Výsledek výzkumu a vývoje projektu TN01000024

Identifikační kód výsledku:	TN01000024 - V45
Název:	SkodaOnto – ontologie procesu výroby a měření kvality
Typ výsledku:	R – Software
Anotace:	Softwarový balík, který obsahuje Java aplikaci zodpovědnou za vytvoření znalostní báze, znalostní bázi samotnou a skripty pro interakci s touto znalostní bází v Pythonu. Znalostní báze je OWL ontologie reprezentující proces obrábění hlav motoru a proces měření znaků kvality vyrobených dílů. Ontologie obsahuje sémantické znalosti o nástrojích a jejich nastavení, obráběných objektech, měřících strojích a jejich čidlech a výsledcích měření znaků kvality. Přiložené skripty jsou příklady využití ontologie pro tři různé aplikace (simulace výroby a měření, diagnostika statisticky nezvládnutého stavu procesu nebo optimalizace ceny zajištění kvality). Java aplikace využívá OWL API pro vytvoření ontologie a načtení individuí. Skripty využívají OwlReady2 balíček pro Python pro interakci se znalostní bází pomocí SPARQL dotazů.
Termín dosažení:	12/2022
Vlastníci výsledku:	ČVUT v Praze, ŠKODA AUTO a.s.

Tento výsledek byl vytvořen se státní podporou Technologické agentury ČR v rámci programu Národní Centra kompetence, projekt č. TN01000024, Národní Centrum Kompetence-Kybernetika a umělá inteligence (NCK KUI).

SkodaOnto - Ontology for representation of manufacturing and measuring process

Petr Jůza, Martin Macaš

January 2023

This technical report describes the proposal of ontology representing the provided Škoda quality control use case. It presents technologies used for the proposed ontology and describes the implementation of the ontology. A more detailed description can be found in bachelor thesis of Petr Jůza [6]. The ontology was created by Petr Jůza within his work on subproject "Big data, security and data integrity" of the project "National Competence Center – Cybernetics and Artificial Intelligence" funded by Technology Agency of the Czech Republic.

1 Quality inspection use case

In this chapter, we analyze the provided use case, describe the given domain and present possible utilization. Section 1 presents a quality control loop which is a significant part of quality management, describes its steps and briefly suggests possible utilization. Section 1.1.5 presents the provided use case focusing on both the manufacturing and quality inspection parts in detail.

1.1 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. 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 [9].

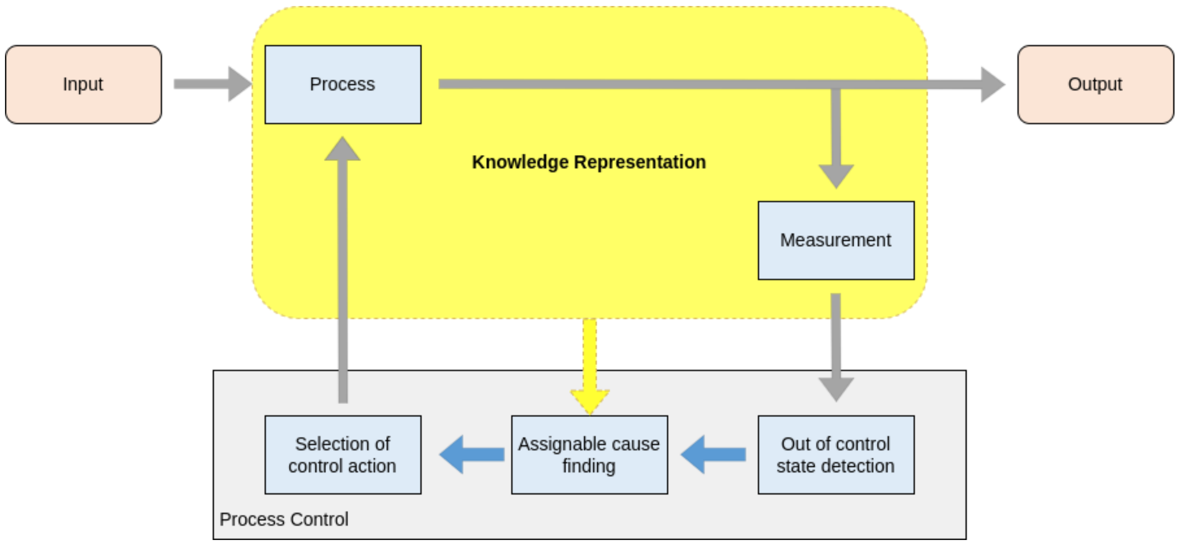


Figure 1: Quality control loop

1.1.1 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 1.2 presents the process representation in detail.

1.1.2 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 [14], 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 1.2.1 focuses on the measuring section of the use case.

1.1.3 Out of control state detection

First, we clarify an out-of-control process concept. Montgomery [10] 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 1.1.1. Two sufficient methods are presented: the Shewhart control chart and its multivariate extension - the Hotelling T^2 control chart.

A paper [9] applies One-Class Support Vector Machine (OSVM) [13] 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.

1.1.4 Assignable cause finding

Positive detection of the OOC state launches an assignable cause-finding procedure [9]. 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.

1.1.5 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.

1.2 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.

1.2.1 Machining Part

At this time, the process handles the manufacturing of eleven engine head types. Fig. 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

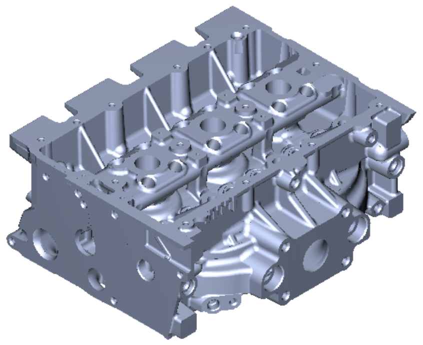


Figure 2: 3D model of an engine head

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. 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). An object is usually machined by multiple tools. The quality of an object is measured in form of characteristics related to a given object.

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

1.2.2 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 (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.

2 Proposal of ontology

In this chapter, we propose SkodaOnto — ontology for manufacturing and measuring processes in Škoda Auto. We used a guide [11] as a support material for the development. The guide mentions three important notes:

[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`

Table 1: Enumeration of all important terms

	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

- 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 future utilization. The following section presents the design process.

2.1 Modeling the ontology

The guide [11] suggests enumerating all important terms in the given domain before we start the designing phase. Table 1 summarizes all concepts and parameters from the machining and measuring part as we presented in section 1.1.5. Concepts are general entities, parameters are numeric values usually defining measure or construction settings.

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.

2.1.1 First stage

The core part of the machining process is a ternary relationship object — tool — NPV. [12] suggests introducing a new concept representing a relation itself. Figure 3 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

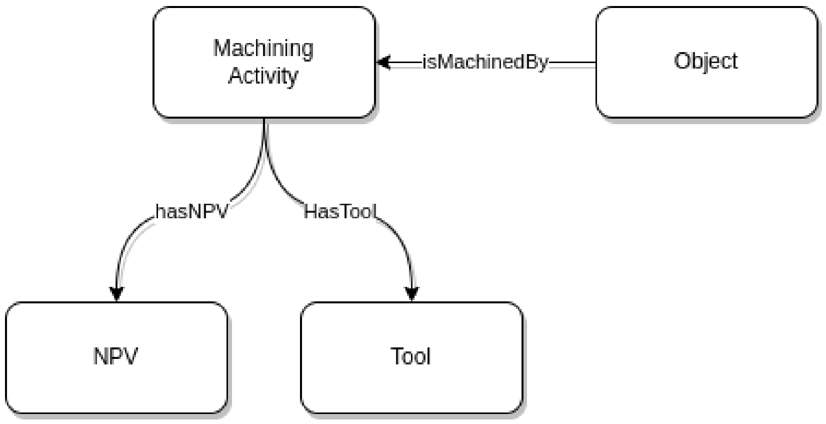


Figure 3: Core part of machining process

other elements or references), therefore, we decided to capture a general relations as shown in Fig. 4. There are four classes (Object, Characteristics, Element, Reference) and four object properties (isDerivedFromReference, isDerivedFromElement, consistsOfElement, hasElement).

The proposed part in Figure 4 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.

2.1.2 Second stage

The first stage (2.1) represents core structures for manufacturing and measuring processes. In the second stage, we define additional concepts representing the measuring process.

Figure 5 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. 6). MeasureSettings (Fig. 6a) is specified by two concepts (MeasureMethod, MeasureStrategy) and their object properties (hasMeasureMethod, hasMeasureStrategy). The rest are data properties (hasSpeed, hasStep, hasDesiredPointCount, hasActualPointCount, hasMeasureTime, hasAngleRange). ConstructionSettings (Fig. 6b) is specified by four concepts (FilteringMethod, FilterType, FilterSubtype, ConstructionMethod) and their object properties (hasFilteringMethod, hasFilterType, hasFilterSubtype, hasConstructionMethod). The rest are data properties (hasParameter, isEliminatingOutliers).

2.1.3 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 7 depicts two new concepts (Nest, MachiningTool) and two new object properties (isPartOfNest, isPartOfMachineTool). Similarly, MeasureSettings class is

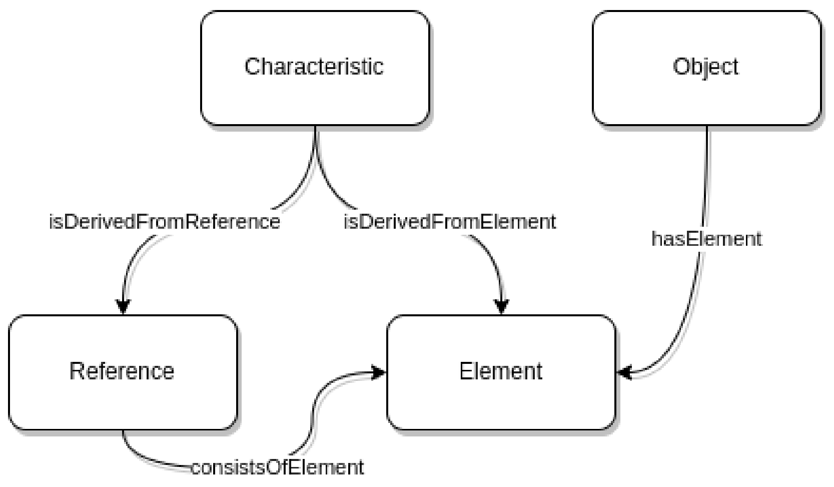


Figure 4: Relation between characteristic, element, reference, and object

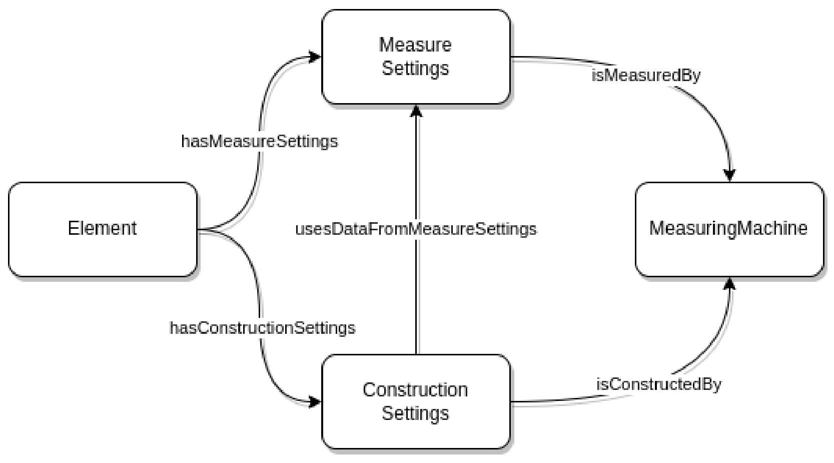


Figure 5: Construction and measure settings with a measuring machine

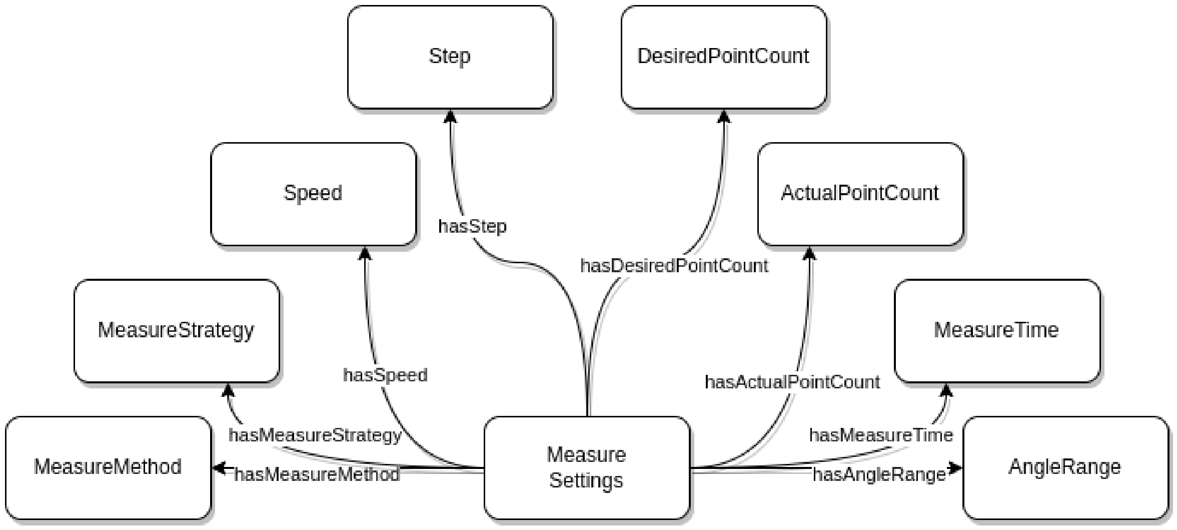


Figure 6: Parameters of measure and construction settings

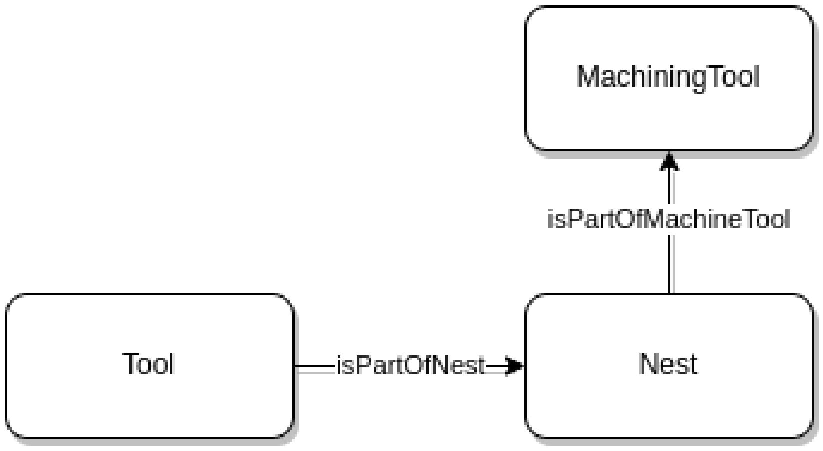


Figure 7: Tool, nest, and machine tool

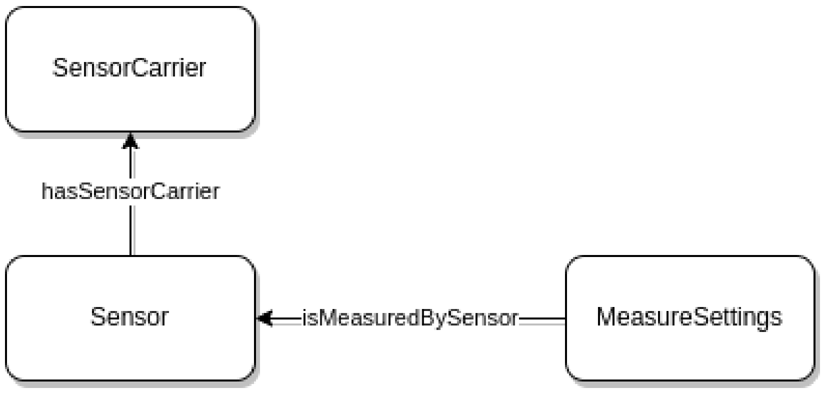


Figure 8: Sensor and sensor carrier

extended by two new concepts (Sensor, SensorCarrier) and two new object properties (isMeasured-BySensor, hasSensorCarrier).

Finally, an overview of the entire ontology is illustrated in Fig. 9.

3 Implementation

This chapter presents an implementation stage of the work. We first introduce a Semantic web in section 3, Semantic web technologies are presented in section 3.1, ontology APIs used in the SkodaOnto are overviewed in section 3.2.4. Finally, section 3.3.2 presents the architecture and implementation of the application.

3.1 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 [1].

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.

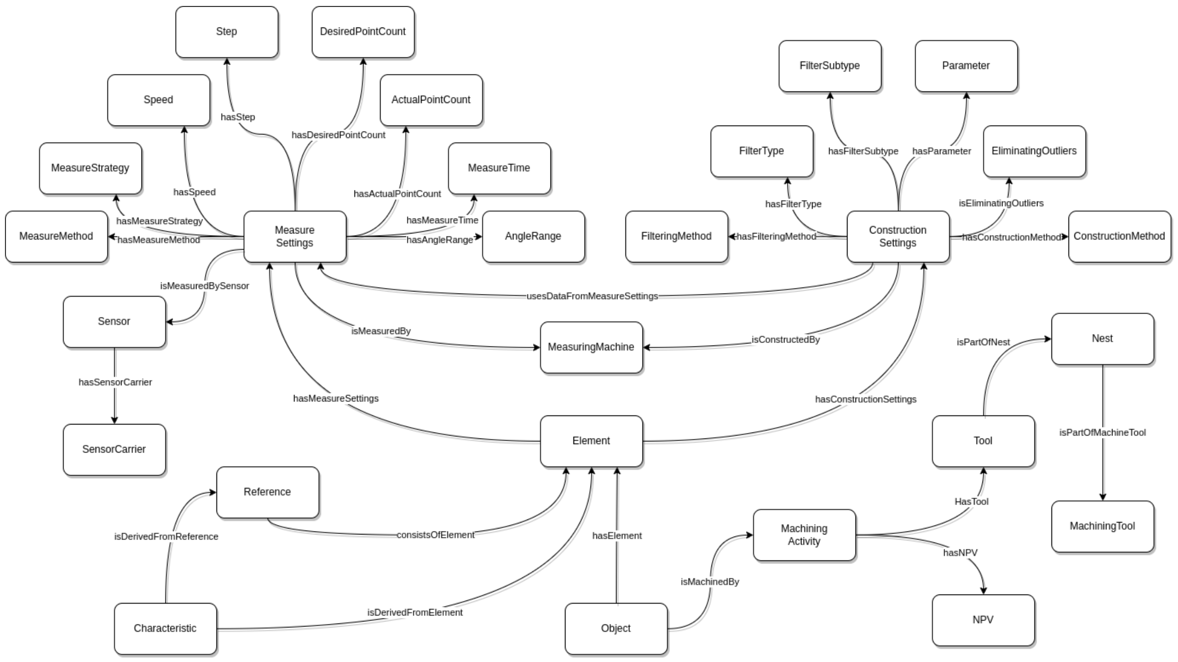


Figure 9: Overview of the ontology

3.2 Semantic web stack

As illustrated in Fig. 10, 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[4] standards and are most relevant to this work. The following subsections present these layers.

3.2.1 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, 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[5] characters. The blank node is a local identifier missing the RDF syntax. The literal is a value for representing strings, numbers, dates, etc [3].

[4]https://www.w3.org/
[5]https://home.unicode.org/about-unicode/

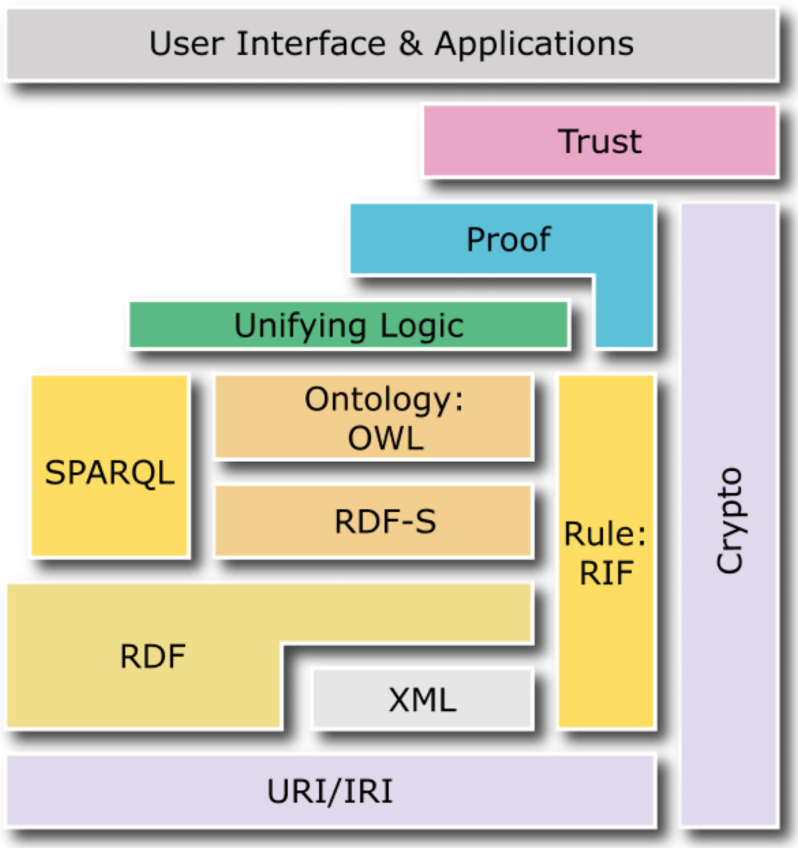


Figure 10: Architecture of semantic web

3.2.2 RDFS

The RDF Schema (RDFS) is a semantic extension of RDF (3.2) 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 [2].

3.2.3 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 (DL) 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 [7].

3.2.4 SPARQL

SPARQL Protocol and RDF Query Language (recursive acronym SPARQL) is a W3C recommendation for querying RDF data. As mentioned in subsection 3.2, RDF represents 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[6], XML[7], CSV[8], or TSV[9] 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 [4].

3.3 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

[6]`https://www.json.org/json-en.html`
[7]`https://www.w3.org/standards/xml/core`
[8]`https://en.wikipedia.org/wiki/Comma-separated_values`
[9]`https://en.wikipedia.org/wiki/Tab-separated_values`

to use two programming interfaces for reusability and further utilization of the SkodaOnto. The SkodaOnto TBox and ABox implementation are achieved via the OWL API [5]. Querying the data stored in the knowledge base is managed using the Owlready2 package [8].

3.3.1 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 [5].

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 (3.2.3) support and its high verbosity. The usage of OWL API is presented in section 3.3.2.

3.3.2 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 [8].

Advantages of choosing Owlready2 are: first, Owlready2 represents a simple and easy-to-use interface; second, the required SPARQL support; and third, since the ontology will be used often by Python modules, Owlready2 overcomes language incompatibility.

3.4 Architecture of the application

Figure 11 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 12 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

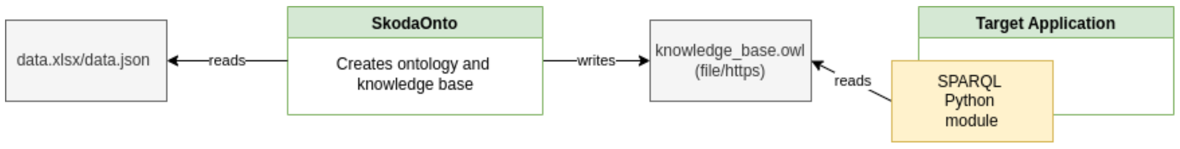


Figure 11: Usage of app

of individuals from the provided dataset (DataLoader class). The application outputs two files — `knowledge_base.owl` and `knowledge_base.ttl` representing our knowledge base.

3.4.1 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 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.

3.4.2 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[10]) constructed by joining several other tables and resources. For illustration, one of the resources is a TSV[11] file exported from a Zeiss Calypso[12] 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 (1.2). Thus, future versions of SkodaOnto will load individuals using a standard data exchange

[10]`https://en.wikipedia.org/wiki/Office_Open_XML`
[11]`https://en.wikipedia.org/wiki/Tab-separated_values`
[12]`https://www.zeiss.com/metrology/products/software/calypso-overview/calypso.html`

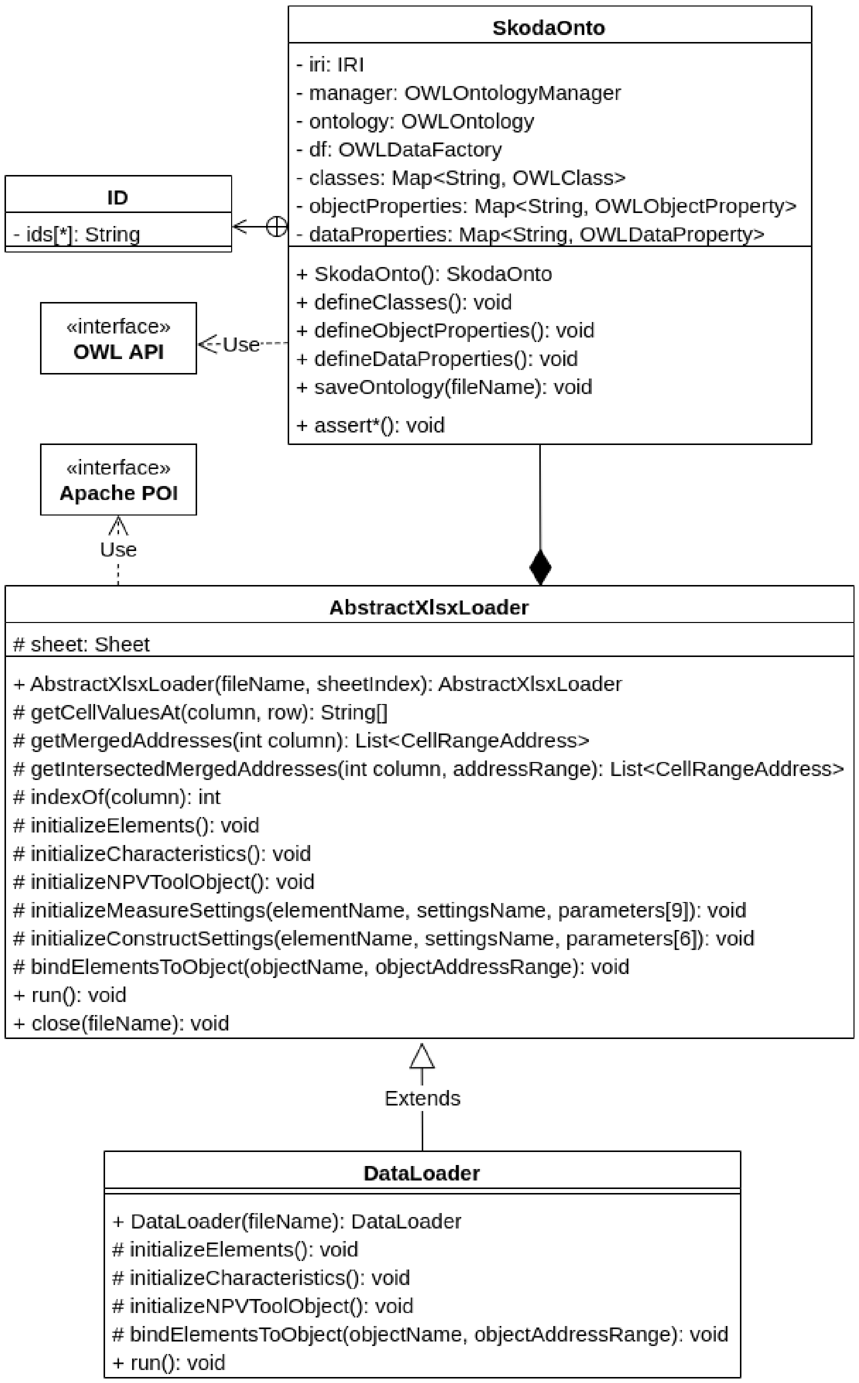
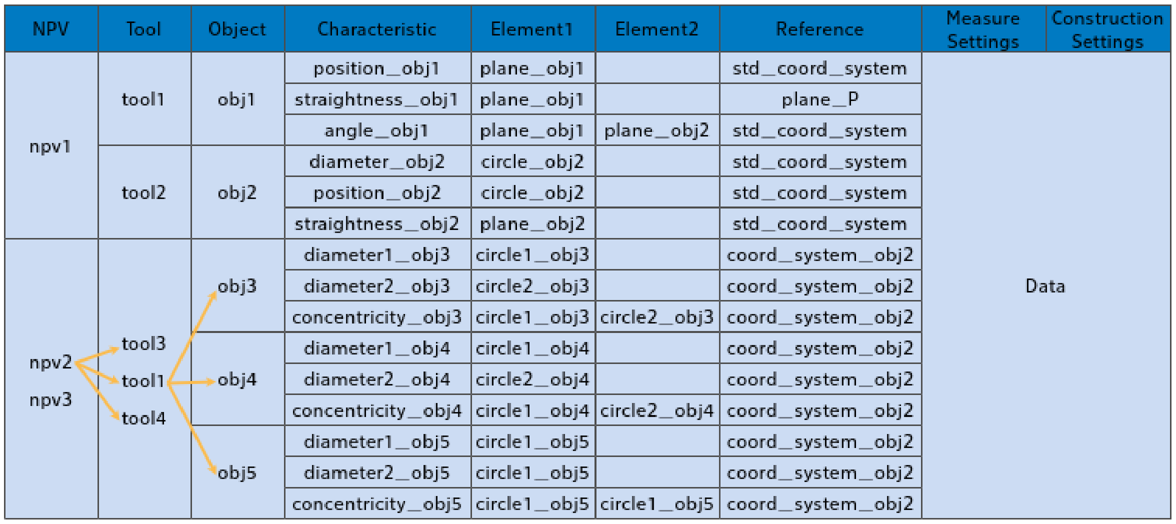

SkodaOnto
- iri: IRI
- manager: OWLOntologyManager
- ontology: OWLOntology
- df: OWLDataFactory
- classes: Map<String, OWLClass>
- objectProperties: Map<String, OWLObjectProperty>
- dataProperties: Map<String, OWLDataProperty>
+ SkodaOnto(): SkodaOnto
+ defineClasses(): void
+ defineObjectProperties(): void
+ defineDataProperties(): void
+ saveOntology(fileName): void
+ assert*(): void
ID
- ids[*]: String
«interface»
OWL API
Use
«interface»
Apache POI
Use
AbstractXlsxLoader
# sheet: Sheet
+ AbstractXlsxLoader(fileName, sheetIndex): AbstractXlsxLoader
# getCellValuesAt(column, row): String[]
# getMergedAddresses(int column): List<CellRangeAddress>
# getIntersectedMergedAddresses(int column, addressRange): List<CellRangeAddress>
# indexOf(column): int
# initializeElements(): void
# initializeCharacteristics(): void
# initializeNPVToolObject(): void
# initializeMeasureSettings(elementName, settingsName, parameters[9]): void
# initializeConstructSettings(elementName, settingsName, parameters[6]): void
# bindElementsToObject(objectName, objectAddressRange): void
+ run(): void
+ close(fileName): void
Extends
DataLoader
+ DataLoader(fileName): DataLoader
# initializeElements(): void
# initializeCharacteristics(): void
# initializeNPVToolObject(): void
# bindElementsToObject(objectName, objectAddressRange): void
+ run(): void

Figure 12: Class diagram



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 13: Simplified structure of the table

format JSON, as was promised by Škoda Auto. Figure 13 shows a simplified version of the final table. Unfortunately, real values are anonymized due to Škoda Auto's policy for internal data sharing. Figure 13 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.

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, 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[13] 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 2.1). Figure 13 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,and5` 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.

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

References

[1] 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*, 05 2001.

[2] Dan Brickley, R.V. Guha, and Brian McBride. Rdf schema 1.1 w3c recommendation 25 february 2014, Feb 2014.

[3] 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, Feb 2014.

[4] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. Sparql 1.1 query language w3c recommendation 21 march 2013, Mar 2013.

[5] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[6] Petr Juza. Formal representation of quality inspection process. Bachelor Thesis, 02 2023. Czech Technical University in Prague.

[7] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview w3c recommendation 10 february 2004, Feb 2004.

[8] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80, 08 2017.

[9] 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, editors, *Proceedings of the 13th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2021)*, pages 384–393, Cham, 2022. Springer International Publishing.

[10] Douglas C Montgomery. *Introduction to statistical quality control*. John Wiley & Sons, Chichester, England, 6 edition, July 2008.

[11] N. Noy and Deborah Mcguinness. Ontology development 101: A guide to creating your first ontology. *Knowledge Systems Laboratory*, 32, 01 2001.

[12] Natasha Noy and Alan Rector. Defining n-ary relations on the semantic web, Apr 2006.

[13] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. volume 12, pages 582–588, 01 1999.

[14] 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*, 20:538–544, 12 2020.