.e., classes of specific elements [11].

3.4 Ontology

Ontology, in a philosophical meaning, is a branch studying existence, while in the field of the Semantic web, Tom Gruber [12] defines the ontology as follows:

Definition 3.1. An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest.

Analyzing the definition, formal means machine-readable and explicit means explicitly defined for a machine. Further, conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose, being shared spreads the conceptualization among users in a community, and finally, domain of interest is that part of the world we want to cover.

The building blocks of ontology are concepts, relations, and instances, all of them having equivalents in semantic networks, first-order logic, and description logics in the form of nodes, edges, predicates, roles, etc. Ontology itself is then a set of such axioms in selected ontology language. The main power of ontologies is their ability to machine interpretation and usage for the reasoning process. Ontology usually describes the domain on the schema level. However, it often includes instance-level axioms, e.g., the set constructor as we mentioned in subsection 3.3.2. Ontology together with a set of individuals composes the knowledge base. One can find various deployment levels, namely knowledge connectivity, knowledge abstraction, or automation in a knowledge processing level. Ontologies have a wide range of usage in the Artificial Intelligence field, namely information integration and retrieval, semantically enhanced content management, knowledge management and community portals, or expert systems [7]. An example of a geographic ontology is graphically represented in Fig. 3.3.

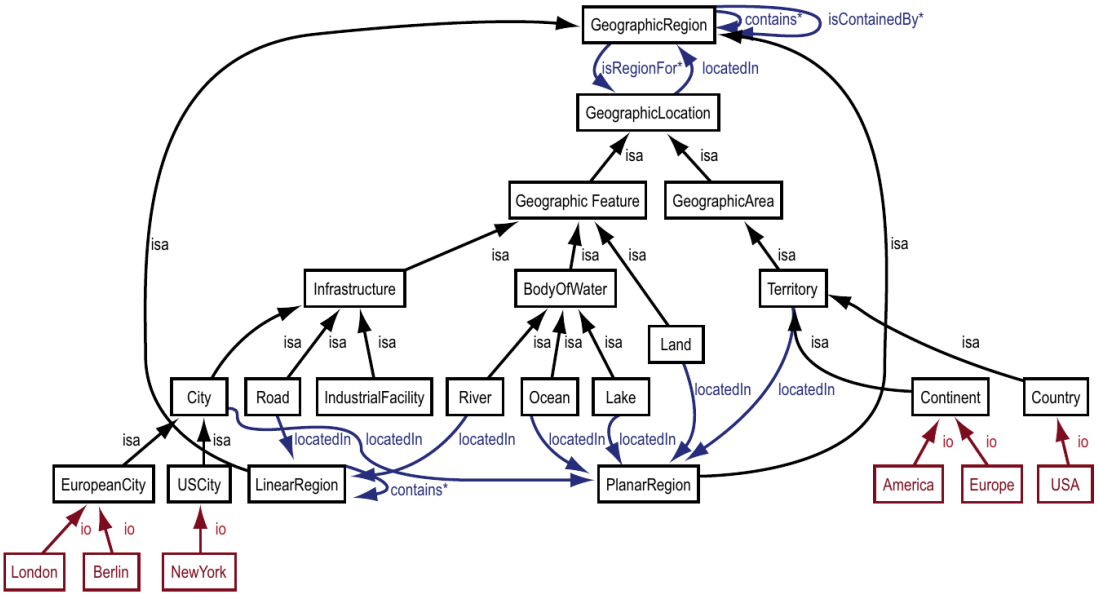

Figure 3.3. Geographical ontology, adopted from [7]

There are generally four types of ontologies depending on the level of generality and reusability. Upper ontologies are the most general, describing abstract concepts like continuants, processes, regions, etc. Significant representatives are Basic Formal

Ontology[2] or DOLCE[3]. Domain and task ontologies characterize a specific domain of interest, varying from biomedicine, genetics, and geography to scientific domain and clinical guidelines. Some examples are Gene Ontology[4] or Ontology for Biomedical Investigations[5]. The last type of ontology is application ontology describing a specific task in an application domain. The last-mentioned ontologies are usually not suitable for further reuse [7].

3.5 Knowledge representation for manufacturing domain

One of the thesis objectives is to conduct a survey of formal representation. The conducted survey includes seven papers referencing the manufacturing domain and formal representation.

According to [13], a manufacturing system is a composition of three dependent concepts, i.e., product, process, and resource. Figure 3.4 depicts a product, process, and resource design with examples, as well as optimization objectives, input models, and the desired output.

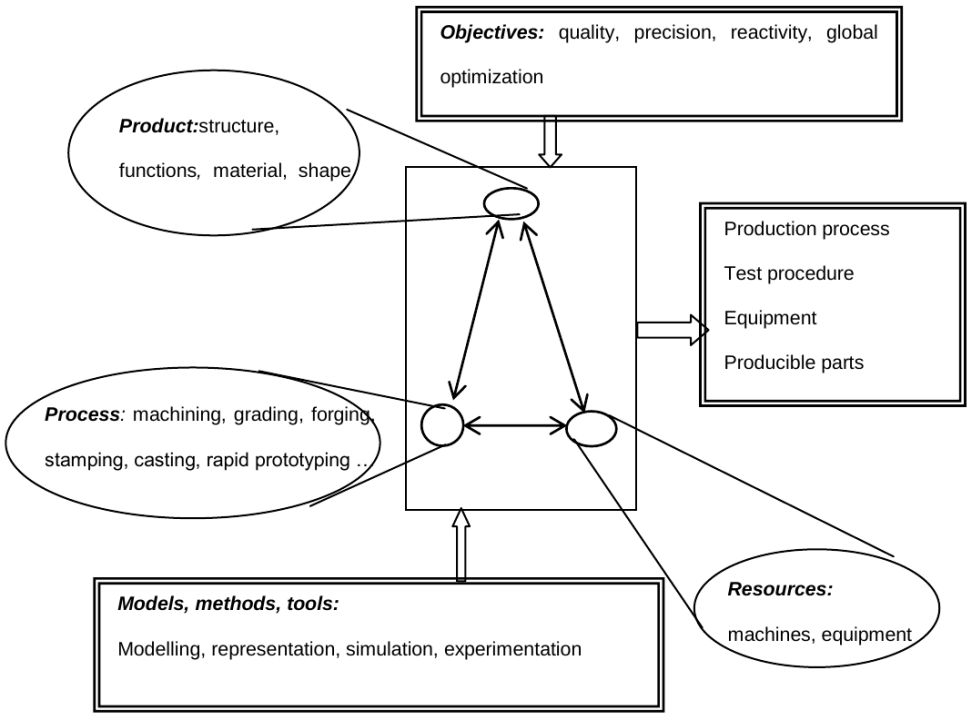

Figure 3.4. Product, process, and resource design, adopted from [13]

Paper [14] conducted a review of ontologies aiming at a representation of at least one of the product, process, and resource concepts. A result is a collection of nine ontologies with a covered scope and typical application.

[2] https://basic-formal-ontology.org/
[3] http://www.loa.istc.cnr.it/dolce/overview.html
[4] http://geneontology.org/
[5] http://obi-ontology.org/

The following paragraphs briefly describe selected ontologies and representations most relevant to the thesis goals and use case. First, we will describe selected ontologies from the mentioned paper [14] including Product Ontology (PRONTO), Manufacturing's Semantics Ontology (MASON), and Adaptive holonic Control Architecture for distributed manufacturing systems ontology (ADACOR). Other representations still relevant to our work (Lightweight Ontology for Sensors, Observations, Samples, and Actuators (SOSA); Design and Process Planning Integration (DPPI); Product, Process, and Resource (PPR)) follows.

Article [15] proposes a Product Ontology (PRONTO) for product modeling. PRONTO's primary motivation is to offer shareability within a single company or across different organizations. Its hierarchy manages the handling of product families, composition and decomposition structures, and constraint specification. The described case study related to the food industry demonstrates the representation of product variants at different levels.

Manufacturing's Semantics Ontology (MASON) [16] is a proposal for an upper ontology in a manufacturing domain as an answer to a demand for interconnected data in intelligent manufacturing. MASON is modeled using the semantic web technologies 5.1. Core concepts forming the ontology consist of entities, operations, and resources. Operations describe how the final product is being manufactured. It includes e.g. manufacturing operations, logistic operations, human operations, or launching operations. Resources cover all supporting elements in product manufacturing, e.g., machine tools, human resources, and geographical resources (plants, workshops, etc.). The paper outlines two applications of MASON, one as a support for an expert system for cost estimation during the design phase of manufacturing. The other is a knowledge representation for multiagent systems where the ontology is mapped to the JADE framework.

Adaptive holonic Control Architecture for distributed manufacturing systems ontology (ADACOR) [17] is formal architecture with it's proprietary upper ontology modeled in a frame-based language. ADACOR defines a taxonomy of classes and relations including concepts such as product, raw material, process plan, operation resources, etc., along with predicate, namely, componentOf, hasTool, Precedence, etc.

Lightweight Ontology for Sensors, Observations, Samples, and Actuators (SOSA) [18] is an ontology defining sensors and their observations, procedures used, observation results and interests, and platforms hosting sensors. SOSA takes inspiration from W3C-XG Semantic Sensor Network (SSN) [19] ontology and replaces its Stimulus Sensor Observation core. It is a W3C recommendation.

Design and Process Planning Integration (DPPI) [20] is a manufacturing process information model designed to overcome interoperability issues among the computer-aided design. The DPPI itself is designed in an object-oriented paradigm using UML[6]. The proposed object model enables estimating overall manufacturing cost and time by taking into account individual elements from product design, process planning, and manufacturing execution areas.

Paper [21] introduces an upper ontology Product, Process, and Resource (PPR) ontology for manufacturing domain. The motivation for this ontology are needs of Industry 4.0 together with interconnected information. The PPR ontology aims to provide integration of various domains by using semantic web technologies (section 5.1). The three underlying concepts building the PPR are products, operations, and workstations. The testing scenario of the PPR was run on the three-part truck assembly by an industrial robot. The utilization of the ontology with data showed an

[6] `https://cs.wikipedia.org/wiki/Unified_Modeling_Language`

evaluation of the available resources for truck assembly by semantic matchmaking.

To conclude our survey, most of the cited works present a manufacturing domain using at least one product, process, or resource. Moreover, the majority of conducted representations use a concept of ontology implemented by semantic web technologies. There are significant tendencies to use ontologies for data sharing in terms of industry 4.0. Finally, some formal representations show successful utilization in the industrial domain.

Chapter 4
Proposal of ontology

In this chapter, we propose SkodaOnto — ontology for manufacturing and measuring processes in Škoda Auto. We used a guide [22] as a support material for the development. The guide mentions three important notes:

- There is no one correct solution how to model an ontology.
- Ontology development is an iterative process.
- Concepts and relations in the ontology should be close to concepts and relations in the domain.

The second point was proven by several meetings with both domain experts from Škoda Auto Measurement Laboratories and Diribet. Every version of our ontology was discussed in order to meet the requirements and to be eligible for the utilization mentioned in the chapter 6. The following section presents the design process.

4.1 Modeling the ontology

The guide [22] suggests enumerating all important terms in the given domain before we start the designing phase. Table 4.1 summarizes all concepts and parameters from the machining and measuring part as we presented in section 2.2. Concepts are general entities, parameters are numeric values usually defining measure or construction settings.

	Machining	**Measuring**
concepts	object, npv, tool, nest, machining tool	characteristic, element, reference, measure machine, measure method, measure strategy, filtering method, filter type, filter subtype, construction method, sensor, sensor carrier
parameters		speed, step, desired point count, actual point count, measure time, angle range, filter parameter, eliminating outliers

Table 4.1. Enumeration of all important terms

The designing phase is divided into three consecutive stages that followed each other in time. Each stage is extending the results of the previous stage. Naturally, we did not manage to design the presented parts of the ontology on the first attempt; however, we present only the last version.

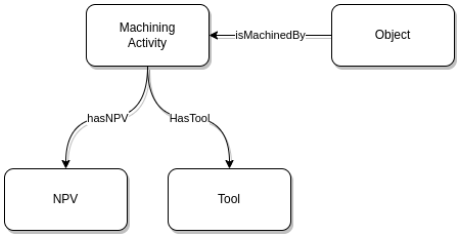

Figure 4.1. Core part of the machining process

4.1.1 First stage

The core part of the machining process is a ternary relationship object — tool — NPV. [23] suggests introducing a new concept representing a relation itself. Figure 4.1 illustrates four concepts (*Object, MachiningActivity, NPV, Tool*) connected with three object properties (*isMachinedBy, hasNPV, hasTool*).

An initial analysis of characteristics and elements revealed various specialized relations between (e.g. characteristic position is derived from a circle or a cone; in other programs, it may be derived from other elements or references), therefore, we decided to capture a general relations as shown in Fig. 4.2. There are four classes (*Object, Characteristics, Element, Reference*) and four object properties (*isDerivedFromReference, isDerivedFromElement, consistsOfElement, hasElement*).

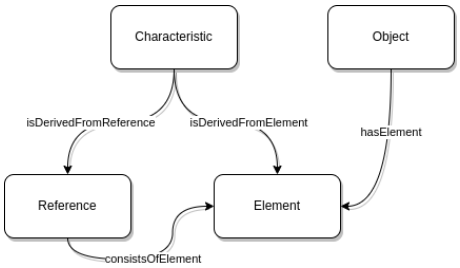

Figure 4.2. Relation between characteristic, element, reference, and object

The proposed part in Figure 4.2 raises a question of whether the characteristic shouldn't have a data property capturing its value. After a debate with our colleagues, we decided not to include it, since these values are already stored in Diribet's statistical software.

4.1.2 Second stage

The first stage (4.1.1) represents core structures for manufacturing and measuring processes. In the second stage, we define additional concepts representing the measuring process.

Figure 4.3 illustrates the fact that an element is measured and constructed with some settings, both of them set in a measuring machine. Thus, we introduce three new concepts (*MeasureSettings, ConstructionSettings, MeasuringMachine*) and five object properties (*hasMeasureSettings, hasConstructionSettings, usesDataFromMeasureSettings, isMeasuredBy, isConstructedBy*).

Both of the settings have their additional parameters (Fig. 4.4). *MeasureSettings* (Fig. 4.4a) is specified by two concepts (*MeasureMethod, MeasureStrategy*) and their object properties (*hasMeasureMethod, hasMeasureStrategy*). The rest are data properties (*hasSpeed, hasStep, hasDesiredPointCount, hasActualPointCount, hasMeasureTime,*

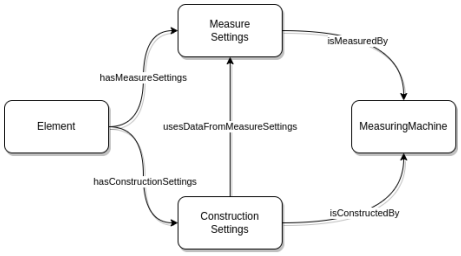

Figure 4.3. Construction and measure settings with a measuring machine

hasAngleRange). *ConstructionSettings* (Fig. 4.4b) is specified by four concepts (*FilteringMethod, FilterType, FilterSubtype, ConstructionMethod*) and their object properties (*hasFilteringMethod, hasFilterType, hasFilterSubtype, hasConstructionMethod*). The rest are data properties (*hasParameter, isEliminatingOutliers*).

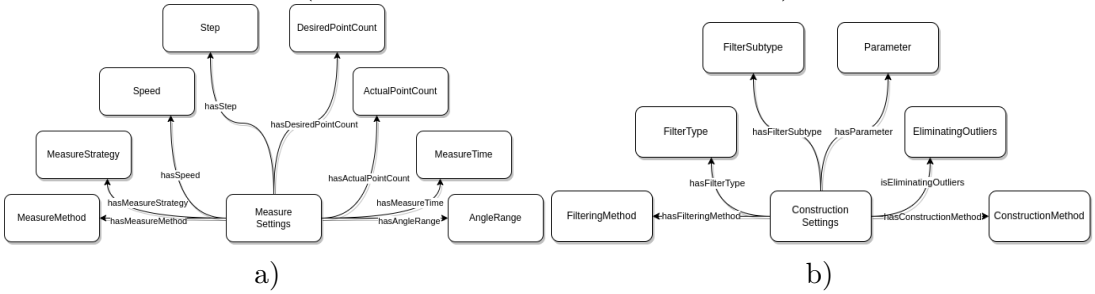

Figure 4.4. Parameters of measure and construction settings

4.1.3 Third stage

The last stage extends the *Tool* class with a machining tool together with a nest and the *MeasureSettings* class with a sensor and its carrier.

Figure 4.5 depicts two new concepts (*Nest, MachiningTool*) and two new object properties (*isPartOfNest, isPartOfMachineTool*).

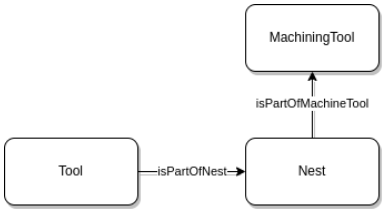

Figure 4.5. Tool, nest, and machine tool

Similarly, *MeasureSettings* class is extended by two new concepts (*Sensor, SensorCarrier*) and two new object properties (*isMeasuredBySensor, hasSensorCarrier*).

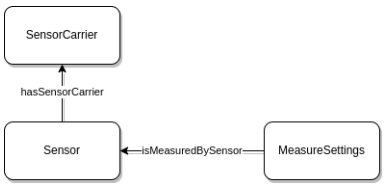

Figure 4.6. Sensor and sensor carrier

Finally, an overview of the entire ontology is illustrated in Fig. 4.7.

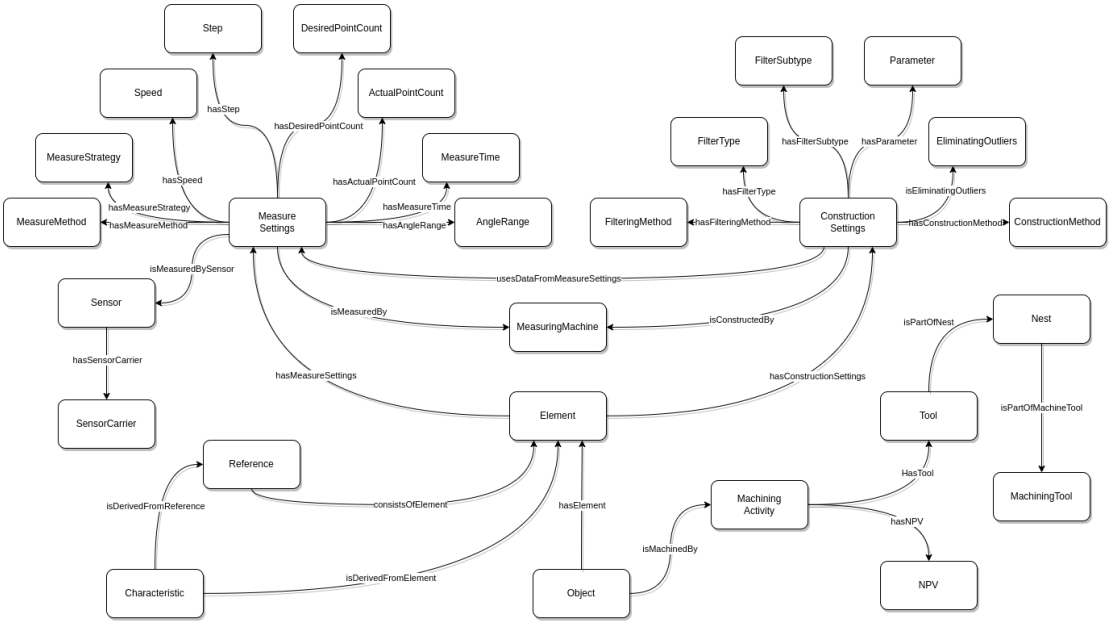

Figure 4.7. Overview of the ontology

Chapter 5
Implementation

This chapter presents an implementation stage of the work. We first introduce a Semantic web in section 5.1, Semantic web technologies are presented in section 5.2, ontology APIs used in the SkodaOnto are overviewed in section 5.3. Finally, section 5.4 presents the architecture and implementation of the application.

5.1 Semantic web

First, we introduce the term Semantic web. The Semantic Web is an extension of the World Wide Web made in a machine-readable way, i.e., with the ability to let computers process the web's semantics. Users interact with the web through agents solving tasks for them. Languages for knowledge representation, such as eXtensible Markup Language (XML) or Resource Description Framework (RDF) captures the data and semantics. XML by adding the structure of the data, while RDF links things, identified by Universal Resource Identifier (URI), together enabling searching for related things. Ontologies, in the area of the Semantic Web, are collections of information formally defining relations among terms. They involve taxonomies for class and relation definition and inference rules for inferring implicit information. Therefore, an ontology in the Semantic Web can be viewed as a vocabulary to let the computer "understand" the web's semantics. Digital signatures let agents verify trusted sources to deliver reliable results [24].

Technologies and languages, such as XML or RDF, mentioned above are only subsections building the Semantic Web. The following text presents the architecture of the Semantic Web and introduces the essential components related to ontologies.

5.2 Semantic web stack

As illustrated in Fig. 5.1, a semantic web stack is an architecture depicting the hierarchy of languages and concepts used in the Semantic Web. The model is built on layers where each layer exploits the layers below. The layers from the bottom of the stack up to the ontology layer are W3C[1] standards and are most relevant to this work. The following subsections present these layers.

5.2.1 RDF

The Resource Description Framework (RDF) is a W3C recommendation for representing information. The core structure of RDF is an RDF triple taking the form of *subject* — *predicate* — *object*. A set of RDF triples forms an RDF graph. A graph parallel to an RDF triple is a *node* — *edge* — *node*, where both subject and object are nodes, and the predicate is a directed edge. The subject can be either an IRI (Internationalized Resource Identifier) or a blank node (anonymous). An object is one of an IRI,

[1] `https://www.w3.org/`

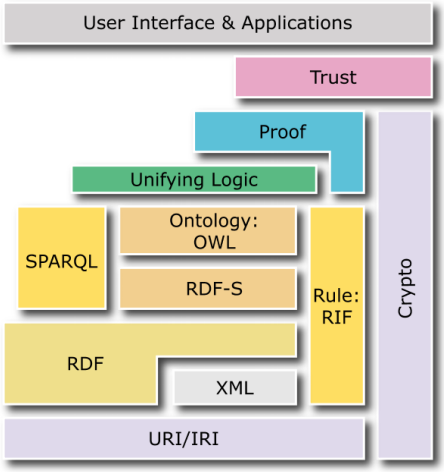

Figure 5.1. Architecture of semantic web, adopted from [25]

a blank node, or a literal. The predicate is the IRI which is a generalized version of URI allowing usage of a broader range of Unicode[2] characters. The blank node is a local identifier missing the RDF syntax. The literal is a value for representing strings, numbers, dates, etc [26].

5.2.2 RDFS

The RDF Schema (RDFS) is a semantic extension of RDF (5.2.1) recommended by W3C. The semantics are added by grouping resources, defining relations, and inferencing the data. This class and property system is similar to object-oriented languages; however, instead of defining properties as attributes (OOP way), RDFS defines property as a relation between resources. RDFS groups resources into class constructs and enables class inheritance. Properties link subject and object resources. The domain and the range modifiers of the property specify the class of instance that participates in a given property. The inheritance of properties is possible through the sub-property construct [27].

5.2.3 OWL

The Web Ontology Language (OWL) enriches the lower layers of the semantic web stack with class and property extension, including class relations, cardinalities, equalities, richer typing and characteristics of properties, and enumerated classes. OWL brings three sub-languages, OWL Lite, OWL DL, and OWL Full. The OWL Lite offers a classification hierarchy and simple constraints. The cardinality constraints are, however, limited only for 0 or 1 values. The OWL DL provides maximum expressiveness while preserving computational completeness and decidability. The DL attribute signifies the Description Logics introduced in section 3.3 acting as a building theory for OWL. Finally, the OWL Full extends the Lite and the DL version for the price of losing computational guarantees. Hence the reasoning may not be fully supported by reasoning software [28].

5.2.4 SPARQL

SPARQL Protocol and RDF Query Language (recursive acronym SPARQL) is a W3C recommendation for querying RDF data. As mentioned in subsection 5.2.1, RDF rep-

[2] `https://home.unicode.org/about-unicode/`

resents a directed labeled graph; thus, SPARQL searches for data via graph patterns. A graph pattern is a set of triple patterns similar to RDF triples, except one of the subject, predicate, or object may be a variable. SPARQL matches RDF subgraphs via these patterns and results in a set or RDF subgraph. SPARQL offers four query forms, particularly SELECT — returns a set of variables specified in a query, CONSTRUCT — returns an RDF graph specified by a graph template, ASK — returns a boolean value indicating whether a graph pattern matches or not, and DESCRIBE — returns an RDF graph describing resources found. Moreover, a set results of the SELECT and the ASK queries are serializable into one of JSON[3], XML[4], CSV[5], or TSV[6] format. Additional query clauses like WHERE (specifies graph pattern), DISTINCT (filters duplicate solutions), PREFIX (specifies namespace), or FILTER (restricts the output of the query) provides a further modification of queries [29].

The usage of SPARQL language together with the Owlready2 interface 5.3.2 is demonstrated in the chapter 6.

5.3 Ontology APIs

When building an ontology, one can choose from several ontology editors (Protégé, NeOn Toolkit, SWOOP, etc.) to assist the development process. Although these editors provide high-level user-friendly graphical interfaces, non-of them can convert data provided in the Škoda Auto use-case. Therefore, the data must be processed and then exposed for utilization programmatically. We decided to use two programming interfaces for reusability and further utilization of the SkodaOnto. The SkodaOnto TBox (3.3.1) and ABox (3.3.2) implementation are achieved via the OWL API [30]. Querying the data stored in the knowledge base is managed using the Owlready2 package [31].

5.3.1 OWL API

OWL API is a Java interface for creating and manipulating OWL ontologies. It is accessible as an open-source project under an LGPL license with various applications, including Protégé or Pellet reasoner. According to the number of downloads (over 34,000 in 2011), the popularity in the software community is remarkable. The design of OWL API presumes ontology as a set of axioms and annotations. The interface *OWLOntologyManager* is responsible for creating, loading, saving, changing, and saving ontologies instantiated as the *OWLOntology* holding instances of *OWLAxiom* class. Despite the lack of SPARQL querying, some essential query support is provided via the *OWLReasoner* interface that further supports consistency checking, class/property hierarchy computation, and entailment of axioms. The API supports serialization and deserialization in the RDF/XML, Turtle, OWL/XML, OWL Functional Syntax, The Manchester OWL Syntax, KRSS Syntax, and the OBO flat file format [30].

We chose OWL API because of its high level of abstraction, independence from concrete serialization, reasoner support, and easy-to-use interface. Disadvantages of using OWL API are its lack of SPARQL (5.2.4) support and its high verbosity. The usage of OWL API is presented in section 5.4.

[3] https://www.json.org/json-en.html
[4] https://www.w3.org/standards/xml/core
[5] https://en.wikipedia.org/wiki/Comma-separated_values
[6] https://en.wikipedia.org/wiki/Tab-separated_values

5.3.2 Owlready2

Owlready2 is a Python package providing ontology-oriented programming. In contrast to OWL API, Owlready2 offers transparent access to OWL ontologies by treating OWL classes as standard Python classes. Most of the OWL constructs, including classes, individuals, properties, datatypes, class expressions, etc., are expressible in Owlready2. Since Python offers object-oriented programming, the dot notation has a wide range of usage, namely, access to entities, access and modification of properties and annotations of entities; access and modification of domain, range, and inverse of properties; and for access and modification of role-fillers (constraints of properties including individuals). Owlready2 provides a native SPARQL engine supporting sufficient constructs for querying. Loading ontologies expects one of NTriples, RDF/XML, or OWL/XML syntaxes. Exporting is available in NTriples and RDF/XML syntaxes. The reasoning procedure is performed via the HermiT reasoner [31].

Advantages of choosing Owlready2 are: first, Owlready2 represents a simple and easy-to-use interface; second, the required SPARQL support; and third, since numerical simulation (6.2) is a Python module, Owlready2 overcomes language incompatibility. Other utilizations using Owlready2 are presented in chapter 6.

5.4 Architecture of the application

Figure 5.2 illustrates the intended usage of our application (SkodaOnto). The application loads the data (currently as a .xlsx table) and outputs the knowledge base. The Python module loads the knowledge directly as a file or can be fetched remotely in future versions.

Figure 5.3 captures the architecture of the SkodaOnto application. The application is written in Java and is responsible for the creation of SkodaOnto ontology (SkodaOnto class) and successive creation of individuals from the provided dataset (DataLoader class). The application outputs two files — *knowledge_base.owl* and *knowledge_base.ttl* representing our knowledge base.

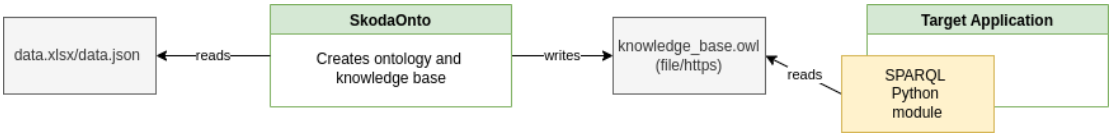

Figure 5.2. Class diagram

5.4.1 Defining the ontology

The SkodaOnto class defines the TBox and provides methods for ABox assertions. The class composes an *AbstractXlsxLoader*, which is responsible for the reading of the data and the TBox assertions calls. The following paragraphs briefly summarize the features of SkodaOnto class.

The base IRI of the SkodaOnto is *"https://skoda-onto.com"*. Since the ontology currently represents one type of engine head, it will be necessary to introduce new namespaces for other engine head types such as *"https://skoda-onto.com/type1"*, *"https://skoda-onto.com/type2"*, etc. in future versions.

Manager, *ontology* and *df* references OWL API objects. The *manager* creates and saves *ontology* that holds all owl axioms. The *df* (data factory) creates OWL API

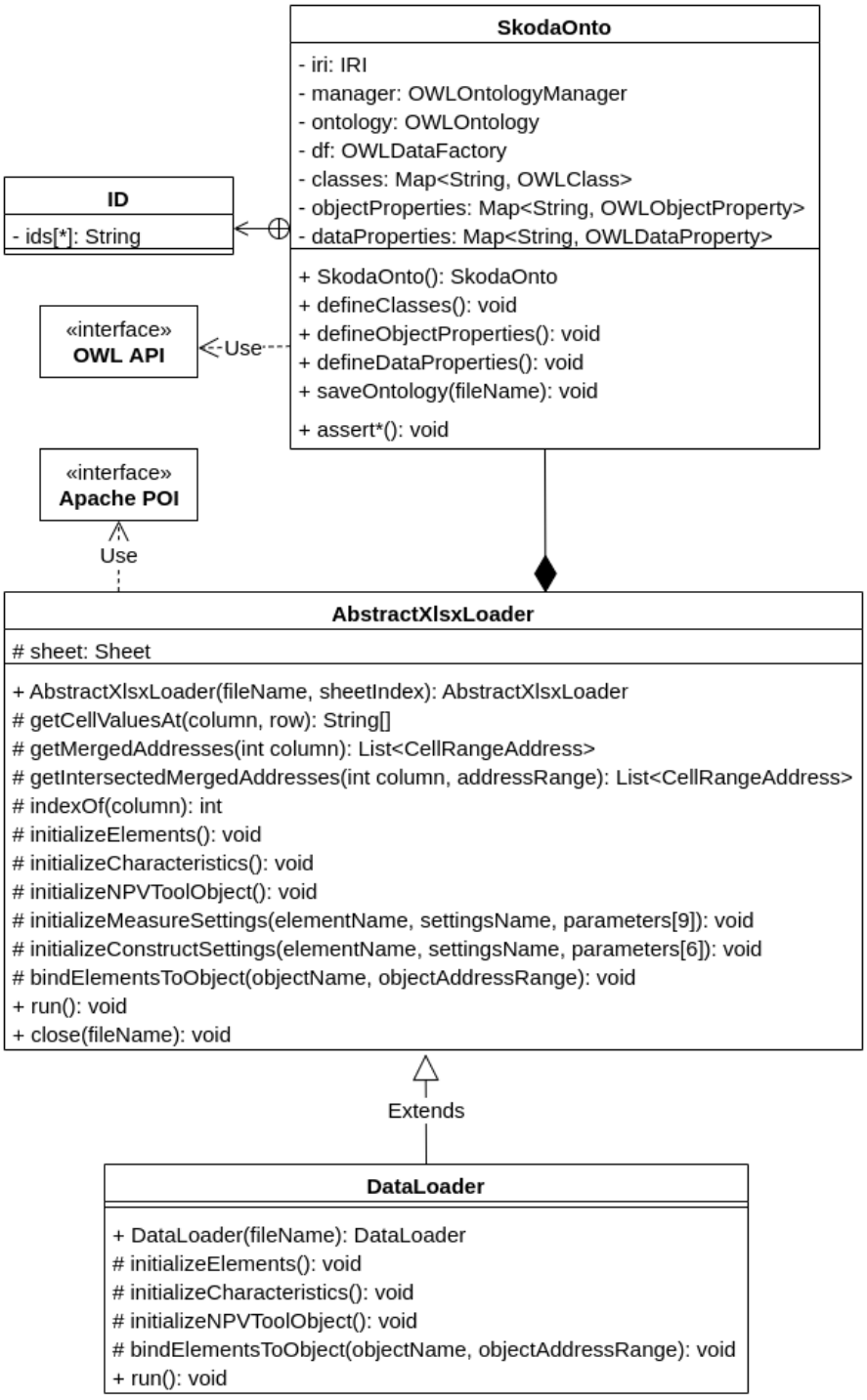

Figure 5.3. Class diagram

objects in a *defineClasses()*, *defineObjectProperties()* and *defineDataProperties()* methods.

Instead of having these objects as class attributes, we decided to use three hash maps (*classes*, *objectProperties* and *dataProperties*) for storing all the class and property objects created by the *df*. Storing objects in hash maps improves the code readability and transparency for the price of a slightly longer access time of objects. Objects can be retrieved by the same identifier as they are defined in the SkodaOnto.

Class and property identifiers are defined in the *ID* inner class. This class holds string identifiers used in the ontology and for retrieving objects from the hash maps.

Besides that, SkodaOnto class exposes methods for individual assertions. These methods and their helper methods are not shown in the class diagram due to their amount.

5.4.2 Defining individuals

Before we describe the process of asserting individuals, it is appropriate to introduce the provided data by Škoda Auto. The data form a table (XLSX format[7]) constructed by joining several other tables and resources. For illustration, one of the resources is a TSV[8] file exported from a Zeiss Calypso[9] metrology software. These tables and resources were manually processed by a colleague from Škoda Auto. If we take into account the time needed for constructing the final table multiplied by the number of engine head types, a huge effort needs to be made to cover the whole *operation 30* (2.2.1). Thus, future versions of SkodaOnto will load individuals using a standard data exchange format JSON, as was promised by Škoda Auto. Figure 5.4 shows a simplified version of the final table. Unfortunately, real values are anonymized due to Škoda Auto's policy for internal data sharing. Figure 5.4 outlines the header of the table and the structure of NPV, Tool, and Object columns. The full table is attached as the *data.xlsx* file.

NPV	Tool	Object	Characteristic	Element1	Element2	Reference	Measure Settings	Construction Settings
npv1	tool1	obj1	position_obj1	plane_obj1		std_coord_system	Data	
			straightness_obj1	plane_obj1		plane_P		
			angle_obj1	plane_obj1	plane_obj2	std_coord_system		
	tool2	obj2	diameter_obj2	circle_obj2		std_coord_system		
			position_obj2	circle_obj2		std_coord_system		
			straightness_obj2	plane_obj2		std_coord_system		
npv2 npv3	tool3 tool1 tool4	obj3	diameter1_obj3	circle1_obj3		coord_system_obj2		
			diameter2_obj3	circle2_obj3		coord_system_obj2		
			concentricity_obj3	circle1_obj3	circle2_obj3	coord_system_obj2		
		obj4	diameter1_obj4	circle1_obj4		coord_system_obj2		
			diameter2_obj4	circle2_obj4		coord_system_obj2		
			concentricity_obj4	circle1_obj4	circle2_obj4	coord_system_obj2		
		obj5	diameter1_obj5	circle1_obj5		coord_system_obj2		
			diameter2_obj5	circle1_obj5		coord_system_obj2		
			concentricity_obj5	circle1_obj5	circle1_obj5	coord_system_obj2		

Figure 5.4. Simplified structure of the table

Assertions of individuals are managed by *DataLoader* class. The class extends *AbstractXlsxLoader* class that acts as a base class for loading data in XLSX format. Abstract class contains a mechanism for loading a spreadsheet file and interacting with it,

[7] https://en.wikipedia.org/wiki/Office_Open_XML
[8] https://en.wikipedia.org/wiki/Tab-separated_values
[9] https://www.zeiss.com/metrology/products/software/calypso-overview/calypso.html

such as getting merged cells in a given column or indexing a column by its name. It also declares abstract methods that should be implemented in a specific loader. Apache POI[10] API manages manipulation of the XLSX file.

The most interesting part of the loader is asserting the ternary relationship npv — tool — object (the ternary relationship is described in subsection 4.1.1). Figure 5.4 depicts relations between these concepts as orange arrows. For example, *npv2* is set to these tools: *tool1*, *tool3*, and *tool4*; *tool1* machines objects *obj3*, *obj4*, *obj5*, and *obj1*; however *obj1* is machined only when *npv1* is set and objects *3, 4, and 5* are machined when *npv2* and *npv3* are set. There are three characteristics measured on *obj1*, namely *position_obj1*, *straightness_obj1*, and *angle_obj1*. Characteristic *angle_obj1* is derived from elements *plane_obj1* and *plane_obj2* and from reference *std_coord_system*. If we imagine relations between NPVs, tools and objects as tree structures (orange arrows), each unique path from the root to the leaf forms the ternary relationship. This assertion is achieved via recursive method *initializeNPVToolObject* implemented in the *DataLoader* class.

[10] `https://poi.apache.org/`

Chapter 6
Usage and results

This chapter presents the utilization of developed ontology. We present three areas (reduction of quality costs, numerical simulation, and diagnostics), where our ontology plays a crucial role. We developed only an interface between the ontology and the target application. The applications are subject to future work or in the case of numerical simulation, developed by Mr. Jaroslav Staněk from Diribet.

We use SPARQL queries (5.2.4) and OwlReady2 (5.3.2) to retrieve all necessary information from our knowledge base. Queries and functions are presented in the following code listings. All SPARQL queries presented in the following text use these prefixes:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <https://skoda-onto.com#>
```

6.1 Reduction of quality costs

Currently, the sampling plan mentioned in chapter 2 is very static. A constant set of more than 700 quality characteristics is prescribed to be measured on one part sampled in constant intervals (e.g., every day). Since many of those quality characteristics never exceeded tolerance or control limits, one can expect that their acquisition can be less frequent, which would reduce measurement costs without any increase in risk of an undetected problem. On the other hand, some quality characteristics are very risky and often get close to the limits or exceed them. It is reasonable to increase the frequency of their acquisition since it could reduce the risk of an undetected problem. The set of acquired quality characteristics can be optimized with respect to the current situation, which would lead to an adaptive sampling plan.

More specifically, Feigenbaum [32] defines two types of quality costs: costs of conformance and costs of non-conformance. The costs of conformance are e.g., costs of inspection, testing, quality planning etc. The costs of non-conformance are e.g., costs of scrap or rework. Very generally, the total quality cost can be defined as: $cost_{total} = cost_{conformance} + cost_{non-conformance}$.

Thus, the general problem of quality control optimization is a minimization of the total quality cost. Although many parameters and settings of the whole quality control loop can influence the total quality costs, one can focus on the optimization of the set of quality characteristics to be acquired. However, the optimization can be advantageously defined as a search for an optimal subset of elements to be measured. This reduces the search space because the number of characteristics is greater than the number of elements. This is due to the fact that multiple quality characteristics are mostly computed from one measured element (e.g., hole diameter, roundness, x and y

position is computed from one circle measured on a hole). If an element is measured, it makes sense to compute all its quality characteristics, since (1) it does not influence conformance cost (cost for computation of characteristics are neglectable compared to costs for measurement of the element) (2) it can reduce the non-conformance costs.

When the optimal subset of elements is found, which minimizes the total quality cost, the optimal subset of quality characteristics is defined as the set of all quality characteristics that are computed from those optimal elements. This set can be obtained by querying our ontology.

The following query returns all elements:

```
SELECT DISTINCT ?element
WHERE {
    ?element rdf:type :Element
}
```

These elements can be processed by the next query. The *el* is substituted by the given element. The query returns all characteristics which are derived from the given element.

```
SELECT DISTINCT ?characteristic
WHERE {
    :characteristic :isDerivedFromElement :el
}
```

Previous queries are encapsulated by the following functions located in the *optimization.py* module:

```
def getAllElements() -> list
def getCharacteristics(element: str) -> set
```

Example of the function call for element *circle2_obj3*:

```
getCharacteristics("circle2_obj3") -> {'diameter2_obj3',
'concentricity_obj3'}
```

Moreover, it should be noted that this describes the potential application of our ontology. The full proposal of sampling plan optimization and adaptation would include also the way of estimation of the costs or the optimization procedure. This is out of the scope of this thesis. Finally, in addition to the optimization of the set of measured elements, the optimization of the sampling interval could be also performed, which would lead to fully adaptively changing prescriptions of which elements to measure and when to measure them.

6.2 Numerical simulation

The numerical simulation is developed by Diribet mainly as a proof of concept for the OOC detector. The testing environment also helps to optimize measurement strategies and estimate the risk. To implement the numerical simulation, a need for process structure emerges.

The numerical simulation simulates *operation 30* as mentioned in subsection 2.2.1. The operation converts inputs to outputs, thus the operation is represented by operation matrices. The simulation is a linear transformation — projection of process parameters (including tools and NPVs) to measured characteristics. The Metropolis-Hastings

algorithm generates process parameters of the first matrix. The rest of the first matrix is an adjacency submatrix representing relations between tools and elements and NPVs and elements. The second matrix is an adjacency matrix representing relations between elements and characteristics.

The numerical simulation requires the construction of three adjacency matrices. The element of the matrix indicates whether pairs of individuals are related or not. I.e., the element $b_{i,j}$ ($i \in I$, $j \in J$; J, I sets of individuals) of the matrix B equals one if individuals i and j are related and zero otherwise. The three matrices are relations between elements — tools, elements — NPVs, and features — elements.

Sets I and J for one matrix are constructed by the following generic SPARQL query. The word *class* is substituted by one of *Tool*, *NullPoint*, *Element*, or *Characteristics*.

```
SELECT DISTINCT ?individual
WHERE {
    ?individual rdf:type :class
}
```

This query searches for all relevant tools for the given element *el*.

```
SELECT DISTINCT ?tool
WHERE {
    ?object :hasElement :el .
    ?object :isMachinedBy ?machining .
    ?machining :hasTool ?tool
}
```

Following query searches for all relevant NPVs for a given element *el*.

```
SELECT DISTINCT ?npv
WHERE {
    ?object :hasElement :el .
    ?object :isMachinedBy ?machining .
    ?machining :hasNullPoint ?npv
}
```

Finally, the last query searches for all elements that construct a given characteristic *char*.

```
SELECT DISTINCT ?element
WHERE {
    :char :isDerivedFromElement ?element
}
```

Presented SPARQL queries are encapsulated and executed in the python module *simulation.py* that can be directly imported by the numerical simulation application. Function headers providing the interface are listed below. The **All*()* functions return a list of individuals, which corresponds to the first mentioned query. The rest of the functions take a string argument parametrizing the the rest of queries respectively.

```
def getAllCharacteristics() -> list:
def getAllTools() -> list:
def getAllNPVs() -> list:
def getAllElements() -> list:
def getTools(element: str) -> set:
```

```
def getNPVs(element: str) -> set:
def getElements(characteristic: str) -> set:
```

6.3 Diagnostics

Diagnostics is a process of identifying a potential cause of the OOC state. We thought out several approaches to solve this problem, nevertheless not all are guaranteed to give a correct solution.

The most simple would be a penalization of individuals that are related to a suspicious characteristic. However, there are some individuals, such as sensors, that are often related to a huge set of other individuals, therefore these individuals would be penalized the most, even though they might not be the cause of the OOC state. Figure 6.1 illustrates the problem, suspicious characteristic "concentricity_obj4" is isolated and its related individuals are penalized by one unit. One can see that sensor "K130-5" together with its carrier "K130" and tool "t1" have already been penalized before. Thus, the high penalization score of the sensor and the tool indicates that the sensor and the tool are probably the cause of the suspicious characteristic. That doesn't have to be true.

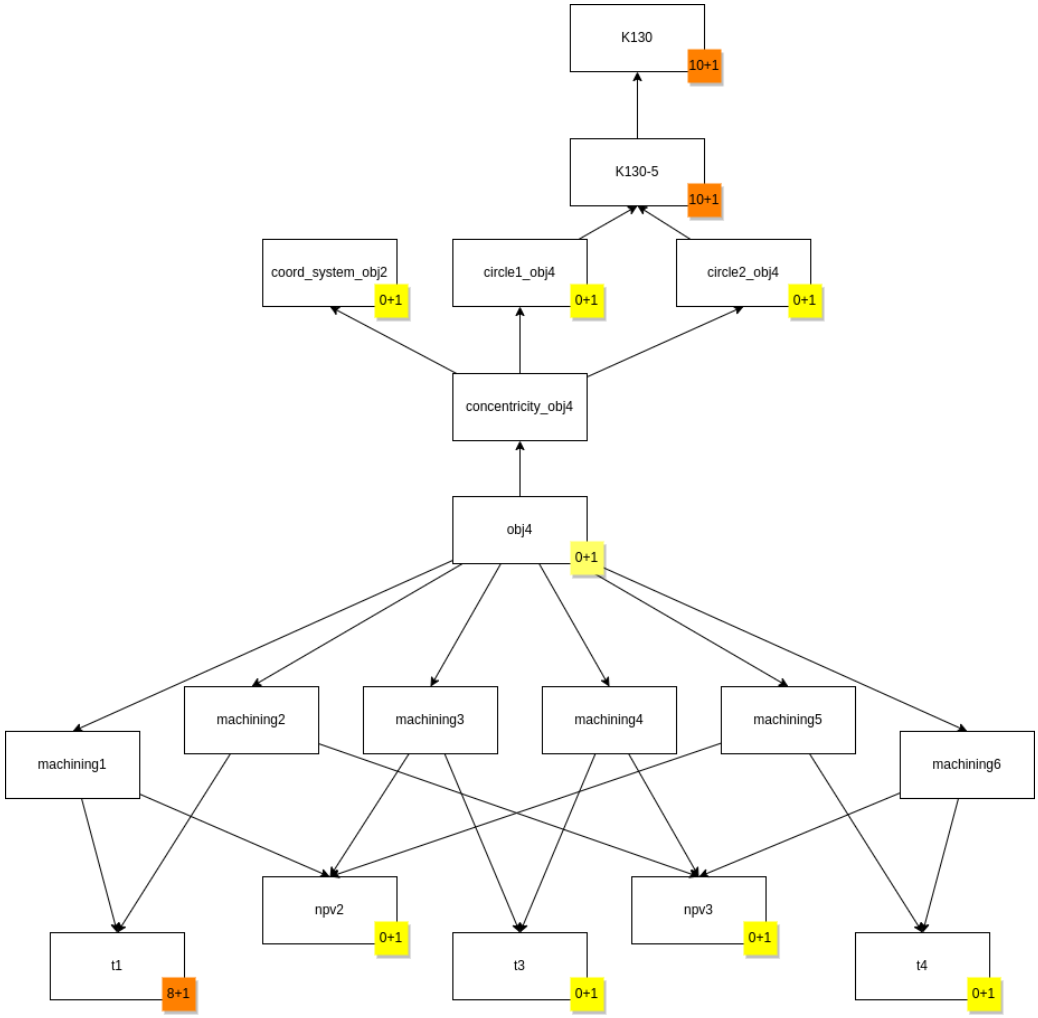

Figure 6.1. Penalization of related individuals

Bayesian networks would present another approach. However, there is not a direct way how to asses the conditional probability distribution.

Finally, the cause-and-effect diagram [5] is a tool for finding a potential cause of the defect. In our case, a suspicious characteristic (a characteristic that probably raised an alarm) is isolated and the cause-and-effect diagram should help to identify the possible cause.

The *diagnostics.py* module offers functions for retrieval sets of related individuals to a given characteristic. The *getAll(characteristic)* returns a set of all related individuals for the given characteristic (as if all functions for specified characteristic were called at once).

```
def getObject(characteristic: str) -> set:
def getNPV(characteristic: str) -> set:
def getTool(characteristic: str) -> set:
def getReference(characteristic: str) -> set:
def getElement(characteristic: str) -> set:
def getSensor(characteristic: str) -> set:
def getSensorCarrier(characteristic: str) -> set:
def getAll(characteristic: str) -> set:
```

Assuming anonymized data from the table 5.4, calling the *getAll()* for the characteristic "concentricity_obj4" returns following:

```
getAll("concentricity_obj4") -> {'t1', 'circle2_obj4', 'K130-5', 'K130',
't3', 'npv3', 'circle1_obj4', 'coord_system_obj2', 'obj4', 't4', 'npv2'}
```

Chapter 7
Conclusion

To conclude, the main objective of the thesis was to formally represent the quality control process in the machining domain. We studied and analyzed the provided use case and proposed possible usage of the formal representation. Then, we conducted the survey of knowledge representation and selected ontology as the best option for our use case. We designed the ontology that fits the provided use case and implemented the proposed ontology together with the assertion of use case data. Finally, we presented the utilization of our ontology and implemented interfaces for accessing our knowledge base.

The result of the work is the java application capable of reading use case data and creating the knowledge base. The knowledge base is built by using semantic web technologies (Web Ontology Language and its underlying formalisms) and OWL API. The application is designed to be easily modified to read data from different formats as well as create only the ontology itself. The attached python scripts allow accessing data from the knowledge base. Scripts use W3C's recommendation SPARQL and Owlready2 python package.

The data retrieved by the numerical simulation script was successfully integrated into the target simulation application. Unfortunately, we cannot show the results as the simulation contains sensitive data.

7.1 Discussion

To compare with our survey, we stuck to the manufacturing system design and implemented product, process, and resource concepts. We also selected the ontological approach as the recent trend in formal representations as well as the use of semantic web technologies.

The general concept of knowledge representation is not suitable only for the area of Industry 4.0, but also for other fields such as healthcare or agriculture. Having a knowledge representation allows us to represent complex real-world systems and let the computer reason about the formal representation. We think this area will be highly utilized in many domains requiring interconnected data.

One of the most challenging parts of this work was data processing. The provided data were constructed from several sources and it was difficult to fully integrate them into the knowledge base. Again, due to the sensitivity of the data, the data had to be anonymized. The programmatical approach of the development taught us the background structure of ontologies as well as SPARQL query processing.

Overall, the design and development of the formal representation required intensive meetings and debates with our colleagues from Škoda Auto and Diribet. It was essential to understand the manufacturing domain enough to create an ontology that is as close as possible to the target domain.

7.2 Future work

Future work will be focused on the diagnostics problem of quality control. The diagnostics will need the structure of both manufacturing and measuring processes, thus the implemented knowledge representation will be utilized in some way. We think that the reasoning feature of the knowledge representation systems would be beneficial to draw inferences from the data. We plan to deploy rules to reason about the data. To do that, we have to explore the manufacturing domain in depth. I.e., understanding the causes and effects of out-of-control states.

The reduction of quality costs will also need to be further explored. We plan to create different measurement plans that will be tested by the numerical simulation. To create these plans, it may be necessary to create more complex structures of individuals that are measured by the CMMs. E.g., adding hierarchical structures of objects and elements.

Naturally, the implemented knowledge representation will be further developed to adapt the utilization. The next major update of the knowledge representation will involve the processing of the data from standard exchange format JSON and extending the knowledge base by including the rest of the engine head types. We also plan to modify relations to optimize the querying time complexity.

References

[1] Francesco Lelli. Interoperability of the Time of Industry 4.0 and the Internet of Things. *Future Internet*. 2019, 11 36. DOI 10.3390/fi11020036.

[2] Thomas Burns, John Cosgrove, and Frank Doyle. A Review of Interoperability Standards for Industry 4.0.. *Procedia Manufacturing*. 2019, 38 646-653. DOI https://doi.org/10.1016/j.promfg.2020.01.083. 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing.

[3] Martin Macas, Diem H. Nguyen, and Charlotte Panuskova. *Support Vector Machines for Control of Multimodal Processes.* In: Ajith Abraham, Andries Engelbrecht, Fabio Scotti, Niketa Gandhi, Pooja Manghirmalani Mishra, Giancarlo Fortino, Virgilijus Sakalauskas, and Sabri Pllana, eds. *Proceedings of the 13th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2021)*. Cham: Springer International Publishing, 2022. 384–393. ISBN 978-3-030-96302-6.

[4] Jan Urban, Libor Beranek, Michal Koptiš, Jan Šimota, and Ondřej Košťák. Influence of CMM scanning speed and inspected feature size on an accuracy of size and form measurement. *Manufacturing Technology*. 2020, 20 538-544. DOI 10.21062/mft.2020.074.

[5] Douglas C Montgomery. *Introduction to statistical quality control*. 6 ed.. Chichester, England: John Wiley & Sons, 2008.

[6] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. *Support Vector Method for Novelty Detection.* In: 1999. 582-588.

[7] Rudi Studer, Stephan Grimm, and Andreas Abecker. *Semantic web services*. 1st ed.. Berlin: Springer-Verlag, 2007. ISBN 978-3-540-70893-3.

[8] Xiuling Li, Shusheng Zhang, Rui Huang, Bo Huang, Changhong Xu, and Yajun Zhang. *A survey of knowledge representation methods and applications in machining process planning*. 2018.
`http://dx.doi.org/10.1007/s00170-018-2433-8`.

[9] Xiaojun Chen, Shengbin Jia, and Yang Xiang. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*. 2020, 141 112948. DOI https://doi.org/10.1016/j.eswa.2019.112948.

[10] Stuart J. Russell, and Peter Norvig. *Artificial intelligence*. 3rd ed.. Upper Saddle River: Prentice Hall, 2010. ISBN 978-0-13-604259-4.

[11] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2007. ISBN 9780511711787.
`https://doi.org/10.1017/cbo9780511711787`.

[12] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*. 1993, 5 (2), 199-220. DOI https://doi.org/10.1006/knac.1993.1008.

[13] Patrick Martin, and Alain d'Acunto. Design of a production system: An application of integration product-process. *Int. J. Computer Integrated Manufacturing*. 2003, 16 509-516. DOI 10.1080/0951192031000115831.

[14] Qiushi Cao, Cecilia Zanni-Merk, and Christoph Reich. *Ontologies for Manufacturing Process Modeling: A Survey*. In: 2018.

[15] Marcela Vegetti, Horacio Leone, and Gabriela Henning. PRONTO: An ontology for comprehensive and consistent representation of product information. *Engineering Applications of Artificial Intelligence*. 2011, 24 (8), 1305-1327. DOI https://doi.org/10.1016/j.engappai.2011.02.014. Semantic-based Information and Engineering Systems.

[16] S. Lemaignan, A. Siadat, J.-Y. Dantan, and A. Semenenko. Mason: A proposal for an ontology of manufacturing domain. *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*. DOI 10.1109/dis.2006.48.

[17] Stefano Borgo, and Paulo Leitão. *Foundations for a Core Ontology of Manufacturing*. In: 2007. 751-775. ISBN 978-0-387-37019-4.

[18] Krzysztof Janowicz, Armin Haller, Simon J.D. Cox, Danh Le Phuoc, and Maxime Lefrançois. Sosa: A lightweight ontology for sensors, observations, samples, and Actuators. *Journal of Web Semantics*. 2019, 56 1–10. DOI 10.1016/j.websem.2018.06.003.

[19] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*. 2012, 17 25-32. DOI https://doi.org/10.1016/j.websem.2012.05.003.

[20] Shaw C. Feng, and Eugene Y. Song. A manufacturing process information model for design and process planning integration. *Journal of Manufacturing Systems*. 2003, 22 (1), 1-15. DOI https://doi.org/10.1016/S0278-6125(03)90001-X.

[21] Václav Jirkovský, Ondřej Šebek, Petr Kadera, Pavel Burget, Sönke Knoch, and Tilman Becker. *Facilitation of Domain-Specific Data Models Design Using Semantic Web Technologies for Manufacturing*. In: New York, NY, USA: Association for Computing Machinery, 2019. 649–653. ISBN 9781450371797.

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

[23] Natasha Noy, and Alan Rector. *Defining N-ary Relations on the Semantic Web*. 2006.
`https://www.w3.org/TR/swbp-n-aryRelations/`.

[24] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities. *ScientificAmerican.com*. 2001,

[25] *Semantic Web, and Other Technologies to Watch: January 2007*.
`https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/`.

[26] Richard Cyganiak, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J. Carroll, and Brian McBride. *RDF 1.1 Concepts and Abstract Syntax W3C Recommendation 25 February 2014*. 2014.
`https://www.w3.org/TR/rdf11-concepts/`.

[27] Dan Brickley, R.V. Guha, and Brian McBride. *RDF Schema 1.1 W3C Recommendation 25 February 2014*. 2014.
`https://www.w3.org/TR/rdf-schema/`.

[28] Deborah L. McGuinness, and Frank van Harmelen. *OWL Web Ontology Language Overview W3C Recommendation 10 February 2004* . 2004.
`https://www.w3.org/TR/owl-features/`.

[29] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. *SPARQL 1.1 Query Language W3C Recommendation 21 March 2013*. 2013.
`https://www.w3.org/TR/sparql11-query/`.

[30] Matthew Horridge, and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*. 2011, 2 (1), 11–21. DOI 10.3233/sw-2011-0025.

[31] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*. 2017, 80 DOI 10.1016/j.artmed.2017.07.002.

[32] A V Feigenbaum. *Total Quality Control*. London, England: McGraw-Hill Education (ISE Editions), 1991.

Appendix A
Attached files

`Main.java`	Class with the main method of the java project.
`AbstractXlsxLoader.java`	Abstract class for reading .xlsx files.
`DataLoader.java`	Class responsible for the assertion of individuals.
`SkodaOnto.java`	Class representing the proposed ontology with assertion methods for individuals.
`data.xlsx`	File containing anonymized data of machining and measuring process.
`knowledge_base.owl`, `knowledge_base.ttl`	Knowledge base files in OWL/XML and Turtle syntax.
`simulation.py`	Python script for accessing knowledge base, specialized for numerical simulation task.
`diagnostics.py`	Python script for accessing knowledge base, specialized for diagnostics task.
`optimization.py`	Python script for accessing knowledge base, specialized for optimization task.