A performance benchmark over semantic rule checking approaches in construction industry

Advanced Engineering Informatics 33 (2017) 68–88

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Full length article

A performance benchmark over semantic rule checking approaches in construction industry

Pieter Pauwels [a,*], Tarcisio Mendes de Farias [c,e], Chi Zhang [b], Ana Roxin [c], Jakob Beetz [b], Jos De Roo [d], Christophe Nicolle [c]

[a] Department of Architecture and Urban Planning, Ghent University, J. Plateaustraat 22, B-9000 Ghent, Belgium
[b] Department of the Built Environment, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
[c] Laboratory LE2I, CNRS, Arts et Métiers, Univ. Bourgogne Franche-Comté (UBFC), Dijon, France
[d] Agfa HealthCare NV, Moutstraat 100, B-9000 Ghent, Belgium
[e] Department of Ecology and Evolution, University of Lausanne, Biophore, CH-1015 Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 9 February 2016
Received in revised form 24 March 2017
Accepted 3 May 2017
Available online 18 May 2017

Keywords:
ifcOWL
Rule checking
Linked data
Reasoning
Semantic web
Benchmark

ABSTRACT

As more and more architectural design and construction data is represented using the Resource Description Framework (RDF) data model, it makes sense to take advantage of the logical basis of RDF and implement a semantic rule checking process as it is currently not available in the architectural design and construction industry. The argument for such a semantic rule checking process has been made a number of times by now. However, there are a number of strategies and approaches that can be followed regarding the realization of such a rule checking process, even when limiting to the use of semantic web technologies. In this article, we compare three reference rule checking approaches that have been reported earlier for semantic rule checking in the domain of architecture, engineering and construction (AEC). Each of these approaches has its advantages and disadvantages. A criterion that is tremendously important to allow adoption and uptake of such semantic rule checking approaches, is *performance*. Hence, this article provides an overview of our collaborative test results in order to obtain a performance benchmark for these approaches. In addition to the benchmark, a documentation of the actual rule checking approaches is discussed. Furthermore, we give an indication of the main features and decisions that impact performance for each of these three approaches, so that system developers in the construction industry can make an informed choice when deciding for one of the documented rule checking approaches.

1. Introduction

1.1. Rule checking in construction industry: application scenarios

Rule checking of building models is one of the key features required for many applications in the domain of architectural design and construction. A large share of the rule checking approaches is located in the realm of building performance checking and regulation compliance checking. Throughout the building life cycle, designs have to be checked for compliance to a vast number of different rules and constraints on international, national, local and even company-specific levels. Rule checking is often present in other application scenarios as well, including automatic query rewriting, building model conversion and subset selection.

With the advent of Building Information Modelling (BIM) tools [1], fundamentally new processes evolve that allow building information to be managed at any point in time. As acknowledged by Eastman et al. [2], more advanced BIM-based rule checking approaches are within reach as a result of this trend. Automated rule checking is defined by Eastman et al. [2] as *"software that does not modify a building design, but rather assesses a design on the basis of the configuration of objects, their relations or attributes*". According to Eastman et al. [2], who refer to the early works by Han et al. [3–5], rule-based systems are understood as systems that *"apply rules, constraints or conditions to a proposed design, with results such as 'pass', 'fail' or 'warning', or 'unknown'.*". So, a rule-based system in construction industry includes at least two critical elements: the design model and the rules.

* Corresponding author.
E-mail addresses: pipauwel.pauwels@ugent.be (P. Pauwels), tarcisio.mendesdefarias@unil.ch (T.M. de Farias), c.zhang@tue.nl (C. Zhang), ana-maria.roxin@u-bourgogne.fr (A. Roxin), j.beetz@bwk.tue.nl (J. Beetz), jos.deroo@agfa.com (J. De Roo), cnicolle@u-bourgogne.fr (C. Nicolle).

http://dx.doi.org/10.1016/j.aei.2017.05.001
1474-0346/

In the AEC industry, the BIM model is typically considered to be the preferred *design model* to start from. Many initiatives that start from such a BIM model furthermore start from a neutral representation of the building model, typically captured in the Industry Foundation Classes (IFC) [6,7], which is developed and maintained by the BuildingSMART organization [8]. The original representation of IFC in EXPRESS is very closely affiliated to a class structure in any programming language or database. Hence, rule checking has typically been implemented in hard-coded rules.

Recently, however, more and more architectural design and construction data is now also represented in the Resource Description Framework (RDF) data model [9], possibly referencing statements and concepts in the Web Ontology Language (OWL2) [10]. The underlying logical basis of OWL can promote the realization of the rule checking process outlined by Eastman et al. [2] using this additional logical basis. Also [11] indicated the usefulness of a logical basis in regulation compliance checking, as early as 2003, which was at that time implemented as an addition to plain XML. The same argument for a semantic or logical basis is also made in Hjelseth and Nisbet [12].

The *rules* come in different forms and shapes. It is certainly not our intention to discuss all these rule representations here, but we can outline a few. For the rule representation forms, Eastman et al. [2] presents three different options, namely:

1. using computer language encoded rules,
2. using parametric tables, and
3. language-driven

Of these three, especially the latter is interesting, as a language-driven approach is particularly good in providing extensibility of the rule set. As long as one can represent rules in the specific rule language that is used, one can supply the system with more rules in an on-demand fashion, which is not possible (or only to a limited extent) when using computer language encoded rules or parametric tables. Eastman et al. [2] further divides the language-driven implementation methods into two alternatives: either as a logic-based language or as a domain-oriented language. An example of the latter is the Building Environment Rule and Analysis (BERA) language that is proposed in Lee [13] and Lee et al. [14]. Another example is the rule checking approach proposed in Solihin and Eastman [15] that relies on Conceptual Graphs (CG), which are grounded in First Order Logic (FOL). This approach is an example of a predicate logic-based language for rule checking in construction industry. With its logical basis in Description Logics (DL) [16], the semantic rule checking approach as proposed in Pauwels et al. [17] is another example of such a logic-based language. It relies on semantic web technologies [18] for the implementation of a limited acoustical performance check. Also Beach et al. [19] relies on semantic web technologies for automated regulatory compliance in the construction sector.

Regardless of the representation that is used to represent the rules, a number of techniques are available to develop the rules. Rules can be developed or created manually, which is near to always the case when representing them in procedural code. When using a (logic-based) language, there is also a possibility of semi-automating the development of rules from its human language representation, as is for example investigated in Zhang and El-Gohary [20,21].

1.2. Semantic rule checking: the basics

In the previous section, we already saw that the design model and the rules are two vital elements in any rule checking process. These elements of the rule checking process have well-defined terms and definitions in any computer science context. The design model can be considered as a sample data set (e.g. an IFC building model). This data set typically follows an agreed structure, vocabulary, or class hierarchy (e.g. the IFC schema). The former, the sample data set, is commonly understood as an assertion box (*the ABox*), whereas the latter, the vocabulary, is commonly known as a terminological box (*the TBox*). ABox and TBox components are often used in scenarios other than rule checking, e.g. query interfaces, data exchange, interface design. In the context of rule checking, the rules form a third box in addition to the ABox and TBox, namely the rule box (*the RBox*).

At the core of any rule checking process are then three key components: (1) a schema that defines what kind of information is used by the rule checking process and how it is structured (the TBox), (2) a set of instances asserting facts based on the concepts defined in the TBox (the ABox), and (3) a set of rules (e.g. `IF-THEN` statements) that can be directly combined with the schema (the RBox). The key advantage in using a 'language-based' rule checking process, is that ABox, TBox and RBox are all stored in a compatible or identical (logic-based) language. A schematic display of this setup is provided in Fig. 1.

These three components are realised in various ways depending on the approach taken and the software system used. In a traditional hard-coded rule checking process, the schema is typically represented by the internal object model of the system including its class hierarchy; the instances are represented by the objects that follow this class hierarchy; and the rules are represented by interconnected procedural functions that can follow any kind of structure, while still being compatible with the class hierarchy of the system.

This is considerably different from the way in which these three components take shape in a semantic language-driven approach. Namely, in this case, the schema is typically represented by an OWL ontology, the instances are represented by the RDF graph using definitions of that OWL ontology, and the rules are logical conjunctions (`AND`) of declarative `IF-THEN` statements. Because of the logical basis of the OWL language in DL [16], the rule checking process is straightforward as soon as all the data and all the rules are available in a complete and consistent shape: inferences are generated by generic reasoning engines and results, asserted as new facts into the graph, are used, e.g. for simple visualisation in a graphical user interface (GUI).

In this article, we specifically look into the rule checking implementation method using semantic web technologies for construction industry. The great advantage of using semantic web technologies is that the schema, the instances, and the rules can all be described using one and the same data model or language. As a result, all three components benefit from the advantages given by Eastman et al. [2] for any language-driven approach, namely:

1. the possibility to easily retarget an implementation to different source formats (e.g. an alternative ontology: a Revit ontology instead of an IFC ontology);
2. portability across contexts, applications and devices, and

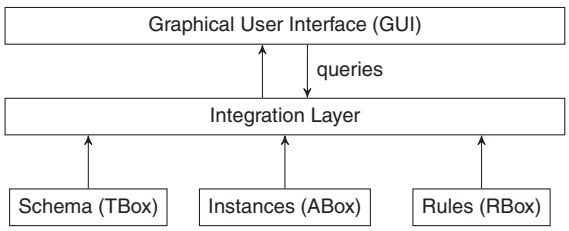

Fig. 1. Schematic view of a rule checking system and its key components.

3. the availability of an unlimited representation wealth, including 'nested conditions' and 'branching of alternative contexts'.

1.3. The need for a performance benchmark

Implementation-wise, however, data, ontologies and rules can be stored in very diverse ways and environments. For example, in some cases, they are shared openly on the World Wide Web (WWW); in other cases, only the ontologies are widely available via the WWW and the rules and the data are kept in local applications; in other cases, all three components are kept solely as in-memory models of an application, making them accessible only via the programming code. All three components may even be generated on-demand from legacy data sources (e.g. SQL databases that are given an RDF interface using R2RML [22]). Furthermore, some opt to represent rules largely using OWL class expressions (restrictions), whereas others rely primarily on IF-THEN rewrite rules (and less on OWL class expressions). In addition, both in the case of OWL class expressions and explicit IF-THEN rewrite rules, diverse levels of semantic expressiveness are available (transitivity, inverse relations, and so forth).

Furthermore, several software solutions are available for storing rules (RBox), ontologies (TBox) and instances (ABox), each with different underlying mathematical basis and thus different performances. When combining different sorts of resources (full RBox, full TBox, full ABox), performance is less than when the reasoning engine is fed with a carefully selected minimal set of resources. The type of reasoning engine and the way in which it is used, e.g. backward vs forward chaining inference, also has an important impact on performance. These and other aspects define the performance of the implemented rule checking procedures.

In conclusion, a wide range of choices awaits when starting the implementation of a semantic rule checking system for architectural design practice and construction industry applications, with each choice considerably affecting the performance and the expressive power of the system. A lot of attention typically goes to the expressiveness of the languages used. In this paper, however, we specifically focus on benchmarking the *performance* of the different systems. There is one main feature that indicates the performance of any system: time. The better performing a system is, the less time it takes to perform its tasks. In this regard, a good and wide performance benchmark has been produced and documented by De Witte et al. [23]. This benchmark compares four Linked Data solutions with commercial support and full SPARQL 1.1 compliance. The four solutions studied are Blazegraph, Enterprise Store I, Enterprise Store II, and Virtuoso. The performance benchmark evaluates the four solutions using their default configuration and with generic artificial data (data sets of 10, 100, and 1000 million triples). This benchmark gives excellent indications on the overall performance of these solutions for very big data sets.

The performance benchmark at which we aim here, is very different. The data sets are a lot smaller (typical in construction industry), but they are from real-world projects and quality. In addition, this benchmark also specifically targets the evaluation of performance in combination with rules (rule checking use cases in construction industry). In other words, the benchmark study discussed in this paper is far more qualitative and aimed towards real-world applications in construction and engineering. Instead of finding the fastest out-of-the-box solution, we wish to pinpoint more precisely what the main causes of performance differences are and how big their impacts are. As a result, we aim at setting up and evaluating systems that have customised settings instead of default settings, as is the case in De Witte et al. [23]. Hence, this paper will not present an exhaustive benchmark, in which all available software is compared. Only a few are selected to build up an initial benchmark. Also, we will not conclude with pinpointing 'the best performing' software and 'the worst performing' software. Instead we will end with a list of those factors that are of key importance when it comes to the performance of a semantic rule checking system in our industry.

For this benchmark, we have selected three setups, namely SPIN, EYE, and Stardog (Sections 4–6). Each of these setups includes a combination of a storage mechanism (e.g. a triple store) with a reasoning engine. Furthermore, each of these setups is a pure semantic web-oriented setup. We decided to stick to semantic web-oriented setups because of the reasons listed in Section 1.2. Clearly, a lot more setups are possible besides these three. Nevertheless, with these three setups, we already cover a lot of ground, as they are very different from each other. The Stardog approach is a purely commercial setup. For such an approach, the highest performance results would be naturally expected, likely accompanied with the least transparent algorithms (closed source). This approach relies on SWRL rules and a backward-chaining approach. The SPIN approach is also commercial, as it is proposed by TopQuadrant. In this approach, we combine SPIN functionality with the Apache Jena engine and its in-house triple store. It is different to Stardog in the sense that it relies more heavily on the SPARQL query language, also for the representation of rules. The EYE approach, finally, is entirely different from the SPIN and Stardog approaches, in the sense that it does not really include a triple store. Furthermore, it relies on N3Logic, a language that is highly expressive, but then also quite different from SWRL and SPIN. The core of the EYE engine is a Prolog virtual machine and the entire code is open-source.

In conclusion, although we do not cover the entire field of potential setups for rule checking using semantic web technologies, we do cover a very diverse range with our selection of three. By additionally focusing on the qualitative analysis (what makes the performance in the construction industry), we present a seldom available type of benchmark.

1.4. Paper outline

In this article, we give a first outline of the main performance differences between key implementation approaches for semantic rule checking systems applied to construction industry. We first give an indication of existing and proposed scenarios in which semantic rule checking takes place or plays an important role (Section 2). This section is based on the initial overview provided in Pauwels and Zhang [24] and will mainly consist of examples, with very limited indications regarding the technologies used. These indications suffice to outline a number of key implementation scenarios, so that a useful test environment for a performance benchmark over existing implementation scenarios and proposals can be established. In Section 3, we document the test environment we have set up for creating the performance benchmark, including software and hardware environments, performance measurement criteria, data sources and relevant test environment settings. This section also gives a high-level overview of the three rule checking approaches that we have selected for testing: SPIN, EYE, and Stardog. Sections 4–6 document the three rule checking approaches that we tested. In Section 7, we present a comparison of the three approaches to provide a performance benchmark for future reference. This section provides a table that lists a quantitative time performance comparison between the three selected approaches. However, the biggest part of this section focuses on listing key features to consider when deciding to set up a system for rule checking in construction industry. The main contributions of this article are in that last area:

- a classification, overview, discussion and comparison of the many aspects and choices that need to be made when implementing a semantic rule checking process in the construction industry,
- a first large open benchmark (data, ontology, rule set, query set) that allows performance comparison for future (semantic) rule checking implementation approaches.

2. Rule checking in construction industry: sample application scenarios

A number of applications have emerged by now that rely on the logical basis of semantic web languages to accommodate some form of semantic rule checking. One example that was already mentioned in the introduction is the effort by Pauwels et al. [17], which aims at accommodating *acoustic regulation compliance checking* for BIM models. In this exploratory article, an indication is made of the way in which N3Logic rules [25] can be represented and used in combination with a domain ontology (TBox) and an instance model (ABox), so that an inference engine immediately indicates whether a building model is compliant or not with the European acoustic regulations. In a full implementation, one might opt to use the IFC data model to fill the TBox and ABox.

A similar proposal is made in Pauwels et al. [26] to convert a geometry representation in IFC to corresponding geometry representations in the X3D and STL schemas, thereby relying on N3Logic rules, the EYE reasoning engine [27,28] and standard semantic web technologies. This *geometry conversion* example is closer to addressing the interoperability challenge in construction industry, rather than being really representative for a rule checking process as it is commonly interpreted in construction industry.

The example rule checking implementation that is documented in Lee et al. [29] might be considered similar to this last geometry conversion proposal, in the sense that it also targets the inference of information in one representation (as required for cost estimation) based entirely on information in another representation (IFC), which relates to the same interoperability challenge. Lee et al. [29] propose two small OWL ontologies, a work item ontology and a work condition ontology, which are engineered with the purpose of helping in the *building cost estimation* process. The authors propose to extract information from an ifcXML file and parse this information as RDF instances of the earlier mentioned OWL ontologies. Using these RDF instances and the OWL ontology, an OWL reasoning engine (in this case the RETE-based Bossam reasoner [30]) is able to infer additional properties and class memberships. A user interface then allows a user to query a SPARQL endpoint holding the resulting graph. The query results can be used more intuitively in a cost estimation application. This example nicely illustrates that a reasoning process can just as well be implemented using OWL class expressions only.

One of the earliest approaches in this domain similarly relies extensively on ontologies and SPARQL, rather than considering dedicated semantic rule languages. Namely, the approach presented by Yurchyshyna et al. [31,32] and Bouzidi et al. [33] makes this proposal towards *regulation-compliance checking* in construction industry. This approach of querying a model or ontology with a query language is still regularly used in semantic rule checking. A similar proposal has namely been proposed by Dimyadi et al. [34] for regulation-compliance audits in general.

Wicaksono et al. [35] rely on semantic web technologies to build an intelligent *energy management system* for buildings. They propose to build an RDF representation of a building model, using an OWL ontology for building information. This ontology includes concepts for the appliances present in the building (dishwasher, fridge). The OWL ontology can then be combined with a number of rules expressed in the Semantic Web Rule Language (SWRL) [36], to enable an inference engine to infer if there are any anomalous activities occurring (e.g. 'heaters' that are 'working' AND 'windows' that are 'open'). This work has been extended in Wicaksono et al. [37], to include an OWL ontology inspired by IFC. In this extended proposal, the authors rely on a rule engine based on SWRLJessBridge, which allows the execution of rules combined with the Protégé API [38]. A SPARQL endpoint [39] is made available on top of this rule engine, so that the end user only has to query for the results of the rules (i.e. are the `EnergyInefficient` or `UsageAnomaly` individuals present? - see complete examples in Wicaksono et al. [37]).

Another example showing the way in which rules can be deployed for construction industry and building information management is provided by Kadolsky et al. [40] and Baumgärtel et al. [41]. In this example, the authors propose to represent a building in an ifcOWL ontology, after which rules can be used to retrieve information that is relevant for *building energy performance*. Baumgärtel et al. [41] specifically show how rules can be used to allow a thermal insulation check: the right-hand side of one of the SWRL rules includes the statement `?summ eeBIM:definition "Thermal insulation check failed"`.

In the area of *Health and Safety (HS)* measures, the "Job Hazard Analysis" (JHA) application proposed by Zhang et al. [42,43] provides another excellent example of a semantic rule checking process. The authors propose to combine an RDF representation of the building model, contained in Tekla Structures, with a number of ontologies and SWRL rules that allow the analysis of the construction project in terms of jobs, tasks, safety procedures, and the resources that are required to allow the safe execution of these job steps. A prototype was implemented in Tekla Structures in the form of a plugin.

A number of example application scenarios has recently been proposed in the context of *semantic data enrichment* or *schema and data transformations*. These kinds of transformation are to some extent similar to what was proposed in Pauwels et al. [26] regarding the automatic transformation of IFC data to X3D data and to STL data (and back). As one of these data transformation approaches, de Farias et al. [44,45] propose the usage of SWRL rules in combination with the ifcOWL ontology, which would allow one to automatically transform data patterns (subset graphs) into parallel and in this case less complex data patterns. The examples used in de Farias et al. [44] particularly propose to simplify IfcRelationship instances following the ifcOWL ontology, and to simplify the representation of external versus internal walls. Automatically generating these parallel representations allows end users to make far simpler and more intuitive queries. A similar proposal was made by van Berlo and van den Helm [46], in this case using N3Logic rules and the EYE engine and focusing even more on a simplified, user-friendly and intuitive query interface.

As all these examples successfully illustrate, the usage of semantic web technologies in Architecture, Engineering and Construction (AEC) industries opens up considerable possibilities in terms of formal rule checking. This rule checking can be useful for a number of different use cases, including building usage analysis, anomaly detection, job hazard analysis, regulation compliance checking, and data transformation.

3. The test environment

Although pilot test-cases have been proposed regarding rule checking in the architectural design and construction industry, no indications are typically given about the performance of the system. We aim to remedy this situation with the performance benchmark proposed in the remainder of this article. We will hereby consider three rule checking procedures, namely SPIN and

Jena [47], EYE [28] and Stardog [48]. As indicated in Section 1.3, these three procedures are chosen because they are very different in kind and thus allow one to make a qualitative comparison across key different approaches (Conclusion in Section 7.3). In order to properly evaluate the three semantic rule checking procedures, we have set up a test environment that allows to compare the test results in a somewhat quantitative manner as well (Conclusion in Section 7.2).

The test environment consists of a TBox, an ABox, an RBox (following the structure displayed in Fig. 1). We will first document in Section 3.1 the ifcOWL ontology or schema which is relied upon in the test environment (TBox in Fig. 1). This ontology was chosen as it is most closely affiliated to the IFC structure of the design models that rule checking processes are supposed to use. In Section 3.2, we will then give an overview of the kinds of building models that have been used in this experiment (ABox in Fig. 1). Section 3.3 outlines which kinds of rules have been experimented with (RBox in Fig. 1) for the performance benchmark. The final component of our test environment is documented in Section 3.4 and consists of the queries that are fired to the application layer that performs the actual rule checking (Integration layer in Fig. 1).

3.1. The ontology

In our test environment, all building models are encoded using the ifcOWL ontology. This ontology has been built up under the impulse of numerous initiatives [49–55]. The most complete overview of the key decisions in constructing this ifcOWL ontology is documented in Pauwels and Terkaj [56]. The ontology that was used for this test is the one that is made publicly available by the BuildingSMART Linked Data Working Group (LDWG) [57,58]. In short, this ontology is very closely aligned with the EXPRESS schema of IFC. It can thus be considered as an OWL version of the full IFC schema and it therefore allows to specify any building information that one would commonly be able to specify using IFC. The full ontology includes 1230 classes, 1578 object properties, 5 data properties, 1627 individuals, and it has a relatively expressive SROIQ(D) level of expressiveness (OWL DL). Furthermore, the following criteria have been used to build the ontology [56] and therefore define the character of the ontology:

1. The ifcOWL ontology must be in OWL2 DL.
2. The ifcOWL ontology should match the original EXPRESS schema as closely as possible.
3. The ifcOWL ontology primarily aims at supporting the conversion of IFC instance files into equivalent RDF files. Thus, herein it is of secondary importance that an instance RDF file can be modelled from scratch using the ifcOWL ontology and an ontology editor.

This ifcOWL ontology is at the moment of writing in a candidate standard status in buildingSMART International, and it is therefore expected to become a reference model for the industrial rule checking processes targeted in our paper.

3.2. The building models

The test environment has one common set of building models following the ifcOWL ontology. These are building information models that were modelled by people outside the current research team for real-world projects. They have been modelled in different BIM modelling environments, including most prominently Tekla Structure (Trimble) and Autodesk Revit. The models were exported to IFC and made available for this project. A number of BIM models are publicly available [59], whereas other models are not disclosed (private models). The test models can be categorised in a number of dimensions, most prominently:

- BIM authoring tool,
- model size,
- IP level.

In total, our test set included 5 undisclosed IFC models and 364 publicly available models. The 5 restricted IFC models were all modelled using Autodesk Revit 2012. The publicly available IFC models were modelled using a range of different BIM environments, and often within different disciplines (architecture, MEP, structure). Table 1 gives an overview of how many public files were available for each BIM environment found (retrieved from IFC file metadata). Most public IFC files were modelled using Tekla Structures (227 out of 364 - 62.3%), followed by unknown or manual (38 out of 364 - 10.4%) and Autodesk Revit (22 out of 364 - 6%). Of course, these numbers are not representative for the market share or market presence of each of these BIM authoring tools; they only illustrate the profile of our test set.

The test files are also quite different in terms of model size. In this regard, we have set an arbitrary distinction between small models (0 to 500,000 IFC instances), medium models (500,000 to 2 million IFC instances), and large models (more than 2 million IFC instances). Most models are relatively small, but a number of larger models were tested as well in this performance benchmark. Table 2 gives an overview of the number of IFC files for each model size range, private as well as public. An indication is also given of the average associated file size (in MB). Note that the number of IFC instances is considered as the reference number for model size. Note that the 321 small-sized IFC models follow an exponential curve (see Fig. 2). In fact, 222 IFC models have less than 30,000 IFC instances (av. equivalent of 1.5 MB).

Each building model was originally available in IFC2X3. These building models were converted to ifcOWL-compliant RDF graphs using the software made available at [59]. This conversion is done prior to the actual performance benchmark test. The conversion resulted in an RDF graph (TTL syntax) for each of the IFC files.

Table 1
Number of public files available for each software environment.

BIM environment	# files
Tekla Structures	227
unknown or manual	38
Autodesk Revit	22
Xella BIM	15
Autodesk AutoCAD	12
iTConcrete	9
SDS	8
Nemetschek AllPlan	7
GraphiSoft ArchiCAD	5
HiCAD	3
IFCEngineDLL	3
MicroStation TriForma	3
STEPCaselib	3
VDI	3
BLISIFC	1
EliteCAD	1
ETABS	1
IfcOpenShell0.5.0-dev	1
RAMCADstudio	1
SketchUpPro	1
Total	364

Table 2
Number of files for each model size (small, medium, large).

# IFC instances	Av. file size (MB)	# files
0 to 500,000	0 to 30 MB	321
500,000 to 2 million	30 to 100 MB	37
more than 2 million	more than 100 MB	11
Total		369

```
ifcowl:hasProperties(?a, ?x)
∧ ifcowl:NominalValue(?x, ?z)
∧ ifcowl:Name(?x, ?y)
∧ ifcowl:RelPropertyDefinition(?b, ?a)
∧ ifcowl:RelObjects(?b, ?c)
∧ ifcowl:IfcWall(?c)
∧ ifcowl:dp_IfcBoolean(?z, true)
∧ ifcowl:dp_IfcIdentifier(?y, "IsExternal")
⇒ aei:IsExternal(?c, true)
```

Listing 1. Example rule in Manchester OWL syntax (dummy example).

3.3. The rules

The performance benchmark experiment relies not only on a representative set of building models and ontology, but also on a representative set of rewrite rules. For this experiment, we have therefore manually built a set of 68 rewrite rules. These 68 rules can be classified in a number of types, depending on content. All 68 rules can be found in [60], in SPIN, SWRL and N3 syntax. All rules are inspired by the work on simplification of ifcOWL graphs, as suggested and initially documented in de Farias et al. [44,45], Pauwels et al. [61], Borgo et al. [62]. As an example, we outline an example rule in Listing 1 using the Manchester OWL syntax. This example was also used in [44]. So, the IF-part of all 68 rules would typically include a complex partial graph following the ifcOWL ontology, whereas the THEN-part of the rules would be a single property that can be inferred from that rule. As a simple example, the first 8 lines in Listing 1 show the very complex way in which one can state that an `IfcWall` is external in ifcOWL, and the last line in Listing 1 then shows the much simpler direct relationship `aei:isExternal` that is generated directly from `IfcWall` instance to the boolean value `true`.

As can be seen in Listing 1, each rule consists of an antecedent and a consequent, distinguishable by the ⇒ symbol. Both antecedent and consequent are conjunctions of atoms written $a_1 \wedge \ldots \wedge a_n$. Variables are indicated using the standard convention of prefixing them with a question mark (e.g., ?x). So, the IF-part of the example in Listing 1 shows 8 statements that are joined (conjuction symbol ∧) into one graph with variables. It is important to clarify that the terms in these 8 statements are the exact same (i.e. they have the exact same Uniform Resource Identifier - URI) as the ones used in the TBox (the ifcOWL ontology - Section 3.2), hence the ifcowl: prefix. So, the objectProperty ifcowl:RelObjects is defined in the ifcOWL ontology. As the ABox (building model) follows the terminology in the TBox (ifcOWL ontology), this same term, also with the exact same URI, is also used in the representation of the building model (Section 3.1). An inference engine thus integrates (see integration layer in Fig. 1) the ABox, TBox and RBox using these terms as defined in the ifcOWL ontology. This immediately explains the importance of having one commonly agreed ifcOWL ontology.

In the following subsections, we outline what kinds of rules are included. We hereby distinguish between 7 rule sets of varying size.

3.3.1. Rule set 1: Rules including property sets (PSet)

A first useful set of rules in combination with the ifcOWL ontology involves the rewriting of property set references. The IFC schema includes a quite complex structure that allows to assign the same *set of properties* to a number of different IFC instances (n-ary relation). For example, the same set of material properties (e.g. brick physical properties) is allowed to be described once and then attached to a number of different walls of different types. The set of properties serves as a package that includes all the individual properties. The representation of this construction in IFC tends to become relatively complex, however.

Using rules, such complex representations can be automatically transformed / rewritten into alternative, more direct and simpler constructs, as proposed in de Farias et al. [44,45], Pauwels et al. [61], Borgo et al. [62]. The rules displayed in Listing 2 and 3 display how these contructs are rewritten in additional single property statements `aei:hasPropertySet` and `aei:hasProperty`. These two single rules compose rule set 1 (RS1). This is a small, yet often

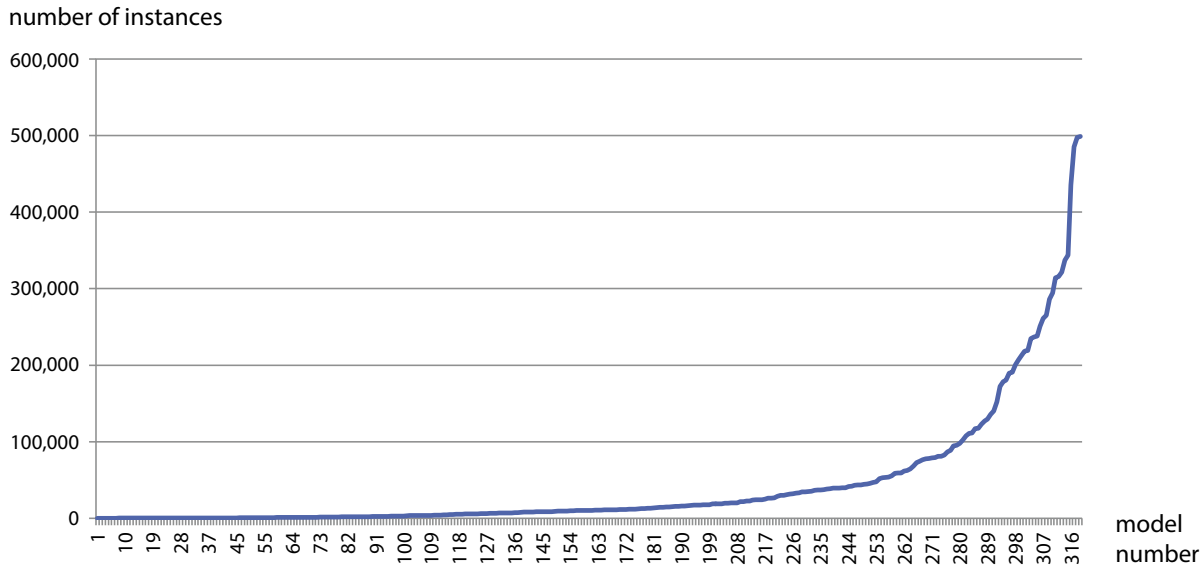

Fig. 2. The set of small IFC models consists for 68.2% of IFC models with a size smaller than 30,000 IFC instances (av. 1.5 MB).

```
ifcowl:IfcObject(?obj)
∧ ifcowl:IfcRelDefines(?rel)
∧ ifcowl:relatedObjects_IfcRelDefines(?rel, ?obj)
∧ ifcowl:
  relatingPropertyDefinition_IfcRelDefinesByProperties(?rel
  , ?pSet)
∧ ifcowl:IfcPropertySet(?pSet)
⇒ aei:hasPropertySet(?obj,?pSet)
```

Listing 2. Rule S1-1 allows to infer `aei:hasPropertySet` properties that directly link objects and their property sets.

```
aei:hasPropertySet(?obj, ?pSet)
∧ ifcowl:hasProperties_IfcPropertySet(?pSet, ?p)
∧ ifcowl:IfcProperty(?p)
⇒ aei:hasProperty(?obj, ?p)
```

Listing 3. Rule S1-2 relies on rule S1-1 (Listing 2) and allows to infer `aei:hasProperty` properties that directly link objects to individual properties.

used rule set that can be used in many contexts to simplify querying and data publication of common simple properties attached to IFC entity instances.

Listing 2 looks for any `IfcObject` instance in the RDF building model (line 1) that is involved in a relationship `IfcRelDefines` (lines 2 and 3) with an `IfcPropertySet` instance (lines 4–7). For any such object, a new object property is then declared that links the `IfcObject` instance directly to that `IfcPropertySet` instance (without all the intermediate nodes and edges), using a new `aei:hasPropertySet` link. Listing 3 then looks for any `IfcObject` that has an `aei:hasPropertySet` property (line 1), including its relations to the actual `IfcProperty` instances (lines 2 and 3). For each such property, a new `aei:hasProperty` link is generated between the object and the property, so that any `IfcObject` (e.g. `IfcWall`) now has a direct link to each of its properties in addition to the regular, but rather complex ifcOWL structure.

3.3.2. Rule set 2: Rules including subtypes of `IfcRelationship`

As the IFC schema allows to define entities and properties, it also allows to define the relationships between the available entities. For these relationships, IFC relies on a particular IFC entity, namely `IfcRelationship`. The `IfcRelationship` is defined as the supertype of `ifcowl:IfcRelAssigns`, `IfcRelDecomposes`, `IfcRelAssociates`, `IfcRelDefines`, `IfcRelConnects`. As `IfcRelDefines` is prominently present in Section 3.3.1, the rules in RS1 could also be considered as examples for rule set 2 (RS2).

A slight difference between `ifcowl:IfcRelDefines` versus `ifcowl:IfcRelAssigns`, `IfcRelDecomposes`, `IfcRelAssociates`, and `IfcRelConnects` is that the latter four are typically used for expressing relationships *between* IFC instances (composition, aggregation, assocation), rather than relationships of IFC entities with property values that typically end into simple data type values (string, boolean, etc.). We consider two example rules here, which are displayed in Listings 4 and 5.

```
ifcowl:IfcRelAggregates(?a1)
∧ ifcowl:relatedObjects_IfcRelDecomposes(?a1, ?a0)
∧ ifcowl:relatingObject_IfcRelDecomposes(?a1, ?a2)
∧ ifcowl:IfcElement(?a0)
∧ ifcowl:IfcElement(?a2)
⇒ aei:hasElementComposition(?a0, ?a2)
```

Listing 4. Rule S2-1 retrieves all occurrences of `ifcowl:IfcRelAggregates` that indicate an aggregation relationship between two `IfcElement` entities.

```
ifcowl:IfcRelAssociatesMaterial(?a1)
∧ ifcowl:relatedObjects_IfcRelAssociates(?a1, ?a0)
∧ ifcowl:relatingMaterial_IfcRelAssociatesMaterial(?a1, ?a2
  )
∧ ifcowl:IfcObjectDefinition(?a0)
∧ ifcowl:IfcMaterial(?a2)
⇒ aei:hasMaterial(?a0, ?a2)
```

Listing 5. Rule S2-1 retrieves all occurrences of `ifcowl:IfcRelAssociatesMaterial` that link an object with a particular material (`ifcowl:IfcMaterial`).

Listing 4 shows in the first 5 lines the structure that is imposed by the ifcOWL ontology (as it is also the standard IFC structure) for declaring that one `IfcElement` (lines 2 and 5) is the aggregation of a number of other elements (lines 3 and 4). For each of these complex aggregation links, a simple link is generated that directly links the element with its composing elements (last line). Listing 5 has an entirely identical structure as Listing 4, but it applies to the relation between an `IfcMaterial` and those objects that have this material associated to them. Rule set 2 includes 31 rules in total, all following the same structure as in Listings 4 and 5. Using this rule set, complex constructs using `IfcRelationship` can be rewritten in far less complex and more direct single relations between IFC elements.

3.3.3. Rule set 3: Rules handling lists

The third rule set (RS3) is related to the usage of lists in the IFC schema. These list constructs are particularly cumbersome and complex, hence they could be rewritten in a shorter and more human-friendly manner (see also [61]). The impact of these list constructs is particularly strong for `IfcCcartesianPoint` instances, because these list constructs are used for representing ordered coordinates (x, y, z) of Cartesian points *and* IFC building models typically have a very large number of Cartesian points. Therefore, we focus here particularly on the x-, y-, and z-coordinates of a Cartesian point. This results in three very similar rules for RS3, of which only one is displayed in Listing 6 (namely for the x-coordinate only).

Listing 6 shows in the IF-side of the rule an `IfcCartesianPoint` and its single `coordinates_IfcCartesianPoint` property. This property points to a list of `IfcLengthMeasure` instances. The THEN-part of this rule shows how this rule allows to directly link the value of each of these `IfcLengthMeasure` instances to the Cartesian point it belongs to, using a direct object property `hasCoordinateX` (similar rule for y- and z-coordinates).

3.3.4. Rule set 4: Simple data types

In the EXPRESS to OWL conversion procedure that is adopted to obtain the ifcOWL ontology used here (see [58,56]), one of the main criteria is to stay as close as possible to the original EXPRESS schema with the ifcOWL ontology. This has resulted in the requirement to 'wrap' simple data types into `owl:Class` instances that refer with an additional data property (`express:hasString`, `express:hasDouble`) to the actual value of the 'wrapped simple data type class'. This is a lengthy construct for a rather simple

```
ifcowl:IfcCartesianPoint(?x)
∧ ifcowl:coordinates_IfcCartesianPoint(?x, ?y)
∧ ifcowl:IfcLengthMeasure_List(?y)
∧ list:hasContents(?y, ?z)
∧ expr:hasDouble(?z, ?val)
⇒ aei:hasCoordinateX(?x, ?val)
```

Listing 6. Rule S3-1 retrieves the x-coordinate of a Cartesian point from the list in which it is recorded.

statement. An example of this construct can be seen in the IF-part of the rule in Listing 7. It shows how any IfcRoot instance (i.e. any object in an IFC file) has a name_IfcRoot object property, which points to an IfcLabel instance, which in turn has a data property hasString. Similar constructs are used for any value (string, int, double, boolean, ...) in IFC.

The rules in this fourth rule set (RS4) allow one to simplify these constructs. At the moment, this rule set includes but one example rule, namely the one in Listing 7, but it can include many more similar rules. The THEN-part of this rule clearly shows how a data property name is inferred, which directly links the IfcRoot object with the value.

3.3.5. Rule set 5: Property shortcuts

The fifth rule set (RS5) extends RS1, which allows to infer the single property statements aei:hasPropertySet and aei:hasProperty. Rule set 5 consists of 20 rules that all follow the structure displayed in Listings 8 and 9 using two of the twenty rules as examples. The rules differ only in the property name (line 3) and in the simple data type that is associated with it (string, boolean, integer - line 5).

The rules in Listings 8 and 9 allow one to infer two direct data properties that connect an object instance (subject of aei:hasProperty) with a precise data value for a specific property, in this case (aei:isExternal and aei:hasTotalArea). It is important to point out that both examples rely on the object property aei:hasProperty, which can be inferred using RS1.

3.3.6. Rule set 6: Internal and external elements

We have included the sixth rule set (RS6) as it further extends RS5 (Section 3.3.5) and RS1 (Section 3.3.1). More particularly, the 6 rules in RS6 rely directly on the inference result of Rule S5-1, which infers whether an object is external or internal to the building. All 6 rules in the rule set have the same structure, differing only in the usage of aei:isExternal false versus aei:isExternal true, and in the usage of IfcDoor, IfcWindow, and IfcWall. An example rule is given in Listing 10 for an aei:ExternalWall. This rule looks for any individual of type IfcWall (line 2). If that individual also has the data property aei:isExternal true (line 1), this wall can be classified as an ExternalWall (line 3). Note that also here, the property in line 1 (aei:isExternal) is in the THEN-part of rule S5-1, which again has the property aei:hasProperty in its IF-part, which is again in the THEN-part of S1-2. These three thus form a clear chain that can be followed by a reasoning engine in forward or backward direction.

3.3.7. Rule set 7: Composition and decomposition

Rule set 7 (RS7) finally includes five rules, of which four are displayed in Listing 11–14. They are all related specifically to composition and decomposition, both of spaces and spatial elements. The rule in Listing 11 allows to infer all spaces that are located in a floor with an elevation higher than 0, and to classify them as such (aei:ElevatedSpace). This rule in particular includes the usage of a mathematical requirement, as one of the parameters needs to be higher than 0.

```
ifcowl:IfcRoot(?x1)
∧ ifcowl:name_IfcRoot(?x1, ?x2)
∧ ifcowl:IfcLabel(?x2)
∧ expr:hasString(?x2, ?x3)
⇒ aei:name(?x1, ?x3)
```

Listing 7. Rule S4-1 retrieves the name property for any instance of ifcowl:IfcRoot and reassigns it using a shorter notation form.

```
aei:hasProperty(?obj, ?p)
∧ ifcowl:name_IfcProperty(?p, ?name)
∧ expr:hasString(?name, "IsExternal"^^xsd:string)
∧ ifcowl:nominalValue_IfcPropertySingleValue(?p, ?val)
∧ expr:hasBoolean(?val, ?str)
⇒ aei:isExternal(?obj, ?str)
```

Listing 8. Rule S5-1 relies on rule S1-2, which in turn relies on rule S1-1 (Listing 2 and 3), and allows to infer the value for the property aei:isExternal, which is typically associated to door, window, and wall elements.

```
aei:hasProperty(?obj, ?p)
∧ ifcowl:name_IfcProperty(?p, ?name)
∧ expr:hasString(?name, "TotalArea"^^xsd:string)
∧ ifcowl:nominalValue_IfcPropertySingleValue(?p, ?val)
∧ expr:hasDouble(?val, ?str)
⇒ aei:hasTotalArea(?obj, ?str)
```

Listing 9. Rule S5-2 relies on rule S1-2, which in turn relies on rule S1-1 (Listing 2 and 3), and allows to infer the value for the property aei:hasTotalArea, which is typically used for calculating floor areas.

```
aei:isExternal(?wall, true)
∧ ifcowl:IfcWall(?wall)
⇒ aei:ExternalWall(?wall)
```

Listing 10. Rule S6-1 relies on rule S5-1 (Listing 8), which furthermore relies on rule S1-2 and S1-1 (Listing 2 and 3). Rule S6-1 allows to infer whether a wall is of type aei:ExternalWall or not.

```
ifcowl:IfcBuildingStorey(?floor)
∧ ifcowl:elevation_IfcBuildingStorey(?floor, ?elev)
∧ expr:hasDouble(?elev, ?y)
∧ math:greaterThan(?y, "0"^^xsd:double)
∧ ifcowl:isDecomposedBy_IfcObjectDefinition(?floor, ?rel)
∧ ifcowl:relatedObjects_IfcRelDecomposes(?rel, ?x)
∧ ifcowl:IfcSpace(?x)
⇒ aei:ElevatedSpace(?x)
```

Listing 11. Rule S7-1 retrieves all ifcowl:IfcSpace entities that are in a floor with an elevation higher than 0, and classifies these ifcowl:IfcSpace entities as aei:ElevatedSpace entities.

```
ifcowl:IfcRelSpaceBoundary(?a1)
∧ ifcowl:relatedBuildingElement_IfcRelSpaceBoundary(?a1, ?
  a2)
∧ ifcowl:relatingSpace_IfcRelSpaceBoundary(?a1, ?a0)
∧ ifcowl:IfcSpace(?a0)
∧ ifcowl:IfcElement(?a2)
⇒ aei:hasSpaceBoundaries(?a0, ?a2)
```

Listing 12. Rule S7-2 allows to retrieve the elements that form the boundary of a space entity.

```
aei:hasSpaceBoundaries(?space1, ?d)
∧ ifcowl:IfcDoor(?d)
∧ aei:hasSpaceBoundaries(?space2, ?d) .
∧ log:notEqualTo(?space1, ?space2)
⇒ aei:isConnectedByDoorTo(?space1 ?space2)
```

Listing 13. Rule S7-3 retrieves all ifcowl:IfcSpace entities that are connected with each other by means of a common door in a common space boundary.

The second and third rule of this rule set are closely related. Rule S7-2 is displayed in Listing 12 and identically follows the

```
ifcowl:IfcRelVoidsElement(?a1)
∧ ifcowl:relatingBuildingElement_IfcRelVoidsElement(?a1, ?
  a0)
∧ ifcowl:relatedOpeningElement_IfcRelVoidsElement(?a1, ?a2)
∧ ifcowl:IfcElement(?a0)
∧ ifcowl:IfcOpeningElement(?a2)
⇒ aei:hasElementVoiding(?a0, ?a2)
```

Listing 14. Rule S7-4 retrieves all `ifcowl:IfcElement` instances that include an `ifcowl:IfcOpeningElement`, and connects both directly with the `aei:hasElementVoiding` property.

structure of the rules in RS2 (Section 3.3.2). This rule can thus also be classified in this rule set. Indeed, the rule includes at its core a subtype of `IfcRelationship`, namely `ifcowl:IfcRelSpaceBoundary`. This relationship entity allows to retrieve the elements (`IfcElement`) that bound a space (`IfcSpace`). The THEN-part of ruls S7-2 thus includes a direct object property (`aei:hasSpaceBoundaries` - line 6) between the `IfcSpace` individuals (line 4) and those `IfcElement` individuals (line 5) that are related through them through a `ifcowl:IfcRelSpaceBoundary` individual (lines 1–3).

Rule S7-2 can easily be combined with rule S7-3 to infer a more particular and concrete property that is implicitly present in an IFC model. Rule S7-3 namely allows to automatically infer whether a door is present that connects two distinct `ifcowl:IfcSpace` entities (see Listing 13). This rule relies on the property `aei:hasSpaceBoundaries` that can be inferred using rule S7-2 in Listing 12. In addition, this rule only applies when the two `ifcowl:IfcSpace` entities are not the same (distinct URI). This is included in Listing 13 using the property `log:notEqualTo`.

Rules S7-4 (Listing 14) and S7-5 depend on each other as well. Rule S7-4 allows to retrieve `ifcowl:IfcElement` instances and the `ifcowl:IfcOpeningElement` instances that reside in them. Whenever both are found (lines 4 and 5 in Listing 14), they are directly related using the object property `aei:hasElementVoiding` (line 6). Rule S7-5 allows to furthermore retrieves all `ifcowl:IfcElement` instances that have such an `aei:hasElementVoiding` property and also an `ifcowl:IfcElement` that fills the same void. The THEN-part of rule S7-5 the infer an `aei:hasElementFilling` object property.

3.4. The queries

With the above components, our test environment contains a large number of the key components that are necessary to build a rule checking system (see Fig. 1). The only part that is missing, is the arrow between the GUI and the integration layer in Fig. 1. In terms of semantic web technologies, this part is to be accommodated using *queries*. These queries can be accommodated in a number of ways, SPARQL queries and API calls being the most common ones.

We have used a limited list of queries to perform the performance benchmark tests. These queries are made available in [60], both in SPARQL and N3 syntax. As the focus of this article is on the rule checking itself, the considered queries are entirely based on the right-hand sides of the rules in Section 3.3. More precisely, each query statement queries for one of the object of data properties that is present in the THEN-part of a rule. E.g. `aei:hasProperty` of Listing 3 is queried for in query 31. They are deliberately kept short, simple, and thus often self-explanatory. A list of the 60 queries that have been used, can be found in Appendix A.

3.5. Running the experiments

In the previous sections, a detailed outline was given for each of the components in our rule checking schema in Fig. 1. All in all, the following data are available for the performance benchmark experiment. These data are enumerated below and made available at [60].

- ontologies (TBox in Fig. 1)
  - the ifcOWL ontology (IFC2X3_TC1.TTL)
- data (ABox in Fig. 1)
  - six representative building models (of the 364 + 5) in TTL syntax
- rules (RBox in Fig. 1)
  - RSTOTAL: SPIN, SWRL and N3
  - RS1_PSets: SPIN, SWRL and N3
  - RS2_relationship: SPIN, SWRL and N3
  - RS3_listsCartP: SPIN, SWRL and N3
  - RS4_simpledatatypes: SPIN, SWRL and N3
  - RS5_simpleproperties: SPIN, SWRL and N3
  - RS6_internalexternal: SPIN, SWRL and N3
  - RS7_spatialcompdecomp: SPIN, SWRL and N3
  - the RDFS vocabulary rule set (rdfsvocab.n3)
- queries (queries in Fig. 1)
  - all 60 query files in SPARQL and N3 syntax

At the center of the test environment is a central server that is used by all implementations to perform the tests. This server was supplied by the University of Burgundy, research group CheckSem, and had the following specifications:

- Ubuntu OS
- Intel Xeon CPU E5-2430 at 2.2 GHz
- 6 cores
- 16 GB of DDR3 RAM memory

Three Virtual Machines (VMs) were set up in this central server and managed as separate test environments. Each of these VMs had 2 cores out of 6 allocated and each contained the above resources (ontologies, data, rules, queries).

- SPIN VM: Jena TDB [47]
- EYE VM: EYE inference engine [28]
- STARDOG VM: Stardog triplestore [48]

An appropriate performance benchmark test requires each query to be run a number of times for each building model. If we run each query 5 times, this would imply $369m \times 60q \times 5 = 110,700$ or more than 110,000 queries for which performance results would become available. Each set of 110,000 queries should be run three times (one time in each environment), which leads to a total of 330,000 queries. Obviously, we cannot document all statistics in full detail in the remainder of this paper. Furthermore, not every query and building model is equally useful. Namely, the building models have different information included, because they were built / modeled in different contexts and purposes. Furthermore, some of the building models are in fact small subsets of a larger building model. So, the building models do not necessarily include all of the properties that were listed in the queries of Section 3.4. E.g. if a building model does not have a staircase (hasNumberOfRiser) or slope (hasSlope), it makes no sense to test how long it takes to query for a staircase or a slope.

Therefore, we will limit ourselves to documenting a combination of six key building models (large to small - see Table 3) and three key queries: a simple query with little results, a simple query with many results, and a complex query that triggers a

Table 3
The building models that will be used for documentation of the performance benchmark in the following sections.

Model name	IFC file size (bytes)	TTL file size (bytes)	Triple count
202672_tractiewerkplaats_Phase 1	429,033,456	1,525,392,784	19,104,134
wereld van theater en verwondering	162,175,017	1,573,743,569	19,652,142
<Private>	232,562,104	1,888,079,836	23,466,825
technisch_instituut_sint_michiel	140,467,471	455,569,422	5,800,067
091210Med_Dent_Clinic_Arch	15,472,026	106,361,365	1,320,515
Duplex_A_20110505	2,366,050	17,608,893	225,135

considerable number of rules (see Table 4). In addition, we choose to focus on those queries and building models that do return results.

The three test environments are discussed in more detail in the following sections (Sections 4–6).

4. Procedure 1: Rule checking over SPARQL and SPIN

Fig. 3 provides an overview of the components that are usually involved in this rule checking implementation approach [63]. In the following sections, we will first present a brief description of the overall implementation, after which we document some essential details related to performance in the given test environment (access to building models and rule examples). Section 4.3 finally gives an indication of the performance results.

4.1. Overview of the implementation approach

This test environment is implemented based on the open source APIs of Topbraid SPIN and Apache Jena [64] [47]. The used revisions of key packages are listed as follows:

- SPIN API 1.4.0
- Jena Core 2.11.0
- Jena ARQ 2.11.0
- Jena TDB 1.0.0

SPARQL is a W3C standard for querying RDF data [39]. As a standard query language, SPARQL is widely implemented by almost all RDF stores. Hence, an advantage of this procedure is that it is relatively easy to reuse defined queries or rules in many other platforms.

SPARQL Inferencing Notation (SPIN) is a W3C Member Submission, which provides an RDF syntax for SPARQL and thereby enables storing queries with RDF data [64]. Listing 15 illustrates a SPIN example, which represents a SPARQL `CONSTRUCT` query for the rule S1-2 presented in Listing 3. SPIN also defines a modelling vocabulary to structure SPARQL queries as rules and functions and arrange their execution order [65]. The SPIN API used in this test environment is developed based on Apache Jena, and its reasoning engine is based on the Jena ARQ query engine. Running SPIN rules can be considered as a set of SPARQL queries which are fired iteratively following random or arranged sequences.

As illustrated in Fig. 3, the Jena TDB triple store is used as the repository for RDF data and OWL ontologies. Each of the RDF files listed in Table 3 is loaded into the TDB store as a separate named graph. The reasoning engines of Jena and SPIN are used to produce inferences using the statements in the ifcOWL ontology and SPIN rules respectively. Instance queries listed in App. A are processed by the Jena ARQ query engine.

Table 4
The queries that will be used for documentation of the performance benchmark in the following sections.

Number	Query
Q1	?obj aei:hasProperty?p
Q31	?point aei:hasCoordinateX?x. ?point aei:hasCoordinateY?y. ?point aei:hasCoordinateZ?z
Q53	?d rdf:type aei:ExternalWall

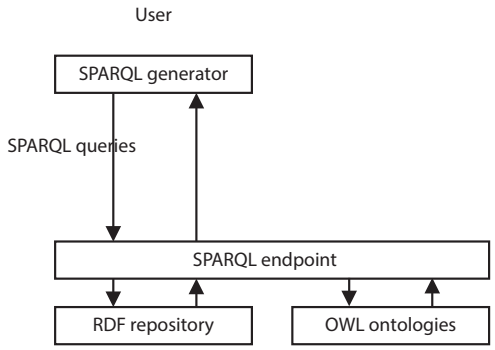

Fig. 3. Indication of the setup that is typically used in the rule checking implementation approach relying on SPARQL and SPIN.

```
owl:Thing
  spin:rule
    [ rdf:type sp:Construct ;
      sp:templates (
        [ sp:object
          [ sp:varName "p"^^xsd:string ] ;
        sp:predicate aei:hasProperty ;
        sp:subject
          [ sp:varName "obj"^^xsd:string ]
      ]) ;
      sp:where (
        [ sp:object
          [ sp:varName "pSet"^^xsd:string ] ;
        sp:predicate aei:hasPropertySet ;
        sp:subject
          [ sp:varName "obj"^^xsd:string ]
        ]
        [ sp:object
          [ sp:varName "p"^^xsd:string ] ;
        sp:predicate ifcowl:hasProperties_IfcPropertySet ;
        sp:subject
          [ sp:varName "pSet"^^xsd:string ]
        ]
        [ sp:object ifcowl:IfcProperty ;
        sp:predicate rdf:type ;
        sp:subject
          [ sp:varName "p"^^xsd:string ]
      ])
    ] ;
```

Listing 15. The SPIN RDF representation for a SPARQL CONSTRUCT query which describes the rule S1-2.

4.2. Details of the implementation in the test environment

Rules listed in Section 3.3 are written in SPARQL and are converted to RDF syntax using the SPIN framework (cfr. Listing 15). These files are available in [60] (SPIN hyperlinks). A small Java program is implemented to read RDF models, schema, rules from the

TDB store and query data. All the SPARQL queries are configured using the Jena `org.apache.jena.sparql.algebra` package to optimise the processing sequence of their triple patterns in order to achieve a better performance.

4.2.1. Methods of writing rules

In the SPIN framework, rules can be written in two ways, allowing either a forward chaining or backward chaining reasoning process. A forward chaining process executes all rules iteratively until no more new facts are generated. This typically occurs before query execution time, resulting in a fully saturated RDF graph stored on disk. No more inferences need to be made at query execution time in this case. It reduces time cost in query execution time, but is usually wasteful in total since it materialises many triples which are not relevant for the final queries. A backward chaining process is goal-driven and it only processes rules which are related to queries, which will tremendously reduce triple materialisation and the total processing time. Choosing which reasoning strategy is essential for the performance of a rule checking environment.

Forward chaining rules are defined using SPARQL `CONSTRUCT` form to capture the `IF-THEN` structure in the SPIN framework. An example is presented in Listing 16 for rule S6-1 (Listing 10), which declares that an `IfcWall` instance which has an `isExternal` property with value `true` is an `ExternalWall`. In the SPIN framework, this query is converted to the RDF syntax as an instance of `sp:Construct` (see Listing 15) and associated with a general concept, e.g. `owl:Thing` or `rdfs:Resource`, using the `spin:rule` property.

Backward chaining rules are implemented by encapsulating a `SELECT` query as a SPARQL property function. For example, the property of `aei:isExternal` in Listing 16 is associated with a `SELECT` query given in Listing 17. In this method, the variable `arg1`, the defined property and the query result are by default interpreted respectively as the subject, predicate and object of the rule consequent statement. This method enables a goal-driven backward chaining strategy–the associated `SELECT` query will only be fired as a sub-query when this property is called in a SPARQL query. A limitation is that some rules cannot be defined using this strategy since their rule consequences use semantically standardized predicates. For example, the rule defined in Listing 10 cannot be written in this method because its consequence uses the predicate `rdf:type`, which is a standardized property representing the "is-a" relationship.

In this test environment, most of the rules are written as backward chaining rules (second method). However, there are in total 7 rules listed in Rule set 6 and 7 that can only be defined using the first method (forward chaining). Therefore, there are both forward chaining and backward chaining rules defined in this test environment (hybrid reasoning process).

4.2.2. Reasoning process

The reasoning process is illustrated in Fig. 4. As described above, rules are written in two ways using the SPIN framework (forward and backward). Forward chaining SPIN rules are firstly pre-processed by the SPIN engine. Those rules that include SPIN functions or statements in OWL ontologies will fire them using the SPIN engine and the Jena engine respectively. This process continues recursively until no more SPIN function is called. Generated triples will go through this process iteratively until no more new facts are generated. At query execution time, the SPARQL query instances also fire backward chaining SPIN functions or OWL statements if they are related to the query.

In this test environment, the Jena reasoner is configured as an RDFS reasoner to avoid unnecessary reasoning processes. It means that although the ifcOWL ontology is developed using the OWL DL profile, only the RDFS vocabulary is supported in this environment.

```
CONSTRUCT {
    ?x a aei:ExternalWall .
  }
WHERE{
    ?x a ifcowl:IfcWall .
    ?x aei:isExternal true .
  }
```

Listing 16. Rule S6-1 declares that an `IfcWall` which has an `isExternal` property of `true` is an `ExternalWall`.

```
SELECT ?bool
WHERE {
    ?arg1 aei:hasProperty ?property .
    ?property ifcowl:name_IfcProperty ?name .
    ?name expr:hasString "IsExternal" .
    ?property ifcowl:nominalValue_IfcPropertySingleValue ?
      val .
    ?val expr:hasBoolean ?bool .
  }
```

Listing 17. Rule 12 retrieves the `"IsExternal"` data property for an object and reassigns it using a shorter notation `aei:isExternal` when this property is called in other queries.

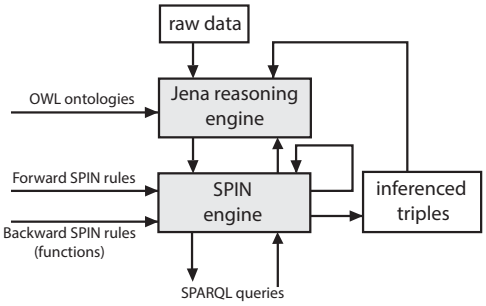

Fig. 4. Data flow of the reasoning process based on two reasoners.

4.3. Performance results

This section gives an indication of the performance results. The performance documented in this section includes the pre-processing time of networking and forward chaining rules, and the query execution time. Table 5 lists the performance in terms of execution time of pre-processing including reasoning with all forward chaining rules (column 2). It also provides the triple count of each RDF model (column 1) and added new triples (column 3). Table 6 documents query execution times of query Q1, Q31 and Q53.

As is shown in Table 5, because only 7 rules are defined as forward chaining rules, the added triples in the pre-processing phase are limited. In Table 6 and Fig. 5, we can see that the query

Table 5
Performance results for pre-processing forward chaining SPIN rules.

Model name	Triple count	Avg. reasoning (s)	Added triples
202672_tractiewerkplaats_Phase 1	19,104,134	8.103	0
wereld van theater en verwondering	19,652,142	1.532	320
<Private>	23,466,825	53.356	4600
technisch_instituut_sint_michiel	5,800,067	1.342	0
091210Med_Dent_Clinic_Arch	1,320,515	23.876	1878
Duplex_A_20110505	225,135	2.117	97

Table 6
Performance results for query Q1, Q31 and Q53 on pre-processed models.

Query	Model name	Triple count	Avg. querying (s)	Stand. derivation	Result count
Q1	202672_tractiewerkplaats_Phase 1	19,135,302	135.364	13.315	3,016,218
Q1	wereld van theater en verwondering	19,683,630	1.469	0.321	23,581
Q1	<Private>	23,502,593	24.009	1.764	394,875
Q1	technisch_instituut_sint_michiel	5,831,235	41.276	4.930	1,016,660
Q1	091210Med_Dent_Clinic_Arch	1,353,561	4.989	0.206	86,243
Q1	Duplex_A_20110505	256,400	0.552	0.042	12,597
Q31	202672_tractiewerkplaats_Phase 1	19,135,302	46.172	3.990	213,627
Q31	wereld van theater en verwondering	19,683,630	92.028	12.328	425,934
Q31	<Private>	23,502,593	82.676	7.849	415,308
Q31	technisch_instituut_sint_michiel	5,831,235	19.926	2.625	102,127
Q31	091210Med_Dent_Clinic_Arch	1,353,561	3.685	0.630	21,109
Q31	Duplex_A_20110505	256,400	0.738	0.115	4,481
Q53	202672_tractiewerkplaats_Phase 1	19,135,302	0.0013	0.00064	0
Q53	wereld van theater en verwondering	19,683,630	0.0057	0.0051	235
Q53	<Private>	23,502,593	0.0017	0.0006	168
Q53	technisch_instituut_sint_michiel	5,831,235	0.0051	0.0079	0
Q53	091210Med_Dent_Clinic_Arch	1,353,561	0.0058	0.0009	1,471
Q53	Duplex_A_20110505	256,400	0.001	0.0008	23

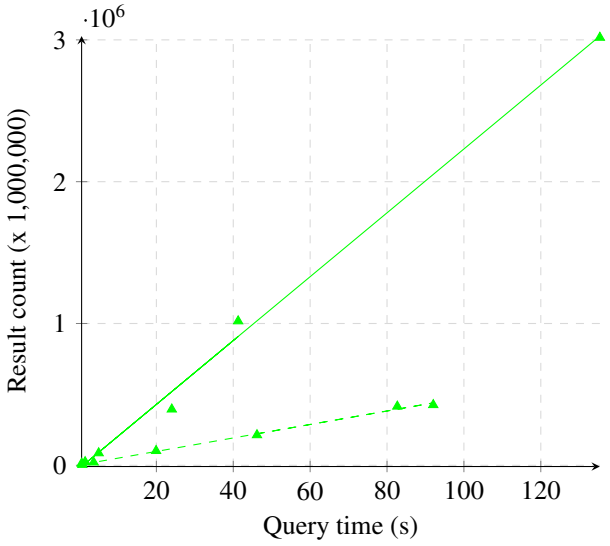

Fig. 5. Indication of the relationship between query execution time (*x*-axis) and result count (*y*-axis) for Q1 (solid) and Q31 (dashed) in Procedure 1.

execution times of Q1 and Q31 have almost linear relationships with their respective result count, and the solid line which represents Q1 has a greater slope. These results are both expected because matching more triples needs longer time and Q1 has less triple patterns. Q53 has far better performance in comparison to the other two queries. This is mainly because it does not fire any SPIN rules in its query execution time, while Q1 and Q31 both fire 3 rules (backward chaining). The Rule S6-1 (Listing 10), which is closely related to Q53 has been defined as a forward chaining rule, hence all the instances of external walls have been explicitly stated before query execution time. In applications, end users can choose to define more forward chaining rules to move more reasoning processes to the pre-processing phase to improve the query performance.

5. Procedure 2: Rule-checking using EYE and N3Logic

Fig. 6 provides an overview of the components that are involved in the second rule checking implementation approach. In the following sections, we will first present a brief description of each

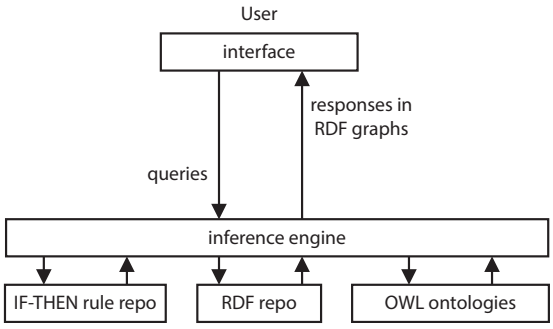

Fig. 6. Indication of the setup that is used in the rule checking implementation approach relying on the EYE inference engine and N3Logic rules.

of the components in this schema, after which we document the way in which this approach works in the given test environment (building models, rules, queries, ontologies).

5.1. Overview of the implementation approach

Unlike the first rule checking approach in Section 4, the approaches documented in Sections 5 and 6 rely entirely on dedicated rule languages instead of the SPARQL query language. The approaches in Sections 5 and 6 rely more particularly on the SWRL [36] and N3Logic rule languages [66]. SWRL was proposed as one of the first semantic rule languages in [36]. Both an abstract and an XML concrete syntax were proposed to describe rules. This allows rules to be expressed both in XML and in a more human-readable abstract syntax. N3Logic is an alternative rule language, proposed in Berners-Lee et al. [66]. Unlike SWRL, this rule language is based on Notation-3 (N3) as its normative syntax, which was presented as a readable alternative for RDF/XML by Berners-Lee and Connolly [67]. Listings 18 and 19 illustrate the same rule in both languages, without prefix declarations, namely the rule that was already illustrated in Listing 3 and for which a SPIN representation was given in Listing 15.

It is important to be clear about the expressiveness of SPARQL versus N3 versus SWRL. In this regard, the graph in Fig. 7 provides a schema of what expressions are available in each of the three languages and more. N3Logic relies on the Full N3 syntax (upper right in Fig. 7), SPARQL is in the SPARQL syntax, and SWRL is in

```
[ rdf:type swrl:Imp ;
  swrl:body [ rdf:type swrl:AtomList ;
      rdf:first [ rdf:type swrl:IndividualPropertyAtom ;
        swrl:propertyPredicate aei:hasPropertySet ;
        swrl:argument1 <urn:swrl#obj> ;
        swrl:argument2 <urn:swrl#pSet>
      ] ;
      rdf:rest [ rdf:type swrl:AtomList ;
        rdf:first [ rdf:type swrl:IndividualPropertyAtom ;
          swrl:propertyPredicate :
            hasProperties_IfcPropertySet ;
          swrl:argument2 <urn:swrl#p> ;
          swrl:argument1 <urn:swrl#pSet>
        ] ;
        rdf:rest [ rdf:type swrl:AtomList ;
          rdf:rest rdf:nil ;
          rdf:first [ rdf:type swrl:ClassAtom ;
            swrl:classPredicate :IfcProperty ;
            swrl:argument1 <urn:swrl#p>
          ]
        ]
      ]
    ] ;
  swrl:head [ rdf:type swrl:AtomList ;
      rdf:rest rdf:nil ;
      rdf:first [ rdf:type swrl:IndividualPropertyAtom ;
        swrl:propertyPredicate aei:hasProperty ;
        swrl:argument1 <urn:swrl#obj> ;
        swrl:argument2 <urn:swrl#p>
      ]
    ]
] .
```

Listing 18. SWRL notation for rule S1-2 displayed earlier in Listing 3.

```
{
  ?obj aei:hasPropertySet ?pSet .
  ?pSet ifcowl:hasProperties_IfcPropertySet ?p .
  ?p rdf:type ifcowl:IfcProperty
}
=>
{
  ?obj aei:hasProperty ?p
}
```

Listing 19. N3 notation for rule S1-2 displayed earlier in Listing 3.

the RDF/XML syntax (lower left in Fig. 7). The building models in ifcOWL are available in TTL syntax, which is one of the smaller circles in the center of the diagram in Fig. 7. As one can see in this Venn-diagram, full N3 syntax provides keywords like =>, {,}, which are not available in SWRL, nor in SPARQL. As a result, rules represented in N3 syntax are typically less complex and long compared to their representations in SPARQL and/or SWRL.

Besides the usage of the N3Logic rule language, the second rule checking approach relies on the Euler Yet another proof Engine (EYE) [69]. In this test, we relied on the EYE version 'EYE-Win ter16.0302.1557', which in turn relies on 'SWI-Prolog 7.2.3 (amd64): Aug 25 2015, 12:24:59'. EYE is a semibackward reasoner enhanced with Euler path detection [69]. Semibackward reasoning is backward reasoning for EYE components, i.e. rules using <= in N3, and forward reasoning for rules using => in N3. As our rule set currently contains only rules using =>, forward reasoning will take place.

5.2. Details of the implementation in the test environment

The test relies on the resources that are made available in [60] and that were documented in Section 3 (ontology, rules, data, queries). These resources were loaded into the VM. Except for the building models (data) and ifcOWL ontology (IFC2X3_TC1.ttl), only N3 statements were used. Individual commands have been

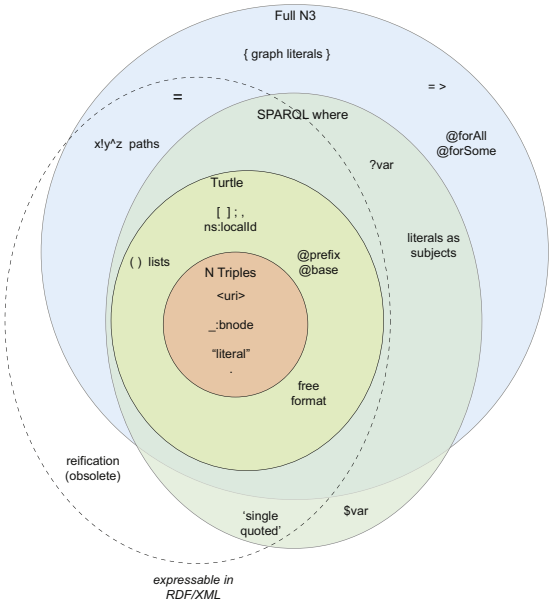

Fig. 7. Venn diagram summarising the expressiveness for the different syntaxes for RDF graphs (original image in Berners-Lee [68], now available at http://www.w3.org/DesignIssues/diagrams/n3/venn).

fired towards the reasoning engine, thus mimicking user interaction (cfr. API calls, query statements). Each command is executed five times to allow calculating the average query execution time.

When following this approach, no triple store is used, unlike the approaches in Sections 4 and 6. Instead, all triples are processed directly from the respective files. This allows to easily reuse files that are readily available on the web. On the other hand, this also means that all loading and all inference still need to be done at run-time. Considering the size of both the building models and the ifcOWL ontology, this can take a long while, even if we limit to RDFS inferences for the ontology. All resources (full building model, full IFC ontology, RDFS vocabulary, rule set, and query) are passed all at once and in an unoptimised manner to the reasoning engine, which then needs to go through all these resources and do a lot of initial networking and reasoning that would better not be done at query execution time (especially not for larger models).

To prevent these long inference runs, it is recommended to configure as much pre-processing as possible, so that this workload does not take place at query execution time. This approach is also taken by the SPIN and Stardog approach in Sections 4 and 6 in the sense that they rely on an optimised triple store with all initial resources.

5.3. Performance results

The presence of a triple store can be mimicked in this EYE approach. The EYE engine relies at its core on Prolog trees. By making all data available in Prolog code, and executing the performance benchmark on this code, a triple store set up is to some extent emulated. 'Networking time' equals the amount of time that is required to load data, rules and queries and convert these into Prolog Virtual Machine (PVM) code, whereas 'reasoning time' equals the amount of time required to answer the queries and emit the results (= query execution + reasoning time). Obtaining this triple store setup is briefly documented here in two steps.

A first step in building an optimal EYE setup, is to decrease the size of the ifcOWL ontology. This ontology contains 30,297 statements. In our reasoning process, however, we only use the RDFS statements of this ontology. It is possible with EYE to generate

Table 7
Performance results for pre-processing resources.

Model name	Triple count	Avg. networking (s)	Avg. reasoning (s)	Result count
202672_tractiewerkplaats_Phase 1	19,104,134	408.88	2,466.79	36,604,298
wereld van theater en verwondering	19,652,142	313.37	8,797.19	43,918,784
<Private>	23,466,825	–	–	–
technisch_instituut_sint_michiel	5,800,067	110.04	998.06	11,770,817
091210Med_Dent_Clinic_Arch	1,320,515	22.25	190.10	2,843,308
Duplex_A_20110505	225,135	4.35	30.27	498,618

Table 8
Performance results for the query execution process from PVM files in Procedure 2.

Query	Model name	Triple count	Avg. networking (s)	Avg. reasoning (s)	Avg. reasoning indexed (s)	Result count
Q1	202672_tractiewerkplaats_Phase 1	36,604,298	23.73	187.93	37.11	3,016,218
Q1	wereld van theater en verwondering	43,918,784	29.05	1.40	0.29	23,581
Q1	<Private>	50,459,123	34.89	23.93	4.87	394,875
Q1	technisch_instituut_sint_michiel	11,770,817	7.44	73.97	12.95	1,016,660
Q1	091210Med_Dent_Clinic_Arch	2,843,308	1.77	5.16	1.05	86,243
Q1	Duplex_A_20110505	498,618	0.28	0.83	0.16	12,597
Q31	202672_tractiewerkplaats_Phase 1	36,604,298	23.76	31.37	2.10	213,627
Q31	wereld van theater en verwondering	43,918,784	28.85	63.28	4.20	425,934
Q31	<Private>	50,459,123	34.42	62.19	4.12	415,308
Q31	technisch_instituut_sint_michiel	11,770,817	7.46	28.32	1.04	102,127
Q31	091210Med_Dent_Clinic_Arch	2,843,308	1.77	3.04	0.21	21,109
Q31	Duplex_A_20110505	498,618	0.28	0.65	0.045	4,481
Q53	202672_tractiewerkplaats_Phase 1	36,604,298	23.60	3.11	0.001	0
Q53	wereld van theater en verwondering	43,918,784	28.78	5.97	0.003	235
Q53	<Private>	50,459,123	34.22	6.74	0.003	168
Q53	technisch_instituut_sint_michiel	11,770,817	7.46	1.03	0.001	0
Q53	091210Med_Dent_Clinic_Arch	2,843,308	1.79	0.44	0.013	1,471
Q53	Duplex_A_20110505	498,618	0.28	0.07	0.001	23

an RDF graph that only includes the ifcOWL statements that might actually be used in this reasoning process. The resulting file (IFC2X3_TC1-parteval-subclass.n3) includes 4,250 statements instead of the original 30,297 statements.

A second step that can be made to further move workload into a pre-processing stage, is to do part of the networking and reasoning beforehand and store the result locally. This mimicks to some extent the triple store setup, except that the triple store still needs to be opened at run-time and is not running in the background. In this approach, a large part of the inferences is already made before end user involvement and does not have to be made at query execution time. It combines the building model, the partial ifcOWL ontology and the total rule set into one fullbuildingmodel file. The times needed for this pre-processing step in our test environment are given in Table 7. In addition, Table 7 lists the initial triple count (2nd column) and the final triple count after pre-processing (last column). Comparing both nicely illustrates that the number of statements in the model doubles after the pre-processing stage.

When relying on the PVM code (Prolog Virtual Machine) of these fullbuildingmodel files, performance results can be obtained as listed in Table 8. Columns 4 and 5 list the results for the reasoning runs when the EYE engine did not use any indexer or other form of internal optimisation algorithm. However, a just-in-time indexer (JITI) is available in the engine. This indexer dynamically and automatically adds indexes in frequently visited relations (after loading the PVM –networking time– once). Column 6 in Table 8 shows the query execution times when this indexer is used and output is given in table format (.csv), like the outcome of a regular SPARQL query.

Similarly to what was found in the SPIN procedure, performance results are linearly related to the number of results that a query produces. In this regard, the graph in Fig. 8 shows the time required for each of the queries Q1 and Q31 to complete (*x*-axis in sec), compared to the number of results that is returned by the query (*y*-axis). This graph indicates that query execution time increases with the number of query results that is retrieved (Q53 < Q31 < Q1). Similar graphs are available for the other models.

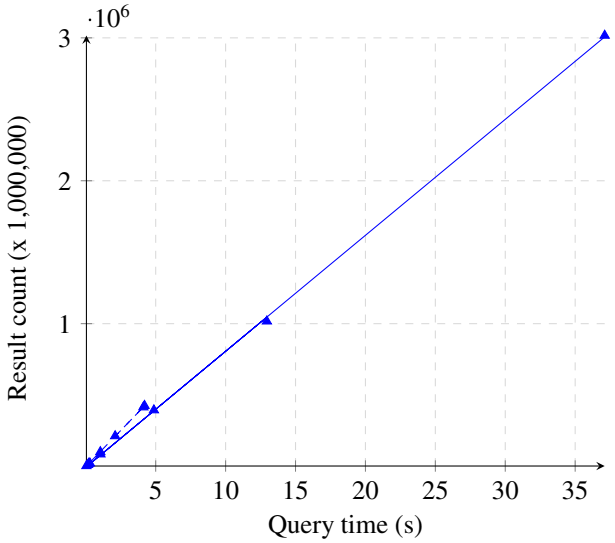

Fig. 8. Indication of the relationship between query execution time (*x*-axis) and result count (*y*-axis) for Q1 (solid) and Q31 (dashed) in Procedure 2.

6. Procedure 3: Rule checking over SWRL rules with a semantic graph database

Fig. 9 provides an overview of the components that are involved in the third rule checking implementation approach tested here. In

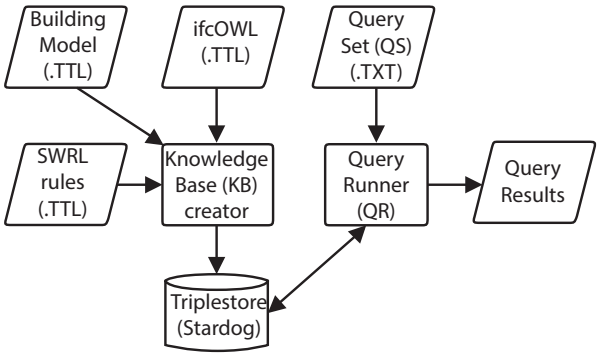

Fig. 9. An overview of the components involved in Procedure 3.

the following sections, we will first present a brief description of each of the components in this schema, after which we document the way in which this approach works in the given test environment (access to building models and rule examples).

6.1. Overview of the implementation approach

This third rule checking approach is implemented using a 4.0.2 Stardog semantic graph database [48]. Stardog implements an OWL reasoner associated to a rule engine. The particular strengths of Stardog are:

- it supports SWRL rules,
- it allows performing reasoning at query execution (backward chaining), and
- it is a large-scale triple store.

In our tests for this article, we used backward chaining reasoning only. In scenarios using forward chaining (e.g. EYE and N3Logic approach, see Section 5), all facts entailed from explicit statements are materialized before query execution. Compared to this approach, backward chaining uses a given query as a starting point (a goal) and works backwards, from a rule's consequent to the rule's antecedent. A backward chaining reasoner will split the query into sub-queries (sub-goals) that are directly matched to existing explicit statements in the knowledge base (KB).

Avoiding triple materialisation through forward chaining reasoning can be an important advantage. Indeed, let us consider the case of an IFC file in Turtle serialization of a size of about 1.5 GB, representing 19,652,142 triples (see model "wereld van theater en verwondering", line 2 in Table 3). In the case of forward chaining, reasoning over such a file would imply the materialisation of all implicit facts. Thus, the number of triples in the knowledge base would be considerably higher. Moreover, if any changes were made to the building model, which often occurs in the architectural design and construction industry, all inferences would have to be re-computed. Re-computing all inferences for a building model such as the "wereld van theater en verwondering" can be a very time-consuming process. As listed in Table 7, re-computing all inferences for this model using Procedure 2 takes 9110.56 s (313.37 s + 8797.19 s).

The fact that Stardog handles large-scale knowledge bases (KB) is also an important advantage in the context of BIM. Stardog 4.0.2 is implemented in Java 8, and supports the RDF 1.1 graph data model, OWL2 profiles and SPARQL 1.1. Stardog allows using user-defined rules for inference, along with closed-world reasoning capabilities. Stardog allows attaining a DL-expressivity level of $SROIQ(D)$. Stardog performs reasoning by applying a query rewriting approach. In this approach, SWRL rules are taken into account during the query rewriting process.

6.2. Details of the implementation in the test environment

The Knowledge Base (KB) creator component, illustrated in Fig. 9, takes as an input the following elements, all converted into Turtle serialization: the considered IFC file (Building Model $*$.ttl in Fig. 9), the ifcOWL ontology and the set of SWRL rules (see RSTOTAL_SWRL.ttl that is available in [60]). The KB creator then conceives a new Stardog repository, and populates it with the RDF triples received as an input. A new repository is created for each IFC file considered for testing. All considered queries are written in a $*$.txt file (a Query Set (QS) input file illustrated in Fig. 9), each query having an identifier. For these tests, 60 queries have been used (see Section 3.4). These queries were written using SPARQL 1.1 syntax, as supported by the Stardog triple store. The Query Runner component takes as an input the QS and executes each query 20 times over each of the repositories previously created. This process outputs the query results, notably the query answer time along with the number of results retrieved by each query.

6.3. Performance results

Table 9 documents the benchmarks for this third approach, notably in terms of query execution time and retrieved results for queries Q1, Q31 and Q53 in QS. For all tests, the Stardog reasoner type was set to SL. SL is a level of expressivity in Stardog that allows supporting a combination of RDFS, OWL 2 QL [10], OWL 2 RL [10], and OWL 2 EL [10] axioms, along with SWRL rules. All other types of axioms will be ignored by the SL reasoner.

Similar to what was found in the SPIN and EYE approach (Sections 4 and 5), Table 9 shows that the execution time for a given query is closely related to the number of retrieved results (see Fig. 10). This is to be expected because returning more results generally implies matching more triples and inferring over more assertions. Inferences and triple matching are time-consuming tasks. Therefore, we can observe that query execution time increases according to the number of retrieved results for Q1 and Q31. For example, let us consider Q31 over KB1 (contains triples for Model 1) and KB2 (contains triples for Model 2). Both KBs have almost the same number of stored triples. When executed over KB1, Q31 retrieves half the number of results returned when executed over KB2. Related query execution time is about 2.3 times faster when executing Q31 over KB1 than when executing Q31 over KB2.

Secondly, the query execution time depends on the data that is available in the models. Comparing the results when querying with Q53 over KB4 to Q53 over KB1 (see Table 9), we notice that, in both cases, no results are retrieved. However, the query execution time for Q53 over KB1 is about 3 times faster than for Q53 over KB4. Actually, the query execution time depends on inferences and triple matching tasks for answering a given query. As a reminder, Q53 queries all external walls of a building. Looking at building Model 1 (KB1), it contains no wall instances. However, building Model 4 (KB4) has about 100 wall instances. This means that more triples have to be matched for answering Q53 over KB4 than over KB1. This naturally increases query execution time. Still, Q53 can trigger 4 rules. Nevertheless, in the case of Q53 over KB1, only the rule described in Listing 10 is triggered. Namely, since KB1 does not contain wall instances, the reasoner does not need to evaluate the `aei:isExternal` predicate, and consequently, Q53 does not trigger the other rules described in Listings 2, 3, and 10 in Section 3.3. Thus, there are less inferences to perform and thus the query execution time is reduced. This is not the case when executing Q53 over KB4, as the `aei:isExternal` predicate is evaluated.

A similar explanation can be given to justify cases where the execution time does not increase in direct correspondence to the number of retrieved results. For example, the same reason as above

Table 9
Query performance results for Q1, Q31, and Q53 over 6 Knowledge Bases (KBs) using Procedure 3.

Query	KB	Model name	Triple count	Average (s)	Standard deviation	Result count
Q1	KB1	202672_tractiewerkplaats_Phase 1	19,104,134	13.44	1.42	3,016,218
Q1	KB2	wereld van theater en verwondering	19,652,142	0.17	0.07	23,581
Q1	KB3	<Private>	23,466,825	1.4	0.11	394,875
Q1	KB4	technisch_instituut_sint_michiel	5.800.067	3.55	0.16	1,016,660
Q1	KB5	091210Med_Dent_Clinic_Arch	1.320.515	0.33	0.05	86,243
Q1	KB6	Duplex_A_20110505	225.135	0.08	0.01	12,597
Q31	KB1	202672_tractiewerkplaats_Phase 1	19.104.134	6.82	1.47	213,627
Q31	KB2	wereld van theater en verwondering	19.652.142	15.83	2.93	425,934
Q31	KB3	<Private>	23,466,825	15.28	1.75	415,308
Q31	KB4	technisch_instituut_sint_michiel	5,800,067	2.81	0.29	102,127
Q31	KB5	091210Med_Dent_Clinic_Arch	1,320,515	1.36	0.14	21,109
Q31	KB6	Duplex_A_20110505	225,135	1.00	0.08	4,481
Q53	KB1	202672_tractiewerkplaats_Phase 1	19,104,134	0.07	0.01	0
Q53	KB2	wereld van theater en verwondering	19,652,142	0.12	0.03	235
Q53	KB3	<Private>	23,466,825	0.31	0.07	168
Q53	KB4	technisch_instituut_sint_michiel	5,800,067	0.20	0.16	0
Q53	KB5	091210Med_Dent_Clinic_Arch	1.320.515	0.20	0.02	1,471
Q53	KB6	Duplex_A_20110505	225.135	0.13	0.02	23

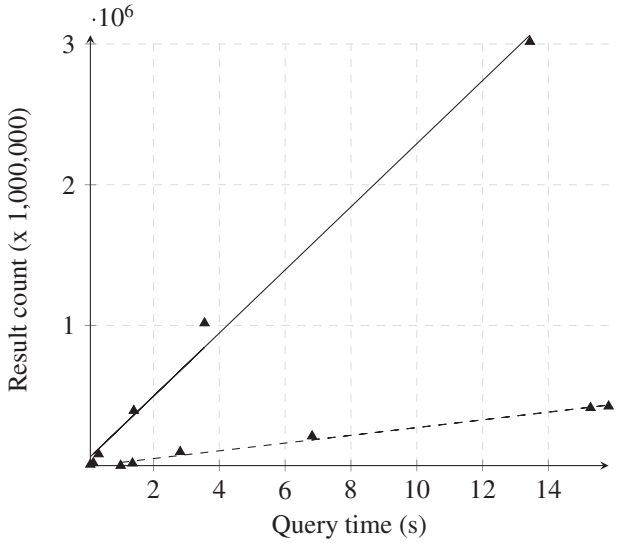

Fig. 10. Indication of the relationship between query execution time (*x*-axis) and result count (*y*-axis) for Q1 (solid) and Q31 (dashed) in Procedure 3.

can explain why Q53 over KB2 retrieves more results and faster than Q53 over KB3 (contains Model 3).

7. Discussion of results

7.1. Overall statistics

The query execution times for each of the three procedures are repeated in Table 10. In addition, the graphs in Figs. 11 and 12 plot the query execution times in this Table with the corresponding result counts, for Q1 and Q31 respectively.

7.2. Which performance benchmark?

Table 10 only lists query execution times, and these query execution times are of slightly different kinds. We repeat portions of this table in this section and discuss the results a bit more closely. It is clear from this paper that a benchmark should in fact be set up for both the pre-processing times (loading in triple store, networking, reasoning, indexing) and the query execution times. It is, however, also really hard to set these two measurements apart for entirely different procedures, like the three procedures considered here. For the third approach, for example, we have only one measurement in the end, which bundles networking, reasoning and query execution time. So, making a comparison of the distinct measurements (pre-processing time vs. query execution time) is already impossible. Our discussion and benchmark will thus be more qualitative. We focus on the query execution times, as they are of highest importance for an engineer interested in using these rule checking procedures.

7.2.1. Backward versus forward chaining reasoning

The SPIN performance results of Q1 and Q31 in Table 10 list the execution times for a backward chaining inference process plus actual query execution time. The SPIN statistics of Q53, however, show the execution times for the query execution time itself, as the forward chaining inference process takes place beforehand. Also for the EYE approach, the time for pre-processing the graph (networking and reasoning) is not listed (forward chaining inference process). Only the times in column 5 of Table 8 are considered in Table 10. In the Stardog approach, the listed times include the backward chaining inference process as well as the actual query execution time.

So, there is a strong mix of backward and forward chaining reasoning performances in Table 10. There are two specific areas in Table 10 that show the impact of using a backward versus a forward reasoning process. These two locations are repeated in Tables 11 and 12. Table 11 shows only the results of Q53 for the three selected procedures. The results for Procedure 1 and 2 (SPIN and EYE) both purely show the query execution times (as forward chaining reasoning has been done beforehand). The last column (Procedure 3 - Stardog), however, lists the backward-chaining reasoning result plus query execution time. As can be seen in Table 11, the performance of Stardog is worse in this case, which may be an argument against backward chaining reasoning. Furthermore, it can be noticed that the performance results of the SPIN/JENA approach are comparable to those of the EYE approach, which is to be expected as they both implement a forward chaining reasoning process.

A second area of reference is repeated in Table 12. This table shows the query performance results for Procedure 2 (EYE) and 3 (Stardog) for Q1 and Q31. The results for Procedure 2 (column 3) are direct query execution times (as forward-reasoning has happened beforehand), whereas the results for Procedure 3 (column 3) include backward reasoning times and query execution times.

Table 10
Comparison between the query performance results for all three considered query procedures, the 6 considered models, and the three considered queries.

Query	Model name	AQT PROC1 (s)	AQT PROC2 (s)	AQT PROC3 (s)
Q1	M1 - 202672_tractiewerkplaats_Phase 1	135.364	37.11	13.44
Q1	M2 - wereld van theater en verwondering	1.469	0.29	0.17
Q1	M3 - <Private>	24.009	4.87	1.4
Q1	M4 - technisch_instituut_sint_michiel	41.276	12.95	3.55
Q1	M5 - 091210Med_Dent_Clinic_Arch	4.989	1.05	0.33
Q1	M6 - Duplex_A_20110505	0.552	0.16	0.08
Q31	M1 - 202672_tractiewerkplaats_Phase 1	46.172	2.10	6.82
Q31	M2 - wereld van theater en verwondering	92.028	4.20	15.83
Q31	M3 - <Private>	82.676	4.12	15.28
Q31	M4 - technisch_instituut_sint_michiel	19.926	1.04	2.81
Q31	M5 - 091210Med_Dent_Clinic_Arch	3.685	0.21	1.36
Q31	M6 - Duplex_A_20110505	0.738	0.045	1.00
Q53	M1 - 202672_tractiewerkplaats_Phase 1	0.0013	0.001	0.07
Q53	M2 - wereld van theater en verwondering	0.0057	0.003	0.12
Q53	M3 - <Private>	0.0017	0.003	0.31
Q53	M4 - technisch_instituut_sint_michiel	0.0051	0.001	0.20
Q53	M5 - 091210Med_Dent_Clinic_Arch	0.0058	0.013	0.20
Q53	M6 - Duplex_A_20110505	0.001	0.001	0.13

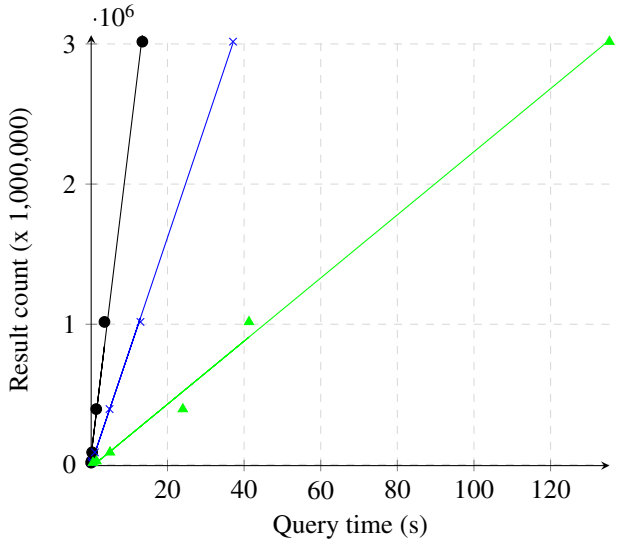

Fig. 11. Plot showing the linear relation between query time (*x*-axis) and result count (*y*-axis) for query Q1 in each of the three implementation procedures (green = SPIN; blue = EYE; black = Stardog). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

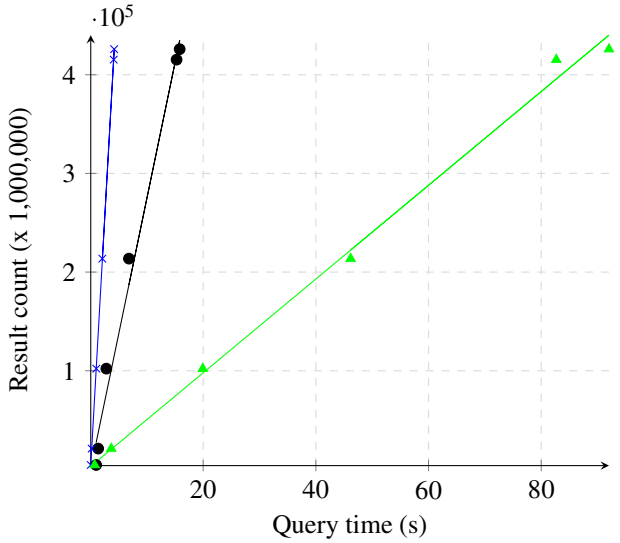

Fig. 12. Plot showing the linear relation between query time (*x*-axis) and result count (*y*-axis) for query Q31 in each of the three implementation procedures (green = SPIN; blue = EYE; black = Stardog). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

This table thus gives a clear comparison for the same queries and models between forward and backward reasoning. Interestingly, Q1 is generally processed more efficiently in Procedure 3 (Stardog), whereas Q31 is generally processed more efficiently in Procedure 2 (EYE). Q1 is a simple query for `aei:hasProperty`, whereas Q31 queries for the three Cartesian points.

Most importantly, these performance results in Table 12 are very similar, which would in this case form an argument in favour of backward chaining reasoning. If query execution times are similar, but Procedure 3 has notably less data on disk, then Procedure 3 (backward) is favourable.

7.2.2. Differences in backward chaining reasoning performances

The results of Q1 and Q31 rely on a backward chaining inference process in the JENA + SPIN and the Stardog approaches. We repeat these specific results in Table 13. These performance results deal with reasoning as well as query execution times. There is a considerable difference between these results. As it is not entirely clear which algorithms and indexing mechanisms are used in the JENA + SPIN system, nor in the Stardog system, it is not really possible to precisely indicate what the reason is. However, the reason likely lies in the way in which both systems are implemented. Namely, the JENA engine relies in the case of a backward chaining inference process on a 'table datalog engine' [70], which is *"a logic programming (LP) engine with a similar execution strategy to Prolog engines"* [70].

Stardog, however, implements the backward chaining reasoning process in a considerably different way. Namely, Stardog relies to a large extent on query rewriting techniques for implementing any form of reasoning process. As is explained in [71], *"Stardog rewrites the user's query with respect to any schema or rules, and then executes the resulting expanded query (EQ) against the data in the normal way. This process is completely automated and requires no intervention from the user."*. In the case of Q31, Stardog takes each of the three associated rules; uses those to expand Q31 until only queryable data remains in one long query; then applies query optimisation strategies on that query (e.g. changing join order) [72]; and finally executes the optimised query. This is all done without user intervention. The query optimisation strategy is thus crucial.

Table 11
Forward chaining versus backward chaining, showing favourable results for forward chaining reasoning (columns PROC1 and PROC2).

Query	Model name	AQT PROC1 (s)	AQT PROC2 (s)	AQT PROC3 (s)
Q53	M1	0.0013	0.001	0.07
Q53	M2	0.0057	0.003	0.12
Q53	M3	0.0017	0.003	0.31
Q53	M4	0.0051	0.001	0.20
Q53	M5	0.0058	0.013	0.20
Q53	M6	0.001	0.001	0.13

Table 12
Forward chaining versus backward chaining, showing favourable results for backward chaining reasoning (column PROC3).

Query	Model name	AQT PROC2 (s)	AQT PROC3 (s)
Q1	M1	37.11	13.44
Q1	M2	0.29	0.17
Q1	M3	4.87	1.4
Q1	M4	12.95	3.55
Q1	M5	1.05	0.33
Q1	M6	0.16	0.08
Q31	M1	2.10	6.82
Q31	M2	4.20	15.83
Q31	M3	4.12	15.28
Q31	M4	1.04	2.81
Q31	M5	0.21	1.36
Q31	M6	0.045	1.00

Table 13
The effect of rule engineering on reasoning time.

Query	Model name	AQT PROC1 (s)	AQT PROC3 (s)
Q1	M1	135.364	13.44
Q1	M2	1.469	0.17
Q1	M3	24.009	1.4
Q1	M4	41.276	3.55
Q1	M5	4.989	0.33
Q1	M6	0.552	0.08
Q31	M1	46.172	6.82
Q31	M2	92.028	15.83
Q31	M3	82.676	15.28
Q31	M4	19.926	2.81
Q31	M5	3.685	1.36
Q31	M6	0.738	1.00

7.3. Aspects affecting performance

From the previous sections, a considerable number of general findings can be made regarding rule checking performance. Many of these findings return in each of the three considered approaches. Hence, we will summarise them in the following paragraphs.

7.3.1. The effect of using a triple store

First of all, the usage of a triple store considerably affects query execution time. Loading files in memory at query execution time leads to considerable delays, as can be found in the discussion in Section 5. In fact, this section shows that querying PVM code, which can be considered similar to a running triple store, occurs many times faster compared to loading everything in-memory at query execution time. The SPIN approach and Stardog approach confirm this, as they rely on their own triple stores and have comparably good results in this setup (Table 10).

7.3.2. The dependency on the number of output results

All three approaches indicated that there is a direct relation between query execution time and the number of results that is eventually retrieved. The graphs in Figs. 11 and 12 indicate this relationship for the three approaches. This was explained in Section 6: as more results are available in the datasets, more triples need to be matched, leading to more assertions. The relation between query execution time and the number of results tends to be linear.

7.3.3. The dependency on the kind of data available in the models

A relation exists as well between query execution time and the kind of data available in the models. This is explained in Section 6 for Q53, which queries for external walls in the dataset. In contrast to Q1 and Q31, this query fires a rule (S6-1 in Listing 10) that in turn fires a number of other rules (namely S5-1, S1-2 and S1-1 in Listings 8, 3, and 2 respectively) in a cascading manner. However, if the first rule (S6-1) has no matches in the provided data (i.e. no Walls are available), then the other three rules do not need to be inferred, thus improving query execution times.

In contrast, Q31 fires relatively long rules, namely S3-1 to S3-3 (Listing 6). It takes more time to make these matches in all three approaches. Additionally, as IFC models typically have many Cartesian points included, processing Q31 takes notably longer.

7.3.4. Forward chaining versus backward chaining

The difference between a forward chaining and a backward chaining approach is very important. This is already clear from the discussion in Section 7.2. The impact in making this choice is most notably seen in Section 4, as this section shows the same procedure (SPIN) both in backward chaining and forward chaining mode. Q1 and Q31 are processed in a backward chaining manner. This means that the inference engine still needs to process the rules at query execution time. However, as it only needs to process those rules that are relevant considering the provided query, this can occur relatively fast. Depending on the engine, speed can be further increased, as is clearly shown in the backward chaining query results in Section 6: the backward chaining performance of procedure 3 is notably better compared to the corresponding performance of procedure 1 (see top left and top right in Table 12).

Inference in a forward chaining manner takes a lot longer, as a lot more triples need to be materialized (inferred and stored!). However, as soon as these additional triples are available, querying those triples can occur with a drastically improved performance. This can be found in the query execution results for Q53 in Section 4 (or bottom left in Table 10). In other words, pre-processing the dataset and rules takes a lot longer, but query execution times improve a lot. Of course, if pre-processing needs to be done time and time again (for instance when the building model changes a lot), such a forward chaining pre-processing procedure will not be a good choice. If data is stable (and storage space is available), then it is a good decision to opt for a forward chaining process.

7.3.5. Indexing algorithms, query rewriting techniques, and rule handling strategies

All in all, the overall strategy in which rules, queries and data are actually handled by the software systems (triple store, inference engine, software components) obviously has the highest impact on performance. This stands also at the core of Section 7.2. In this regard, the three considered procedures are quite far apart

from each other, explaining the considerable performance differences, not only between the procedures (Table 10), but also between diverse usages within one and the same system.

For each of the three approaches, it is not entirely known which algorithms and optimisation techniques are used, apart from the limited amounts of information communicated in the system documentation files. As a result, it is not really possible to make a fair comparison between the considered approaches concerning differences in indexing algorithms, query rewriting techniques and rule handling strategies used.

Nevertheless, we can see the impact of the indexing algorithm for Procedure 2 (EYE) in Table 8 (column 5 versus 6). In this table, we noticed considerable improvements as soon as the internal JITI indexer system was used. This indexer places an index on often visited predicates (e.g. `aei:hasCoordinateY`), thus considerably improving the amount of time required to retrieve and check those predicates. Similar improvements are clearly also made (or possible) in the two other procedures (SPIN and Stardog).

8. Conclusion

Each of the three procedures tested in this paper have their own specifics and characteristics. As all three procedures rely on components that are implemented in very diverse ways (commercial application versus Prolog implementation versus an approach relying on the Jena software libraries), the performance results for the three procedures are not directly comparable. Although we do present indicative query execution times, the creation of a full benchmark will require further work. The main focus of the work presented here is to give an indicative and comprehensive overview of three key implementation approaches for rule checking. Our discussion outlines an unprecedented list of key decisions and choices for anyone willing to implement a semantic rule checking process based on Semantic Web technologies in construction industry. Namely, when implementing a semantic rule checking procedure for the architectural design and construction industry, the following decisions are most important, in order of impact on performance.

1. Indexing algorithms, query rewriting techniques, and rule handling strategies.
2. Forward chaining versus backward chaining.
3. The dependency on the kind of data available in the models.
4. The effect of using a triple store.
5. The dependency on the number of output results.

Future work consists of further elaborating this initial performance benchmark with additional data and rules and comparing results on a higher scale for the individual approaches separately. At the moment, we have not analysed *all* 369 models and 69 rules and 60 queries, as it would take us too far to document all this for the three approaches. Future work will consider each approach and other currently not considered approaches separately and indicate performance results for larger numbers of queries, rules and building models.

Acknowledgments

For their financial support, the authors would like to thank the Special Research Fund (BOF) of Ghent University, the China Scholarship Council (CSC), the French company ACTIVe3D and the Burgundy Regional Council.

Appendix A

1. `?obj aei:hasProperty ?p`
2. `?x aei:hasElementCovering ?y`
3. `?x aei:hasGroupAssignment ?y`
4. `?x aei:hasServiceSystem ?y`
5. `?x aei:hasSpatialContainment ?y`
6. `?x aei:hasQuantity ?y`
7. `?x aei:hasElementComposition ?y`
8. `?x aei:hasSpatialContainer ?y`
9. `?x aei:hasProjectClassificationInformation ?y`
10. `?x aei:hasProductAssignment ?y`
11. `?x aei:hasProcessAssignment ?y`
12. `?x aei:hasProjectLibraryInformation ?y`
13. `?x aei:hasPortNesting ?y`
14. `?x aei:hasObjectNesting ?y`
15. `?x aei:hasControlAssignment ?y`
16. `?x aei:hasElementDecomposition ?y`
17. `?x aei:hasClassificationAssociation ?y`
18. `?x aei:hasMaterial ?y`
19. `?x aei:hasApprovalAssociation ?y`
20. `?x aei:hasSpatialDecomposition ?y`
21. `?x aei:hasControlFlow ?y`
22. `?x aei:hasObjectTyping ?y`
23. `?x aei:hasResourceAssignment ?y`
24. `?x aei:hasStructuralActivity ?y`
25. `?x aei:hasDocumentAssociation ?y`
26. `?x aei:hasLibraryAssociation ?y`
27. `?x aei:hasSuccessorProcess ?y`
28. `?x aei:hasActorAssignment ?y`
29. `?x aei:hasStructuralConnectivity ?y`
30. `?x aei:hasProjectDocumentInformation ?y`
31. `?point aei:hasCoordinateX ?x. ?point aei:hasCoordinateY ?y. ?point aei:hasCoordinateZ ?z`
32. `?xl aei:name ?x3`
33. `?obj aei:hasNumberOfRiser ?str`
34. `?obj aei:hasNumberOfTreads ?str`
35. `?obj aei:hasTreadLengthAtInnerSide ?str`
36. `?obj aei:hasProjectedArea ?str`
37. `?obj aei:isExtendToStructure ?str`
38. `?obj aei:hasSlope ?str`
39. `?obj aei:hasWaistThickness ?str`
40. `?obj aei:hasNumberOfStoreys ?str`
41. `?obj aei:hasTreadLengthAtOffset ?str`
42. `?obj aei:hasWalkingLineOffset ?str`
43. `?obj aei:hasTotalArea ?str`
44. `?obj aei:hasReference ?str`
45. `?obj aei:hasNosingLength ?str`
46. `?obj aei:isLoadBearing ?str`
47. `?obj aei:hasSpan ?str`
48. `?obj aei:hasRiserHeight ?str`
49. `?obj aei:hasHeight ?str`
50. `?obj aei:hasFireRating ?str`
51. `?obj aei:hasTreadLength ?str`
52. `?d rdf:type aei:InternalWall`
53. `?d rdf:type aei:ExternalWall`
54. `?d rdf:type aei:InternalDoor`
55. `?d rdf:type aei:ExternalDoor`
56. `?d rdf:type aei:InternalWindow`
57. `?d rdf:type aei:ExternalWindow`
58. `?x rdf:type aei:ElevatedSpace`
59. `?spacel aei:isConnectedByDoorTo ?space2`
60. `?x aei:hasElementFilling ?y`

References

[1] C.M. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM handbook: a guide to building information modeling for owners, managers, architects, engineers, contractors, and fabricators, John Wiley & Sons, Hoboken, NJ, USA, 2008.

[2] C.M. Eastman, J.-m. Lee, Y.-s. Jeong, J.-K. Lee, Automatic rule-based checking of building designs, Automation in Construction 18 (8) (2009) 1011–1033.

[3] C.S. Han, J. Kunz, K.H. Law, Making automated building code checking a reality, Facility Management Journal (1997) 22–28.

[4] C.S. Han, J. Kunz, K.H. Law, Client/server framework for on-line building code checking, ASCE Journal on Computing in Civil Engineering (1998) 181–194.

[5] C.S. Han, J. Kunz, K.H. Law, Building design services in a distributed architecture, ASCE Journal on Computing in Civil Engineering (1999) 12–22.

[6] T. Liebich, Y. Adachi, J. Forester, J. Hyvarinen, S. Richter, T. Chipman, M. Weise, J. Wix, Industry Foundation Classes IFC4 Official Release, 2013. <http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm> Last accessed on 16 August 2016.

[7] BuildingSMART International, Summary of IFC releases, 2014. <http://www.buildingsmart-tech.org/specifications/ifc-releases/> Last accessed on 16 August 2016.

[8] BuildingSMART International, BuildingSMART - International home of openBIM, 2014. <http://www.buildingsmart.org/> Last accessed on 16 August 2016.

[9] G. Schreiber, Y. Raimond, RDF 1.1 Primer - W3C Working Group Note 24 June 2014, W3C Working Group Note, 2014. <http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/> Last accessed on 16 August 2016.

[10] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer (Second Edition) - W3C Recommendation 11 December 2012, W3C Recommendation, 2012. <http://www.w3.org/TR/2012/REC-owl2-primer-20121211/. Last accessed on 16 August 2016.

[11] S. Kerrigan, K.H. Law, Logic-based regulation compliance-assistance, in: Proc. of the Ninth International Conference on Artificial Intelligence and Law (ICAIL), 2003, pp. 126–135.

[12] E. Hjelseth, N. Nisbet, Exploring semantic based model checking, in: Proceedings of the 27th CIB W78 Conference, 2010, pp. 21–24.

[13] J.K. Lee, Building Environment Rule and Analysis (BERA) Language and its Application for Evaluating Building Circulation and Spatial Program Ph.D. thesis, Georgia Institute of Technology, 2011.

[14] J. Lee, C. Eastman, Y. Lee, Implementation of a BIM domain-specific language for the building environment rule and analysis, J. Intell. Robot. Syst. 79 (2015) 507–522.

[15] W. Solihin, C. Eastman, A knowledge representation approach to capturing BIM based rule checking requirements using conceptual graph, in: Proceedings of the 32rd International CIB W78 Conference, Eindhoven, Netherlands, 2015, pp. 686–695.

[16] F. Baader, W. Nutt, Basic description logics, in: F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, Cambridge, MA, USA, 2003, pp. 47–100.

[17] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle, J. Van Campenhout, A semantic rule checking environment for building performance checking, Autom. Construct. 20 (5) (2011) 506–518.

[18] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Sci. Am. 284 (5) (2001) 35–43.

[19] T.H. Beach, Y. Rezgui, H. Li, T. Kasim, A rule-based semantic approach for automated regulatory compliance in the construction sector, Expert Syst. Appl. 42 (12) (2015) 5219–5231.

[20] J. Zhang, N. El-Gohary, Automated information transformation for automated regulatory compliance checking in construction, J. Comput. Civ. Eng. 29 (4) (2015).

[21] J. Zhang, N. El-Gohary, Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking, J. Comput. Civ. Eng. 30 (2) (2016).

[22] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012, 2012. <http://www.w3.org/TR/r2rml/> Last accessed on 16 August 2016.

[23] D. De Witte, L. De Vocht, R. Verborgh, K. Knecht, F. Pattyn, H. Constandt, E. Mannens, R. Van de Walle, Big linked data ETL benchmark on cloud commodity hardware, in: Proceedings of the International Workshop on Semantic Big Data, 2016.

[24] P. Pauwels, S. Zhang, Semantic rule-checking for regulation compliance checking: An overview of strategies and approaches, in: Proceedings of the 32rd International CIB W78 Conference, Eindhoven, NL, 2015, pp. 619–628.

[25] T. Berners-Lee, Notation 3 Logic: an RDF language for the semantic web, 2005. <http://www.w3.org/DesignIssues/Notation3.html> Last accessed on 16 August 2016.

[26] P. Pauwels, D. Van Deursen, J. De Roo, T. Van Ackere, R. De Meyer, R. Van de Walle, J. Van Campenhout, Three-dimensional information exchange over the semantic web for the domain of architecture, engineering, and construction, Artif. Intell. Eng. Des. Anal. Manuf. 25 (4) (2011) 317–332.

[27] J. De Roo, Euler YAP Engine, a birds EYE view, 2011. <http://www.agfa.com/w3c/Talks/2011/01swig/> Last accessed on 16 August 2016.

[28] J. De Roo, Euler Yet another proof Engine, 2015. <http://eulersharp.sourceforge.net/> Last accessed on 16 August 2016.

[29] S.-K. Lee, K.-R. Kim, J.-H. Yu, BIM and ontology-based approach for building cost estimation, Autom. Construct. 41 (2014) 96–105.

[30] M. Jang, J.-C. Sohn, Bossam: an extended rule engine for OWL inferencing, in: RuleML 2004, Lecture Notes in Computer Science (LNCS), vol. 3323, Springer, 2004, pp. 128–138.

[31] A. Yurchyshyna, C. Faron-Zucker, N. Le Thanh, A. Zarli, Towards an ontology-based approach for formalising expert knowledge in the conformity-checking model in construction, in: Proc. of the 7th European Conference on Product and Process Modelling, 2008, pp. 447–456.

[32] A. Yurchyshyna, C. Faron-Zucker, N. Le Thanh, A. Zarli, Towards an ontology-enabled approach for modeling the process of conformity checking in construction, in: Proceedings of the CAiSE'08 Forum, 2008, pp. 21–24.

[33] K.R. Bouzidi, B. Fies, C. Faron-Zucker, A. Zarli, N. Le Thanh, Semantic web approach to ease regulation compliance checking in construction industry, Future Internet 4 (3) (2012) 830–851.

[34] J. Dimyadi, P. Pauwels, M. Spearpoint, C. Clifton, R. Amor, Querying a regulatory model for compliant building design audit, in: Proceedings of the 32rd International CIB W78 Conference, Eindhoven, NL, 2015, pp. 139–148.

[35] H. Wicaksono, R. Sven, E. Kusnady, Knowledge-based intelligent energy management using building automation system, in: Proc. of the 2010 IPEC Conference, 2010, pp. 1140–1145.

[36] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al., SWRL: a semantic web rule language combining OWL and RuleML, W3C Member Submission 21 (2004) 79.

[37] H. Wicaksono, P. Dobreva, P. Häfner, R. Sven, Ontology development towards expressive and reasoning-enabled building information model for an intelligent energy management system, in: Proc. of the 5th International Conference on Knowledge Engineering and Ontology Development, 2013, pp. 38–47.

[38] H. Knublauch, R.W. Fergerson, N.F. Noy, M.A. Musen, The Protégé OWL plugin: an open development environment for semantic web applications, in: The Semantic Web–ISWC 2004, Springer, 2004, pp. 229–243.

[39] E. Prud'hommeaux, A. Seaborne, SPARQL query language for RDF - W3C Recommendation 15 January 2008, W3C Recommendation, 2008. <http://www.w3.org/TR/rdf-sparql-query/> Last accessed on 16 August 2016.

[40] M. Kadolsky, K. Baumgärtel, R.J. Scherer, An ontology framework for rule-based inspection of eeBIM-systems, Proc. Eng. 85 (2014) 293–301.

[41] K. Baumgärtel, M. Kadolsky, R.J. Scherer, An ontology framework for improving building energy performance by utilizing energy saving regulations, in: Proc. of the 10th European Conference on Product and Process Modelling (ECPPM), 2014, pp. 519–526.

[42] S. Zhang, J. Teizer, J.-K. Lee, C.M. Eastman, M. Venugopal, Building information modeling (BIM) and safety: automatic safety checking of construction models and schedules, Autom. Construct. 29 (2013) 183–195.

[43] S. Zhang, F. Boukamp, J. Teizer, Ontology-based semantic modeling of construction safety knowledge: towards automated safety planning for job hazard analysis (JHA), Autom. Construct. 52 (2015) 29–41.

[44] T.M. de Farias, A. Roxin, C. Nicolle, A rule based system for semantical enrichment of building information exchange, CEUR Proceedings of RuleML (4th Doctoral Consortium), vol. 1211, 2014, pp. 2–9.

[45] T. de Farias, A. Roxin, C. Nicolle, IfcWoD, semantically adapting IFC model relations into OWL properties, in: Proceedings of the 32rd International CIB W78 Conference, Eindhoven, NL, 2015, pp. 175–185.

[46] L. van Berlo, P. van den Helm, IFC Model checking with N3, Presentation at the 3rd International Workshop on Linked Data in Architecture and Construction (LDAC), 2015.

[47] Apache, Apache Jena, 2015. <https://jena.apache.org/> Last accessed on 16 August 2016.

[48] Complexible Inc. , STARDOG 4: THE MANUAL, 2016. <http://docs.stardog.com/> Last accessed on 16 August 2016.

[49] H. Schevers, R. Drogemuller, Converting the industry foundation classes to the web ontology language, in: Proceedings of the First International Conference on Semantics, Knowledge and Grid, IEEE Computer Society, Washington, DC, 2005, pp. 556–560.

[50] J. Beetz, J. van Leeuwen, B. de Vries, An ontology web language notation of the industry foundation classes, in: Proceedings of the 22nd CIB W78 Conference on Information Technology in Construction, 2005, pp. 193–198.

[51] J. Beetz, J. Van Leeuwen, B. de Vries, IfcOWL: a case of transforming EXPRESS schemas into ontologies, Artif. Intell. Eng. Des. Anal. Manuf. 23 (1) (2009) 89–101.

[52] P. Pauwels, D.V. Deursen, IFC/RDF: adaptation, aggregation and enrichment, in: Report of the First International Workshop on Linked Data in Architecture and Construction, Ghent, Belgium, 2012, pp. 2–5.

[53] R. Barbau, S. Krima, S. Rachuri, A. Narayanan, X. Fiorentini, S. Foufou, R.D. Sriram, OntoSTEP: enriching product model data using ontologies, Comput.-Aid. Des. 44 (6) (2012) 575–590.

[54] L. Zhang, R.R. Issa, Development of IFC-based construction industry ontology for information retrieval from IFC Models, in: Proceedings of the 2011 eg-ice Workshop, University of Twente, The Netherlands, 2011, pp. 6–8, July.

[55] S. Krima, R. Barbau, X. Fiorentini, R. Sudarsan, R. Sriram, OntoSTEP: OWL-DL ontology for STEP, National Institue of Standards and Technology, NISTIR 7561, in: Proceedings of the 6th International Conference on Product Lifecycle Management, 2009.

[56] P. Pauwels, W. Terkaj, EXPRESS to OWL for construction industry: towards a recommendable and usable ifcOWL ontology, Autom. Construct. 63 (2016) 100–133.

[57] BuildingSMART LDWG, IFC2X3_TC1 - OWL ontology for the IFC conceptual data schema and exchange file format for Building Information Model (BIM) data, 2015. <http://www.buildingsmart-tech.org/future/linked-data/ifcowl/20150925_latest/IFC2X3_TC1.owl> Last accessed on 16 August 2016.

[58] BuildingSMART LDWG, Overview page for the Linked Data Working Group, 2015. <http://www.buildingsmart-tech.org/future/linked-data/> Last accessed on 16 August 2016.
[59] P. Pauwels, IFC repository, 2015. <http://smartlab1.elis.ugent.be:8889/IFC-repo/> Last accessed on 16 August 2016.
[60] P. Pauwels, T. Mendes de Farias, C. Zhang, A. Roxin, J. Beetz, J.D. Roo, C. Nicolle, Additional Data AEI Article 2016, 2016. <http://perfbench.linkedbuildingdata.net/> Last accessed on 16 August 2016.
[61] P. Pauwels, W. Terkaj, T. Krijnen, J. Beetz, Coping with lists in the ifcOWL ontology, in: Proceedings of the 22nd EG-ICE International Workshop, Eindhoven, Netherlands, 2015, pp. 113–122.
[62] S. Borgo, E.M. Sanfilippo, A. Sojic, W. Terkaj, Ontological analysis and engineering standards: an initial study of IFC, in: V. Ebrahimipour, S. Yacout (Eds.), Ontology Modeling in Physical Asset Integrity Management, Springer, 2015, pp. 17–43.
[63] C. Zhang, J. Beetz, Model checking on the semantic web: IFC validation using modularized and distributed constraints, in: Proceedings of the 32rd International CIB W78 Conference, Eindhoven, NL, 2015, pp. 819–827.
[64] H. Knublauch, J.A. Hendler, K. Idehen, SPIN - Overview and Motivation, 2011. <http://www.w3.org/Submission/spin-overview/> Last accessed on 16 August 2016.
[65] H. Knublauch, SPIN - Modeling Vocabulary, 2011. <http://spinrdf.org/spin.html/> Last accessed on 16 August 2016.
[66] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, J. Hendler, N3Logic: a logical framework for the World Wide Web, Theory Pract. Logic Programm. 8 (2008) 249–269.
[67] T. Berners-Lee, D. Connolly, Notation 3 (N3): A readable RDF Syntax - W3C Team Submission 28 March 2011, 2011. <http://www.w3.org/TeamSubmission/n3/> Last accessed on 16 August 2016.
[68] T. Berners-Lee, Notation 3 (N3): A readable language for data on the web, 2005. <http://www.w3.org/DesignIssues/Notation3.html> Last accessed on 16 August 2016.
[69] J. De Roo, Euler Yet another proof Engine - EYE, 2016. <http://eulersharp.sourceforge.net/> Last accessed on 16 August 2016.
[70] A. Jena, Reasoners and rule engines: Jena inference support, 2016. <https://jena.apache.org/documentation/inference/> Last accessed on 16 August 2016.
[71] C. Inc., Stardog 4: The Manual - Query Rewriting, 2016. <http://docs.stardog.com/#_query_rewriting> Last accessed on 16 August 2016.
[72] T.M. de Farias, A. Roxin, C. Nicolle, {SWRL} rule-selection methodology for ontology interoperability, Data Knowl. Eng., ISSN: 0169-023X, <http://www.sciencedirect.com/science/article/pii/S0169023X15000713>.