Semantic Web 15 (2024) 517–554
DOI 10.3233/SW-233342
IOS Press

Deriving semantic validation rules from industrial standards: An OPC UA study

Yashoda Saisree Bareedu [a,*,**], Thomas Frühwirth [b], Christoph Niedermeier [a], Marta Sabou [c,d], Gernot Steindl [b], Aparna Saisree Thuluva [a], Stefani Tsaneva [c,d] and Nilay Tufek Ozkaya [a]

[a] *Corporate Technology, Siemens AG, Munich, Germany*
E-mails: bysaisree@gmail.com, christoph.niedermeier@siemens.com, aparna.thuluva@siemens.com, nilay.tuefek-oezkaya@siemens.com
[b] *Institute of Computer Engineering, TU Wien, Vienna, Austria*
E-mails: thomas.fruehwirth@tuwien.ac.at, gernot.steindl@tuwien.ac.at
[c] *Institute of Information Systems Engineering, TU Wien, Vienna, Austria*
E-mails: marta.sabou@ifs.tuwien.ac.at, stefani.tsaneva@tuwien.ac.at
[d] *Institute for Data, Process and Knowledge Management, Vienna University of Economics and Business, Austria*
E-mails: marta.sabou@ifs.tuwien.ac.at, stefani.tsaneva@wu.ac.at

Editors: Bahar Aameri, University of Toronto, Canada; María Poveda-Villalón, Universidad Politécnica de Madrid, Spain; Emilio M. Sanfilippo, ISTC-CNR Laboratory for Applied Ontology, Italy; Walter Terkaj, STIIMA-CNR, Italy
Solicited reviews: Jürgen Bock, Technische Hochschule Ingolstadt, Germany; Gianfranco Modoni, STIIMA-CNR, Italy; one anonymous reviewer

Abstract. Industrial standards provide guidelines for data modeling to ensure interoperability between stakeholders of an industry branch (e.g., robotics). Most frequently, such guidelines are provided in an unstructured format (e.g., pdf documents) which hampers the automated validations of information objects (e.g., data models) that rely on such standards in terms of their compliance with the modeling constraints prescribed by the guidelines. This raises the risk of costly interoperability errors induced by the incorrect use of the standards. There is, therefore, an increased interest in automatic semantic validation of information objects based on industrial standards. In this paper we focus on an approach to semantic validation by formally representing the modeling constraints from unstructured documents as explicit, machine-actionable rules (to be then used for semantic validation) and (semi-)automatically extracting such rules from pdf documents. While our approach aims to be generically applicable, we exemplify an adaptation of the approach in the concrete context of the OPC UA industrial standard, given its large-scale adoption among important industrial stakeholders and the OPC UA internal efforts towards semantic validation. We conclude that (i) it is feasible to represent modeling constraints from the standard specifications as rules, which can be organized in a taxonomy and represented using Semantic Web technologies such as OWL and SPARQL; (ii) we could automatically identify modeling constraints in the specification documents by inspecting the tables ($P = 87\%$) and text of these documents (F1 up to 94%); (iii) the translation of the modeling constraints into formal rules could be fully automated when constraints were extracted from tables and required a Human-in-the-loop approach for constraints extracted from text.

Keywords: Semantic validation, information extraction, natural language processing, human-in-the-loop, OPC UA

[*] All authors have contributed with equal effort to the work and therefore are listed in alphabetical order.
[**] Corresponding author. E-mail: bysaisree@gmail.com.

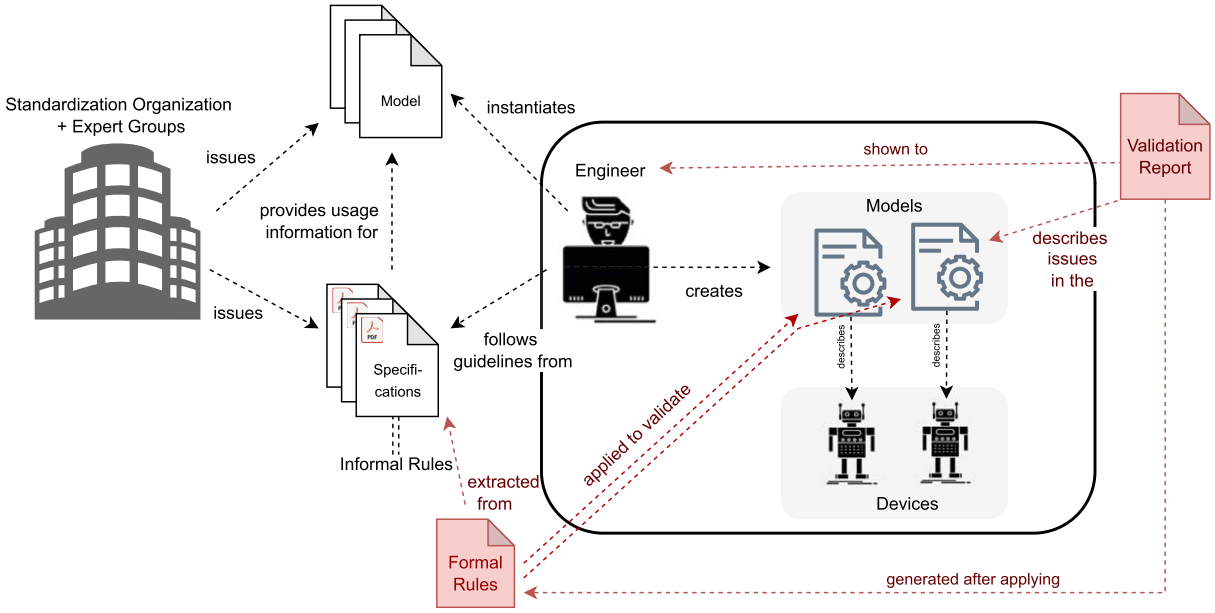


Fig. 1. Standard development, standard usage and the need for semantic validation (shown in red).

1. Introduction

Interoperability in the industry has been a research topic since the 1970s [23] and became even more relevant during the fourth industrial revolution, as Cyber-Physical Production Systems (CPPS) rely on networked manufacturing equipment that needs to be seamlessly integrated, often in run-time, dynamic workflows [10]. To ensure such interoperability, industrial standards are core to many industries and represent vital elements in the ecosystem of that industrial domain by providing shared guidelines for data representation and exchange. For example, the International Electrotechnical Commission's (IEC) Common Information Model (CIM) provides a data model for standardized information exchange of energy grid descriptions [42]. Another example is OPC Unified Architecture (OPC UA, https://opcfoundation.org/) – an industrial standard that ensures interoperability between industrial devices on the technical, syntactical, and semantic levels: while the transport protocols and payload formats are defined on the technical and syntactical levels, semantic interoperability is achieved through extensible information models, which capture domain-specific knowledge. OPC UA is becoming a very important and widely used standard across the industry. Leading global IoT vendors including AWS, Google Cloud, IBM, Microsoft, SAP, and Siemens are using OPC UA as the industrial standard. There are millions of industrial devices already supporting OPC UA. Therefore, in this paper, we focus on this standard for a use case study.

We distinguish two core stages in the life-cycle of an industrial standard, which we depict in Fig. 1 and discuss in the context of OPC UA. First, *standard development* is undertaken by an industrial standardization body. The outcomes of this process are formal (machine-readable) models accompanied by additional (human-readable) documents that describe how the models shall be used. Thereby, the human-readable document may include restrictions beyond the formal model's expressiveness. In the case of OPC UA, standard development is performed by the OPC Foundation which involves experts from the industry and academia around the world (left part of Fig. 1). The OPC Foundation has defined rich *domain independent* semantics for OPC UA information models captured in one of the *OPC UA base specifications*. Several domain-specific working groups are part of the OPC Foundations. Their role is to issue domain-specific information models, compliant with the meta-model defined in the base specification, in the form of *OPC UA companion specifications*, e.g., in the domains of Robotics, Machinery, etc.

A second key stage is the *standard usage*, when stakeholders in the domain of the standard create information objects that follow the guidelines proposed by the standard, thus achieving interoperability in that industrial domain. In the case of OPC UA, several manufacturers around the world rely on this standard to ensure the interoperability

of their devices with the devices from other manufacturers. To that end, *engineers* employed by these manufacturers need to write OPC UA information models of such devices. These models are serialized as structured, XML-based documents (also referred to as NodeSet files) capturing the structure and functionality of the device by following the OPC UA standard laid out in the base and (relevant) companion specifications (right side of Fig. 1).

Provided that the information objects fully comply with the guidelines of the standard, interoperability across stakeholders subscribing to the standard is achieved. However, the semantics defined by (most of) the standards are often only available as unstructured (pdf) documents. Currently, the assumption is that the *engineer* creating information models has a thorough understanding of the base/companion specifications and has correctly applied all pertinent guidelines described in several hundred pages. In practice, this is unrealistic and often leads to the incorrect application of the standard which triggers costly interoperability errors. There is, therefore, a need for *automatic semantic validation*, in order to easily and reliably check the compliance of an (OPC UA) information model with relevant standard specifications.

In this work, we address this common pattern in the landscape of industrial standards, with a focus on OPC UA. Automatic semantic validation is crucial in the concrete case of OPC UA. Besides its widespread use in industry to describe millions of industrial devices, the number of companion specifications is also significantly increasing year by year. As of 2021, there were 28 companion specifications. By 2022 36 companion specifications are available and many are under development. This situation prompted the OPC Foundation to initiate an OPC UA Semantic Validation Working Group in July 2019 with OPC UA and semantic experts as members with the primary goal to provide the foundation to create valid, consistent NodeSets. As a part of this goal, the OPC UA experts identified the modeling constraints in the OPC UA core specifications manually by reading the specification documents and understanding them. Semantic experts are working towards a formal representation of the constraints as formal, machine-actionable rules in order to validate them on the nodesets. However, this process of manually collecting the constraints and formalizing them as rules is a tedious and time-consuming activity. Motivated by this stringent need in the case of OPC UA, our work focuses exactly on addressing such scenarios in industrial standards.

Concretely, we propose an approach to semantic validation in which modeling guidelines available in non-structured specifications are translated into formally represented rules that are machine-actionable and can be used to automatically validate the correctness of the information models in terms of their compliance with the specifications (red elements in Fig. 1). We investigate the following research questions:

- *RQ1: To what extent can informal modeling guidelines be captured into formal rules?* Is it possible to identify such rules? Can they be organized in a taxonomy? What is the best way to represent them? If capturing informal guidelines into formal rules is possible, it is unfeasible to expect that this process will be performed manually by interested stakeholders, such as the specification authors. Therefore, this process should be supported as much as possible, as addressed by the next two RQs.
- *RQ2: To what extent can modeling guidelines be automatically identified in the specification documents?* How complex is the task of identifying modeling constraints in specifications? What methods are amenable for this task?
- *RQ3: To what extent can informal modeling guidelines be automatically mapped to formal rules?* What methods can be used to that end?

We answer these research questions through the following methodology leading to several contributions:

- We propose a *high-level approach* for the extraction of modeling constraints from specifications and their formal representation, e.g., as SPARQL rules. The proposed technical solution has several core components: (1) a catalog of major rule types and their corresponding representations as SPARQL query (templates); (2) a component to automatically identify constraints in specification documents; (3) methods for generating rules from textual constraints identified in stage (2) according to templates proposed in stage (1).
- We apply the steps of this approach in the concrete case of OPC UA specifications, resulting in contributions such as: (a) an *OPC UA specific rule catalog and classification* including the corresponding SPARQL rules; (b) *OPC UA specific semi-automatic methods* for extracting rules from pdf documentations.
- We evaluate the performance of the technical components individually. Then we provide a feasibility evaluation of the proposed method for one concrete OPC UA specification in the Machinery domain and perform an evaluation campaign involving Machinery working group experts.

We continue by providing further motivation for our work and background related to OPC UA in Section 2. In Section 3 we describe the proposed high-level approach that can be potentially applied to other standard documents. The following sections investigate our research questions within the context of our use case in this paper, which is OPC UA, including the rule taxonomy (Section 4), methods for automated modeling constraint extraction (Section 5), and methods for generating formal rules from modeling constraints (Section 6). These individual components are evaluated in Section 7. We discuss related work and concluding remarks in Sections 8 and 9 respectively.

2. Background and motivation

2.1. Background: OPC UA

Basic OPC UA notions OPC UA is a framework for industrial communication that additionally provides information modeling capability. The communication of OPC UA is based on the client/server principle, but part 14 of the specification also introduces a publish/subscribe communication paradigm.

OPC UA information modeling provides structure and context to the data of a production facility. It facilitates the description of devices, such as sensors, actuators, as well as whole production machines, in an object-oriented and semantically meaningful way [24].

The basic elements of the information model are: (1) nodes that represent objects, variables, methods, etc., and (2) references which are used to model relations between nodes. Eight different node classes are defined by OPC UA: *ObjectType*, *Object*, *DataType*, *VariableType*, *Variable*, *ReferenceType*, *Method*, and *View*. Depending on the node class, a set of attributes is defined for each node. One of the most important attributes, which is supported by every node class, is the *NodeId*. The *NodeId* is used to unambiguously identify each node by a so-called *NamespaceIndex* and an *Identifier*. Some other attributes are the *NodeClass* itself, the *DisplayName* (a human-readable name for the node), *Description* (a human-readable description of the purpose of the node), maybe a *Value*, and many more. Nodes can be instantiated based on the defined *ObjectTypes* or *VariableTypes*, similarly to object-oriented programming. These objects, variables, and methods are called *instance Nodes*.

The OPC UA information model is extensible, and it is used by domain experts for certain domains (e.g., machinery) to create Companion Specifications. The experts agree upon the modeling and release it as an industry-standard model, which can be used free of charge by any other stakeholder in that domain, e.g., other machine vendors. Thus, these OPC UA Companion Specifications facilitate interoperability at the semantic level.

Figure 2 shows an example of an information model from the Machinery Companion Specification using the graphical notation defined in the OPC UA base specification part 3. The abstract *ObjectType MachineryItemIdentificationType* has two variables *ManufacturerUri* and *YearOfConstruction*. These variables are related to the *ObjectType* via a *HasProperty* reference. The *HasSubType* reference is used to specify another *ObjectType* called *MachineIdentificationType* which inherits the properties from *MachineryItemIdentificationType*. As this *ObjectType*

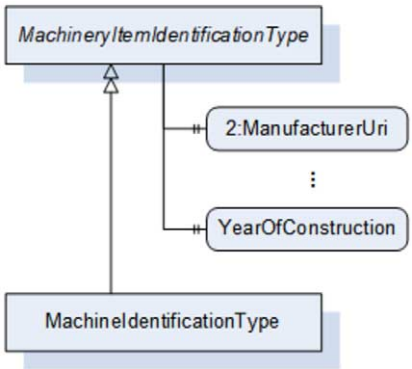


Fig. 2. Part of the OPC UA information model as it is defined in the machinery companion specification.

is not defined as abstract, instances of this type (Objects) can be created. Even if no instance is shown in this example, and not every aspect of the model is explained, it provides insight into the OPC UA information modeling concepts. More details can be found in [25].

OPC UA specifies an XML Schema, which defines the information model as an XML serialization. Such XML files are called *OPC UA NodeSet files* and can be generated with both open-source and commercial tools. The NodeSet file can be loaded and instantiated by an OPC UA server to make the information available to other clients. This exposed information model is the server's *address space*.

Need for semantic validation Even if OPC UA provides sophisticated information modeling possibilities, it lacks the ability to define restrictions on the model. Such restrictions could be beneficial to enforcing semantic interoperability when applying the information model in a certain use case. Currently, constraints on the model have to be documented in text, using natural language. Thus, currently, they cannot be checked automatically. As an example, the Machinery companion specification states, that the property variable *YearOfConstruction*, as shown in Fig. 2, shall be a four-digit number, such as "2022" or "2023". However, this constraint is only specified in the companion specification (textual) and not in the information model. Thus, nothing prevents a machine manufacturer from *incorrectly* using the information model as defined in the companion specification and assigning the value "22" instead of "2022". The problem occurs when a system in the factory tries to automatically schedule maintenance actions based on the age of the machine. The system will not be able to interpret "22" as the year "2022" and will fail. Other examples of information that currently cannot be modeled in OPC UA are exact multiplicities (e.g., an object shall reference exactly *n* objects of a specific type), or limitations on the structural model (e.g., a folder shall contain only objects of a set of specific types). This shows that the semantic validation of machine information models against the standard is crucial to ensure interoperability between machines and thus avoid costly errors and malfunctioning due to the incorrect use of the standard.

Such errors could be avoided by ensuring that the NodeSet correctly follows the specification guideline. This "validation" process is currently the task of the engineer that creates such NodeSets and it is unrealistic since the OPC UA base specification itself contains about one thousand pages and the number of companion specifications is rapidly increasing yearly to address the demand for ensuring interoperability between machines from different domains.

There is therefore a need for a *paradigm shift* towards automated, semantic validation of OPC UA information models. To that end, the OPC Foundation already investigates the possibility of incorporating rules in the OPC UA NodeSet files in the future. Figure 3 depicts how this could be modeled in OPC UA in a very abstract way using the graphical OPC UA notation. The exact implementation is still subject to debate and not the focus of this paper. However, the figure illustrates the basic idea, which is specifying rules for individual nodes within the OPC UA information model. Generally speaking, these rules express restrictions on the node or instances of the node.

To enable such a paradigm shift within OPC UA motivated our work towards a (semi-)automated approach for extracting the constraints from the specifications, organizing them into a rule taxonomy, and formalizing them in a formal query language. Thereby, a transformation from OPC UA NodeSets to OWL [37] enables the use of Semantic Web technologies, e.g., SPARQL for rule specifications (a technical realization of such a validation process is reported in Section 7.2).

One or more OPC UA Nodesets are transformed into an OWL ontology by following the transformation rules defined in [37]. The transformation is done by mapping each OPC UA node, its attributes and references to the pertinent OWL elements, namely OWL class, property (ObjectProperty, DataTypeProperty, AnnotationProperty), OWL named individual, etc. As an example, the result of the transformation process applied to the excerpt of the

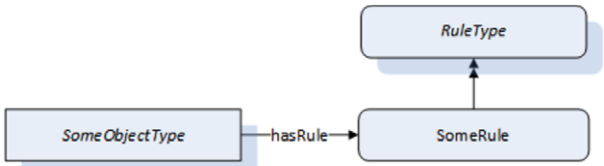


Fig. 3. Rules defined in the OPC UA information model.

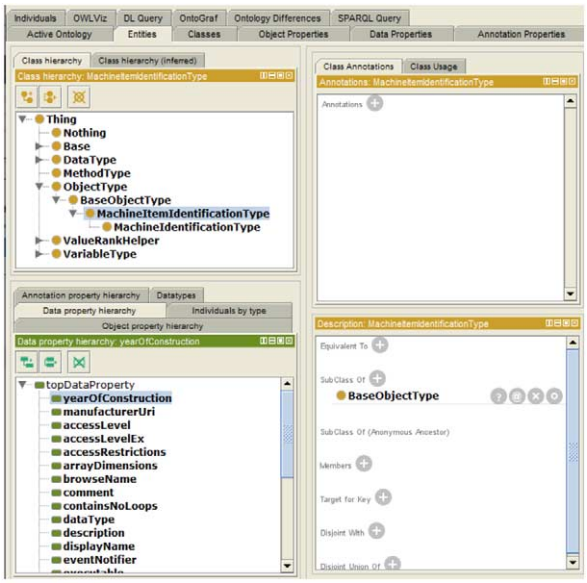


Fig. 4. Part of the OPC UA information model as it is defined in the machinery companion specification transformed to OWL.

Machinery Companion Specification from Fig. 2 is illustrated in Fig. 4. This OWL ontology can then be loaded into a SPARQL server, such as Apache Jena Fuseki, and queried via SPARQL.

2.2. *Motivation: Stakeholders use case*

Several stakeholder groups can benefit from our work that supports this paradigm shift towards semantic validation. Firstly, *users of the Companion Specifications, e.g., a machine vendor* will be able to check the conformance of the created address space of, e.g., a machine with the applied Companion Specification. As a result, *plant operators* can rely on conformance with companion specifications and simplify integration procedures.

Secondly, *Companion Specification creators* (i.e., members of the OPC UA working groups) will be able to check consistency between their companion specification and NodeSet files created in terms of this specification. They will also be able to provide a mechanism to enforce semantic interoperability (by providing formal rules in addition to the pdf-based companion specification). Capturing the informal specification of standards in formal rules will make sure that the standards are correctly applied, thus supporting interoperability. This stakeholder group greatly benefits from the automated support for the process of creating formal rules. The generated rules can, furthermore, be supplemented with additional, manually formulated rules that are either not present in the pdf document or could not be extracted automatically.

On a long term, work on formal verification of compliance with standard specifications addresses the needs of all stakeholder groups. This paper primarily focuses on supporting *companion specification creators* in turning their specifications into formal rules. Our goal is that OPC UA experts shall, in the future, use the results and tools described in this paper and apply them on existing and new companion specifications. This will provide them with an extensive set of automatically generated rules, which can then be extended as necessary.

3. Overall approach: Deriving rules from industrial standard specifications

We propose a high-level approach to support translating modeling guidelines expressed in specification documents into formal, machine-actionable rules that can be used for semantic validation. To that end, we introduce the following terminology:

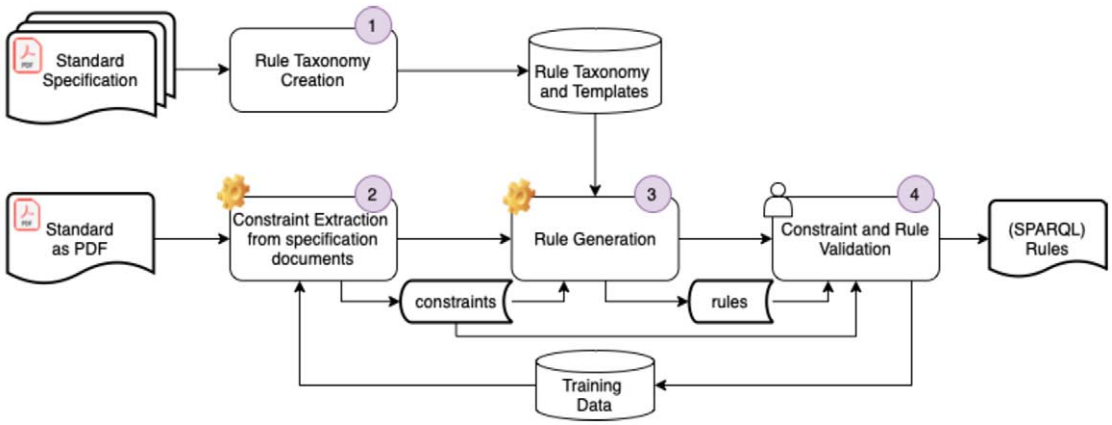


Fig. 5. Overall approach for deriving rules from specifications of industrial standards.

- *Constraints* represent concrete snippets in the standard specifications that express modeling constraints which are candidates for being formalized into rules. Modeling constraints can be specified either in an unstructured (textual) or semi-structured (tabular, list) form. Constraints are not machine-actionable but have an *informal* character to the reader of the specification. For example in tables, the terms *WidgetType, Reference names, NodeClass, TypeDefintion*, etc., signify the constraints allowed with respect to *WidgetType* node.
- *Rules* are used to express constraints. Rules may be expressed in different languages. We propose the term *semi-formal rules* for rules that follow a specific structure but are not intended to be machine-actionable (e.g., Semi-Formal Notation (SFN) used in Section 4.3), and the term *formal rules* for rules that are structured, formally represented and machine-actionable information (e.g., rules represented in SPARQL, SHACL). Thus, formal rules can be used on the information models in order to verify that they comply with the constraints expressed in the specification. Example rules generated from the constraint information available in tables are: *WidgetType IsAbstract False*, *WidgetType HasProperty Color which should be of datatype String*. Complex modeling constraints might require representation through multiple rules, known as *Rule Sets*.
- *Rule Templates* are generalized versions of rules that use template variables. Rules (both semi-formal and formal) are derived from templates by replacing the template variables with specific values. Rule templates are an important mechanism for supporting automatic rule extraction by being populated with automatically extracted values.

The overall approach for rule extraction encompasses several key stages as visualized in Fig. 5:

1. *Stage 1: Rule taxonomy creation*: a first stage is understanding whether modeling constraints described in a standard's specification are amenable to be represented as formal rules. If this is feasible, rules that can capture such constraints need to be identified, represented in a language with formal semantics that is machine-actionable (e.g., SPARQL) and organized in a taxonomy. Some modeling constraints are common across the standards, while some are specific to each standard. However, the representation of modeling constraints as (semi-) formal rules is highly specific to the standard.
2. *Stage 2: Constraint extraction* from specification documents, identifies those snippets in the specifications that contain information that should be represented as formal rules. Such information could be present through multiple modalities including textual descriptions, tables, or images. In principle, these different modalities can be used individually or in tandem to increase the performance of the extraction process.
   We propose here two methods: (1) one based on extracting constraints from tables in the specification and (2) focused on identifying language snippets that extract constraints. Several steps are involved in extracting modeling constraints from tables which include: (a) identifying the tables from specification documents; (b) the tables can be categorized into multiple table types based on the row and column headings of the tables which are the key terms that differentiate between multiple table types. However, the identification of table types is a domain-dependent task and it should be performed by identifying the key terms with respect to

each table type. These lists of key terms are standard and are applied in the table and constraint extraction algorithm that functions dynamically in extracting constraints from a target table from any companion specification document (from the tested 28 specifications) given as input to this algorithm.; (c) once the table types are identified, the modeling constraints specific to those tables can be extracted easily.

3. *Stage 3: Rule generation* represents making the transition from the modeling constraints extracted from the specification document (in Stage 2) into formally represented rules as identified in Stage 1. One or more rule templates are created for each rule type identified in the rule taxonomy, which can be used to automate this process when a clear mapping can be established between automatically extracted information from the standard specifications and the template variables (e.g., in case of generating the rules from constraints extracted from tables). If such a mapping is not possible, human intervention might be needed (e.g., this might be required for constraints extracted from text).
   This process requires a method for classifying modeling constraints to rule types: in our approach, the extracted constraints are first classified into several predefined rule types defined in the rule taxonomy. This can be done automatically for the constraints extracted from tables since the constraints are extracted based on table types; for constraints extracted from text human intervention is required. In this paper, we mainly focus on the classification of constraints from tables. Classification of constraints extracted from text is subject to future work.
4. *Stage 4: Constraint and rule validation.* Although the aim is to automate the constraint extraction and rule generation, automated methods rarely provide perfect results. Therefore, a concluding stage in this pipeline involves domain experts who validate (and if needed correct) the outputs of stages 2 and 3 above. With this process, ground truth data is collected from experts to inform the further development and extension of the existing methods.

The concrete techniques used to implement the stages of our approach highly depend on the characteristics of the standard at hand. Therefore, some of the steps described above are generic and can be applied across standards while others are specific to the standard under consideration. However, the overall pipeline can be potentially applied to standard documents from different domains by performing the domain-specific steps when required. In the next sections, we describe how we implemented these stages in the context of the OPC UA standard.

4. Stage 1: Rule taxonomy creation

The creation of a rule taxonomy and the corresponding rule templates addresses RQ1. The process involved the following four steps (see Fig. 6):

- *Step 1: Identify constraint types.* First, we evaluated whether the tables present in the OPC UA companion specifications represent modeling constraints with regard to the OPC UA information model. The core specification was not considered, because a preliminary analysis showed that it contains unique constraints and, therefore, it provides limited potential for identifying rule templates. The output of this step was the identification of various modeling constraint types, e.g., *ObjectTypeDefinition* constraint. Examples of identified constraints can be found in Section 5.
- *Step 2: Structure rule taxonomy.* As stated above, modeling constraints can be verified by checking one or often multiple rules. The different rules were structured in a taxonomy (see Section 4.2) but not yet formulated.
- *Step 3: Formulate rules in SFN/SPARQL.* An example for each rule within the taxonomy was formulated in SFN. In addition, some rules were also expressed in SPARQL, which allows them to be verified against existing OPC UA Node Sets after they are translated to OWL, as described in Section 2.1 (see Section 4.3).
- *Step 4: Create rule templates.* The constraint-specific information of each rule was replaced by template variables, resulting in *rule templates*. Finally, the rule templates were also organized in a taxonomy (see Section 4.3).

Additional information about each step and intermediate results are provided in the following sections.

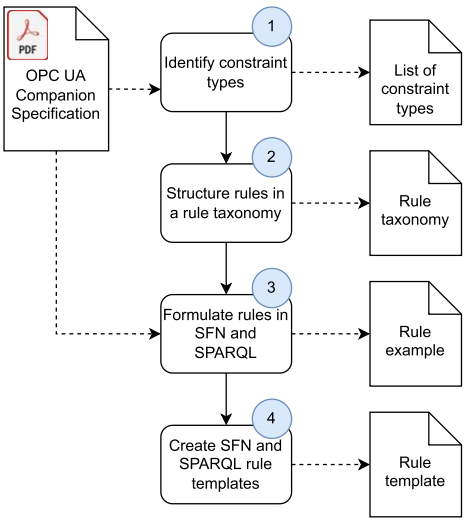


Fig. 6. Process for creating the rule taxonomy and templates.

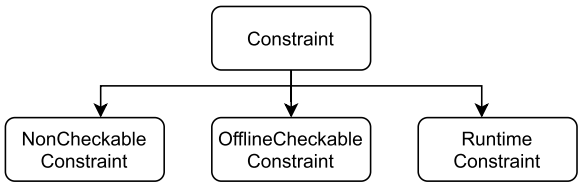


Fig. 7. Top-level constraint classes.

4.1. Step 1: Identify constraint types

Modeling constraint types identified during the analysis of OPC UA companion specifications were organized in a constraint taxonomy (Fig. 7). Thereby, constraint types were associated to one of three different constraint classes:

- *Non-Checkable Constraint* – The OPC UA specification contains information that could be interpreted as a modeling constraint but cannot actually be checked on the information model. For example, the specifications describe which of many OPC UA Status/Error codes should be used in specific circumstances. Nevertheless, it may be useful to also identify and document these Non-Checkable Constraints.
- *Runtime Constraint* – Some of the information provided by an OPC UA server is not present in the NodeSet file but is dynamically created when the server starts or while it is running.
- *Offline-Checkable Constraint* The main focus of this work is on Offline-Checkable Constraints that can be verified directly on the NodeSet, or on the OWL-transformed version of the NodeSet.

A modeling constraint can be expressed via one or multiple rules. Therefore, the next step is to define rules for each constraint.

4.2. Step 2: Structure rule taxonomy

Based on the modeling constraint types identified in Step 1, a number of individual rules were derived and subsequently classified into a rule taxonomy. The top-level hierarchy of this taxonomy (Fig. 8) distinguishes between (1) *Global Rules*, (2) *Node Rules*, and (3) *Type Rules* based on the scope of the rule application.

- *Global Rule* – A *Global Rule* is not associated with a specific node in the NodeSet but expresses some general rule to be fulfilled within the entire NodeSet (e.g., within the entire device description). For example, the

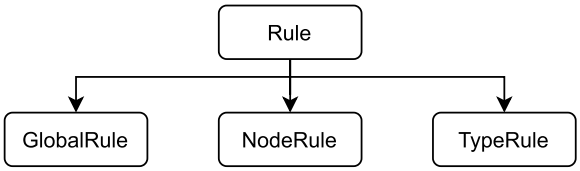


Fig. 8. Top-level rule classes.

following semi-formal rule expressed in SFN checks that the node MotionDeviceSystemType exists in the information model (see also Fig. 9).
Example in SFN: *The node MotionDeviceSystemType exists.*

- *Node Rule* – A Node Rule only applies to the node that the rule is associated with.
  Example in SFN: *The node MotionDeviceSystemType references a node MotionDevices. This node has ModellingRule Mandatory.*
- *Type Rule* – A Type Rule applies to all nodes derived from the type node that the rule is associated with.
  Example in SFN: *For all instances of MotionDeviceSystemType: The instance references exactly 1 MotionDevices node.*

These three basic top-level categories are refined, as shown in Fig. 28–30 (the Appendix), resulting in 3 *Global Rules*, 24 *Node Rules* and 18 *Type Rules*.

Currently, there exist only three different types of Global Rules (Fig. 28). The most important one is to check whether a specific node exists in the information model. The other two are concerned with the initialization of read-only and write variables. As they cannot be associated with a specific node, such rules are defined globally on the entire NodeSet file.

Restrictions regarding the attributes of a node, the references between them, and a referenced node have to be checked at the type level as well as on node instances of an OPC UA information model. Thus, these classes can be found in the taxonomy for *NodeRules* and *TypeRules*.

Additionally, Node Rules have restrictions on general *Data Type Structures* and *Enumerations*. These rules and the further refinement of these rule classes are depicted in Fig. 30.

For *TypeRules*, an additional rule is needed to check that an abstract ObjectType is not instantiated as an Object (Fig. 29).

4.3. Step 3–4: Rule examples and templates

For each rule present in the taxonomy, an example was formulated in SFN and SPARQL using information from companion specifications. These rules were then generalized into rule templates by replacing constraint-specific information with variables.

We decided to use SFN and SPARQL in this example for several reasons. First, while SFN cannot be used for automatic verification, it provides a human-readable form of documentation describing the purpose of the rule. Second, after consulting with OPC UA experts, it was decided to use SPARQL as the first candidate for a formal rule specification as it has broad support in semantic web tools and OPC UA NodeSets can be translated into OWL [37] as shown in Section 2.1. And third, using SFN and SPARQL simultaneously demonstrates that the approach is not limited to one specific language but can be extended by other languages, e.g., SHACL or technologies from model-driven engineering.

The rules and rule templates are not presented as a whole here, but the following examples should provide an impression of general concepts and the general form of rules formulated in SFN and as SPARQL queries. In general, SPARQL rules produce an error message if the rule is violated, and otherwise, they return no result. The rule, presented in Fig. 9, checks the existence of a node *MotionDeviceSystemType* on the global, NodeSet file level. The SPARQL query results in a message if no such node could be found and returns no results otherwise.

The SFN and SPARQL *rule templates*, depicted in Fig. 10, are generalized versions of the rule which is depicted in Fig. 9. To derive these templates, constraint-specific information (i.e., the information provided by the contents of

```
SFN:

The node MotionDeviceSystemType exists.
SPARQL:

PREFIX ta: <http://opcfoundation.org/UA/Meta/TA/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?msg
WHERE {
  FILTER NOT EXISTS { ?s ta:browseName
"http://opcfoundation.org/UA/Robotics/MotionDeviceSystemType"^^xsd:anyURI }
  BIND(STR('Node http://opcfoundation.org/UA/Robotics/MotionDeviceSystemType
does not exist') as ?msg)
}
```

Fig. 9. SFN and SPARQL rule examples.

```
SFN:

The node @@BrowseName@@ exists.
SPARQL:

PREFIX ta: <http://opcfoundation.org/UA/Meta/TA/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?msg
WHERE {
  FILTER NOT EXISTS { ?s ta:browseName "@@BrowseName@@"^^xsd:anyURI }
  BIND(STR('Node @@BrowseName@@ does not exist') as ?msg)
}
```

Fig. 10. SFN and SPARQL rule template examples.

the table cells) of the rules created in the previous step was replaced with template variables. The examples depicted in Fig. 9 verify that a node with a specific browse name, provided via the template variable @@BrowseName@@, exists in the ontology representing the NodeSet file. Such templates are important because they can be used as a basis for generating concrete rules by replacing the template variables with concrete values.

The process of formulating rules and, consequently, rule templates is a trade-off between complexity and re-usability. Complex rules covering many aspects at once (e.g., the existence of a node, its data type, and its value) are possible but are less re-usable across multiple modeling constraint types and more rule templates would be required. We, therefore, opted to formulate more fine-grained rules checking each of the exemplified aspects separately.

5. Stage 2: Constraint extraction from specifications

To address RQ2, we investigated various methods to support the automatic detection of modeling constraints in OPC UA specifications. The OPC UA industrial standards, published in pdf format, convey information by relying on three modalities: (i) tables capturing in a structured way modeling guidelines pertinent for the corresponding domain; (ii) graphical charts depicting the components of the information model (see example in Fig. 2); (iii) textual descriptions explaining the information represented in tables and graphical charts. We investigate to which extent information related to modeling constraints can be extracted automatically from tables (Section 5.1) and text (Section 5.2) in OPC UA specification documents.

5.1. Constraint extraction from tables

We report an initial study to clarify whether tables are a good source for modeling constraint extraction (Section 5.1.1), the creation of a data set as a basis for this process (Section 5.1.2), as well as an approach for extracting modeling constraints from tables (Section 5.1.3).

5.1.1. Preparatory study: Which tables are useful for constraint extraction?

To extract all relevant modeling constraints from specification documents an understanding of the information structured in tables is needed and whether tables contain information that is relevant for the constraint extraction. Three of the authors performed an analysis of 543 tables from 10 different companion specifications and concluded that:

– tables are a rich and structured source of modeling constraints;

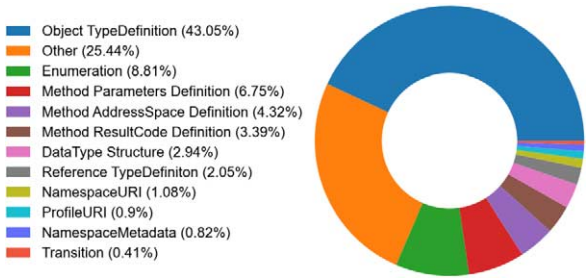


Fig. 11. Distribution of table types in 28 preselected OPC UA companion specifications.

– there are several *table types*, and each table type conveys specific OPC UA modeling constraint types. For instance, a *Reference TypeDefinition* table can include a symmetric modeling constraint and an inverse name modeling constraint while an *Object TypeDefinition* table contains among others cardinality and existence modeling constraints.

As part of the conducted table analysis, 42 table types are identified, 9 of which (e.g., *Example*, *Common-terms* table types, etc.) are considered irrelevant for the modeling constraint extraction task. Based on the number of instances (e.g., concrete tables) a table type has and the number of specification documents that used this table type, the top most frequent 11 table types are selected and used as input to the automated extraction process described in Section 5.1.3. These table types are: *Object TypeDefinition*, *Enumeration*, *Method Parameters Definition*, *Method AddressSpace Definition*, *Method ResultCode Definition*, *DataType Structure*, *Reference TypeDefinition*, *NamespaceURI*, *ProfileURI*, *NamespaceMetadata*, *Transition*.

5.1.2. Data set creation

For the task of automatically extracting modeling constraints from tables, 28 companion specification pdf documents are considered. Figure 11 visualizes the distribution of the selected table types in those 28 documents. It shows that *Object TypeDefinition* type tables appear most frequently and account for 43% of all tables in the inspected specifications. *Enumeration* type tables make up 9% percent of all the tables in these specifications. The tables from the 11 table types selected as most frequent during the analysis phase (Section 5.1.1) cover almost 75% of all occurring tables. The rest 25.5% of tables (shown as *Other* in Fig. 11) belong to table types that are very infrequent or are specific to only a single document.

5.1.3. Approach for constraint extraction from tables

In this section, we focus on the process of automatically extracting OPC UA modeling constraints from tables. Figure 12 shows an overview of the approach. In general, OPC UA Companion Specification documents have multimodal forms of data such as tables, texts, and flowcharts with constraint information. As this task focuses on the extraction of modeling constraints from tables firstly, all tables from a pdf document are extracted. From these extracted tables, target table types are identified and simultaneously cell-wise modeling constraint information in each concrete table is pinpointed, extracted, and formulated as a generated (semi-)formal rule. Each step of this process is explained next.

Step 1 – table detection and extraction This step takes as input a pdf Companion Specification document and returns a set of tables. It relies on *Camelot* (https://camelot-py.readthedocs.io), an open-source python library, specialized in the automatic detection and individual extraction of different tables from pdf documents. For the extracted tables a categorical separation of the required target table types is carried out as explained next.

Step 2 – table categorization In this step, the set of tables from the previous step is processed and each table is categorized into its corresponding table type.

We apply a heuristics-based approach that relies on making use of a filter based on two types of manually specified lists of key terms: (i) the first list contains terms that indicate that a table is of a certain type; and (ii) the second list contains terms that are not specific to that table type. These lists of key terms are domain-dependent. For instance, here the domain is OPC UA industrial standard and the key terms are column names of the table along with a few

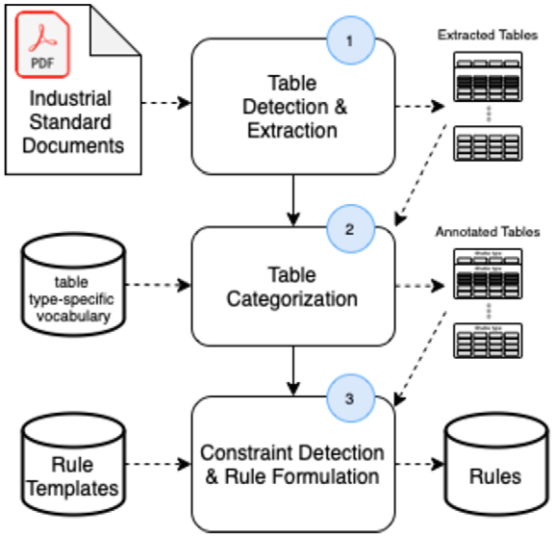


Fig. 12. Process of extracting constraints from tables.

Table 1
Number of strings for all considered table types

Table type	indicative strings	non-indicative strings
ObjectType Type Definition	8	25
ReferenceType	3	0
MethodAddressSpace Type	3	11
DataType Structure	10	33
EnumerationType	11	55
MethodResultCode Type	4	1
MethodParameter Type	1	4
Transition Type	3	15
NamespaceType	2	18
Profile URI Type	4	6

specifically occurring words in these tables, for example, IsAbstract, EnumType, InverseName, etc. These key term lists are constant for every companion specification within OPC UA whereas for other industrial standards these lists of key terms must be customized depending on their domain-specific vocabulary. Table 1 gives an overview of the number of indicative and non-indicative strings with respect to each type of table, applicable to all the Companion Specifications considered in this task.

Algorithm 1 describes the steps taken to decide which tables are of a certain table type. It takes as input all extracted tables (*tables*) as well as the lists of indicative and non-indicative terms for a given table type (*i_list* and *ni_list* respectively). For each table (*table*) and for all strings appearing in the table, the algorithm checks whether any string present in the table appears in the list of terms indicative of this table type, while not appearing in the list of non-specific terms for that table type. The algorithm outputs a vector of all tables that comply to the currently checked table type.

The 11 table types and the 10 pairs of lists of indicative and non-indicative keywords with respect to each of these types that we picked for this experiment, align with the companion specification template provided by the OPC UA Foundation. But as these key terms signified for one table type can also appear for other table types, further optimization of the keyword-based approach is required. Hence another list of non-key terms or non-indicative list of strings is also added with respect to each table type along with an indicative list of strings. Namespace metadata and Namespace URI type tables can be obtained by one common filter defined for finding Namespace type tables.

Algorithm 1 Step 2 table categorization into one table type

```
procedure SELECT(tables, i_list, ni_list)
    output ← []
    for table in tables do
        for any str in table do
            if str in i_list then
                if str not in ni_list then
                    output ← output + table
    output ← output.unique()
    return output
```

```
objtypdef = ["IsAbstract False True"," HasComponent
HasProperty Requires", "NodeClass Object  Variable",
"TypeDefinition", "ModellingRule MandatoryPlaceholder
OptionalPlaceholder", "OrganizedBy", "Organized by",
"Powerlink Attributes", "Details", "DisplayName", "Access
level", "ValueRank"]
nonobjtypedef = ["InverseName","Symmetric", "Subtype of
HierarchialReferences defined", "Argument[]",
"InputArguments","OutputArguments","Namespace","Conf
ormanceUnit","Conformance Unit","ToState","FromState",
"HasEffect","Notes –  Notes referencing footnotes of the
table content.", "NOTE  Notes referencing footnotes of the
table content.","SourceBrowsePath","Source Path"]

for table in tables:
if any(s in table.df.to_string() for s in objtypdef):
if not any(s in table.df.to_string() for s in nonobjtypedef):
```

Fig. 13. Filtering *Object TypeDefinition* table types by using lists of indicative/non-indicative strings.

Therefore, a total of 10 pairs of both indicative and non-indicative lists of key terms have been used to categorically filter required table types from all the tables extracted from pdf in Step 1.

Figure 13 shows an example of a filter used to identify whether a table is of type *Object TypeDefinition*. Strings such as "TypeDefinition" and "isAbstract False True" indicate that the table is of type *Object TypeDefinition*, while the keywords "ToState" and "FromState" refer to a *Transition* table and would thus classify a table as not-*ObjectTypeDefinition*. Using this technique, whenever we want to separate and identify a table type, the filter associated with respect to that table type is used to further filter out that table type resulting in the extraction of their exact types.

The main challenge in this step remains the inconsistencies in the appearance of a specific table type in the companion specification documents. Since not all tables are exactly compliant with the template defined by the OPC UA companion specification, a table could sometimes have different structural formats with new or varying column names, new positioning of the columns in the table, missing or misaligned rows and columns, as well as typographical errors. Therefore, the classification of the table types had to be further customized to increase the correctness of table extraction. Since only known inconsistencies observed among the same table type were considered, the results of the algorithm might vary when a new inconsistency appears. Hence, further optimization to achieve a fully accurate constraint information extraction from tables is considered as a future goal.

Step 3 – constraint identification & rule formulation Once the types of the extracted tables are known, for each table type a set of its respective type-specific modeling constraints are identified and formulated into (semi-)formal rules using rule templates. The constraints are identified in the following way (also sketched in Algorithm 2). Firstly, by looping iteratively over the rows throughout the length of the table, the required values present in the cells of each

Algorithm 2 Step 3 constraint identification and rule formulation

```
1: procedure SELECT(var1, var2, n, template)
2:     for var2 in range(n, length(var1)) do
3:         col1 ← value in (row,col) from target col1
4:         col2 ← value in (row,col) from target col2          ▷ Example of rule formulation
5:         "Node with" + col1 + " has a " + col2 + "."
6:         rule = populateTemplate(template, col1, col2))     ▷ For example, if col1 = Axis, col2 = variable
7:     return rule
```

identified table are extracted. Some variables are defined for every required column in the table for holding these values extracted from every table. Hence, an automatic filling of these values into respective variables takes place. For instance, a variable named 'Sourcenode' holds the 'Sourcenode BrowseName' value that appears in the cell at the 2nd row and 2nd column of ObjectType tables. Similarly, the 'References' variable holds all the 'Reference' values present in the Reference column starting from the cell at the 4th row, 1st column, with reference values in this column extending throughout the length of the table until the last row. In Algorithm 2, *var*1 holds a table from separated tables obtained from step 2, *var*2 holds the row index value of table present in *var*1 and *n* is the row index value from the starting row of actual values in the target column.

Subsequently, these variables that hold the modeling constraint information automatically extracted from tables are formulated as (semi-)formal rules by using pre-defined rule templates, finally providing the generated rules(rule templates are explained in detail in Section 4). This is feasible because the rule taxonomy provides a mapping between constraints and rule templates (as explained in Section 6 and shown in Fig. 17). This mapping suggests potential rule templates for the given constraint. However, only rule templates that can be fully populated (the values of all template variables required by the rule template have to be provided by the table) are selected and result in rules.

An example of a rule template is, '*The* ⟨*Sourcenode BrowseName*⟩ ⟨*Reference*⟩ ⟨*Targetnode BrowseName*⟩ *which should be of datatype* ⟨*Datatype*⟩'. By using the variables that carry the values extracted from cells of tables, we can formulate constraints as described in the following example, '*The WidgetType HasProperty Color which should be of datatype String*', using the concept of string concatenation. The variables are concatenated with strings in rule templates to formulate (semi-)formal rules corresponding to the modeling constraints captured in tables. In this way, from every identified and separated table type in step 2, constraints are extracted and generated as (semi-)formal rules in step 3 simultaneously. An evaluation of tabular constraint extraction is provided in Section 7.3.

Figure 14 shows how a (semi-)formal rule can be generated from the information extracted from tables. These (semi-)formal rules are generated in order to check the redundancy of rules described in tables and text in the companion specification documents, this is subject to future work. There is another output from the table extraction which is presented in Fig. 18. This figure shows that we can formalize the rules by populating the SPARQL templates with information extracted from tables. Therefore, the table extraction is used for two purposes and gives two outputs. Detailed description about populating the SPARQL templates is described in Section 6.1, 6.2.

5.2. Constraint extraction from text

OPC UA companion specification documents include large amounts of tables and figures. Nevertheless, it can be observed that the text surrounding the tables can extend the modeling constraints defined in the tables or introduce new modeling constraints. Therefore, the extraction of modeling constraints from text is an important task. While extracting modeling constraints from tables is a straightforward process, extractions from text require a well-trained algorithm and a human-in-the-loop for verification.

We investigate two different approaches- a machine-learning-based classification and a lexical-pattern-based extraction, introduced in the next sections.

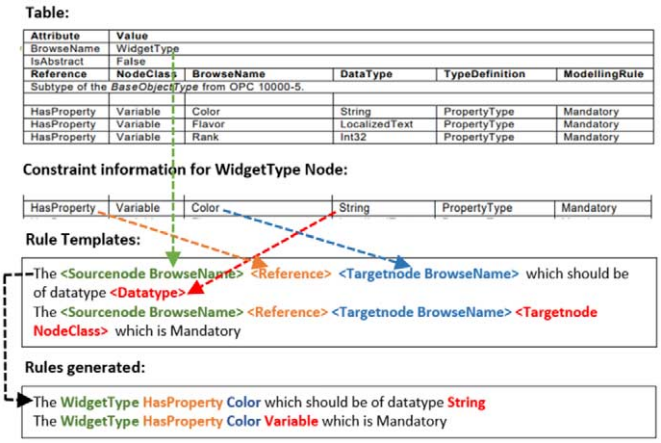


Fig. 14. *Constraint identification and rule formulation* from an Object TypeDefinition table.

Table 2
A sample of the gold standard data for the *PackML* companion specification

Text Snippet	isConstraint
UnitCurrentMode – is used to display the current mode of the instance of this type. The DataType is Enumeration which is abstract, but an instance shall be assigned a concrete enumeration, which corresponds to the enumeration listed in SupportedModes.	yes
EquipmentInterlock.Blocked – If TRUE, then processing is suspended because downstream equipment is unable to receive material (e.g. downstream buffer is full).	no

5.2.1. Machine learning (ML)-based approach

The first approach frames the modeling constraint identification as a binary classification problem. It involves training a classification model using labeled sentences and then classifying new text inputs as constraints or not-constraints.

Training set A prerequisite for this approach is the availability of an annotated data set for the training of the classification algorithm. For this purpose one companion specification (*PackML*) was analyzed by the authors of this paper (who are also OPC UA members) and modeling constraints were manually identified. The resulting training data set includes 198 text snippets which consist of either a single sentence or several sequential sentences extracted from the specification and their binary classification as a constraint (approx. 20% of the data) or not-constraint (80%). A sample of the data can be seen in Table 2.

Figure 15 shows an overview of the extraction process as discussed next.

Step 1 – model training By using the pre-annotated text snippets 8 *Scikit-learn* (https://scikit-learn.org) based machine learning models (*Nearest Neighbors, Linear Support-Vector Machine (SVM), Radial Basis function SVM (RBF SVM), Standard Gradient Descent (SGD), Decision Tree, Random Forest, Neural Network* and *Adaptive Boosting (AdaBoost)*) are trained to recognize different types of output – in our context, to differentiate between constraints and not-constraints. The annotated sentences from the PackML training data were split into Train and Test sets with a ratio of 4:1. When trained with enough example data, the models can then predict the type of new text inputs.

Step 2 – sentence extraction The next step is to extract all sentences from the pdf specification document from which the constraints should be derived.

Since the first chapters of each companion specification include general information about OPC UA, terms explanations as well as introductory examples, they are not considered for this task. For the extraction of the rest of the sections, *Spacy* (https://spacy.io) and *PyPDF2* (https://pypi.org/project/pyPdf/), Python libraries supporting various Natural Language Processing tasks such as information extraction, are used.

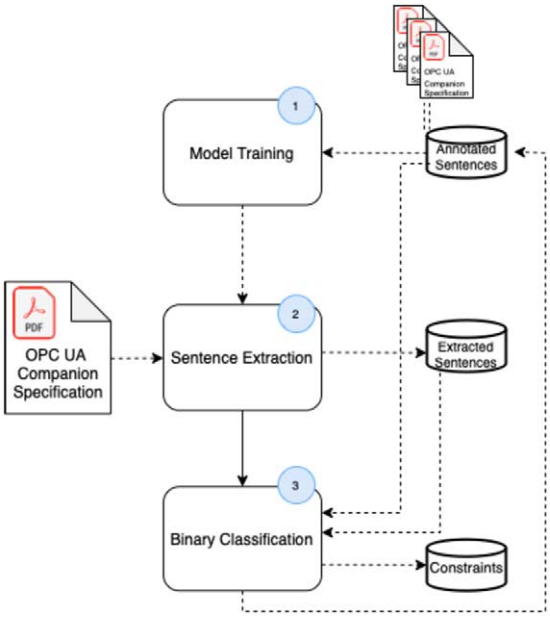


Fig. 15. Machine learning based process for extracting constraints from text.

Step 3 – binary classification After a set of sentences is extracted, each sentence needs to be categorized as constraint or not-constraint. First, the extracted sentences, as well as the training data, are pre-processed using stop word removing, lemmatization and tokenization. Second, each of the trained models is applied to the extracted sentences and their results are compared. The results are discussed in Section 7.4.1.

5.2.2. Lexical-pattern-based approach

In the English language, a sentence can be considered a rule if it conveys a semantic meaning of some function or some property or something that "has to be present" in some entity or needs to be necessarily followed. Such sentences have a syntactical structure that contains representative words such as *must*, *have to*, *should*, *will*, etc. Such words are called auxiliary verbs and play a key role in identifying if a statement is a "compulsory condition" and thus a constraint. This observation prompted us to experiment with a lexical-pattern-based constraint extraction, depicted in Fig. 16 and explained next.

Step 1 – linguistic patterns identification The first step of the approach is to analyze the structure of sentences likely to express modeling constraints. For instance, a modeling constraint could be expressed with an auxiliary verb in a lexical pattern structure such as *pronoun/noun/(proper noun) + auxiliary verb + verb*. An example is: "At least one instance of a MotionDeviceSystemType must be instantiated in the DeviceSet.". The word *MotionDeviceSystemType* is a proper noun and vocabulary belonging to the OPC UA context, *must be* is an auxiliary verb, and these words are followed by the verb *instantiated*. Subsequently, the following lexical patterns are formulated:

1. pattern1 = ['POS':'SCONJ','OP': '?', 'POS':'DET', 'POS': 'NOUN','OP': '?', 'POS': 'PROPN']
2. pattern2 = ['POS': 'PROPN', 'POS': 'NOUN','OP': '?', 'POS': 'PUNCT','OP': '?', 'POS': 'AUX', 'POS': 'PUNCT', 'OP': '?', 'POS': 'DET','OP': '?', 'POS': 'ADV', 'OP': '?', 'POS': 'VERB']

Here POS is a linguistic attribute that defines the parts of speech of the word in our pattern. Operators and quantifiers define how often the token must be matched. In the above pattern the operator OP: '?' makes the Parts of speech or POS token with the AUX or auxiliary verb matching, optional, by allowing to match 0 or 1 times. Similarly, PROPN – represents a proper noun, PUNCT – represents punctuation, DET – represents determiner, ADV – represents adverb, SCONJ – represents subordinating conjunction (e.g., *if, while, that*).

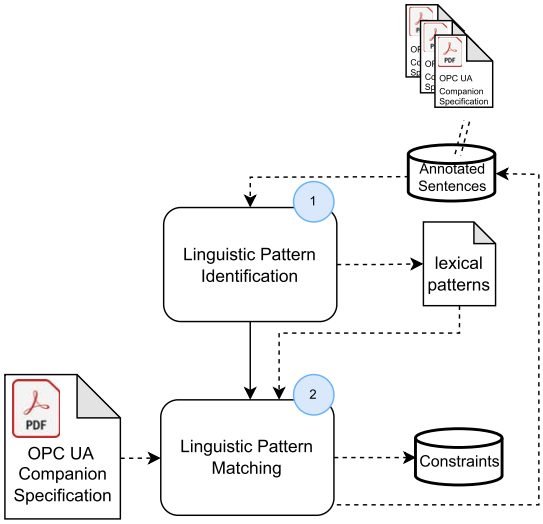


Fig. 16. Rule-based approach for extracting constraints from text.

Step 2 – linguistic patterns matching For this task, we use *Spacy* and its *Matcher* object, which allows for the specification of linguistic patterns and leads to extracting only sentences structured according to the preassigned pattern, which are likely to be modeling constraints. We evaluate this method in Section 7.4.2.

6. Stages 3&4: Rule generation and validation

Our third research question focuses on the feasibility of mapping modeling constraints specified in the tables and text of specification documents to rules. In the case of OPC UA, there is a strong link between table types and rule types, and rules can be generated automatically from tables as we describe in Section 6.1 (and based on the algorithm introduced in Section 5.1.3). As it is more difficult to connect textual constraints to (semi-)formal rules, we rely on a Human-in-the-loop approach to identify correct rules for modeling constraints extracted from text (Section 6.2).

6.1. Generating rules from tables

All tables found in the companion specifications are classified as NonCheckableConstraint, RuntimeConstraint, or OfflineCheckableConstraint (as discussed in Section 4.1). NonCheckableConstraints are tables with information that can not be checked on the information model. An example would be the description of parameters. Runtime-Constraints can only be checked on a running instance of an OPC UA server because the information is not directly available in a NodeSet file. Examples would be Server Profiles or Namespace URIs. Our work focuses on the so-called OfflineCheckableConstraints, which are concerned with information about the structure of the information model.

To document the constraint taxonomy, rule taxonomy, relations between constraints and rules, and rule templates, we created the ontology illustrated in Fig. 17. The specific snippet depicted illustrates the class structure (TBox) of the ontology and the rules that can be used to express and ObjectTypeDefinition constraint. Concrete constraints found in the specifications are instances of the corresponding constraint class. Likewise, necessary rules to express the constraint are instances of the corresponding rule class.

To *automate the generation of rules from tabular data*, we relate each one of an OfflineCheckableConstraint to a set of possible rules via an OWL class definition. If a table is found in a companion specification, the first step is to look up if the table represents an OfflineCheckableConstraint. If so, the information from the table is parsed (using techniques described in Section 5.1), and an applicable rule template (based on the constraint to rule mappings provided) is retrieved from the set. Afterward, the template variables from the rule template are populated with the

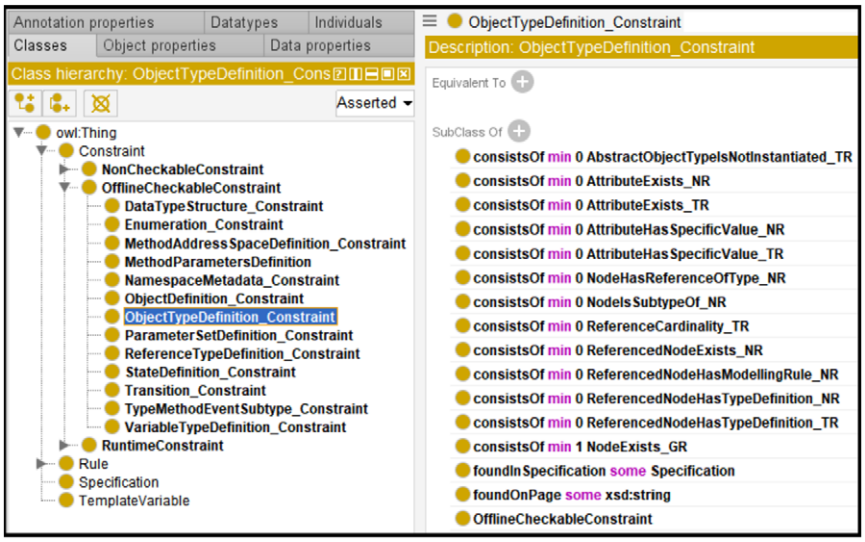


Fig. 17. Snippet of the Rule Taxonomy, which is an ontology describing the types of modeling constraints, types of rules as well as the mappings from constraint types to rule types.

retrieved information from the table (with the algorithm described in Section 5.1.3). This procedure is repeated until all necessary rules are created for the particular constraint.

As an example, 13 different rule templates are available to express an ObjectTypeDefinition constraint (Fig. 17). The needed information, e.g., that a node has a certain attribute value or the reference is of a certain type, is extracted from the tables and used to fill in the template variables in the related SPARQL rule templates. These completed rule templates can now be executed on the SPARQL endpoint, which has loaded the OPC UA information model. Violations of the rules are reported as error messages.

Figure 18 shows an example of this procedure. In the upper part of Fig. 18, a snippet of the ObjectTypeDefinition table is shown for the WidgetType, as defined in the Machinery companion specification. It shows the part of the table where it is stated that the WidgetType is not abstract. This is done by setting the value of the IsAbstract attribute to False.

Among others, the ObjectTypeDefinition constraint is related to the *AttributeHasSpecificValue_NR* rule template, which checks if a certain attribute has a specific value. In this case, it checks if the attribute IsAbstract has the value “False”. The template of the *AttributeHasSpecificValue_NR* rule is depicted in the lower part of Fig. 18. The template variables are now assigned with the values from the table to make the SPARQL rule executable. For this, the algorithm from Section 5.1.3 is used but with a SPARQL output. The SPARQL rule will lead to the error message if the WidgetType is defined as abstract in the OPC UA information model.

6.2. *Generating rules from text with human-in-the-loop*

While the types of modeling constraints expressed in certain table types can be mapped to suitable (semi-)formal rule templates thus enabling the automated generation of (semi-)formal rules from tabular data (Section 6.1), generating rules from text is a much harder process. As currently there is insufficient training data to train automatic classifiers for this task, we propose a *Human-in-the-Loop (HiL)* approach to map textual modeling constraints to (semi-)formal rules. The approach involves three tasks in order to (1) verify the correctness of textual modeling constraints to remove noise introduced by the automated extraction modules; (2) classify modeling constraints into categories linked to rule types; (3) validate the resulting rules. As such, these tasks also serve as an approach to implement *Stage 4: Constraint and Rule Validation* of the overall approach presented in this paper (see Section 3, Fig. 5).

We rely on Human Computation (HC) techniques for the HiL approach. HC implies outsourcing specific tasks of a system, which cannot be fully automated, to human participants and leveraging the human processing power

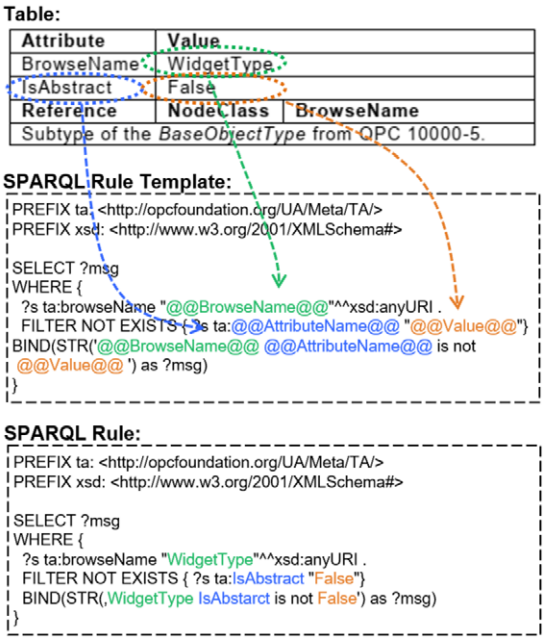


Fig. 18. Procedure of generating SPARQL rules from tables.

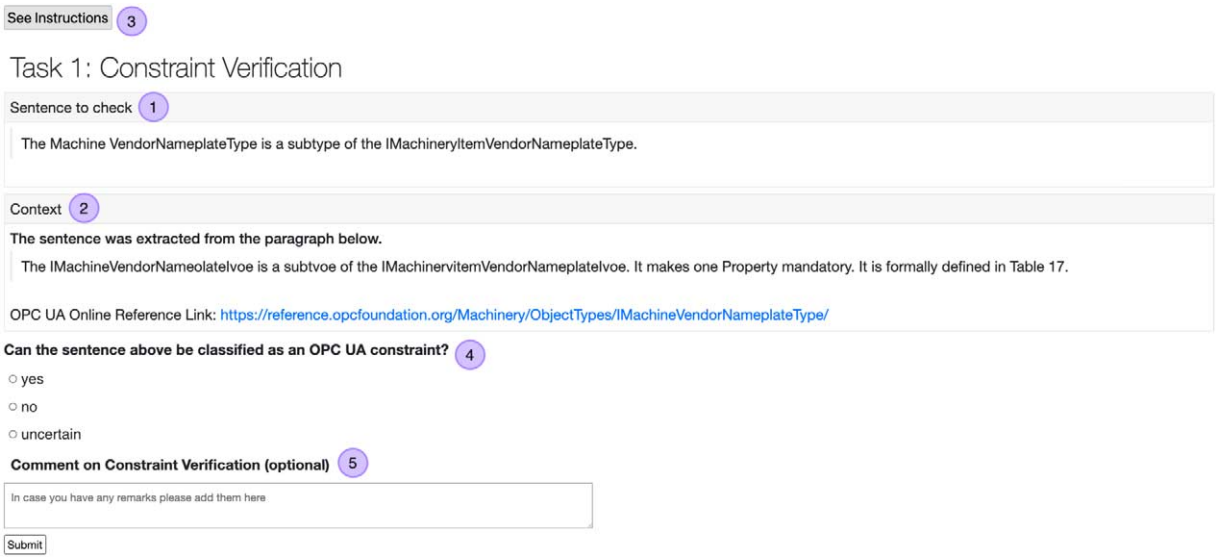


Fig. 19. HC task designed for constraint verification.

to solve those tasks. HC has been already successfully applied in a variety of domains for verification tasks [34] and we believe it is an important building block of a semi-automated method to support the steps needed for the derivation of (semi-)formal rules from pdf-based specifications. Next, the designed HC approach is explained and later in Section 7.5 a discussion of the results of an expert-sourcing validation campaign is presented.

Human Computation task design The HC solution consists of three tasks, which are designed to be completed by experts from the OPC UA working groups. In a first task, as shown in Fig. 19, the evaluators are asked to verify whether an extracted sentence (1 in Fig. 19) represents a modeling constraint. To provide enough background

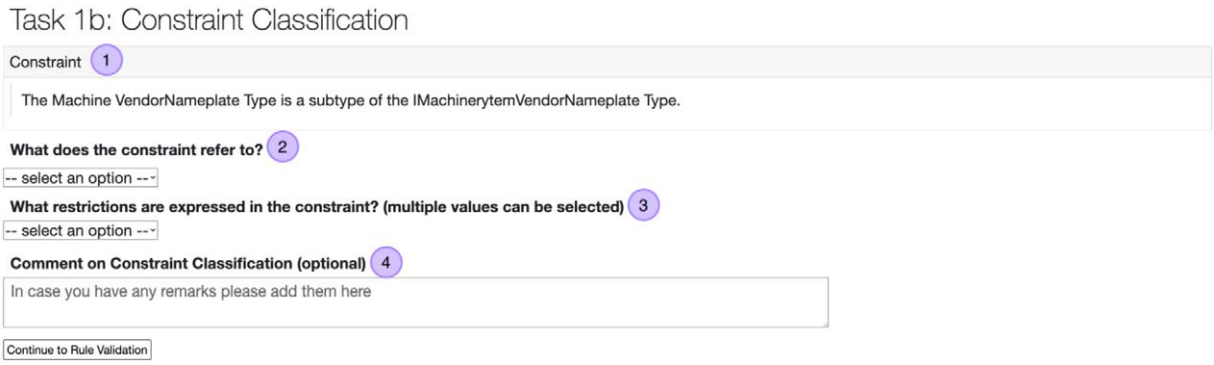


Fig. 20. HC task designed for constraint classification.

information for the decision the task includes additional context (2) such as the paragraph from which the sentence is extracted as well as relevant materials from the OPC UA Online Reference tool (https://reference.opcfoundation.org). To further support the experts, Instructions (3) are available in which nomenclature is explained and detailed task explanations with examples are provided. To allow for easy aggregation of responses provided by multiple experts, the task is designed as closed-ended (4). In case the evaluator is unsure whether the sentence refers to a modeling constraint they can select the *Uncertain* option. A comment field (5) is also available for remarks that the experts would like to share.

When the evaluator selects that the sentence is a modeling constraint, Task 1b, shown in Fig. 20, is displayed. The goal is to classify the constraint (1 in Fig. 20) into the rule type to which it refers to (2). For instance, the constraint can be related to an *Object TypeDefinition* or an *Enumeration*. To support the generation of rules the user is also asked what restrictions (e.g., cardinality) (3) are defined in the constraint. As with a previous task, a comment field (4) is available as well.

Based on the selection in Task 1b and the created Rule Taxonomy a set of rules can be generated which are to be used to validate that the constraint is adhered to and Task 2, shown in Fig. 21, follows. Here the participant's role is to validate whether the shown (semi-)formal rule (set) (2 in Fig. 21) correctly and completely reflects the constraint expressed in the extracted sentence (1).

The form in which the rules are shown is a variable and should be decided based on the evaluation population. In the shown example in Fig. 21 a structured natural language was selected, however, a formal rule language such as SPARQL could be used as well – the choice ultimately depends on the technical expertise of the participants. As with Task 1 Instructions (3) are available and a *Yes/No/Uncertain* answer (4) is expected. In case the person does not agree with the proposed rule (set) they are asked to select whether the rule (set) is *incorrect, incomplete, incorrect and incomplete* or *superfluous*. A comment filed (5) is included in case the evaluator wants to share some further insights regarding the rule.

While human effort is still needed for the constraint extraction and rule validation, by applying the designed HiL approach the manual effort of the experts is significantly reduced. Indeed, with this approach we manage to circumvent the efforts of manually inspecting the full specification document, extracting modeling constraints, and formulating (semi-)formal rules so that the experts are *only* required to perform a verification of the automatically produced results.

7. Evaluation

Since some methods presented above are still experimental, they are not all integrated in the current workflow of the proposed approach. Having said that, there is an automated core processing pipeline that consists of the rule taxonomy and the rule templates (Stage1) which inform the rule generation (Stage3) based on data extracted from the tables in the OPC UA specifications (Stage2). For the methods for modeling constraint extraction from text, a

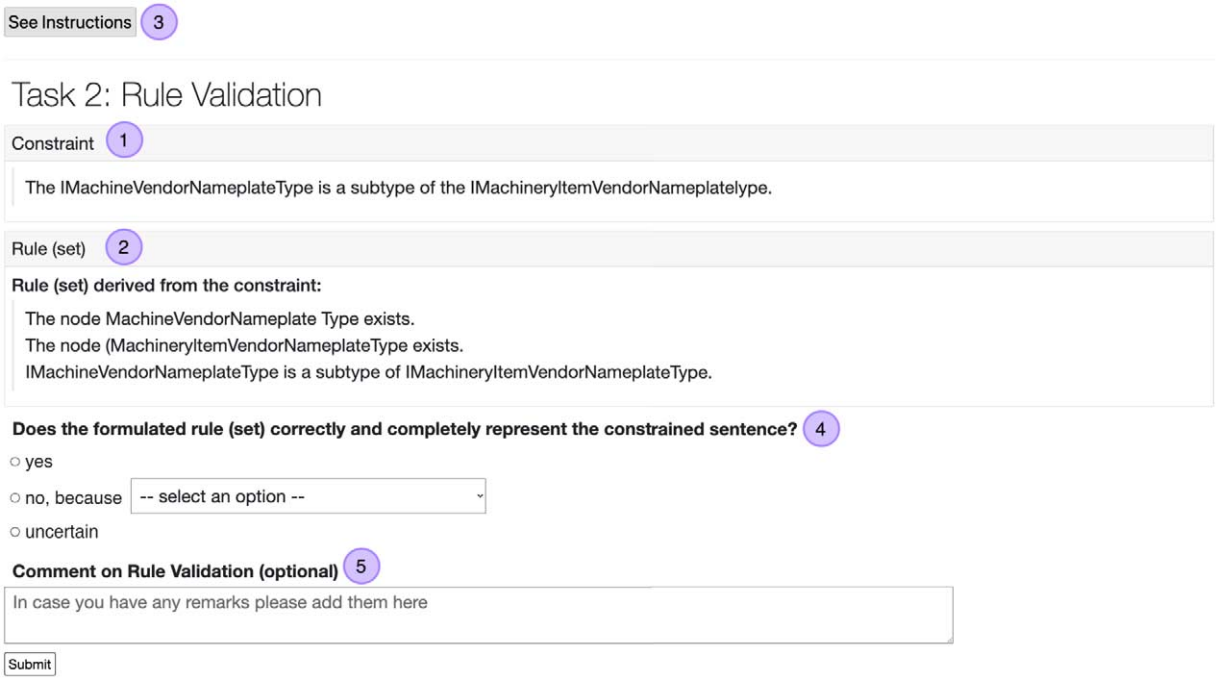


Fig. 21. HC task designed for rule validation.

complete automated workflow is not possible, therefore a HiL approach is chosen. Here modeling constraints identified in text (Stage2) are converted into semi-formal rules by mapping to the elements of the taxonomy developed in Stage1 through HiL methods proposed in Stage4.

Therefore, at this stage, evaluation primarily focused on assessing the performance and quality of individual elements of the proposed approach including: (i) the rule taxonomy and rules defined in Stage 1 (Section 7.1 and 7.2); (ii) the methods for extracting constraints from tables and text (Section 7.3 and 7.4) and (iii) the feasibility of HC tasks for constraint and rule validation as envisioned in Stage 4 by performing an evaluation campaign with OPC UA experts from the Machinery domain (Section 7.5). With such focused evaluations the goal was to understand the feasibility and behavior of the proposed individual elements across several specifications. An end-to-end evaluation of the entire approach is ongoing work and will focus on well-defined use cases from selected OPC UA specifications.

7.1. Evaluation of rule taxonomy

As described in Section 5.1, the rule taxonomy was created based on the analysis of the modeling constraints related to a type of *table* in a companion specification. Therefore, the question arose whether and to what extent this rule taxonomy would enable expressing constraints present in the *textual* part of the specifications.

To validate the coverage of the rule taxonomy, we manually identified the textual modeling constraints in the Machinery companion specification. Afterward, these constraints were analyzed to see if the already identified rules could also be used to express them. We found that the rule taxonomy can already cover the vast majority of these textual modeling constraints. Only two global rules had to be added, which are concerned with instantiating variables. This seems reasonable as this kind of information can hardly be expressed in a table and needs additional context.

We conclude that, although the rule taxonomy was created based on modeling constraints expressed in tables, it is sufficiently complete to also express constraints described in the textual part of the specifications. Having said that, the rule taxonomy is by no means a final collection of rules, but a core set of rules that can (and should) be extended as required.

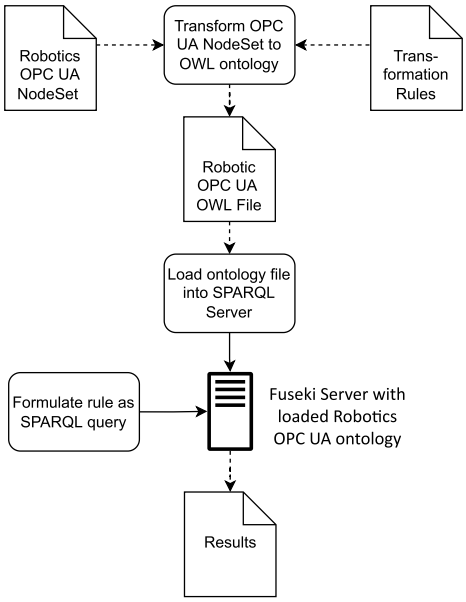


Fig. 22. Process for evaluating SPARQL rules.

7.2. Evaluation of SPARQL rules

Another important question to clarify was whether the SPARQL rules created as part of the rule taxonomy could be applied successfully on a concrete OPC UA information model.

To that end, in conformance with the Robotics companion specification, an information model was created, including instance nodes. These instance nodes are needed for evaluating some of the rules. Figure 22 illustrates the applied process. As SPARQL cannot be executed directly on the OPC UA NodeSet file, a transformation from an OPC UA NodeSet to an OWL ontology was required. More details about the OPC UA to OWL transformation can be found in [37] and Section 2. Afterward, the resulting ontology was loaded into a triplestore (Apache Jena Fuseki) to exemplify rule checking based on 12 different SPARQL rules. These rules were evaluated by checking the rules' outputs against the expected results. As the information model provides only a limited amount of possible ObjectTypes, Objects, Variables, etc., only a subset of the defined rules was implemented in SPARQL.

7.3. Evaluation of constraint extraction from tables

To assess the performance of the modeling constraint extraction from tables (described in Section 5.1), we compute: (i) True Positives (TP) which occurs when a table of a certain type (e.g., *Enumeration*) is extracted and classified correctly as its type; (ii) False Positives (FP) for tables of a specific type (e.g., *Reference TypeDefinition*) that are extracted and classified as a table of a different, incorrect type (e.g., *Enumeration*); (iii) Precision as formalized in Eq. (1). Due to the nature of the input sources and the implemented solution False Negative and True Negative situations arise very rarely and are not considered for the purpose of this task.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Since the constraints are formulated using the values obtained from the tables, it can be stated that once a table is extracted correctly, the constraints associated with it will also be properly extracted. In the majority of cases, the constraint extraction is directly proportional to the correctness of the table extraction as constraints are derived from the table values. Therefore, the main focus of the evaluation is on table extraction and categorization, also as a proxy for the correctness of constraint extraction.

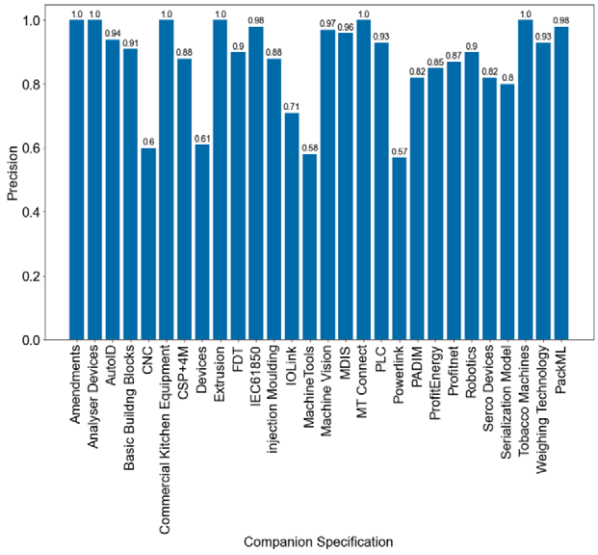


Fig. 23. Precision of table extraction per companion specification.

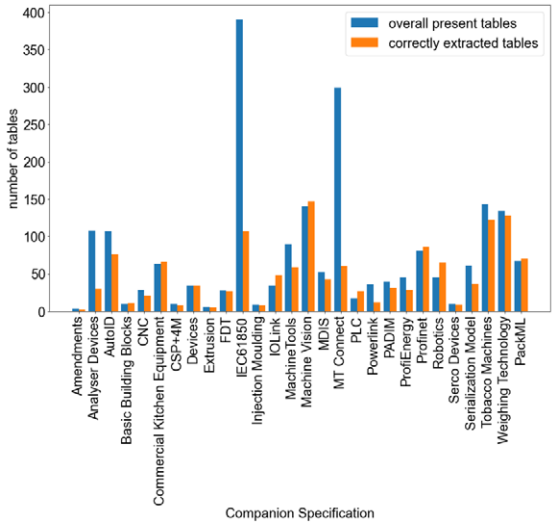


Fig. 24. Comparison between the total number of tables and the correctly extracted tables for each companion specification.

7.3.1. Table extraction and categorization

Figure 23 shows the precision of extraction from each of the 28 selected companion specifications, thereby resulting in an average precision of 0.87 and standard deviation of 0.134. As mentioned in Step 2 some customization of the algorithm is needed for documents that do not follow the guidance and template provided by OPC UA exactly. The more inconsistencies in the layout the lower the precision scores are. In Section 7.3.2 a more detailed error analysis is provided.

Figure 24 illustrates the Recall analysis. Recall can be formalized in terms of True Positive and False Negative as expressed in Eq. (2).

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

There are a total of 1998 target tables (tables of the types selected for the extraction) in the considered companion specification documents. With the used algorithms 1357 True Positive table extractions were possible thereby also leading to proper constraint extractions for those tables. This results in a recall of approx. 68% for the automatic extraction of tables and the corresponding modeling constraints captured in these tables. However, True Positive extractions are in some cases higher in numbers than the actual number of tables present. This can occur when a large table split into several pages is perceived as multiple tables by the extraction algorithm. Another case where a single table is extracted as multiple smaller tables is when there is irregular spacing or positioning of columns in the table. In some other scenarios, a single table is extracted twice because it fits the conditions of more than one table type. Such special scenarios that lead to some challenges or complexities in extraction are discussed in detail in the following section.

7.3.2. Error analysis

We hereby discuss the difficulties in the extraction of tables (and constraints) from pdf documents. Overall six different error types were identified which led to challenges or imperfections in extraction. Figure 25 shows an overview of the occurrence of those errors in the processing of each companion specification.

- *Error 1: Error due to fixed line-scale value.* The line-scale value is a feature in Camelot and plays a key role in the proper extraction of data in tabular format from a pdf document. It has the purpose of a line-size scaling factor. After testing different values on the companion specification, we empirically determined a line-scale value of 80. Nevertheless, the fixed value becomes an issue for tables on which the algorithm has not been trained and which are not defined in the OPC UA specification templates.
- *Error 2: Error due to the intersection of words in table-type-specific vocabulary lists.* In Step 2 from Section 5.1.3 the usage of table-type-specific vocabulary lists for table categorization is discussed. While in most cases the keywords inside the table are specific to a single table type, there are few cases where the same identification words are included in several table types. This causes some tables to be classified incorrectly (False Positive).
- *Error 3: Error due to page-breaks*. Page-breaks are observed in many companion specifications documents and become an issue for the extraction when a table is split into multiple pages. In cases where the table continuation does not include headings or column names, it does not get extracted and a loss of constraints can be observed. On the other hand, when the split table includes a heading on each page, this can result in the extraction of several tables instead of one, which is not an issue for the correct extraction of constraints, however, the performance metrics get affected.
- *Error 4: Error due to improper alignment of columns and table structures.* Improper alignment of columns inside the tables also leads to incorrect or missing constraint extractions even when the table extraction is correct. Such problems are mainly observed in *NamespaceMetadata* and *NamespaceURI* tables.
- *Error 5: Typographical errors or unexpected special characters.* Other factors that can negatively influence table extractions are typographical errors or misprints. Newline characters in unexpected positions in tables or incorrect column names also lead to errors in the extraction of constraints.

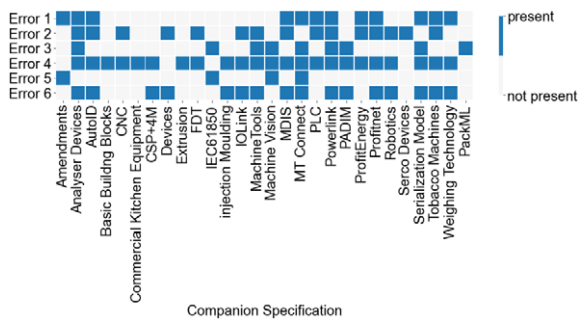


Fig. 25. Error types encountered during the processing of each companion specification.

– *Error 6: Error due to some unique structure of a table type that is not covered within the algorithm.* In a few cases no particular error in the algorithm was found causing the incorrect or missing extractions. A possible explanation could be the nonconformity of some tables to the standard companion specification template structures.

To conclude, the approach for the automatic extraction of tables and corresponding modeling constraints from OPC UA pdf specification documents achieves a Recall of 68% and an average Precision of 87%. Customizing the algorithm to the unique structures in different documents could result in better performance, however, it would also reduce the dynamic nature of the algorithm and would not improve extractions from future documents if they use incompatible structures. Moreover, the currently used Camelot software has some limitations as discussed above for extracting information from tables in textual documents. This software should be improved in order to increase the overall performance of information extraction from tables. On the other hand, the structuring of tables in the OPC UA documents and the correct usage of table templates provided by the OPC Foundation while creating the companion specification documents could further contribute to an increased quality of the information extraction algorithms.

7.4. Evaluation of constraint extraction from text

For the evaluation of the constraint extraction from text, one further companion specification document (*Machinery*) was selected, for which a gold standard was manually created. The data set was retrieved semi-automatically by using an automatic extraction method and the expertise of the authors for corrections and extensions of the extracted data. In total the gold standard data set includes 44 constraints and 229 non-constraints.

7.4.1. Machine learning (ML)-based approach

The eight supervised machine-learning-based models (Nearest Neighbors, Linear SVM, RBF SVM, SGD, Decision Tree, Random Forest, Neural Network, AdaBoost) were chosen because they are widely used scikit-learn library-based classifiers [28]. The configurations of the classifiers were default with small changes (see Table 3) and *TfidfVectorizer*[1] was used as feature extractor.

The machine-learning-based approach was first tested on the 20% of *PackML* gold-standard which was not used for the training of the algorithms. The models previously trained with 80% of the *PackML* gold-standard data are evaluated by testing the performance of each classifier in terms of precision, recall, F1 score, and accuracy. The results are shown in Table 4. Due to the imbalance of the data weighted average is used for the evaluation metrics. Results indicate that Neural Network (Multi-layer Perceptron) outperforms the rest of the models and produces the best results with a recall of 0.95.

Table 3
The configuration of Scikit-learn based binary classifiers

Classifier	Configuration
Nearest Neighbors	KNeighborsClassifier(10)
Linear SVM	SVC(kernel="linear", C=0.025)
RBF SVM	SVC(gamma=0.002, C=1000)
SGD	SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random_state=42,max_iter=5, tol=None)
Decision Tree	DecisionTreeClassifier(max_depth=5)
Random Forest	RandomForestClassifier(max_depth=5, n_estimators=10, max_features=10)
Neural Network	MLPClassifier(alpha=1, max_iter=1000)
Ada Boost	AdaBoostClassifier()

[1] https://scikit-learn.org/stable/modules/generated/sklearn.featurenew_extraction.text.TfidfVectorizer.html

Table 4
PackML test results in weighted average

Classifier	Precision	Recall	F1-score	Accuracy
Nearest Neighbors	0.93	0.93	0.93	0.93
Linear SVM	0.76	0.87	0.81	0.87
RBF SVM	0.93	0.92	0.93	0.92
SGD	0.90	0.90	0.90	0.90
Decision Tree	0.92	0.92	0.92	0.92
Random Forest	0.90	0.87	0.80	0.87
Neural Network	**0.95**	**0.95**	**0.94**	**0.95**
AdaBoost	0.90	0.90	0.90	0.90

Table 5
Machinery test results on the entire data

Classifier	Precision	Recall	F1-score	Accuracy
Nearest Neighbors	0.79	0.83	0.81	0.83
Linear SVM	**0.74**	**0.86**	**0.80**	**0.86**
RBF SVM	0.80	0.77	0.78	0.77
SGD	0.79	0.76	0.77	0.77
Decision Tree	0.80	0.81	0.80	0.81
Random Forest	**0.74**	**0.86**	**0.80**	**0.86**
Neural Network	0.79	0.83	0.81	0.83
AdaBoost	0.79	0.79	0.79	0.79

Recall indicates how many relevant items are retrieved. In other words, how many constraints are detected correctly by the algorithms. Among the evaluation metrics shown in Table 4, we chose to focus on recall due to the unbalanced data set consisting of only a small number of constraints, each of which is highly valuable for the extraction. Additionally, we consider False Positives as less important than False Negatives because for the validation approach, it is important that no constraints are missed.

To further assess the constraint extraction approach, the *PackML* trained models are tested on the *Machinery* specification document as well. The results are shown in Table 5 and indicate that the best performance is achieved using Linear SVM and Random Forest. The recall values reach 86% which is sufficiently good for such an experiment (training with one document and testing on another one).

In the light of these experiments, we conclude that, when the training and test data are similar (from the same set), Neural Networks work very well. However, Linear SVM and Random Forest achieve more consistent results, even if there is a considerable difference between the training and test set. Therefore, we can interpret that, Neural Networks might have the problem of overfitting while Linear SVM and Random Forest can be generalized to new data more easily.

7.4.2. Lexical-pattern-based approach

The evaluation of the lexical pattern-based approach consists in comparing the performance of the linguistic pattern matching to a manually curated gold standard. Using the *Machinery* companion specification as an input document, the linguistic patterns defined in Section 5.2.2 are applied. As a result, all sentences are extracted that fit the constraint patterns, which leads to (i) 26 True Positives (extracted sentences labeled as constraints in the gold standard); (ii) 131 False Positives (extracted sentences labeled as not-constraints in the gold standard); (iii) 18 False Negatives (sentences that did not get extracted but are labeled as constraints in the gold standard). Using Eq. (1) and Eq. (2) we calculate a precision score of approximately 0.17 and a recall of 0.59.

In comparison to the ML-based approach evaluated in the previous section, the modeling constraints extraction based on linguistic patterns does not achieve good results. Since the OPC UA data is large, natural-language-based and diverse, modeling constraints can be expressed in different structures. Nevertheless, formulating a higher number of linguistic patterns would result in a higher number of False Positives. To conclude, the ML-based extraction

proved to be more flexible and thus outperformed the lexical pattern-based approach. However, training an ML model that offers high accuracy predictions is a more complex and expensive approach. Therefore, based on the requirements and desired results one should make a trade-off between high accuracy and low resource extraction.

7.5. Feasibility evaluation with domain experts

To evaluate the usefulness and clarity of the designed HC Tasks for the verification of automatically extracted constraints and generated rules (Stage 4 of our approach) an expert-sourcing validation campaign was conducted in the concrete context of the OPC UA Machinery specification.

7.5.1. Expert validation campaign of the Machinery specification

From the gold-standard *Machinery* data created by the authors, we selected: (i) 60 sentences, 70% of which were considered by authors to represent modeling constraints; and (ii) the rules that could be formulated for the constraints expressed by these sentences. Six OPC UA experts (named in the Acknowledgement section) that were involved in the creation of the chosen companion specification document (*Machinery*) were asked to perform the validation tasks. The rules in Task 2 (Fig. 21) were shown in a structured natural language, rather than SPARQL so that they are understood also by experts not familiar with the SPARQL syntax.

The validation campaign was run on Amazon Mechanical Turk (https://www.mturk.com) – a crowdsourcing platform that offers the possibility to implement HC processes. The platform allows for easy results aggregation and makes it possible to avoid sequence bias by showing the tasks in a random order to each of the evaluators.

For the campaign, the extracted sentences were split into two batches of 30 sentences (and their corresponding rules), and 3 experts were assigned to each batch. The evaluators completed the tasks within two weeks and each of them submitted 25–30 responses. This setup allowed for the collection of 3 responses on average per sentence and provided 168 judgments in total.

Since there was no variety in the classes to which the selected sentences referred to (e.g., *Object TypeDefinition*), Task 1b (Constraint Classification) was not included in the validation campaign. In the next subsections, the results of both HC tasks are analyzed to identify areas for improvement of the approach.

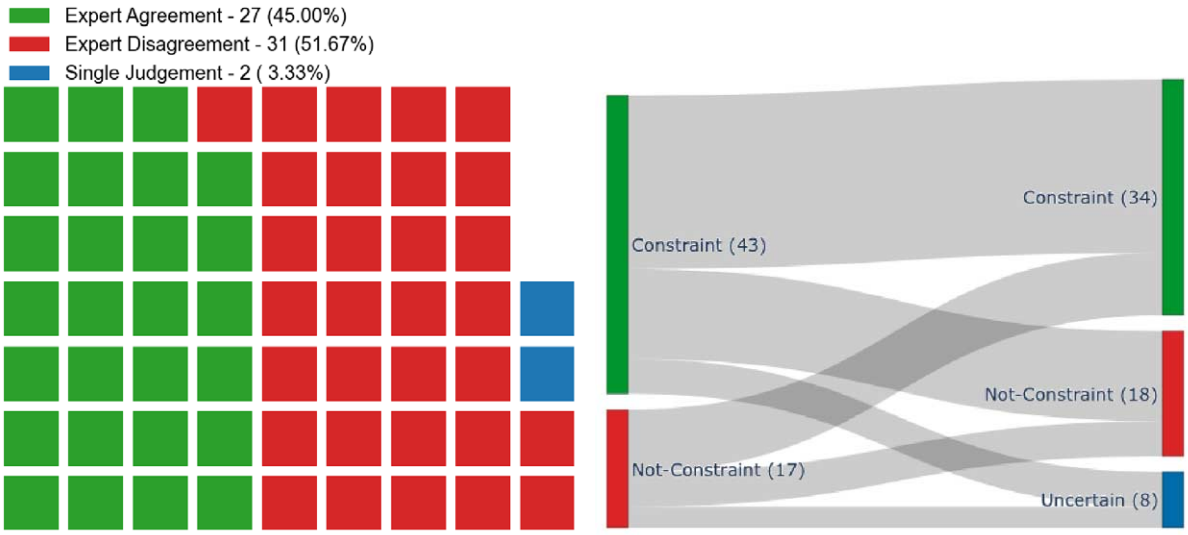
7.5.2. Constraint extraction from text (Task 1)

Figure 26 shows an overview of the results of the constraint classification task. Based on the experts' judgments a majority vote could be calculated for almost 90% of the sentences, however, a complete agreement among the experts was not achieved on over 50% of the data items as it is seen in Fig. 26a. Since for each task a variable number of evaluations were collected, Krippendorff's alpha coefficient was used to calculate the inter-rater agreement among the experts. The alpha score of 0.19 indicates very low agreement and shows that the identification of constraints is a difficult task.

In Fig. 26b a comparison between the classification proposed by the authors and the classification resulting from the experts' validations is shown. Sentences, for which a majority vote could not be established, have been added to the category "Uncertain". When comparing the majority vote of the experts against the authors' classification there is an overlap on less than 60% of the sentences. These results add to the statement that identifying constraints in OPC UA documents is a complex problem and textual definitions are open to interpretation. Therefore identifying textual constraints and defining formal rules to verify them is an important task for ensuring the conformity of OPC UA Nodeset files to the OPC UA base/companion specifications, which can be enabled by the proposed validation approach.

7.5.3. Rule validation (Task 2)

In this section, the ability of the proposed rules to completely and correctly capture textual constraints is examined. Because of the low agreement among the experts on the constraint verification task, rules were not validated by all participants and a majority voting was not feasible for all judgments. Moreover, 45% of the rule validations were evaluated only by a single expert and for 67% of the rules defects were identified by one evaluator only. Since an inter-rater agreement score is not meaningful considering the gathered data, we compute for each expert the percentage of their agreement with the proposed rules.



(a) Overview of the agreement among experts on the classification of the input sentences in absolute numbers and in percentages.

(b) Overview of the classification of the input sentences by the authors and by the experts in absolute numbers.

Fig. 26. Results of the constraint classification task in terms of (a) agreement among experts on the classifications, and (b) alignment between the classification proposed by the authors (left) and by the experts' majority vote (right).

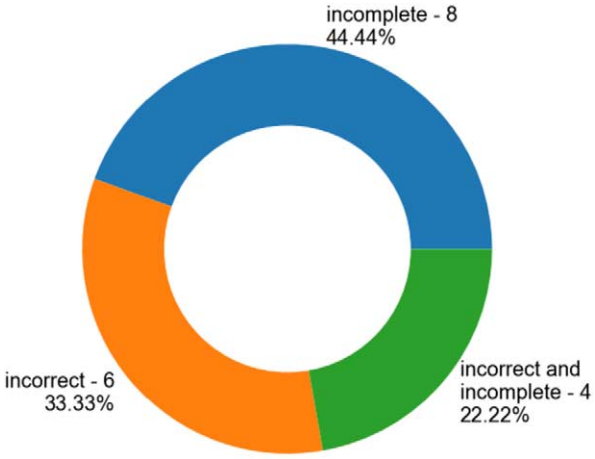


Fig. 27. Frequency of the defects identified by the experts for rules that were marked as invalid in absolute numbers and in percentages.

The expert scores vary from 33% to 100% and result in an average of 68% acceptance of the proposed rules. The frequency of found defects for the 18 rules, for which a defect was selected by at least one evaluator, can be seen in Fig. 27. While no rule set was found to be superfluous, 8 rules were judged as incomplete and 6 as incorrect. In 4 of the rule sets both defects (incompleteness and incorrectness) were found, where each have been selected by at least one expert. In the comments added by the experts for this task, exact mistakes and proposals for improvements were identified.

To conclude, the campaign results show that (i) the task of constraint identification is difficult even for experts, and (ii) the human-in-the-loop approach enables the successful identification and classification of invalid constraints and rules which is essential for improving the automatic algorithms.

It is important to note that the feasibility evaluation involved a large amount of manual effort in preparing the input data, however, with the improvement of the automatic extraction algorithms, these efforts will be reduced. An

important outcome of the performed validation campaign is the establishment of a gold standard for the Machinery specification, which will act as training data for our automatic approaches so that for the extraction of constraints from further specifications the manual effort will be further minimized.

8. Related work

We position our work in the landscape of using Semantic Web technologies in industrial settings (Section 8.1), and then discuss related work on information extraction in general (Section 8.2), and semantic information extraction (Section 8.3) in particular.

8.1. Application of Semantic Web technologies in industrial settings

Semantic Web technologies have been extensively applied to support various industrial applications in the last decade ranging from combining sensor networks with the Web [6] to augmenting products with semantic descriptions [33] or enabling smart city infrastructures [5]. The application of these technologies has been taking place in a variety of (mission-critical) domains, such as manufacturing [2,27], electric grids [18], or buildings [4] and addressed tasks both during the *engineering* (e.g., engineering model integration, digital twin model consistency management) and *operation&maintenance* (monitoring and anomaly detection, optimization and reconfiguration) phases of cyber-physical systems in such domains [35]. Furthermore, an important task is the verification of technical information objects (e.g., ontologies, semantic data sets) in terms of complying with constraints specified in languages such as SHACL, SWRL, or SPARQL.

Against this backdrop, in this paper, we investigate a *novel setting* for the application of Semantic Web technologies in the context of industrial standards, in particular as a solution option for automating validation processes of information models based on standards. Our work is in line with the current paradigm shift in such standards towards automatic standard validations. In particular, the OPC UA Semantic Validation Working Group (which we described in Section 1) focuses on providing the foundation to create valid, consistent NodeSets. However, their work is hampered by the high manual effort to identify modeling constraints in text and capturing them in a formal, machine-actionable rule language. Therefore, in this study, we worked towards a (semi-) automatic approach to validate the OPC UA information models and to the best of our knowledge, we are the first to work on such an approach.

8.2. Information extraction

A key goal of our work is extracting relevant constraint information from unstructured documents that can be then translated into (semi-) formal rules. This is in essence an information extraction task. Information extraction (IE) takes natural language-based text as input and produces detailed, fixed-format data using technology-based methods [9]. The challenge depends on the complexity of the data [20]. The closer the data is to natural language, the greater the required effort. The input data might come from social media, e-mails, chatbots, websites, ontologies, documents, etc [17]. The information can be extracted by using various NLP, text mining and machine learning (ML) methods [7] depending on the desired modeling. Thus, the significant interplay between syntax and semantic modeling should be considered [14]. Moreover, the methods might be a part of supervised or unsupervised concepts such as sentence classification, estimation, text generation, similarity, graph creation, ontology-learning, etc. In either case, IE aims to convert the knowledge into a structured form suitable for computer manipulation, opening up many possibilities for using it [17]. In contrast to this state of the art, we are working not only on text but also on table forms and aiming to use mixed methods, including classification, text similarities, and text generation.

Hence, several studies which use information extraction methods in the NLP domain, such as text classification, intent extraction, text similarity, and paraphrasing, exist. In order to achieve one of those tasks, information extraction can be used as a step for a higher purpose such as grammatical knowledge extraction. Although the studies in the English language dominate the field, there are many studies for other languages as well. For example, there

is a study [29] about information extraction on Past, Present, and Future Tenses, and there is another example of Hungarian noun phrase extraction from textual documents [32].

Information extraction from text-based documents has been investigated for various purposes for a couple of decades [8]. Sentiment analysis – for instance – is one of the main working areas in text mining and IE. It is the base of the emotion and intent classification of texts. By tagging the sentences or paragraphs as positive, negative and neutral, we can make sense of the text. Therefore, it can be used for child filters of the internet, analyzing the user comments as customer service, etc. Another study from the economy domain is interested in information extraction for qualitative financial data [7]. The project aims to prevent text-based qualitative data overload from different text-based sources by using NLP techniques. A literature review in the medical domain about information extraction from clinical data [41], claims that around 300 different articles were examined in terms of IE tools, methods, data sources, and applications. The FRODO (A Framework for Distributed Organizational Memories) [39] project, for example, started before 2000 and focused on document analysis to extract relations and entities.

The other technical method for document analysis is text classification. In [31], text documents in the business domain such as e-mails, office documents, pdf files, etc., are classified using a knowledge base. An additional interesting example of the studies in the industry is resume automation [36] for human resource departments. It is a different domain regarding being interested in personal and other structured data types.

Text-based regulations are very similar and interesting documents for us because they are giving information about constraints and rules regarding the domain. There is an example from the Game industry about documents called Automated Extraction and Classification of Slot Machine Requirements from Gaming Regulations [30]. The rules from state and federal laws and regulations were extracted using primitive rule-based algorithms and Naive Bayes. Another similar work on legal regulations focuses on obtaining machine-readable data from legal sources to create an appropriate computable representation of building regulations [1]. Finally, security-related terms from various unstructured data sources were extracted in [21].

As it is seen, there is an endless amount of work in the field of information extraction.

8.3. Semantic information extraction

A core part of our work is proposing methods to (semi-)automatically extract validation rules from the standard specifications, thus being related to the large body of work on information extraction (IE) in the *context of the Semantic Web* (i.e., *semantic information extraction*) recently reviewed in [26]. This review identified that IE systems have been applied in numerous domains and have been developed for different types of sources – structured text, semi-structured text, unstructured text, images, and tables or mixed multi-modal forms [13]. While many studies focus on text-based sources, there are other modalities that can be used as a base for information extraction, such as tables and schema. In [12] the authors aim to make a classification of information, structured in tables, and define the relations between cell information. In a few studies, multiple modalities such as either graph and text together or a combination of table and text are used for information extraction. In those studies, the main goal is to extract semantic and well-formed machine-readable data from huge online or offline data [11].

An important distinction also refers to the extracted information which can be *entities*, *concepts*, *binary relations* (i.e., triples) or more complex *n-ary relations* (i.e., rules, axioms). As the focus of our work is on extracting rules, which can be seen as complex n-ary relations, we identified the following examples of works focused on extracting n-ary relations. In [16], semantic relations between concepts are extracted from medical semi-structured text based on belief states. The use of IE for the generation of triples from documents is discussed in [40]. A more complex problem is acquiring rules, which often can contain several triples. Rules have been acquired from web content [19,38] or textual content, e.g., in the medical domain [3].

There are also approaches where the output of IE results in a complex knowledge structure such as a knowledge graph. For example, in [15] the main goal is the creation of a legal knowledge graph based on the Austrian platform RIS (https://www.ris.bka.gv.at). For the population of the knowledge graph legal entities (such as legal rules, references, contributors, etc.) are extracted from structured resources and from text. For the extraction, both machine and deep learning techniques are used as well as a rule-based approach.

There are also studies about information extraction from communication standards. Information extraction from smart devices in home Wi-Fi networks consists in a string matching between input data and a prebuilt smart device

rule database. Therefore, the focus is on device information extraction from unstructured strings rather than complex human language-based documents. One of the latest studies [22] which is closer to our purpose is setting up an error-tolerant procedure that extracts information from Natural Language (NL) Communication Standard Documents. They handle Alternating Bit Protocol (ABP) as a use case. Above all, we deal with multi-modal textual data for complex technical documents to generate semantic validation rules. While [22] mainly focuses on inconsistency inside the document, we would like to extract the semantic net and query mechanism for information models.

In summary, our work focuses on technical documents, explores both textual and tabular modalities, and aims to extract rules (e.g., complex n-ary relations). This is the first effort to perform this complex information extraction task in communication standards to the best of our knowledge. OPC UA specifications are handled in the project's first phase scope. However, in the future, we would like to expand our approach to other industrial communication standards, such as Asset Administration Shell.[2]

9. Conclusions and future work

The use of industrial standards is core to industrial engineering across application domains to introduce cannons for digital communication, data exchange, etc. that ensure interoperability across domain stakeholders. However, a commonly taken approach is that standard specifications are provided in non/semi-structured documents which makes it difficult to automate compliance checks of information artifacts relying on them, as demonstrated by the concrete use case of OPC UA. Therefore, a paradigm shift in the area of industrial standards is needed towards more machine-processable, explicitly represented standard specifications, additionally to textual specifications, so that the validation of information artifacts can be automated.

To support such a paradigm shift towards automated semantic validation, we proposed a high-level approach for representing and (semi-)automatically extracting formal rules from unstructured standard documents and then instantiated this approach in the case of OPC UA and reached the following conclusions.

In terms of *RQ1*, we found that it was feasible to represent modeling guidelines from the specifications as formal rules. Furthermore, these rules could be organised in a taxonomy represented by means of an OWL ontology, thus benefiting from the capabilities of explicit semantic representation of the Semantic Web technologies. We used SPARQL to provide rule-templates that can be instantiated into concrete rules and also tested a subset of these rules on the Robotics companion specification. We found that the current taxonomy of rules could express constrains available both in tabular and textual format, and can be easily extended as needed.

Related to *RQ2*, as a first step towards automating the derivation of rules from specifications, in the context of OPC UA, both tabular and textual information can be processed to identify modeling constraints with a high precision ($P = 87\%$) for tables and high performance for extraction from text (F1 up to 94%). In terms of methods, although they require training data, machine learning methods lead to more promising results than approaches relying on lexico-syntactic patterns, which are hampered by the high variety in the style of the text across domains as well as specification creators. Furthermore, the evaluation campaign with the Machinery experts showed that the way modeling constraints are expressed in text is often highly ambiguous and leads to low agreement even across experts. This further motivates the need for methods as presented in this paper where a combination of techniques are used to identify, verify and explicitly define formal rules corresponding to such constraints to reduce the ambiguity of the textual specifications.

Related to *RQ3*, generating rules based on automatically identified modeling constraints in the specifications could be solved in two ways in the OPC UA context. Firstly, fully-automated generation was possible based on information available in tables as the table types are already indicators of types of rules that are expressed; pre-requisites were a clear understanding of modeling constraint and formal rule types, a mapping between these as well as the use of rule templates as defined in the rule taxonomy. Second, to connect textual modeling constraints to formal rules a Human-in-the-loop approach was proposed, given the lack of training data for automating this task.

[2] www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf?__blob=publicationFile&v=12

Limitations and Future work. The following future work items address the current limitations of this work:

- *Reduce manual effort* required by some of the methods. The extraction methods in Stage2 require manual input in terms of keywords for table identification or training data. As this work advances, it is to be expected that after the repeated application to a high number of specifications, the manual effort of method adaptation will decrease (e.g., keywords specified for extracting tables for one specification will be largely applicable to the next specifications). Furthermore, in ongoing work we are experimenting with a rule-extraction method that is semi-automated (i.e., extraction rules are learned automatically from text and validated by an expert), thus further reducing the amount of manual input. Future work on testing this approach on other OPC UA specifications will explicitly consider and quantify the (decrease of) manual effort when the approach is applied to a new specification.
- *End-to-end evaluation of proposed approach.* While the individual steps of the proposed approach have been tested on a significant OPC UA corpus, evaluation has focused on the functioning of individual methods. Ongoing efforts focus on the identification of several use cases that allow the end-to-end evaluation of the proposed approach, with all its individual methods. While the core building blocks of the approach have been investigated and evaluated on several companion specifications, an integrated tool-chain for the end-to-end process is part of the future work.
- *Improve rule taxonomy.* It was found that both the taxonomy and rule template definitions heavily depend on the language used to express the rules and, in the case of SPARQL, also on the specific OPC UA NodeSet to OWL Ontology transformation. Thus, if the OPC UA NodeSet to OWL Ontology transformation changes, the SPARQL rules may also require adaptations. Also, a two step approach for rules, separating the navigation to the focus node from the actual rule application would help to reduce the amount of needed rules. Therefore, we will investigate alternative approaches for representing the rule taxonomy in order to reduce the complexity of the taxonomy and provide simpler rule templates.
- *Explore multi-modal constraint extraction.* While modeling constraints could be identified both in tables and text, we observed an overlap of content across the information expressed in these modalities which currently is not taken into account. Therefore, we wish to advance the constraint extraction methods by leveraging this overlap as a signal of which information is a modeling constraint (e.g., strong signal when the constraint is mentioned in both modalities).
- *Provide rule explanations through provenance information.* The multi-model extraction will lead to a more complex rule extraction workflow. Therefore, providing traces on how a rule was derived and what it means (e.g., based on the originating part of a specification) is important for OPC UA Specification authors and end-users of these specifications that want to validate their files. Ideally, if a rule identifies an issue in a file, it is important to provide rule provenance information in terms of tables, text snippets that lead to its derivation as well as the version of specification it was derived from as a first immediate documentation for the end-user to debug/improve his files. To achieve such features, we will enable the auditability of the rule extraction process through a provenance tracing and representation middleware.
- *Extend to other OPC UA specifications* beyond those considered as part of current work.
- *Apply approach to other industrial standards.* While the points above deepen the work in the context of the OPC UA standard, an orthogonal effort will be investigating (i) to what extent the methods and algorithms developed for OPC UA are applicable in the context of other industrial standards; (ii) what challenges need to be solved when applied to other standard; and (iii) what is a typical effort during such adaptation. While we expect that the core stages of the approach will be applicable for the case of most standards, methods in individual stages will have to be adapted to the particularities of that standard (e.g., the rule catalog will have to reflect rules relevant for that standard, the constraint extraction methods will have to be modified to the particularities of the documents that capture the standard, etc.). Having said that, the work presented in this paper will offer not only a general process to follow but also valuable suggestions about the range of possible methods to be used in each individual stage and their typical challenges and performance. We are currently investigating a new application use case for the Asset Administration Shell communication standard.

With the future work mentioned above, this work has the potential to bring a major contribution towards automatic, semantic validation within the widely used OPC UA standard, as well as other industrial standards, thus leading to improved interoperability in industrial engineering settings and exploring the capabilities of Semantic Web technologies for this novel and important problem.

Acknowledgements

We thank the six OPC UA Machinery working group experts (Sebastian Friedl, Götz Görisch, Tonja Heinemann, Timo Helfrich, Heiko Herden and Wolfgang Mahnke) for taking part in the feasibility evaluation campaign. We also thank the experts from the OPC UA Semantic Validation working group, who helped us pilot the Human Computation tasks and provided feedback on possible improvements. Parts of this work related to Human Computation were funded through the FWF HOnEst project (V 745-N) and the OntoCommons project funded by the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 958371.

Appendix. Rule taxonomies

Figures in this appendix depict the created rule taxonomies.

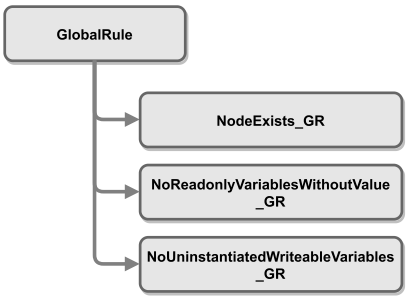


Fig. 28. Identified global rules.

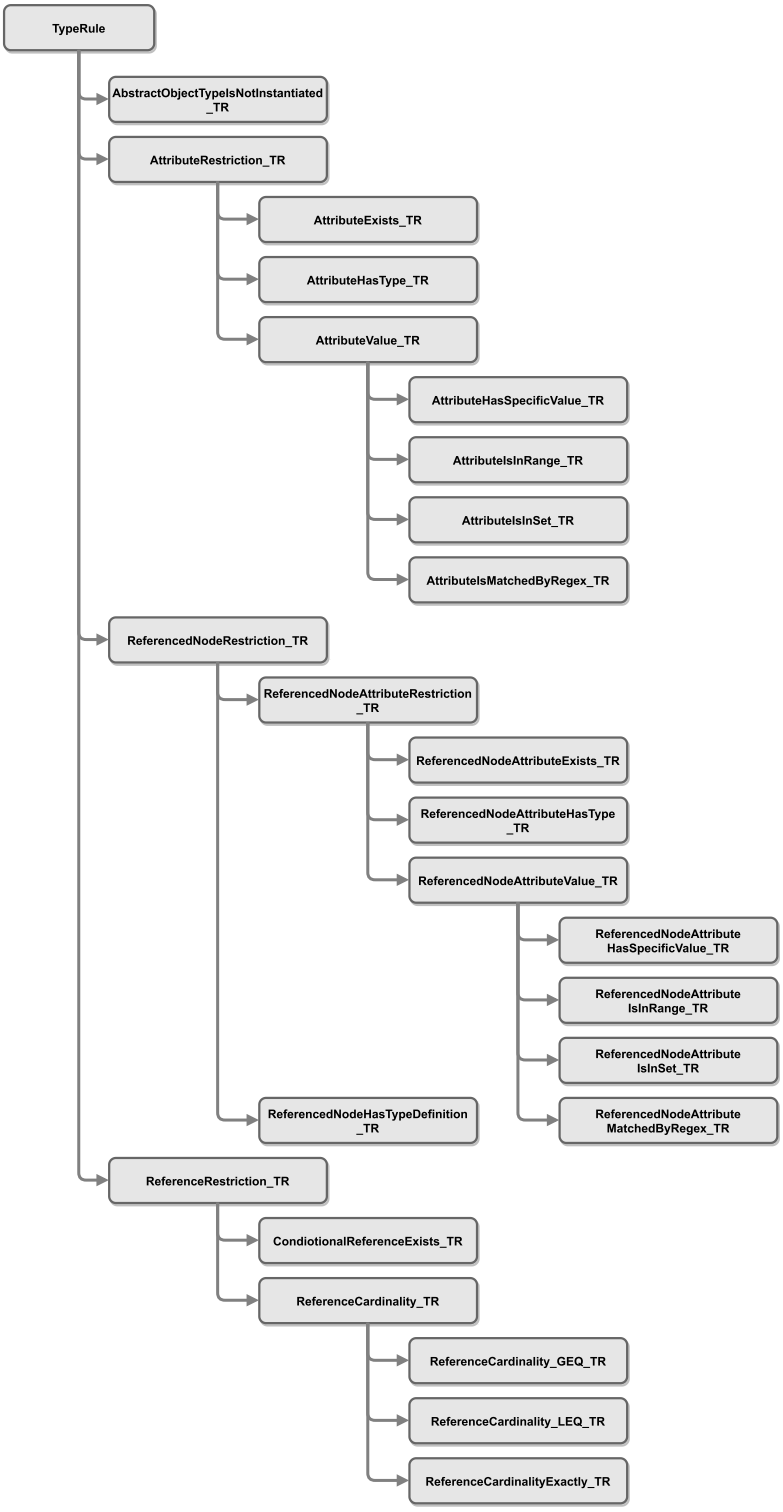


Fig. 29. Identified type rules.

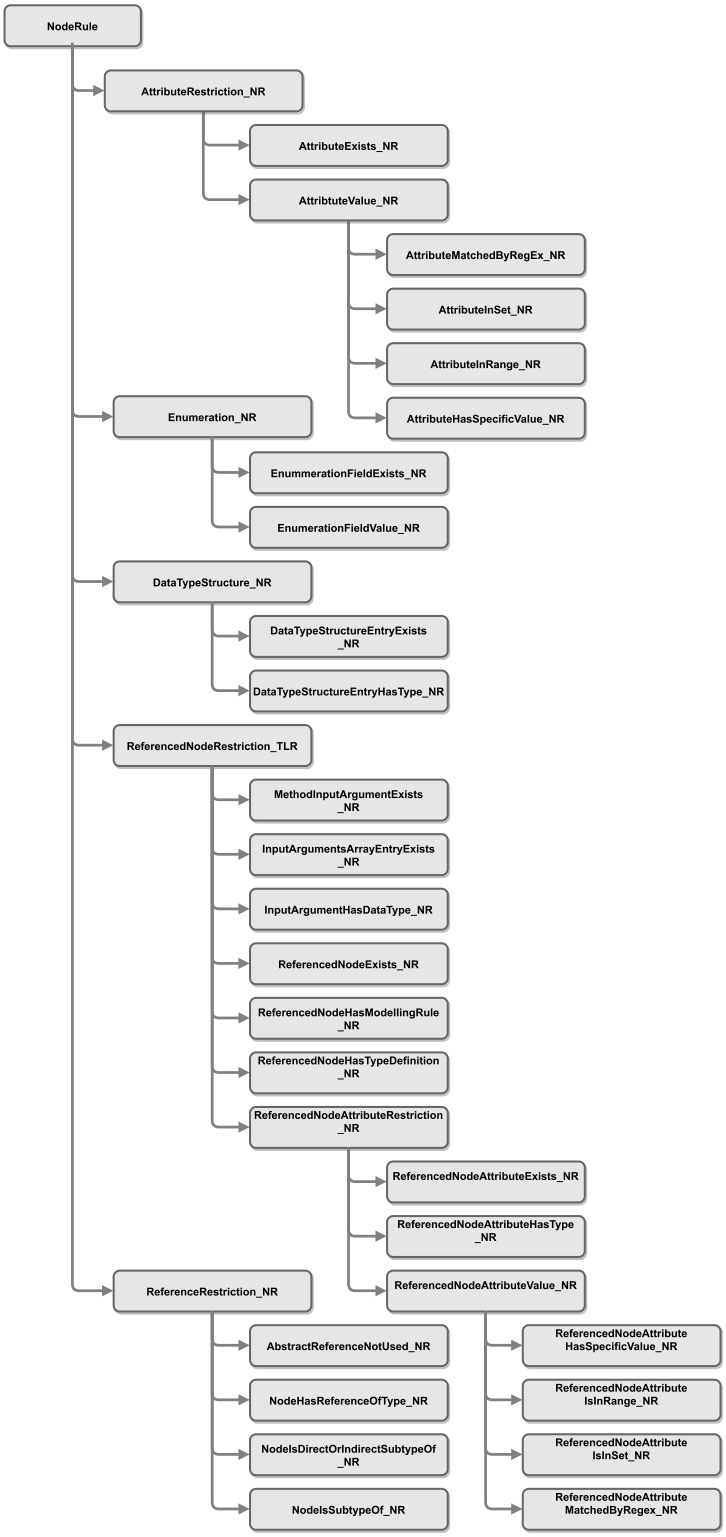


Fig. 30. Identified node rules.

References

[1] M. Aydın and H. Yaman, Domain knowledge representation languages and methods for building regulations, in: *Eurasian BIM Forum*, Springer, 2019, pp. 101–121.

[2] S. Biffl and M. Sabou, *Semantic Web Technologies for Intelligent Engineering Applications*, Springer, 2016, pp. 1–405. doi:10.1007/978-3-319-41490-4.

[3] A. Boufrida and Z. Boufaida, Rule extraction from scientific texts: Evaluation in the specialty of gynecology, *Journal of King Saud University – Computer and Information Sciences* (2020).

[4] B. Butzin, F. Golatowski and D. Timmermann, A survey on information modeling and ontologies in building automation, in: *Proc. of Annual Conf. of the IEEE Industrial Electronics Society*, IEEE, 2017, pp. 8615–8621. doi:10.1109/IECON.2017.8217514.

[5] I. Celino and S. Kotoulas, Smart cities [guest editors' introduction], *IEEE Internet Computing* **17**(6) (2013), 8–11. doi:10.1109/MIC.2013.117.

[6] O. Corcho and R. García-Castro, Five challenges for the semantic sensor web, *Semantic Web* **1**(1–2) (2010), 121–125. doi:10.3233/SW-2010-0005.

[7] M. Costantino, R.G. Morgan, R.J. Collingham and R. Carigliano, Natural language processing and information extraction: Qualitative analysis of financial news articles, in: *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, 1997, pp. 116–122. doi:10.1109/CIFER.1997.618923.

[8] J. Cowie and W. Lehnert, Information extraction, *Commun. ACM* **39**(1) (1996), 80–91. doi:10.1145/234173.234209.

[9] H. Cunningham, Information extraction, automatic, *Encyclopedia of Language and Linguistics* **3**(8) (2005), 10.

[10] H. da Rocha, A. Espirito-Santo and R. Abrishambaf, Semantic interoperability in the industry 4.0 using the IEEE 1451 standard, in: *IECON 2020 the 46th Annual Conference of the IEEE*, Industrial Electronics Society, 2020, pp. 5243–5248. doi:10.1109/IECON43393.2020.9254274.

[11] P. Dolog and W. Nejdl, Challenges and benefits of the semantic web for user modelling, in: *Proceedings of the Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2003) at 12th International World Wide Web Conference*, Budapest, 2003.

[12] H. Dong, S. Liu, Z. Fu, S. Han and D. Zhang, Semantic structure extraction for spreadsheet tables with a multi-task learning architecture, in: *Workshop on Document Intelligence at NeurIPS 2019*, 2019.

[13] L. Eikvil, Information extraction from world wide web-a survey, Technical Report, Citeseer, 1999.

[14] R. Engels and B. Bremdal, Information extraction: State-of-the-art report, On-To-Knowledge Consortium, 2000.

[15] E. Filtz, S. Kirrane and A. Polleres, The linked legal data landscape: Linking legal data across different countries, *Artificial Intelligence and Law* **29**(4) (2021), 485–539. doi:10.1007/s10506-021-09282-8.

[16] T. Goodwin and S.M. Harabagiu, Automatic generation of a qualified medical knowledge graph and its usage for retrieving patient cohorts from electronic medical records, in: *2013 IEEE Seventh International Conference on Semantic Computing*, IEEE, 2013, pp. 363–370. doi:10.1109/ICSC.2013.68.

[17] R. Grishman, Information extraction, *IEEE Intelligent Systems* **30**(5) (2015), 8–15. doi:10.1109/MIS.2015.68.

[18] S. Howell, Y. Rezgui, J.-L. Hippolyte, B. Jayan and H. Li, Towards the next generation of smart grids: Semantic and holonic multi-agent management of distributed energy resources, *Renewable and Sustainable Energy Reviews* **77** (2017), 193–214. doi:10.1016/j.rser.2017.03.107.

[19] J. Kang and J.K. Lee, Rule identification from web pages by the XRML approach, *Decision Support Systems* **41**(1) (2005), 205–227. doi:10.1016/j.dss.2005.01.004.

[20] E. Kumar, *Natural Language Processing*, IK International Pvt Ltd, 2013.

[21] R. Lal et al., Information Extraction of Security related entities and concepts from unstructured text, 2013.

[22] S. León, J.A. Rodríguez-Mondéjar and C. Puente, Inconsistency detection on data communication standards using information extraction techniques: The ABP case, in: *International Workshop on Soft Computing Models in Industrial and Environmental Applications*, Springer, 2019, pp. 291–300.

[23] Y. Liao, L.F.P. Ramos, M. Saturno, F. Deschamps, E. de Freitas Rocha Loures and A.L. Szejka, The role of interoperability in the fourth industrial revolution era, *IFAC-PapersOnLine* **50**(1) (2017), 12434–12439, 20th IFAC World Congress, https://www.sciencedirect.com/science/article/pii/S2405896317317615. doi:10.1016/j.ifacol.2017.08.1248.

[24] W. Mahnke and S.-H. Leitner, OPC unified architecture – the future standard for communication and information modeling in automation, *ABB Review* **2009** (2009), 3.

[25] W. Mahnke, S.-H. Leitner and M. Damm, *OPC Unified Architecture*, Springer, Berlin, 2009. ISBN 978-3-540-68898-3. doi:10.1007/978-3-540-68899-0.

[26] J.L. Martinez-Rodriguez, A. Hogan and I. Lopez-Arevalo, Information extraction meets the semantic web: A survey, *Semantic Web* **11**(2) (2020), 255–335. doi:10.3233/SW-180333.

[27] F. Ocker, C.J.J. Paredis and B. Vogel-Heuser, Applying knowledge bases to make factories smarter, *at – Automatisierungstechnik* **67**(6) (2019), 504–517. doi:10.1515/auto-2018-0138.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., Scikit-learn: Machine learning in Python, *The Journal Of Machine Learning Research* **12** (2011), 2825–2830.

[29] J. Piskorski and R. Yangarber, Information extraction: Past, present and future, in: *Multi-Source, Multilingual Information Extraction and Summarization*, Springer, 2013, pp. 23–49. doi:10.1007/978-3-642-28569-1_2.

[30] M.D. Prendergast, Automated extraction and classification of slot machine requirements from gaming regulations, in: *2021 IEEE International Systems Conference (SysCon)*, 2021, pp. 1–6. doi:10.1109/SysCon48628.2021.9447144.
[31] K. Rajbabu, H. Srinivas and S. Sudha, Industrial information extraction through multi-phase classification using ontology for unstructured documents, *Computers in Industry* **100** (2018), 137–147. doi:10.1016/j.compind.2018.04.007.
[32] G. Recski, Hungarian noun phrase extraction using rule-based and hybrid methods, *Acta Cybernetica* **21**(3) (2014), 461–479. doi:10.14232/actacyb.21.3.2014.11.
[33] M. Sabou, Smart objects: Challenges for semantic web research, *Semantic Web* **1**(1–2) (2010), 127–130. doi:10.3233/SW-2010-0011.
[34] M. Sabou, L. Aroyo, K. Bontcheva, A. Bozzon and R.K. Qarout, Semantic web and human computation: The status of an emerging field, *Semantic Web* **9**(3) (2018), 291–302. doi:10.3233/SW-180292.
[35] M. Sabou, S. Biffl, A. Einfalt, L. Krammer, W. Kastner and F.J. Ekaputra, Semantics for cyber-physical systems: A cross-domain perspective, *Semantic Web* **11**(1) (2020), 115–124. doi:10.3233/SW-190381.
[36] S. Sanyal, S. Hazra, S. Adhikary and N. Ghosh, Resume parser with natural language processing, *International Journal of Engineering Science* **4484** (2017).
[37] R. Schiekofer, S. Grimm, M. Milicic Brandt and M. Weyrich, A formal mapping between OPC UA and the Semantic Web, 2019, pp. 33–40. doi:10.1109/INDIN41052.2019.8972102.
[38] S. Schoenmackers, J. Davis, O. Etzioni and D. Weld, Learning first-order horn clauses from web text, in: *Proceedings of the 2010 Conference on Empirical Methods on Natural Language Processing*, 2010, pp. 1088–1098.
[39] M. Sintek, M. Junker, L. Van Elst and A. Abecker, Using information extraction rules for extending domain ontologies, in: *Workshop on Ontology Learning*, 2001.
[40] R. Upadhyay and A. Fujii, Semantic knowledge extraction from research documents, in: *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, IEEE, 2016, pp. 439–445. doi:10.15439/2016F221.
[41] Y. Wang, L. Wang, M. Rastegar-Mojarad, S. Moon, F. Shen, N. Afzal, S. Liu, Y. Zeng, S. Mehrabi, S. Sohn et al., Clinical information extraction applications: A literature review, *Journal of Biomedical Informatics* **77** (2018), 34–49. doi:10.1016/j.jbi.2017.11.011.
[42] B. Wollenberg, J. Britton, E. Dobrowolski, R. Podmore, J. Resek, J. Scheidt, J. Russell, T. Saxton and C. Ivanov, A brief history: The common information model [in my view], *IEEE Power and Energy Magazine* **14**(1) (2016), 128–126. doi:10.1109/MPE.2015.2481787.