The Numerate Web: Mathematical Formulas and Computations on the Web of Data

Ken Wenzel
Digitalization in Production, Fraunhofer-Institute for Machine Tools and Forming Technology IWU, Chemnitz, Germany
E-mail: ken.wenzel@iwu.fraunhofer.de

Abstract. Ontologies and related Semantic Web technologies are applied in many areas where mathematical relationships are essential to the domain knowledge. However, unlike ontologies and logical rule languages, mathematical expressions and calculation rules are not an intrinsic part of the linked data representation. Therefore, additional mapping processes between semantic domain models and the programs executing the mathematical computations are usually required. The Numerate Web is an approach to representing mathematical models with RDF, linking them to RDF resources and properties, running computations, and finally also making the results available as part of the RDF representation.

Keywords: rdf, math, ontology, models, computation, linked data

1. Introduction

Mathematical formulas and computations are ubiquitous in academics, industry, and everyday life. On the one hand, they are required to determine the properties of complex technical systems. On the other hand, they are also necessary to simply calculate the cost of a dinner in a restaurant, maybe also including taxes and tips.

Especially for the first case, conceptual models are essential to describe the technical systems, with their components, properties, physical quantities, and measurement units. Semantic Web technologies were also already successfully applied in this field to create semantic models of such systems [1–5]. These models then usually provide the input for mathematical models to compute static or dynamic properties of the system components.

While current Semantic Web technologies provide extensive knowledge modeling and logical reasoning capabilities, support for mathematical computations is limited. Therefore mathematical computations are either realized using domain-specific ontologies interpreted by specialized programs or by translating the mathematical expressions to a logical rule language like SWRL [6] or to SPARQL [7] queries.

A simple example for a mathematical computation would be a house with multiple rooms where the total area of the house should be calculated as the sum of the areas of its rooms.

When using the first approach a program needs to retrieve all contained rooms with their areas and calculate the sum. The downside of using such an external program is that the logic of the mathematical computation is not explicitly available as knowledge within an ontology and therefore cannot be formally verified nor changed or extended if required for slightly different use cases.

The second approach has the significant advantage that SWRL and SPARQL are already available in most Semantic Web implementations and that both have RDF serializations[1].

[1] For representing SPARQL in RDF the SPIN – SPARQL syntax [8] can be used.

When using SWRL, the mathematical formulas have to be translated into logical rules in the form of **antecedent** $\Longrightarrow$ **consequent**. Even for simple calculations, like the one shown in Listing 1 taken from a recent publication about assembly planning [9], the final rule is hard to read in comparison to its simple mathematical meaning: **comprehensive_indicator**$(x) =$ **datum_indicator**$(x) *$ **datum_indicator_weight**$(x) + \ldots$

```
assembly_object(?x) ^ swrlb:multiply(?a1, ?z, ?y) ^
datum_indicator(?x, ?y) ^ datum_indicator_weight(?x, ?z) ^
...
swrlb:add(?F1, ?a1, ?a2, ?a3, ?a4, ?a5) -> comprehensive_indicator(?x, ?F1)
```

Listing 1: Excerpt of an SWRL rule calculating indicators for assembly planning (see [9])

Another publication about modeling of engineering information with Semantic Web Technologies discusses the limits of SWRL and mentions extended mathematical calculations as a desired feature for reasoning within the engineering domain [10].

As SWRL is a subset of Datalog, it also has no built-in support for aggregations [11]. Therefore, either custom extensions or other mechanisms like the SWRL-based query language SQWRL [12] are required to compute aggregations like the total of multiple area values, as described in the example above.

On the other hand, SPARQL supports aggregations and could be used for the example calculation, as shown in Listing 2, which uses a hypothetical vocabulary for types and properties.

```
select ?house (sum(?area) as ?totalArea) {
  ?house a :House ; :room ?room .
  ?room :area ?area
} group by ?house
```

Listing 2: Example of using SPARQL to compute the total areas of houses

However, as with SWRL, the mathematical expressions need to be translated into specialized queries where the original semantics get lost. Furthermore, SPARQL's support for aggregate functions is limited to the operators: `count`, `sum`, `min`, `max`, `avg`, `group_concat`, and `sample`. Also mathematical constructs like vectors, matrizes, lambda expressions or more complex mathematical operations are not supported.

The latter is a strength of mathematical markup languages like MathML [13] and OpenMath [14]. These allow exchanging of complex mathematical expressions on the internet and already provide the base for multiple approaches to mathematical knowledge management [15–18].

MathML is based on XML and defines two flavors: *Presentation MathML* and *Content MathML*. The Figures 1 and 2 depict possible representations of the formula $\sum_{x=1}^{n} x^2$. Figure 1 uses Presentation MathML to define a specific graphical representation of the sum and therefore uses vocabulary for visual arrangement.

In turn, Figure 2 shows the same formula as Content MathML, that is able to convey its meaning including the operators `sum` and `power` as well as the binding of the variable x to different values in the range 1 to n.

While Presentation MathML mainly targets the correct visual representation of mathematical expressions for human readers, Content MathML can be used as semantic markup for automatic processing by machines.

This paper proposes an approach to combine this semantic mathematical markup with the technology stack of the Semantic Web for realizing a Linked Data Web that supports mathematical expressions and models as first-class citizens.

The following section reviews relevant related work. Section 3 introduces the Numerate Web concepts and gives an outline for the structure of the remainder of this paper.

2. Related work

In 2003 Marchiori outlined the idea and possible applications of representing mathematical expressions as part of the Semantic Web [19]. Advantages are seen in an RDF [20, 21] representation that enables the reuse of Semantic

```
<math>
  <munderover>
    <mo>&sum;</mo>
    <mrow>
      <mrow><mi>x</mi></mrow>
      <mo>=</mo>
      <mn>1</mn>
    </mrow>
    <mn>n</mn>
  </munderover>
  <msup>
    <mi>x</mi>
    <mi>2</mi>
  </msup>
</math>
```

Fig. 1. Presentation MathML

```
<math>
  <apply>
    <sum/>
    <bvar><ci>x</ci></bvar>
    <lowlimit><cn>1</cn></lowlimit>
    <uplimit><ci>n</ci></uplimit>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
  </apply>
</math>
```

Fig. 2. Content MathML

Web languages and tools to support functions like search, annotation, or inference on mathematical knowledge. The paper also gives basic ideas for directly converting MathML to RDF, but without an explicit ontology. Marchiori also names the possible computability as a "cool functionality" of mathematical formulas on the Semantic Web.

In 2011 Lange [16] worked on methods for the collaborative creation and exchange of semiformal mathematical content. The authors introduce the OMDoc ontology (Open Mathematical Documents) for the exchange of mathematical statements and theories on the internet. The ontology is defined in OMDoc itself since the authors state that the expressiveness of OWL is insuffient for the representation of all aspects of OMDoc [16, S. 102]. Additional to OMDoc the work introduces an OWL-based ontology for the description of OpenMath Content Dictionaries. It can represent metadata about symbols and their usage within mathematical expressions but not the expressions themselves.

In 2012 Ferré [22] proposed a lightweight RDF vocabulary for the representation of mathematical expressions mainly for the use case of content-based search. The vocabulary solely uses existing RDF and RDFS properties, so there is no explicit ontology. The property `rdf:type` is used as the constructor of mathematical operations where each object is an instance of `rdfs:Container` and the properties rdf:_1, rdf:_2, ..., rdf:_n represent its arguments.
For example, the expression $(a + 2) * 3$ would be represented as

```
[ a math:Times ;
  rdf:_1 [ a math:Plus ; rdf:_1 _:a ; rdf:_2 2 ] ; rdf:_2 3 ] .
_:a rdfs:label "a" .
```

in Turtle [23] format. This syntax is comparable to the notation used by the programming language Lisp that may represent the expression as

```
(math:Times (math:Plus a 2) 3)
```

Due to the missing ontology, the semantics of the RDF representation is limited. For example, constants, and variables are only implicitly distinguishable based on their node kind (constants are RDF literals and variables are blank nodes with a label). Since the representation was developed for structural search, a language construct for binding the variables of a lambda function as required for computations is not supported.

In 2014 Muñoz et al. [24] developed an ontology for mathematical expressions and equations. The ontology also allows references from mathematical models to elements of the domain models. Thus, mathematical expressions can point to certain individual elements of the domain model. However, it remains unclear how numerical attributes of

the referenced domain objects can be used as values in the mathematical model. Furthermore, the explicit referencing of particular domain objects severely restricts the reuse of mathematical formulas. The approach does not align well with existing standards for the representation of mathematical expressions. The ontology is only loosely based on MathML vocabulary.

An ontology-based approach to the representation of mathematical knowledge is also provided by the works of Nevzorova et al. [25] and Elizarov et al. [26] in 2014. The developed OWL ontology *OntoMath^{PRO}* defines taxonomies for mathematical domains such as algebra or number theory and mathematical knowledge objects such as formulas or geometric objects. Application areas of *OntoMath^{PRO}* are, for example, extraction of information from mathematical documents and concept-based search for formulas. *OntoMath^{PRO}* defines a pure classification structure and no vocabulary to represent mathematical expressions or equations themselves.

3. The Numerate Web

The core idea of the Numerate Web is the consistent application of linked data principles to uniformly represent domain models and the related mathematical calculation rules and link them to each other by using a rule language.

Figure 3 depicts an overview of existing Semantic Web technologies along with new building blocks that are introduced in the scope of this work. The illustration is based on a version created by Tim Berners-Lee for the Semantic Web architecture, which has since been adapted and slightly modified in other publications [27, p. 7].

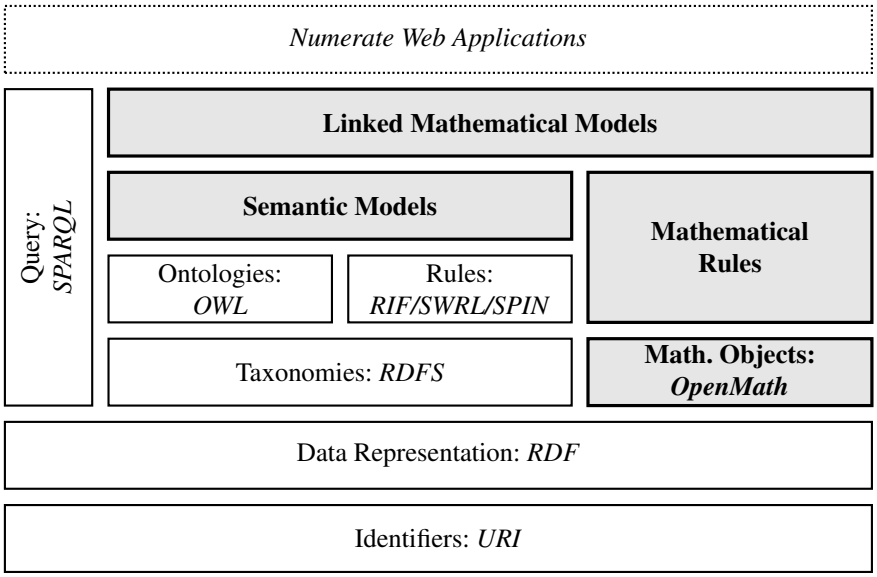

Fig. 3. Extension of the Semantic Web Stack with mathematical objects and rules

This work builds on this already existing basis and develops methods and concepts for the representation of *linked mathematical models*. These are based on semantic models and are RDF-compatible representations of mathematical objects and rules.

Since the term "semantic model" can stand for different types of models in general usage, the following definition will be used in the context of this work.

Definition 1 (Semantic model)**.** *A semantic model is a conceptual model of objects in a particular domain with their properties, relationships, and logical constraints that apply to them.*

The task of such a semantic model is to describe the physical and logical structures of a system and provide them as a basis for mathematical computations.

Building on semantic models, the Numerate Web will provide concepts and methods for representing mathematical objects and performing computations. Thereby, a close link between mathematical formulas and the data of the systems described with the help of semantic models should be established, so that physical and logical system structures can be used as the basis for the execution of computations.

An example calculation where the system structure plays a role is the determination of the total mass of a system consisting of several subsystems and components. Suppose the mathematical model has information about the corresponding part-whole relationships and the masses of the single components. In that case, the total mass can be calculated by the recursive traversal of all components and summing up their corresponding masses.

In the context of this paper, a mathematical model of this type, based on the semantic model of a system, is called a *linked mathematical model*. Thus, the following definition is used.

Definition 2 (Linked mathematical model)**.** *A linked mathematical model is a mathematical model that is defined as an extension of a semantic model and is closely related to its classes, properties, and relationships.*

For the description and exchange of linked mathematical models, this work presents concepts and methods based on existing Semantic Web technologies that build on each other and combine them with the mathematical markup language OpenMath.

This work contributes in particular to the following aspects:

- introduction of an approach for structured semantic models of systems (Section 4)
- introduction of an ontology for mathematical expressions and design of a textual syntax (Section 5)
- methods for accessing RDF data within mathematical formulas (Section 6)
- representation and execution of mathematical rules (Section 7)
  - design of a rule language and a textual syntax
  - algorithms for the execution of calculations
  - concepts for considering physical quantities and units

The proposed concepts are evaluated in Section 8 in two case studies in the fields of mathematical knowledge management and planning of manufacturing process chains, including a discussion of the results.

The paper finally closes with a conclusion and open issues that may be addressed by future work.

4. Semantic models

At the core of the Numerate Web, semantic models represent domain objects with their properties and connections. Although these semantic models may be freely structured, some concepts from systems modeling may be applied, especially in domains like engineering. One interesting approach that tries to find a common theory for modeling complex systems in different domains is multiscale modeling [28, 29]. It helps to better understand systems by breaking them down into subsystems with components and couplings between those components. Such a decomposition is depicted by Figure 4.

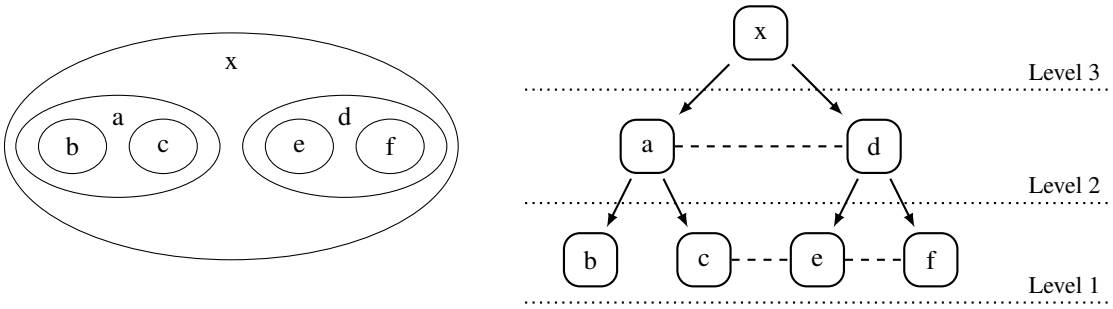

Fig. 4. Composition, levels and couplings of system components (see [29, pp. 825 and 828])

As an example, the approach can be applied to break down a simple gear motor into components and couplings. Figure 5 shows a possible RDF model of such a motor that uses a depth of three levels to represent the relevant components by using the property `:part`. As can be seen, couplings are modeled between components on the same levels by using the property `:connectedTo`. Multiscale modeling also permits couplings between components on different levels, which is not used in this example.

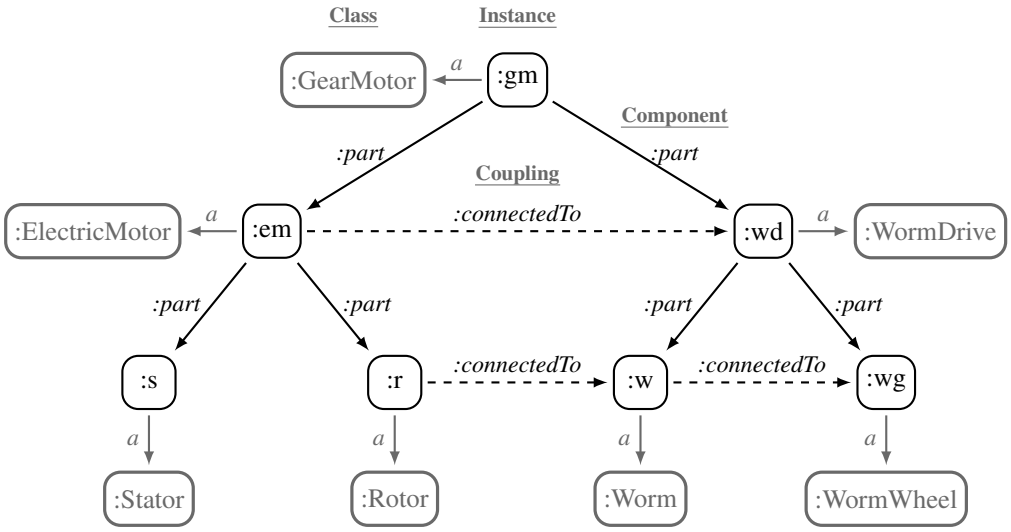

Fig. 5. Example for representig a gear motor as RDF model

Beyond this structural representation, the ontological definition of multiscale models by Yang and Marquardt [29] even goes one step further. It defines different types of laws that hold for properties of system components or couplings between components within the same level or between components on different levels, or even in different subsystems.

Possible laws in regard to the gear motor example above are: "The maximum speed of a motor is 10,000 min^{-1}.", "A worm gear consists of a worm and a worm wheel." or "The transmission ratio of a worm gear is calculated as the quotient of the number of teeth in the worm wheel divided by the number of starts in the worm."

While the first two laws may be represented by OWL constructs, the latter requires the mathematical formula $i = z_2/z_1$ for the gear ratio i that divides the number of teeth z_2 by the number of starts z_1. These mathematical laws are the reason why the Numerate Web requires means for the representation of mathematical objects, which are introduced in the next section.

5. Mathematical objects

5.1. OpenMath-RDF: an RDF vocabulary for mathematical objects

Mathematical objects are the base for expressing mathematical knowledge and rules within the Numerate Web. As mentioned in Section 1 these objects may be represented using Content MathML as markup language.

In MathML 3, a subset, or profile, of Content MathML called *Strict Content MathML* was introduced because Content MathML still provides multiple ways to express the same mathematical object. The strict subset defines only a minimal but sufficient set of elements to represent the meaning of mathematical expressions.

Strict Content MathML is designed to be fully compatible with OpenMath. In fact, it is just a specific XML serialization of OpenMath objects. Therefore, it can also rely on OpenMath's extension mechanism via *Content Dictionaries*. These define the semantics of mathematical symbols by combining textual descriptions with formal

Table 1
Excerpt of the description for *sin** within the Content Dictionary *transc1*

Role	application
Description	This symbol represents the sin function as described in Abramowitz and Stegun, section 4.3. It takes one argument.
Formal Mathematical Property (FMP)	$\sin(x) = \frac{\exp(ix) - \exp(-ix)}{2i}$

* https://www.openmath.org/cd/transc1.html#sin (23/05/2022)

mathematical properties and examples, which themselves are also encoded as OpenMath objects. Table 1 shows an example for definitions of the symbol *sin* contained in the Content Dictionary *transc1*.

Besides using Strict Content MathML, OpenMath objects can be serialized via XML, binary, or JSON encodings. When using one of these encodings, the mathematical objects have to be represented as literals in RDF, making these objects more or less black boxes for SPARQL and RDF toolings. Therefore an OWL ontology for OpenMath was developed [30]. This ontology was recently extended to a full OpenMath-RDF format and a formal bi-directional mapping between OpenMath-XML and OpenMath-RDF was defined [31].

Listing 3 shows the OpenMath-RDF serialization of the expression $i = z_2/z_1$ in Turtle syntax. As can be seen, argument lists are represented by RDF collections that also have a concise Turtle notation. Regarding semantics, it does not matter whether the structure is built with blank nodes or with URIs. However, using URIs makes it possible to globally share and refer to subexpressions as it is common practice with any other RDF resources.

```
[] a :Application ; :operator <http://www.openmath.org/cd/relation1#eq> ;
  :arguments (
    [ a :Variable ; :name "i" ]
    [ a :Application ; :operator <http://www.openmath.org/cd/arith1#divide> ;
      :arguments ( [ a :Variable ; :name "z2" ] [ a :Variable ; :name "z1" ] )
    ]
  ) .
```

Listing 3: RDF serialization of the expression $i = z_2/z_1$

5.2. POPCORN-LD: a textual syntax for OpenMath-RDF

As can be seen in Listing 3, the Turtle-based representation of OpenMath-RDF is hard to read and does not directly convey the meaning of the original mathematical expressions. By using a textual notation, the expression $i = z_2/z_1$ could be represented as `relation1:eq($i, arith1:divide($z2, $z1))` or even in infix notation as `$i = $z2 / $z1`, which is similar to the syntax used by standard spreadsheet software and mathematical programs.

For OpenMath, a textual notation called *POPCORN* (*Possibly Only Practical Convenient OpenMath Replacement Notation*) [32] was developed, allowing user-friendly input and output of mathematical expressions.

The POPCORN language makes some assumptions regarding symbol names and references that are not compatible with OpenMath-RDF:

Symbols (OMS) are always encoded as `cdName.symbolName` and solely resolved within the namespace `http://www.openmath.org/cd/`. This makes the use of symbols from other namespaces impossible and restricts the arbitrary extension of the mathematical language.

References (OMR) to local objects are expressed as `#id`. This may lead to misunderstandings in combination with RDF because namespaces may also start with slashes as in `http://example.org/math/`.

Both SPARQL [7] and the Turtle format [23] represent URIs in the form `<URI>` or by using a short *prefixed name* as in `ex:id` where the namespace `ex` is bound to a URI like `http://example.org/math#`.

For aligning the syntax of POPCORN to existing Semantic Web languages, a modified version of the language called *POPCORN-LD (POPCORN for Linked Data)* is introduced.

POPCORN-LD uses the following notations for basic OpenMath objects and prefix definitions:

Integers and floating-point numbers (OMI und OMF) can be directly represented. However, OpenMath's hexadecimal notation is not supported in favor of using RDF's XML Schema datatypes. Examples are `23` or `3.14159`.

Strings (OMSTR) are supported in one-line format enclosed in `"` or `'`, respectively in multi-line format enclosed in `"""` or `'''`. Examples are `"This is a string."` and `"""This is a string."""`.

Variables (OMV) may not contain spaces and start with a dollar sign `$` like `$a` or `$length`.

Symbols (OMS) are represented in compact form as `prefix:symbolName` or as full URIs `<URI>`. If the `prefix` is known within a POPCORN-LD expression then the symbol URI is the concatenation of `prefix` and `symbolName`.

If the `prefix` is unknown then it is interpreted as the name of a Content Dictionary and resolved within the namespace `http://www.openmath.org/cd/` to support references to known OpenMath symbols like `transc1:sin` which is resolved to `http://www.openmath.org/cd/transc1#sin` without the need for declaring prefixes.

References (OMR) are, analogous to symbols, represented in compact form as `#prefix:name` or by using full URIs as `#<URI>`. The `prefix` must be defined, otherwise the expression is considered invalid. The empty prefix `:` is allowed.

Prefixes for namespaces may be defined using the keyword `prefix` followed by the prefix name, a colon and the related namespace as in the example `prefix ex: <http://example.com/> ex:op(3)`.

POPCORN-LD also supports compound OpenMath objects with the following notation:

Function applications (OMA) use the prefix notation `operator(arguments)` where `operator` is either a symbol or a function object. Multiple arguments are separated by commas.

An example is `arith1:plus(2, 3)`.

Variable bindings (OMBIND) create bound `variables` within the scope of an `expression`. They use the notation `operator[variables -> expression]` as in `quant1.forall[$x, $y -> ...]`. For lambda expressions the short form `variables -> expression` can be used instead of the full form `lambda[variables -> expression]`.

Attributions (OMATTR) are written as `object{S1 -> A1, S2 -> A2, ...}` with symbols `S1` and `S2` and attribute values `A1` and `A2`. An example is `sin($x){mathmlattr:style -> "color: red"}`.

Errors (OME) use the notation `symbol!(arguments)`. The type of the error is defined by `symbol` and the `arguments` represent the context of the error using arbitrary OpenMath.

An example is `moreerrors:unexpected!("Computation failed.")`.

Foreign objects (OMFOREIGN) are represented as `` `encoding<content>` ``.

An example is `` `xhtml 1.0<h1>text</h1>` ``.

For common binary functions like `arith1:plus` POPCORN-LD defines infix operators allowing to write `2 + 3` instead of `arith1:plus(2, 3)`. For other standard functions like `transc1:sin`, POPCORN-LD defines short versions where the CD name may be omitted. The infix operators and shortcut symbols are listed within the POPCORN definition[2].

Table 2 compares examples of mathematical expressions in POPCORN-LD to the equivalent OpenMath-RDF syntax in Turtle notation.

[2]The original POPCORN definition is no longer available but an archived version can be accessed at https://web.archive.org/web/20150822191347/https:/java.symcomp.org/FormalPopcorn.html.

Table 2
Examples of POPCORN-LD and OpenMath-RDF

POPCORN-LD	OpenMath-RDF

`sin($x + $y)`	`@base <http://www.openmath.org/cd/> .`
	`[] a :Application ; :operator <transc1#sin> ;`
	`  :arguments ( [ a :Application ;`
	`  :operator <arith1#sin> ;`
	`  :arguments (`
	`  [ a :Variable ; :name "x" ]`
	`  [ a :Variable ; :name "y" ]`
	`  ) ]`
	`  ) .`

`sum(1..10, $x -> $x^2)`	`@base <http://www.openmath.org/cd/> .`
	`[] a :Application ; :operator <arith1#sum> ;`
	`  :arguments ( [ a :Application ;`
	`  :operator <interval1#interval> ; :arguments (`
	`  [ a :Literal ; :value 1 ]`
	`  [ a :Literal ; :value 10 ]`
	`  ) ] [ a :Binding ; :binder <fns#lambda> ;`
	`  :variables ( [ a :Variable ; :name "x" ] ) ;`
	`  :body [ a :Application ; :operator <arith1#power> ;`
	`  :arguments (`
	`  [ a :Variable ; :name "x" ]`
	`  [ a :Literal ; :value 2 ]`
	`  ) ] ] ) .`

`1.2{mathmlattr:style -> "'color: red"'}`	`@base <http://www.openmath.org/cd/> .`
	`[] a :Attribution ; :arguments (`
	`  [ a :AttributionPair ;`
	`  :attributeKey <mathmlattr#style> ;`
	`  :attributeValue [ a :Literal ; :value "color:␣red" ]`
	`  ] ) ;`
	`  :target [ a :Literal ; :value 1.2 ] .`

6. Linking of mathematical objects to RDF data

OpenMath-RDF enables the RDF-based representation of OpenMath objects. However, up to this point these mathematical expressions are not connected to any RDF data. If, for example, the expression $i = z_2/z_1$ should be evaluated then it is the program's obligation to fill in the required values for the variables z_1 and z_2.

Analogous to connecting programming languages to SPARQL endpoints via APIs a hypothetical Content Dictionary `sparql` with an OpenMath operator `query` could be defined for querying of RDF data as part of the evaluation process of the mathematical expressions.

Listing 4 depicts an example for using the operator `sparql:query` in POPCORN-LD notation. It shows the summation of masses (property `:mass`) for a set of components (type `:Component`) using the `sum` operator. The sum is determined by iterating over the set of components which is generated by `sparql:query` under the assumption that the operator returns a set of values if only one variable occurs in the select clause. The function over the running variable `$c` relies on another operator `sparql:querySingle` that ensures only single results and not sets are returned.

Furthermore, it is to be noted that the query for the weight has to be parameterized with the results from the first query. Therefore, it is required to add the additional arguments `"c"` for the variable name within the SPARQL query and `$c` for the component value to the operator `sparql:querySingle`.

```
$totalMass := sum(sparql:query("select ?c where { ?c a :Component }"),
    $c -> sparql:querySingle("select ?w where { ?c :mass ?w }", "c", $c)
)
```

Listing 4: Example for calculating the total mass of multiple components with OpenMath

The integration of SPARQL operators with OpenMath could bring great flexibility as it would allow the execution of arbitrary SPARQL queries as part of mathematical computations. However, the formulation and required parameterization of SPARQL queries are complex for users without profound knowledge of the query language. Furthermore, the embedded queries have to be represented as string literals which makes them opaque for RDF processors. Moreover, they also limit the readability of the mathematical expressions.

In [30] we already proposed a concept that uses specific RDF operators for integrating RDF data with OpenMath expressions. This concept was developed further regarding its syntax and semantics. The resulting `rdf` Content Dictionary is reviewed and available on the OpenMath website[3].

6.1. Operators for resources

This section introduces the operators `rdf:resource` and `rdf:resourceset` that allow the representation of concrete RDF resources as OpenMath objects and the selection of multiple resources based on their RDF type.

A concrete node can be represented with the `rdf:resource`. This operator receives the name of the node as a parameter in form of a character string. Thus, for example, the expression

```
$machine := rdf:resource("<http://example.org/machine1>")
```

creates the resource `<http://example.org/machine1>` and stores it in the variable `$machine`.

To select a particular subset of all resources for further processing, a concept for the application of filters is necessary. A common task is the selection of specific objects by their type. For example, in the context of production, machines of a particular type, such as milling machines, could be selected to calculate their average availabilities using a formula. To enable such queries, the mathematical operator `rdf:resourceset` is introduced. Analogous to the operator `rdf:resource` operator, this operator takes the name of an RDF type as a parameter, but generates a set of `rdf:resource` objects. As an example, the expression

```
$machines := rdf:resourceset("<http://example.org/MillingMachine>")
```

retrieves all resources of type `<http://example.org/MillingMachine>` and stores the result in the variable `$machines`.

6.2. Operators for properties

With `rdf:resource` and `rdf:resourceset` it is possible to select particular nodes of an RDF graph. However, for traversing the edges further operators are necessary. For this purpose, two additional operators for RDF properties with one and multiple values are defined: `rdf:value` and `rdf:valueset`.

The `rdf:value` operator determines the value of an RDF property by its name for a given resource. As an example, the expression

```
$mass := rdf:value("<http://example.org/mass>",
    rdf:resource("<http://example.org/machine1>"))
```

[3]The `rdf` Content Dictionary is available at https://openmath.org/cd/rdf. (06/02/2022)

retrieves the mass of `machine1` and assigns the value to the variable `$mass`. As can be seen, `rdf:value` expects an RDF resource as the second argument instead of a string value. This decision was made in order to implement a certain degree of type safety and also to enable the direct use of results from the operators `rdf:resource` and `rdf:resourceset` without conversions. The first parameter, the name of the property, is specified as a string similar to the operators for RDF resources. Although this could also be an RDF resource, it would complicate the use of the operator `rdf:value`, without real benefits for applications.

Complementary to the operator `rdf:value` the operator `rdf:valueset` is able to return several property values as a set. It is useful, for example, to traverse hierarchical structures and neighborhood relationships. Thus the expression

```
$totalMass := sum(rdf:valueset("<http://example.org/part>",
        rdf:resource("<http://example.org/machine1>")),
    $p -> rdf:value("<http://example.org/mass>", $p))
```

calculates the total mass of all parts of `machine1` and assigns the result to the variable `$totalMass`.

6.3. Representation of RDF literals

In addition to resources, an RDF graph contains literals that also require a corresponding representation as mathematical objects.

For a subset of RDF literals, OpenMath already contains direct encodings. Supported elements are `OMI` for integers, `OMF` for floating-point numbers, `OMSTR` for strings, and `OMB` for binary encoded content (byte strings). Corresponding rules for the translation between OpenMath and RDF for these cases can be found in [31].

RDF, however, goes beyond the possibilities of OpenMath by allowing literals with arbitrary data types as well as internationalized strings [21, Section 3.3]. RDF itself uses a Unicode-encoded [33] string for the label with an IRI for the data type. An additional language tag according to the Internet standard *Tags for Identifying Languages* [34] is assigned if the data type is `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`.

A similar encoding can also be implemented using OpenMath *attributions*, which may be either semantic or non-semantic. Semantic attributions change the meaning of an object while non-semantic attributions provide additional information like text colors.

Data types and language labels change the meaning of literals and must be represented accordingly by semantic attributions. For this purpose, the two annotations `rdf:literal_type` and `rdf:literal_lang` with the role `semantic-attribution` are defined.

Analogous to the representation in RDF, the annotation `rdf:literal_type` expects as parameter a datatype IRI as `rdf:resource`, while `rdf:literal_lang` expects a language label. Both annotations are applicable to strings for extending them with the corresponding information. On this basis, an RDF literal for the date `05.01.2012` can be represented as follows:

```
"2012-01-05"{rdf:literal_type ->
    rdf.resource("<http://www.w3.org/2001/XMLSchema#date>")
}
```

A literal with the content "‘`This is an English text.`"’ and the language label "‘`en`"’ is representable with the annotation `rdf:literal_lang` as follows:

```
"This is an English text."{rdf:literal_lang -> "en"}
```

6.4. Short forms for URIs

Common serialization forms for RDF data like *Turtle*, *RDF/XML* as well as the query language *SPARQL* support short forms for URIs, in order to improve readability and reduce the amount of data required for encoding[4].

[4]POPCORN-LD also supports prefix declarations. However, these are on a syntactic level and are not preserved within the resulting OpenMath objects. The prefix mechanism introduced here is usable across different OpenMath serialization formats.

For the RDF operators defined in the previous sections short forms for URIs are not necessary for the functionality. However, they do increase readability and thus acceptance by potential users. The languages Turtle and SPARQL support prefix declarations to assign parts of of URIs to corresponding short names and thus make them reusable. As an example, Listing 5 shows the use of prefixes in Turtle. In this case, the prefixes `foaf` for the namespace of the FOAF ontology and `persons` for the namespace of a fictitious ontology about persons. As can be can be seen, the URIs are shortened significantly.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix persons: <http://example.org/persons#> .

persons:Alice foaf:knows persons:Bob .
persons:Bob foaf:age 27 .
```

Listing 5: Prefix declarations in Turtle

In order to support prefix declarations in OpenMath *semantic attributions* could be used, comparable to their use for the definition of literal datatypes and languages.

Therefore, the annotation `rdf:prefixes` together with the symbol `rdf:prefix` are introduced to allow the definition of multiple prefixes as a semantic attribution:

```
rdf:resourceset("foaf:Person"){rdf:prefixes -> {
  rdf:prefix("foaf", "http://xmlns.com/foaf/0.1/")
}}
```

A defined prefix is valid for both the attributed root object and all its child objects. It is possible to overwrite a prefix within a child object is possible.

It should be noted that OpenMath attributes are not inherited along the object hierarchy. For example, in the object

```
rdf:value("foaf:age", rdf:resource("persons:Alice")){rdf:prefixes -> {
  rdf:prefix("foaf", "http://xmlns.com/foaf/0.1/"),
  rdf:prefix("persons", "http://example.org/persons#")
}}
```

the attribute `rdf:prefixes` is only directly linked to the `rdf:value` application and not propagated to the nested `rdf:resource` application. A software that supports the `rdf:prefixes` attribute must accordingly perform the inheritance of the prefixes to child objects itself.

6.5. Complex queries for resources and filters

While SPARQL supports extensive functions for querying RDF data with triple patterns and filter operators, the operator `rdf:resourceset` only allows querying resources of a specific RDF type, such as all gear motors with the expression

```
rdf:resourceset(":GearMotor")
```

One way to extend the expressive capability is to use mathematical operators to construct and work with sets. Such operators are already defined in the `set1` content dictionary.

Using the operator `set1:suchthat` to construct sets, whose elements fulfil a certain condition, for example gear motors with an efficiency of at least 0.85 could be searched for as follows:

$$\{m \in \texttt{rdf:resourceset(":GearMotor")} \mid \texttt{rdf:value(":efficiency", } m\texttt{)} >= 0.85\} \quad (1)$$

This approach makes it possible to realize complex filter functions and thus to achieve an expressiveness comparable to SPARQL. A disadvantage, however, is that the processing of such set operations would have to take place

during the evaluation of the mathematical expressions. Necessary data, such as in the example shown the efficiency `rdf:value(":efficiency", m)` of the gear motor *m*, has to be loaded from the from the RDF database.

If the filter condition could be pushed down to the database then this would allow considerable speed improvements. This is possible by directly using OWL to specify class description that can be used as parameters for `rdf:resourceset`.

Thus, the class `EfficientGearMotor` could be defined according to Listing 6 which uses the *Manchester OWL Syntax* [35, 36]. Then a query of the form `rdf:resourceset(":EfficientGearMotor")` would return the gear motors with an efficiency of at least 0.85.

```
Class: EfficientGearMotor
    EquivalentTo: :GearMotor and :efficiency some xsd:decimal[>= 0.85]
```

Listing 6: Definition of an OWL class `:EfficientGearMotor` for gear motors with an efficiency of 85% and above

To further reduce the effort for users of OWL-based filters, the operator `rdf:resourceset` allows the direct specification of such class descriptions as an argument. Thus, the following compact expression is possible:

```
rdf:resourceset(":GearMotor and :efficiency some xsd:decimal[>= 0.85]")
```

The extension to include OWL class descriptions as parameters for `rdf:resourceset` greatly increases the expressiveness of the RDF operator, but also places higher demands on applications that support this operator. These must be enabled to use the so-called *Class descriptions* of the Manchester OWL syntax.

Systems like *OWL2SPARQL*[5] or *OWLET*[6] can translate OWL class descriptions directly into SPARQL expressions and may be used to implement the operator `rdf:resourceset`.

Another solution is to move the evaluation directly into the database system if it supports OWL-based reasoning. For this purpose, the class descriptions are converted into RDF triples and transferred directly into the database as named OWL classes with unique, generated URIs. Then the database automatically deduces which RDF resources belong to the respective OWL class. Thus, in this case, it is sufficient to use `rdf:resourceset` with the name of the OWL class to retrieve all related resources of this type. Within the OpenMath-RDF serialization, it is possible to directly reference the OWL class by its URI (or blank node ID) instead of using an object of type `:Literal` as an argument for `rdf:resourceset`. Compared to using character strings, this procedure has the advantage that the class description itself is treated as part of the RDF graph. Therefore it can be checked for consistency by OWL reasoners, and it can be automatically adapted in the event of changes – for example, to class and property names.

6.6. Extend POPCORN-LD with RDF operators

In order to improve the usability of mathematical expressions input and output when using the RDF operators, POPCORN-LD is extended with the following compact syntax:

```
'@' '@'? <RDF property>? (
    '(' <RDF resource> ')' |
    '[' <RDF type or class description (Manchester OWL syntax)> ']'
)
```

[5] https://github.com/SmartDataAnalytics/OWL2SPARQL (05/25/2022)
[6] https://github.com/phenoscape/owlet (05/25/2022)

```
ex:Person rdfs:subClassOf
  [
    a owl:Restriction ;
    owl:onProperty ex:ssn ;
    owl:maxCardinality 1
  ], [
    a owl:Restriction ;
    owl:onProperty ex:worksFor ;
    owl:allValuesFrom [
      owl:intersectionOf (
        ex:Company owl:NamedIndividual
      ) ]
  ] .
```

Listing 7 Example of OWL description

```
ex:PersonShape a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:ssn ;
    sh:maxCount 1 ;
    sh:datatype xsd:string ;
    sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
  ] ;
  sh:property [
    sh:path ex:worksFor ;
    sh:class ex:Company ;
    sh:nodeKind sh:IRI ;
  ] .
```

Listing 8 Example of SHACL rules (based on an example from [44])

The basic idea is to introduce references to RDF data using the `@` character. A single `@` indicates that the operator will produce a single value as a result, while `@@` indicates that it returns a, potentially empty, set of values. The following POPCORN-LD expressions demonstrate some examples for referencing RDF resources and properties:

```
rdf:resourceset("foaf:Person")            → @@[foaf:Person]
rdf:resource("<http://example.org/Bob>")  → @(<http://example.org/Bob>)
$alice := rdf:resource("example:Alice")   → $alice := @(example:Alice)
rdf:valueset("foaf:knows", $alice)        → @@foaf:knows($alice)
rdf:value("foaf:age", $alice)             → @foaf:age($alice)
```

7. Mathematical rules

So far, concepts for the definition of *semantic models* as well as for the representation of mathematical expressions and their linkage with with RDF resources and properties are introduced. On this basis the creation of mathematical formulas that use data from semantic models as input parameters is possible.

In order to use these mathematical formulas in a targeted way to compute the values of RDF properties, a rule system is necessary. It should support the assignment of mathematical computation rules to the classes and properties of an ontology and the definition of consecutive calculations that build on each other.

7.1. Existing languages for reasoning and validation

Besides OWL, which is based on the so-called open-world assumption [37][p. 50 ff.] and therefore is not suitable for checking the completeness of RDF data, further languages have been developed for validation purposes [38]. These include ShEx [39], ReSh [40], DSP [41], SPIN [42, 43] and its successor SHACL [44].

The W3C has standardized OWL and SHACL. For this reason, it is advisable to use similar concepts for the definition of mathematical rules.

Listings 7 and 8 contain examples of OWL-based and SHACL-based rules. While OWL uses one instance of type `owl:Restriction` for each property, SHACL uses node shapes of type `sh:NodeShape` to define count, data type and value range for multiple properties. To link several constraints with each other, OWL uses set operations, while SHACL uses boolean operators like `sh:or` and `sh:and` for combining node shapes to more complex constructs. SHACL uses the shape properties `sh:targetNode`, `sh:targetClass`, `sh:targetSubjectsOf`, and `sh:targetObjectsOf` to apply rules to resources, classes, and subjects or objects of specific properties.

7.2. A language for mathematical rules

The mathematical rules language is intended to allow calculations of the form

```
<RDF property> = <mathematical expression>
```

where the left-hand side, the value of an RDF property, is obtained by calculating the mathematical expression on the right-hand side.

The central element for representing these rules in RDF is the class `:Constraint` with the two attributes `:onProperty` and `:expression`, as shown in Figure 6. As can be seen, it borrows its syntax from `owl:Restriction` and `sh:NodeShape`. The attribute `onProperty` specifies the RDF property whose value is given by the mathematical expression represented via `:expression`. The expression itself is specified in the form of an OpenMath-RDF object.

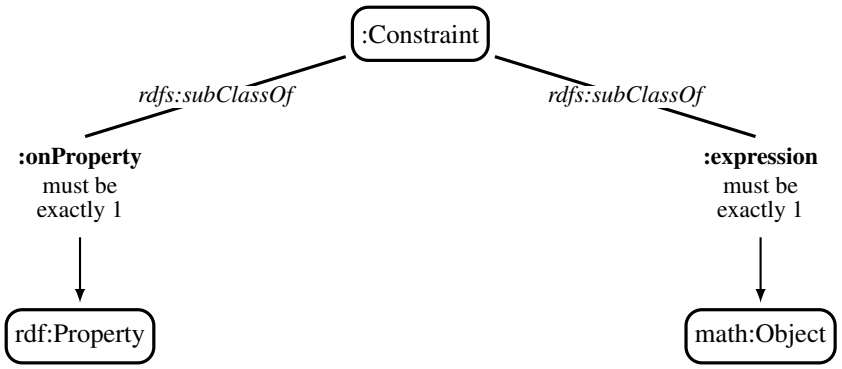

Fig. 6. Class `:Constraint` for representing mathematical rules

While OWL relies on `rdfs:subClassOf` and SHACL uses `sh:targetClass` and others to assign rules to classes, the rule language defined here uses the property `:constraint`. It is an `owl:AnnotationProperty` and relates an OWL or RDFS class to one or more instances of `:Constraint`. This design was chosen to allow a user-friendly syntax that can be embedded within a Manchester OWL specification, as described in the following section.

7.3. A user-friendly syntax for mathematical rules

A textual syntax is helpful for user-friendly input and output of mathematical rules. Based on a combination of POPCORN-LD and the Manchester OWL syntax, a rule language can be defined, which allows the definition of an ontology with its classes and properties while also including mathematical rules.

The listing 9 shows a syntax example for mathematical rules that calculate the area a and the volume v of a cylinder.

```
Class: Cylinder
Constraint:
  a = pi * @r^2,
  v = @a * @h
```

Listing 9: Textual syntax for mathematical rules

The definition of classes was taken from the Manchester OWL syntax with the keyword `Class:` followed by the class name. In addition to this syntax, the new keyword `Constraint:` initiates the definition of mathematical rules in the form

```
<RDF property> = <mathematical expression in POPCORN-LD>
```

for the RDF property given on the left side of the equals sign. The right side defines a POPCORN-LD expression to compute the respective property value. Each rule described textually in this way corresponds to an instance of the class `:Constraint` from Figure 6.

In order to be able to describe not only mathematical rules but also classes, properties, and objects in the form of an OWL ontology, it is possible to use the full Manchester OWL syntax. It could be extended by the `Constraint:` keyword and the POPCORN-LD syntax to combine description logic with mathematical models.

Listing 10 shows an example of what a combination of both languages could look like. In the example, first, the class `Cylinder` is defined with its attributes *r*, *h*, *a* and *v*. These have constraints on cardinality and property ranges (in this case, floating-point numbers), formulated as OWL restrictions. In addition, the calculation rules for the area *a* and the volume *v* of the cylinder are specified by using the keyword `Constraint:`. The instance `ComputeVolume`, representing a concrete cylinder with a radius of 2.0 and a height of 5.0, is again described using the conventional Manchester OWL syntax.

```
Class: Cylinder
SubClassOf: r exactly 1 and r only xsd:double,
  h exactly 1 and h only xsd:double,
  a exactly 1 and a only xsd:double,
  v exactly 1 and v only xsd:double
Constraint:
  a = pi * @r^2,
  v = @a * @h

Individual: ComputeVolume
Types: Cylinder
Facts: r 2.0, h 5.0
```

Listing 10: Example of combining the Manchester OWL syntax with mathematical rules

7.4. Calculation direction of rules

Methods for mathematical modeling are fundamentally distinguishable into *causal* and *acausal* [45]. Causal modeling specifies the direction of computation and thus the data flow, while acausal modeling chooses the direction of computation itself.

Using the Pythagorean theorem $a^2 + b^2 = c^2$ as an example, with acausal modeling both *c* (*a* and *b* known) and *a* (*b* and *c*) as well as *b* (*a* and *c* known) can be computed.

A causal modeling approach would have to solve the equation according to the target variable and write an explicit rule with a fixed calculation direction as in $c = \sqrt{a^2 + b^2}$. Thus, a different rule is necessary depending on the variable *a*, *b*, or *c* to be calculated. Hence the modeler has to decide at an early stage on the direction of the calculation and thus on known or unknown variables [46, pp. 2-7].

Acausal models require a solver to compute unknown variables in a system of equations. For example, OpenModelica normally uses DASSL [47], but also supports solvers included in the SUNDIALS software Suite (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers [48]).

Without loss of generality, a causal modeling methodology is chosen where the computational direction needs to be specified by the modeler to limit the complexity of implementing Numerate Web rules in different systems. Accordingly, the language defined in section 7.2 defines the mathematical rules in explicit form with exactly one variable (in this case, an RDF property) on the left-hand side and a mathematical expression on the right.

7.5. Algorithms for computation

The computation of the rules defined in Section 7.2 is comparable to approaches for logical reasoning. These usually fall into two categories [49, p. 278]: forward chaining and backward chaining.

Forward chaining starts with known facts and iteratively applies rules to them. In doing so, it adds the conclusions of the data basis, which can be used as premises for rules in the next step. This process is repeated until no more new conclusions can be generated. In order to efficiently find applicable rules in each calculation step for the given facts, tailored solutions like the Rete algorithm were developed [50].

In contrast to forward chaining, backward chaining starts with a statement and checks whether it can be generated transitively by a chain of rules. A typical representative of the strategy is the logical programming language Prolog [51]. It uses the so-called SLD resolution (Selective Linear Definite clause resolution) [52, ch. 9], in order to check statements for their validity.

Depending upon the kind and expression strength of the rules, computations of logical rules can have a high computational complexity or even be undecidable. To deal with this problem, OWL, for example, defines so-called profiles [53], which reduce the scope of the language and thus limit the complexity of reasoning.

In contrast to logical reasoning, the mathematical rules defined here can be used without complex search operations in the context of both forward as well as backward chaining. Corresponding algorithms are described below. The definitions are based on the assumption that RDFS inference for the predicate `rdf:type` is used to determine all direct and indirect classes of a resource. If this is not the case in an implementation, then the classes of a resource r can be retrieved, for example, by the following SPARQL query using a path expression for the `rdfs:subClassOf` property:

```
select ?type where {
    ?r a [ rdfs:subClassOf* ?type ] .
}
```

The Algorithm 1[7] implements the forward chaining of the mathematical rules for an RDF graph S, which represents as a set of triples $t \in S$ with $t = \langle s, p, o \rangle$. The function EVALUATERULES is the entry point, which iterates over all known classes in the graph S and executes for each class the function EVALUATERULESFORCLASS.

EVALUATERULESFORCLASS first determines for the given class c the applicable rules, and for each rule, the corresponding RDF property, and the mathematical expression. Then a loop is executed for each instance of c to compute the property value using the function EVALUATEEXPRESSION based on the given mathematical expression. The result is either one value or a set of multiple values stored as objects of triples for the corresponding instance and property.

The function EVALUATEEXPRESSION determines for an instance i the value of a given expression e, which is an OpenMath object in the form of a tuple

$$e = (T, a_1, \ldots, a_n) \tag{2}$$

where T represents the type of the OpenMath object[8] and $a_1, \ldots, a_n$ specify the respective arguments. If f is the general evaluation function EVALUATEEXPRESSION and f_T represents the specific evaluation function for a certain type T, then the evaluation is performed recursively according to the formula

$$v_e = f(e) = f_T(f(a_1), \ldots, f(a_n)) \tag{3}$$

to compute the value v_e of the expression e. In addition to the OpenMath object types, certain symbols, such as constants or operators of application objects, must be mapped by appropriate logic in order to obtain a practically

[7] The algorithms in this section are presented in pseudocode [54, p. 27], a simplified program code that is similar to higher-level programming languages. The pseudocode is combined with mathematical expressions for the representation of sets and tuples.

[8] according to the naming in OpenMath-XML, possible types are: `OMS`, `OMV`, `OMI`, `OMB`, `OMSTR`, `OMF`, `OMA`, `OMBIND`, `OME`, `OMATTR`, `OMR`, `OMATP`, `OMFOREIGN`, and `OMBVAR`

Algorithm 1 Forward chaining of mathematical rules

```
1: function EvaluateRules                                   ▷ computes rules for RDF graph S
2:     for c ← {c | ⟨c, rdf:type, rdfs:Class⟩ ∈ S} do       ▷ class c
3:         EvaluateRulesForClass(ic)
4:     end for
5: end function
6:
7: function EvaluateRulesForClass(c)                       ▷ computes rules for class c
8:     for r ← {r | ⟨c, mathrl:constraint, r⟩ ∈ S} do       ▷ constraint r of c
9:         for (p, e) ← {(p, e) |                           ▷ property p and expression e
               ⟨r, mathrl:onProperty, p⟩ ∈ S ∧
               ⟨r, mathrl:expression, e⟩ ∈ S} do
10:            for i ← {i | ⟨i, rdf:type, c⟩ ∈ S} do         ▷ i is an instance of c
11:                for v ← EvaluateExpression(i, e) do
12:                    S ← S ∪ {⟨i, p, v⟩}                  ▷ add triple to graph S
13:                end for
14:            end for
15:        end for
16:    end for
17: end function
```

Algorithm 2 Evaluate a mathematical expression

```
1: function EvaluateExpression(i, e)                        ▷ evaluates expression e for instance i
2:     if e = (OMA, rdf:valueset, p, e_o) then
3:         o ← EvaluateExpression(i, e_o)
4:         return EvaluateProperty(o, p)
5:     else if e = (OMA, rdf:value, p, e_o) then
6:         o ← EvaluateExpression(i, e_o)
7:         V ← EvaluateProperty(o, p)
8:         if |V| ≠ 1 then
9:             return {(OME, moreerrors:algorithm, 'Error: Expected exactly one property value.')}
10:        end if
11:        return V
12:    else if e = (OMA, rdf:resource) then
13:        return i
14:    else if ... then
15:        ...
16:    end if
17:    ...
18: end function
```

applicable implementation. The algorithm 2 therefore merely represents the principle of the function EVALUATEEXPRESSION in pseudo code.

The parameter i of the function EVALUATEEXPRESSION specifies the execution context of the computation, analogous to object-oriented programming languages like Java, C#, or Python, where the keywords `this` or `self` are used.

For referencing the resource i within the mathematical expression, the definition of a convention for a particular variable like `$this` would be possible. A reserved variable name might collide with existing expressions, and OpenMath also has a better extension mechanism via Content Dictionaries.

Thus, a unique mathematical symbol defined in a Content Dictionary can be used to represent the context resource. The operator `rdf:resource` for representing references to concrete RDF resources is already included in the RDF Content Dictionary. As an extension, the parameterless use of `rdf:resource` is allowed to reference the current context resource. The Listing 11 contains POPCORN-LD expressions that use `rdf:resource` to refer to concrete RDF resources and context resources. It should be noted in particular that an expression of the form `@<property>(@())` is equivalent to the expression `@<property>`, which is similar to the omission of the keyword `this` in an object-oriented programming language when referring to attributes or methods. For example, in Java `this.name` may be shortened to `name` while in POPCORN-LD `@name(@())` may be shortened to `@name`. The operator is implemented by calling the function EVALUATEEXPRESSION, which returns the context

```
$alice := rdf:resource("example:Alice")
$alice := @(example:Alice)
$this := rdf:resource()
$this := @()
rdf:value("foaf:age", rdf:resource()) = @foaf:age(@()) = @foaf:age
```

Listing 11: Examples for referencing RDF resources in POPCORN-LD

resource i as a value when it finds an application of `rdf:resource` with an empty parameter list (line 12 of the Algorithm 2).

Algorithm 3 Compute the value of a single property

1: **function** EVALUATEPROPERTY(i, p) ▷ determines the value of property p for instance i
2: $R \leftarrow \big\{ r \mid (\exists C) \big[\langle i, \text{rdf:type}, C \rangle \in S \wedge \langle C, \text{mathrl:constraint}, r \rangle \in S \wedge \langle r, \text{mathrl:onProperty}, p \rangle \in S \big] \big\}$
3: **if** $R \neq \emptyset$ **then** ▷ a rule exists for the value of p
4: **for** $e \leftarrow \{ e \mid \langle r, \text{mathrl:expression}, e \rangle \in S \}$ **do** ▷ expression e
5: **return** EVALUATEEXPRESSION(i, e)
6: **end for**
7: **return** $\{(\text{OME}, \text{moreerrors:algorithm}, \text{'Error: Constraint without expression.'})\}$
8: **else**
9: **return** $\{$ ASOPENMATH(o) $\mid \langle i, p, o \rangle \in S \}$
10: **end if**
11: **end function**

The operators `rdf:value` and `rdf:valueset` are supported by a function called EVALUATEPROPERTY (Algorithm 3). It expects an RDF resource i and an RDF property p as parameters and computes a possibly empty set of corresponding values. The function first checks whether, for a class of resource i, a mathematical rule exists for the property p. If this is the case, then the function EVALUATEEXPRESSION is used to evaluate the corresponding expression. If such a rule does not exist, then the function retrieves triples of the form $\langle i, p, o \rangle$ from the RDF graph S, transforms the objects o into OpenMath objects and returns them as the property values.

7.6. Inheritance of mathematical rules

Unlike object-oriented programming languages, which typically support the inheritance and overwriting of properties and methods in subclasses, OWL considers classes as sets for which certain logical rules apply. Thus, using OWL rules in derived classes cannot be overwritten and changed. Rather rules can only be extended along the class hierarchy with additional conditions, whereby the rules of the parent classes continue to be valid.

Even if the inheritance and the overwriting of mathematical rules in derived classes is not absolutely necessary, the modeling can be simplified in certain cases. In contrast to OWL, the Numerate Web rule language should therefore

allow the overwriting of mathematical rules of a parent class in child classes. To support this, the algorithms 1 and 3 must be adapted accordingly so that they only apply the rule for a given property defined on a class, which is on the shortest path in the `rdfs:subClassOf` inheritance hierarchy.

An example depicts Figure 7, which shows the calculation of the total energy consumption E_{total} of a machine. Many machines are operated with electrical energy, which generally results in $E_{\text{total}} = E_{\text{el}}$ and thus can be added as a rule to the class `:Machine`. If the machine uses other forms of energy, for example chemical energy in the case of a gas-fired continuous furnace, then the equation expands to $E_{\text{total}} = E_{\text{el}} + E_{\text{chem}}$.

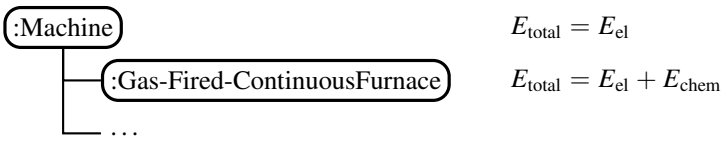

Fig. 7. Overwriting of mathematical rules in derived classes

7.7. Consideration of physical quantities and units

While it is possible to build mathematical models exclusively based on equations with variables, including quantities and units for the semantic description of the variables can improve comprehensibility, especially for complex models. Furthermore, it enables the automatic conversion between different units and the automatic checking of the equations for dimensional compatibility. A detailed listing of use cases for semantic modeling of quantity and unit systems is also given in [55].

Applications that only need to ensure semantically consistent use of units can rely on the *Measurement Units Ontology*[9] *(MUO)*. The ontology defines dimensions (named `PhysicalQuality` in MUO) and units, which can be used with these dimensions.

The QUDT ontologies[10] describe quantities, units, dimensions, and their mathematical relationships. In contrast to MUO, the QUDT ontologies also contain mathematical definitions for the dimensions of physical quantities. A detailed example of the application of SPARQL for checking dimension compatibility with QUDT contains [56, pp. 294].

The Ontology of Units of Measure and Related Concepts (OM) [55] was designed in parallel to QUDT and is also continuously developed. Besides the representation of quantity values, the ontology can also be used to convert units and check the dimensional compatibility of mathematical formulas.

A detailed evaluation of MUO, QUDT, OM, and other unit ontologies can be found in [57].

However, the rules and computational algorithms defined so far are agnostic to units and operate purely on numerical values. The Listing 12 shows three examples for representing the `:diameter` of a `:hole` in Turtle syntax. This quantity value is represented without units and with the unit centimeter in OM and QUDT. Independent of the specific representation of the quantity value, the area of the hole could be calculated according to the formula `area = pi / 4 * @diameter^2`. However, to correctly evaluate this formula, it is necessary to convert the quantity values encoded in OM or QUDT into a simple numerical value. Both QUDT and OM provide the required conversion factors and offsets for converting quantity values into their base units. The logic for the conversion can be easily integrated into the algorithm 3 that computes the property values.

Thus, the calculation results are also available in base units and can be converted into the desired target units by using QUDT or OM. However, the definition of target units requires means to include them as part of the model.

A simple solution for the use with *causal* modeling is the extension of the class `:Constraint` by an additional property `:targetUnit` to define the respective target unit. An example is shown in Listing 13[11], where the unit for the `:area` of class `:Hole` is set to square decimeter dm^2: Assuming that calculation results are available in

[9] http://elite.polito.it/ontologies/muo-vocab.owl and http://elite.polito.it/ontologies/ucum-instances.owl (03/01/2015)
[10] http://www.qudt.org/ (05/27/2022)
[11] The mathematical expression has been omitted for simplicity.

```
@base <http://example.org/> .

# diameter represented without unit
:hole :diameter 10 .

# diameter represented with OM
@prefix om: <http://www.ontology-of-units-of-measure.org/resource/om-2/> .
:hole :diameter [
  a om:Measure ;
  om:hasNumericalValue 10 ;
  om:hasUnit om:centimetre
] .

# diameter represented with QUDT
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix qudt-unit: <http://qudt.org/vocab/unit/> .
:hole :diameter [
  a qudt:QuantityValue ;
  qudt:value 10 ;
  qudt:unit qudt-unit:CentiM
] .
```

Listing 12: Examples for representing a diameter with and without units

```
@base <http://example.org/> .
@prefix mathrl: <http://numerateweb.org/vocab/math/rules#> .
@prefix qudt-unit: <http://qudt.org/vocab/unit/> .

:Hole a owl:Class ;
    mathrl:constraint [
        mathrl:onProperty :area ;
        mathrl:targetUnit qudt-unit:DeciM2 ;
        mathrl:expression [ ... ]
    ] .

:hole :a :Hole .
```

Listing 13: Example for specifying the target unit by using the property `:targetUnit`

their base units, in this example m^2, converting them into their target units is easy. For this purpose the conversion factor m (`qudt:conversionMultiplier`) and the conversion offset (`qudt:conversionOffset`) defined in QUDT can be used. For the given example therefore the conversion from m^2 to dm^2 by $dm^2 = m^2/0.01 + 0$ is possible.

In order to be able to use this extension also in combination with the POPCORN-LD-based rule syntax as introduced in Section 7.3, it can be extended as follows:

```
<RDF property> unit <unit> = <expression in POPCORN-LD>.
```

This would extend the rule for calculating the area of a hole to:

```
area unit qudt-unit:DeciM2 = pi / 4 * @diameter^2
```

As a further simplification, it would also be possible to specify the rule as:

```
area unit `dm^2` = pi / 4 * @diameter^2
```

However, for this, the expression `'dm^2'` has to be evaluated and be mapped to a corresponding QUDT or OM unit.

In order to keep the rule syntax agnostic to units and also to support *acausal* modeling approaches, the specification of the units can be implemented via OWL restrictions as shown in Listing 14. This kind of representation would

```
Prefix : <http://example.org/>
Prefix qudt: <http://qudt.org/schema/qudt/> .
Prefix qudt-unit: <http://qudt.org/vocab/unit/>

Class: Hole
SubClassOf: area exactly 1,
  area only (qudt:unit value qudt-unit:DeciM2)
Constraint:
  area = pi / 4 * @diameter^2
```

Listing 14: Example for using OWL to specify target units of computations

also allow rules with multiple variables on the left side for acausal modeling. However, in this case, a computational algorithm must also evaluate the relevant constraints defined with OWL besides the mathematical rules themselves.

8. Evaluation

In order to evaluate the presented concepts and methods, two case studies are conducted.

The first case study *OpenMath Content Dictionaries* (Section 8.2) investigates the representation of existing OpenMath libraries as RDF, including symbol definitions and examples. The focus lies on the representation of a wide range of mathematical expressions and introspection with the query language SPARQL to enable search queries for mathematical content.

The second case study *process chain planning and evaluation* (Section 8.3) investigates the mapping of different technological process chains with the aim of calculating the energy demand and the resulting production costs, thus enabling comparisons of different planning variants. In this case study, the particular focus is on representing the structures of the processes while considering the production programs, technologies, and resources. Furthermore, the reuse of partial models is explored.

In order to enable the implementation of these case studies and also to create the basis for other use cases, the software framework described insection 8.1 was developed, which contains various modules for the implementation of Numerate Web applications.

8.1. Software framework and base modules

The implementation of the software components for the presented concepts is based on the *Knowledge Modeling and Management Architecture (KOMMA)* [58, 59], a framework for building RDF-based applications using Java. KOMMA contains essential components that, based on the classes and properties described in OWL ontologies, can be used to create RDF-based applications, including the corresponding user interfaces, quickly.

For the implementation of the Numerate Web concepts, KOMMA was extended by appropriate modules that enable the processing and representation of mathematical objects and the execution of calculations[12].

[12]The Java implementation of the Numerate Web components is available at https://github.com/numerateweb/numerateweb (06/02/2022)

Figure 8 shows the resulting software architecture. Components highlighted in gray represent new or further developments within the scope of the work.

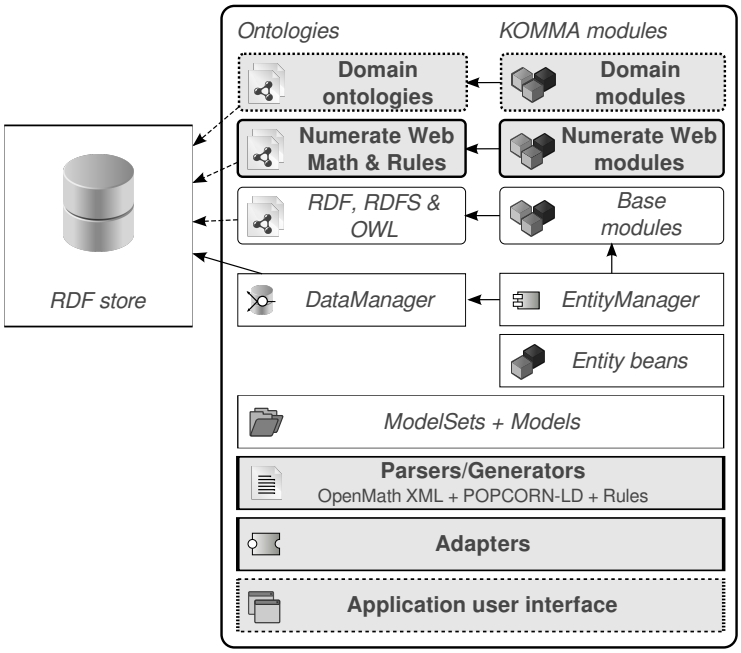

Fig. 8. Software architecture for implementing the Numerate Web concepts

Typically, a KOMMA application consists of ontologies (on the left side of the Figure 8), which are complemented by so-called KOMMA modules. A KOMMA module contains interface and behavior classes that enable object-oriented access to RDF data and bind logic to concepts and properties of the ontology. As a basis for Numerate Web applications, KOMMA modules for mathematical objects and rules (Numerate Web modules) have been developed. In order to realize specific applications for different use cases, the extension with additional domain ontologies is possible.

For the textual input and output of mathematical expressions and computation rules, parsers and generators for POPCORN-LD and the associated rule language (Figure 8 lower part) have been developed.

Correspondingly, the adapters used by KOMMA for the representation and editing of model objects were extended. This adapter mechanism allows existing applications based on the KOMMA framework to deal with Numerate Web objects. For example, the KOMMA ontology editor, for example, supports textual input and output of mathematical expressions in POPCORN-LD.

8.2. Case study 1: OpenMath Content Dictionaries

As already described in section 5, OpenMath uses *Content Dictionaries* to capture the meaning of mathematical symbols. This case study aims to convert the existing content dictionaries accessible via the OpenMath website into RDF, visualize them, and query the mathematical objects they contain with a query language.

For the RDF representation of the metadata described in XML, an ontology was developed to capture mathematical symbols and their meaning. A detailed description of the ontology and its elements can be found in [31].

For the verification of the RDF encoding based on the developed ontology, 214 content dictionaries published via the OpenMath website with a total number of 1578 symbols[13] were converted into an RDF representation.

[13] as of 07/04/2019

8.2.1. Webapplication to display the Content Dictionaries

To investigate the possibilities of implementing knowledge management applications based on the RDF representation of mathematical objects, a catalog application was developed for content dictionaries and their contained symbols. This application uses the software platform *eniLINK*[14], an extension of the software architecture from Figure 8 to include the web framework *Lift*[15] for implementing RDF-based web applications.

The platform uses HTML+RDFa[16] templates that combine HTML with embedded RDF annotations (RDF in Attributes, RDFa[17] for short). The platform translates these templates into SPARQL queries, parameterizes them, and applies them to the corresponding RDF data to generate full HTML representations.

Figure 9 shows the procedure for generating an HTML representation for symbol descriptions, including a graphical representation of their mathematical properties based on the RDF representation of the OpenMath content dictionaries.

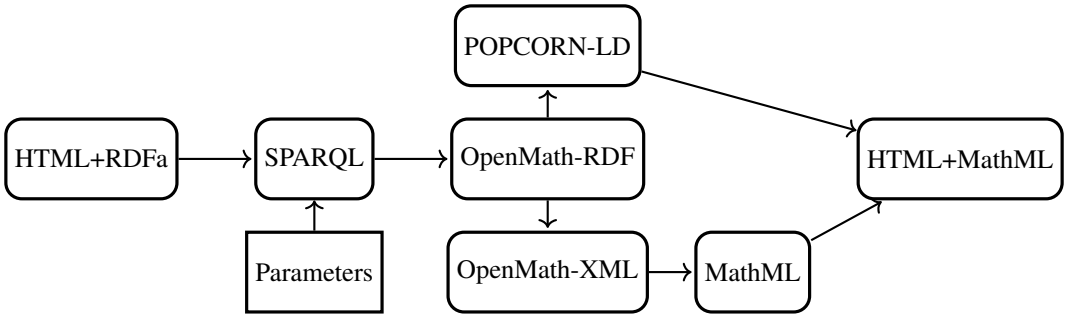

Fig. 9. Generation of an HTML representation based on HTML+RDFa templates

Based on an HTML+RDFa template, the Content Dictionaries and contained symbols are visualized. Mathematical expressions in OpenMath-RDF format are converted to POPCORN-LD and OpenMath-XML for further translation into Presentation MathML. The functions for translating OpenMath-RDF to POPCORN-LD and OpenMath-XML are already provided by the software framework described in Section 8.1. For a graphical representation, the conversion of OpenMath-XML into Presentation MathML is necessary. For this, the existing transformation rules[18] provided by the OpenMath community can be reused.

For each symbol in a Content Dictionary, the data stored for it is prepared and visualized accordingly. As described above, the representation is directly based on RDF data and thus adapts to changes. Furthermore, navigation between symbols is possible via the red highlighted references within the POPCORN-LD expressions. The application example shows that OpenMath-RDF, in combination with the textual notation POPCORN-LD, can represent the existing OpenMath language and can thus be used as a basis for implementing linked data-based tools for mathematical knowledge management.

In the following, the extension of the application with search functionality for mathematical expressions is described to demonstrate that the developed concepts are particularly applicable for the introspection of mathematical expressions.

8.2.2. Search in mathematical expressions

Information retrieval in the field of mathematics is mainly concerned with three problems: the search for mathematical documents with specific contents, the synthesis of mathematical documents, for example, for teaching purposes, and the pattern-based search for formulas using search queries with mathematical expressions and variables [60].

[14] http://platform.enilink.net/ (05/31/2022)
[15] http://liftweb.net/ (05/31/2022)
[16] https://www.w3.org/TR/html-rdfa/ (05/31/2022)
[17] https://www.w3.org/TR/rdfa-primer/ (05/31/2022)
[18] https://github.com/OpenMath/CDs/tree/master/lib/xsl (05/30/2022)

times http://www.openmath.org/cd/arith1#times
in *http://www.openmath.org/cd/arith1*

The symbol representing an n-ary multiplication function.

CMP

for all a,b | a * 0 = 0 and a * b = a * (b - 1) + a

CMP

for all a,b,c | a*(b+c) = a*b + a*c

FMP describe

$$\forall\ a, b.\ a0 = 0 \wedge ab = a(b-1) + a$$

```
quant1:forall[$a, $b -> $a * alg1:zero = alg1:zero and $a * $b = $a * ($b - alg1:one) + $a]
```

XML

FMP describe

$$\forall\ a, b, c.\ a(b+c) = ab + ac$$

```
quant1:forall[$a, $b, $c -> $a * ($b + $c) = $a * $b + $a * $c]
```

XML

Example describe

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

```
meta:Example(linalg2:matrix(linalg2:matrixrow(1, 2), linalg2:matrixrow(3, 4)) * linalg2:matrix(linalg2:matrixrow(5,
6), linalg2:matrixrow(7, 8)) = linalg2:matrix(linalg2:matrixrow(19, 22), linalg2:matrixrow(43, 50)))
```

XML

Fig. 10. Representation of the symbol `arith1:times` within a web application

Based on the formulas contained in the OpenMath Content Dictionaries, it can be shown that an RDF representation in combination with the SPARQL query language is suitable for implementing pattern-based search methods on mathematical formulas.

The search patterns themselves are also encoded as mathematical expressions in POPCORN-LD and are stored according to the step chain from Figure 11 first in OpenMath-XML, then in OpenMath-RDF, and finally into the query language SPARQL.

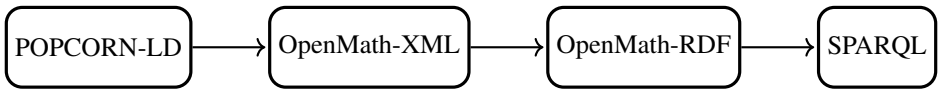

Fig. 11. Translation of POPCORN-LD queries to SPARQL

In order to encode search queries in POPCORN-LD, a Content Dictionary named `patterns` has been developed. This Content Dictionary and an extension for POPCORN-LD are described in [61].

Using the `patterns` Content Dictionary, for example, the description of patterns like

```
patterns:root(patterns:descendant(
  patterns:any_of(arith1:sum, arith1:product)
))
```

in POPCORN-LD, respectively, using short forms for the symbols as

```
.^(..+(.|(sum, product)))
```

is possible. The example pattern finds mathematical expressions that contain sums or products in in any nesting depth.

With the transformation rules defined in [61], this expression can be transformed into the following equivalent SPARQL query[19].

```
SELECT ?result WHERE {
  {
    ?n0 (<>)? <http://www.openmath.org/cd/arith1#sum> .
  } UNION {
    ?n0 (<>)? <http://www.openmath.org/cd/arith1#product> .
  }
  ?result (math:arguments|math:symbol|...|rdf:rest)+ ?n0 .
  FILTER NOT EXISTS {
    [] math:arguments|math:symbol|...|rdf:rest ?result .
  }
}
```

As can be seen, the formulation of the search pattern using POPCORN-LD allows for a much more compact and understandable representation than using the query language SPARQL. The patterns also enable people to formulate appropriate queries without more profound knowledge about RDF and SPARQL.

A screenshot of the corresponding prototypical application is shown in Figure 12.

The visual representation comprises the definitions of the symbols contained in the existing OpenMath Content Dictionaries and the mathematical expressions included in the respective examples.

For the example expression `.^(..+(.|(sum, product)))`, the generated SPARQL query returns multiple results since the existing Content Dictionaries include several definitions and examples containing sums or products as subexpressions. The first three results are visible in Figure 12. The first hit corresponds to an example of the application of a symbol (recognizable by the operator `meta:Example`), whereas the other two are expressions for the formal definition of symbols.

8.3. Case study 2: process chain planning and evaluation

The second case study has its roots in the *Cluster of Excellence eniPROD* (Energy-Efficient Product and Process Innovations in Production Engineering), where, amongst others, investigations on the optimization of a process chain for the hot forming of sheet metal were carried out.

Figure 13 illustrates the examined reference process chain for the production of a passenger car B-pillar foot. Besides the development and optimization of various technologies for press hardening, methods for the evaluation of the energy consumption [62, 63] and the economic efficiency [64] of process chains were explored.

The evaluation of economic efficiency emerges from the fact that the realization of the individual process steps is possible with different technologies. For example, both shear cutting and laser cutting are feasible technologies for the initial cutting of the blanks. For both processes, different machines with different technological properties concerning working range, working speed, or maximum processable sheet thicknesses are usable.

In order to evaluate the cost-effectiveness of a specific version of the process chain, it is necessary to consider the investment and production costs of the processes [64, p. 385]. In the context of the work [64], spreadsheets for the corresponding calculations wer developed with the software *Microsoft Excel*.

8.3.1. Ontology for processes and resources

As an example of the cost calculations for individual sub-processes of the process chain, Figure 14 presents the calculation sheets for a handling process and the related master data.

[19] The property path `math:arguments|math:symbol|...|rdf:rest` is an abbreviation for `math:arguments|math:symbol|math:operator|math:target|math:variables|math:binder|math:body|math:attributeKey|math:attributeValue|rdf:first|rdf:rest`, which can be used to traverse OpenMath-RDF objects.

.^(..+(.|(sum, product)))

Examples SPARQL

describe

Example $\left(\text{This represents the summation of the reciprocals of all the integers between 1 and 10 inclusive.}, \sum_{x=1}^{10} \frac{1}{x}\right)$

```
meta:Example("This represents the summation of the reciprocals of all the integers between\n 1 and 10 inclusive.", su
m(interval1:integer_interval(1, 10), $x -> 1 / $x))
```

XML

describe

$$\text{Bell}(n) = \sum_{k=0}^{n} \text{Stirling2}(n, k)$$

```
combinat1:Bell($n) = sum(interval1:integer_interval(alg1:zero, $n), $k -> combinat1:Stirling2($n, $k))
```

XML

describe

$$\text{Stirling1}(n, m) = \sum_{k=0}^{n-m} -1^k \times \binom{n-1+k}{n-m+k} \times \binom{2n-m}{n-m-k} \times \text{Stirling2}(n, m)$$

```
combinat1:Stirling1($n, $m) = sum(interval1:integer_interval(alg1:zero, $n - $m), $k -> -(alg1:one) ^ $k * binomial
($n - alg1:one + $k, $n - $m + $k) * binomial(2 * $n - $m, ($n - $m) - $k) * combinat1:Stirling2($n, $m))
```

XML

Fig. 12. Search results within the web application

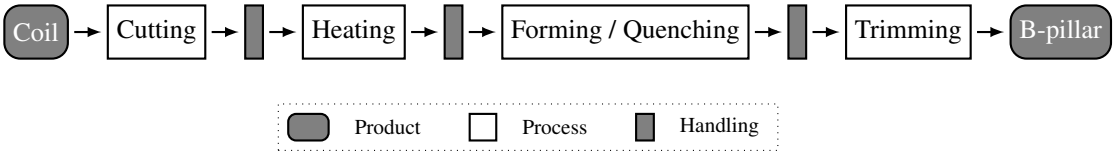

Fig. 13. Process chain for direct press hardening

In reality, the data contained in the two tables relate to different objects. However, they are mashed up within a single view. For example, the attributes *investment costs* and *required space* refer to the operating resource, *power consumption* and *utilization* to the process, *working hours* to the production program, and the *occupancy cost rate* to the company's business data. Using such a spreadsheet, a clear separation of the individual domain objects is not apparent. With a semantic model of the domain, a well-structured representation can be achieved.

Different approaches can be followed to create an ontology for a semantic model out of spreadsheet data. For example, when using a low modeling depth, the ontology could reflect the spreadsheets and solely consist of the classes *Process* to represent all properties of a process and *MasterData* for the essential business data. However, according to the design criteria by Gruber [65] such an ontology would at least violate the criteria *clarity* and *extensibility* since, by using a mixture of properties, a clear definition of the class *Process* would not be given. To better meet these criteria, the following classes were defined in the ontology:

ProcessBluePrint describes the individual processes in a process chain, including predecessors and successors.
MasterData represents essential business data.
Plan summarizes data on the production program with corresponding quantities and the related business data.

Handling process 1: Unstacking and loading of furnace	
Investment costs [€]	130,000.00
Economic life [years]	15.00
Depreciation [€/year]	**8,666.67**
Interest rate	0.07
Imputed interest [€/year]	**4,550.00**
Required space [m²]	60.00
Occupancy cost rate [€/m²]	160.00
Occupancy costs [€/year]	**9,600.00**
Input power electrical [kW]	45.60
Energy price electrical [€/kWh]	0.08
Work hours [h/year]	6,000.00
Utilization	0.93
Full load hours [h/year]	5,555.56
Energy costs electrical [€/year]	**20,266.67**
Maintenance costs [€/year]	**1,500.00**
Costs per year [€/year]	**44,583.33**
Work hours per year [h/year]	**6,000.00**
Utilization	**0.93**
Full load hours [h/year]	**5,555.56**
Machine hour rate [€/h]	**8.03**
Costs per second [€/s]	**0.0022**
Taktime [s]	**25.00**
Costs per process [€/process]	**0.05573**
Costs per part (two parts per process) [€/part]	**0.02786**

Master data	
Production volume [pcs./year] *	1,600,000
Production volume per day [pcs./day] *	6,400
Working days: [number] *	250
Working hours [h/year] *	6,000
Working hours [h/day] *	24
Shifts per day [number] *	3
Shifts per week: [number] *	21
Wage rate for personnel [€/h]	35
Material price 22MnB5 [€/t] *	1,000
Degree of material utilization *	0.70
Net weight B-pillar reinforcement [kg] *	4.25
Interest rate [%]	7
Occupancy cost rate [€/m²]	160
Energy price electric [€/kWh]	0.08
Energy price gas [€/kWh]	0.03

Fig. 14. Spreadsheets for a handling process and related master data

Process describes a specific realization of a process (`ProcessBluePrint`) with an operating resource (`Resource`) according to a given production program (`Plan`).

ProcessWithGas[20] comprises process realizations that use gas in addition to electrical energy.

Resource describes an operating resource (robot, machine tool, and others) with its properties and required tools (`Tool`).

Tool represents tools that are used by an operating resource (`Resource`). `Tool` is a subclass of `Resource`.

The ontology structure is designed to separate the description of the subprocesses of the process chain (`ProcessBluePrint`) from the actual realization of the individual processes (`Process`). This procedure allows the modeling of different realization variants that are linked to the same process chain structure. An overview of the properties of the individual classes is given in Table 3.

Object properties establish relationships between objects. They are used to describe and link the structure of the process chain, process realizations, required operating resources, the production program, and the master data. Datatype properties represent numerical attributes of the objects and have either integer or floating-point numbers as values. A semantic model of the process chain and the necessary operating resources can be created based on the ontology. Figure 15 shows a modeled process chain in a KOMMA-based editor.

The left view represents its hierarchical structure, while the right view allows editing the properties of individual elements. Unlike in a spreadsheet, the semantic model of the objects is visible as a tree. Properties of the ob-

[20]The class `ProcessWithGas` is used here as an example of specialization of `Process` in order to show the further possibilities for the inheritance of mathematical rules.

Table 3
Classes and properties of the process chain ontology

Class	Properties
ProcessBluePrint	**subProcess: ProcessBluePrint [0..*]**, **precedes: ProcessBluePrint [0..*]**
MasterData	energyPriceElectrical, energyPriceGas, interestRate, occupancyCostRate, workHoursPerDay, workHoursPerYear
Plan	**masterData: MasterData**, partsPerHour, partsPerYear, taktTime
Process, ProcessWithGas	**plan: Plan**, **realizes: ProcessBluePrint**, **resource: Resource**, costsPerPart, costsPerYear, energyCosts, energyCostsElectrical, energyCostsGas, fullLoadHours, imputedInterest, machineHourRate, maxPartsPerHour, occupancyCosts, partsPerCycle, resourceCount, utilization
Resource, Tool	**uses: Tool [0..*]**, capacity, depreciation, economicLife, inputPowerElectrical, inputPowerGas, investmentCosts, maintenanceCosts, price, processTime, requiredSpace

owl:ObjectProperty: range [multiplicity], owl:DatatypeProperty

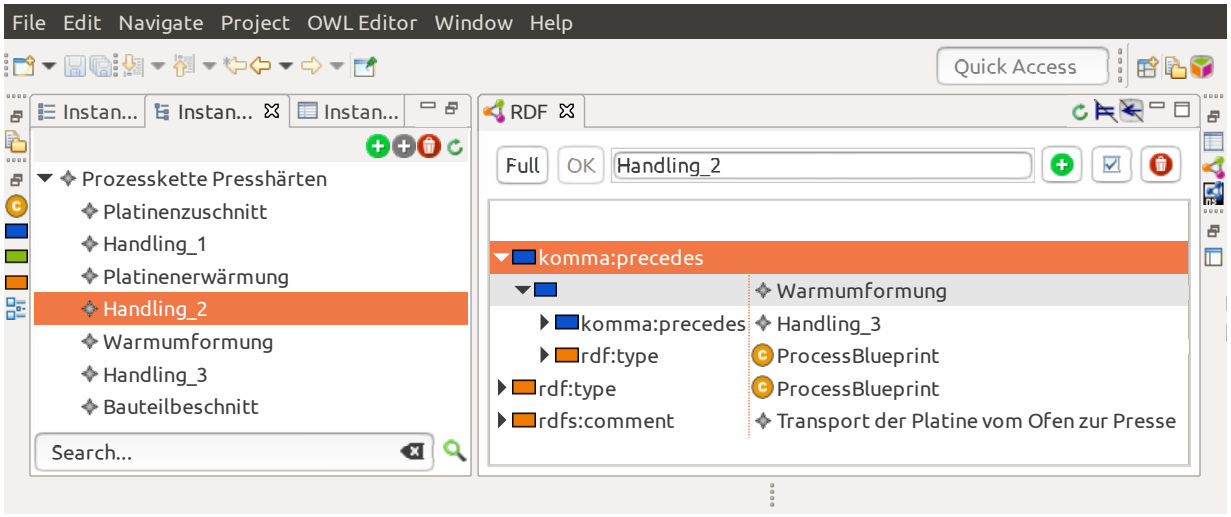

Fig. 15. RDF-based model of the press hardening process chain

jects and relationships between them, such as predecessors and successors of a process, are described semantically unambiguously by the RDF-based model and can be edited by the user according to the underlying ontology.

8.3.2. Mathematical modeling

The next step was to build the mathematical model for cost calculation, whereby certain assumptions and simplifications were made. Specifically for the properties `inputPowerElectrical`, `inputPowerGas` and `processTime`, a comparatively pragmatic approach was chosen. Each of these provides a fixed value for the respective operating resource. In reality, the values usually depend on the properties of the workpiece and the process parameters. However, this does not represent a limitation since extensibility can be achieved through inheritance and overwriting the mathematical rules in subclasses.

Listing 15 contains a selection of the relevant rules. The ontology defines the class `ProcessWithGas` as a subclass of `Process`. Thus, `ProcessWithGas` inherits the OWL constraints and the mathematical rules de-

fined on `Process`. The rule for the property `energyCosts` is a particular case since it is redefined by the `ProcessWithGas` class. When calculating the total energy costs, the extended formula includes gas consumption and electrical energy costs.

Class: Process
Constraint:
energyCosts = **@energyCostsElectrical**,
energyCostsElectrical = **@resourceCount** ∗ **@energyElectrical** ∗
@energyPriceElectrical(**@masterData**(**@plan**)),
energyElectrical = **@inputPowerElectrical**(**@resource**) ∗ **@fullLoadHours**,
processTime = **@processTime**(**@resource**),
resourceCount = ceiling((**@processTime** / **@capacity**(**@resource**)) / **@taktTime**(**@plan**)),

Class: ProcessWithGas
SubClassOf: Process
Constraint:
energyCosts = **@energyCostsElectrical** + **@energyCostsGas**
energyCostsGas = **@resourceCount** ∗ **@energyGas** ∗
@energyPriceGas(**@masterData**(**@plan**)),
energyGas = **@inputPowerGas**(**@resource**) ∗ **@fullLoadHours**

Listing 15: Excerpt of the rules for calculating the number of resources and the enery costs

If the specification of a constant processing time is not sufficient, then it can be calculated via a rule as shown in Listing 16. The example assumes that products are described by a set of features of the type `Feature`, which are characterized by a certain cutting length (`cuttingLength`). A specific feature type is a circular hole (`CircularHole`), whose cutting length depends on the respective hole diameter. The processing time (`processTime`) for laser cutting (`LaserProcess`) can then be calculated as the sum of the cutting lengths of the individual features multiplied by the cutting speed (`cuttingSpeed`). The speed is assumed to be constant in the example but could, in turn, also depend on the laser power, the material, the sheet thickness, and the desired quality.

Class: LaserProcess
SubClassOf: Process, cuttingSpeed exactly 1, product some Product
Constraint:
processTime = sum(**@@feature**(**@product**), $f -> **@cuttingLength**($f) ∗ **@cuttingSpeed**)

Class: Feature
SubClassOf: cuttingLength exactly 1

Class: CircularHole
SubClassOf: Feature, diameter exactly 1
Constraint:
cuttingLength = pi ∗ **@diameter**

Class: Product
SubClassOf: feature only Feature

Listing 16: Calulation of processing times based on product features

8.3.3. Module system and external data

It is possible to combine OWL classes and properties, mathematical rules and related instance data in a single ontology. However, in order to support the criterion of *extensibility* according to Gruber [65], it is helpful to at least separate the terminology and the instance data.

The structure shown in Figure 16 goes one step further and also separates the vocabulary for processes and resources from the mathematical rules for the calculations. The module system uses the import mechanism of OWL. Each of the blocks corresponds to an OWL ontology and the arrows to an import relation, which is represented in an RDF graph via the property `owl:imports`. Modules with solid outlines were developed for the use case described here, while modules with dashed borders represent existing ontologies. The latter include the Numerate Web ontologies and a base ontology of the KOMMA framework for the description of parent-child and predecessor-successor relationships.

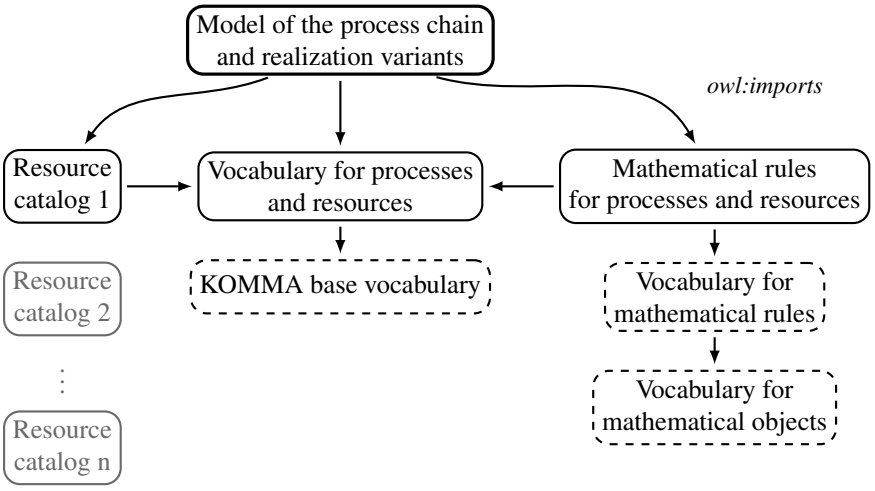

Fig. 16. Modules for the description of process chains and realization variants

As can be seen, the module *Resource catalog* imports only the module *Vocabulary for processes and resources* and therefore does not require knowledge about mathematical relationships between the properties of the modeled elements. Elements of the resource catalog are either machines or tools. The following is an example of a laser cutting machine `:LaserCuttingStation` with the necessary properties for cost calculation. The URI `http://example.org/processes#` corresponds to the namespace of the ontology for describing processes and resources and is abbreviated in the example by the prefix `p`.

```
@prefix p: <http://example.org/processes#> .
:LaserCuttingStation a p:Resource; p:capacity 1;
  p:economicLife 15; p:inputPowerElectrical 6.0;
  p:maintenanceCosts 31110.0; p:price 700000.0;
  p:processTime 120.0; p:requiredSpace 50.0 .
```

In addition to the data required for cost calculation, a resource can be associated with other properties and data. Corresponding vocabulary is, for example, available in the GoodRelations ontology [66], the definitions of which also have been integrated into the Schema.org vocabulary[21]. An example of the use of the GoodRelations ontology for the domain *mountain sports equipment* gives [67]. Although these are different products, the basic concepts can also be applied to machines and tools.

[21] http://schema.org/ (12/20/2019)

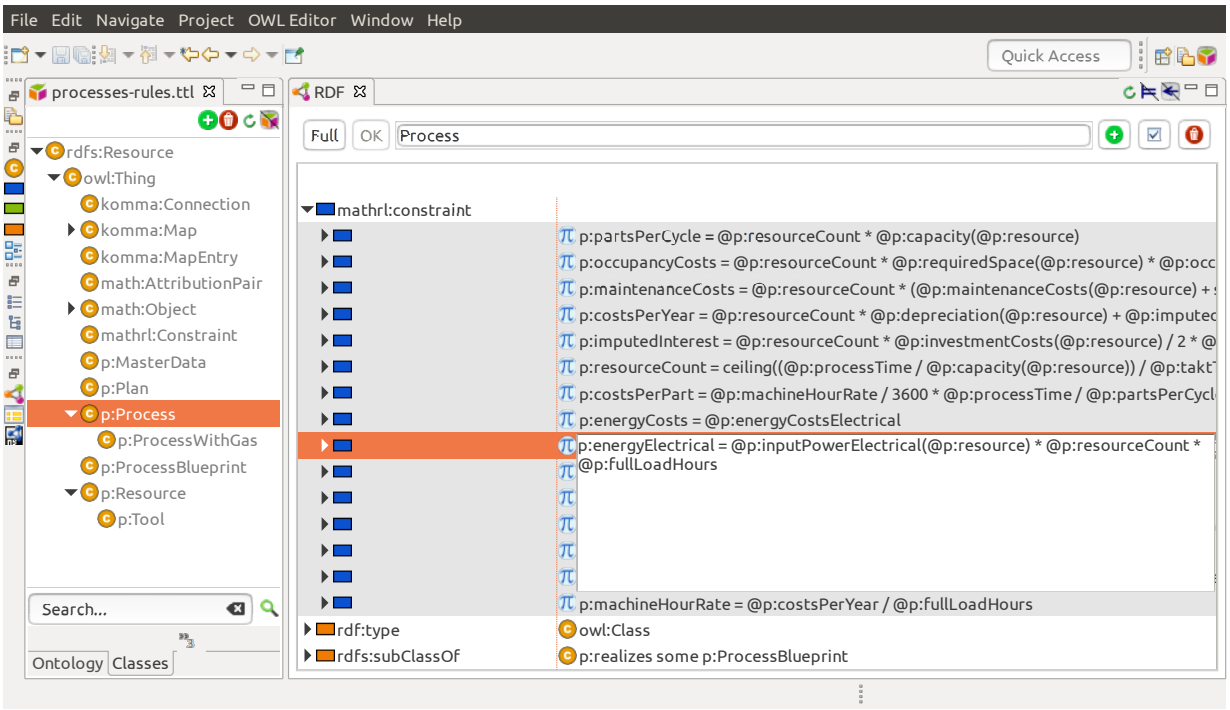

Fig. 17. Editing of the rules for the class `Process`

The resource catalogs used for modeling are flexibly selectable. Any additional catalog can be added to the model, as shown in the following example, via the import mechanism of OWL. The use of `owl:imports` has the advantage that any OWL-compatible software is able to load the submodules automatically.

```
@base <http://example.org/> .
<model> a owl:Ontology; owl:imports <vocab/processes>, <processes/rules>,
  <resources>,
  <http://other-company.com/machines>, ..., <resources_n> .
```

The data can also be retrieved from external web servers. For example, a hypothetical company with a website at `other-company.com` may host its resource catalog at `http://other-company.com/machines`. Thus, the integration of external data in mathematical models is possible, if it is available in an RDF format and an aligned vocabulary, in this case, the *vocabulary for processes and resources*.

8.3.4. Software for modeling and computation

For modeling the classes and properties as well as the associated computation rules, the software framework presented in section 8.1 extended by appropriate editing functions for mathematical expressions are used. Figure 17 shows on the left side the class hierarchy of the imported ontologies as well as on the right side the properties of the class `Process`, including the computation rules associated with it. By appropriate extensions developed in the context of this work, the editor allows the representation and editing of mathematical formulas in the POPCORN-LD syntax. Therefore, like other RDF properties, individual mathematical rules are editable via text fields, comparable to a spreadsheet.

Based on the defined ontologies, instances of the predefined classes can be created and edited. The figure 18 shows the model of a handling process along with the associated production program and master data.

All numerical properties of the handling process are calculated using mathematical rules, analogous to the initially presented spreadsheet in Figure 14. A comparison of the results indicates that the calculations of the RDF-based solution match the spreadsheet's calculations. Only the presentation of the numbers differs because the spread-

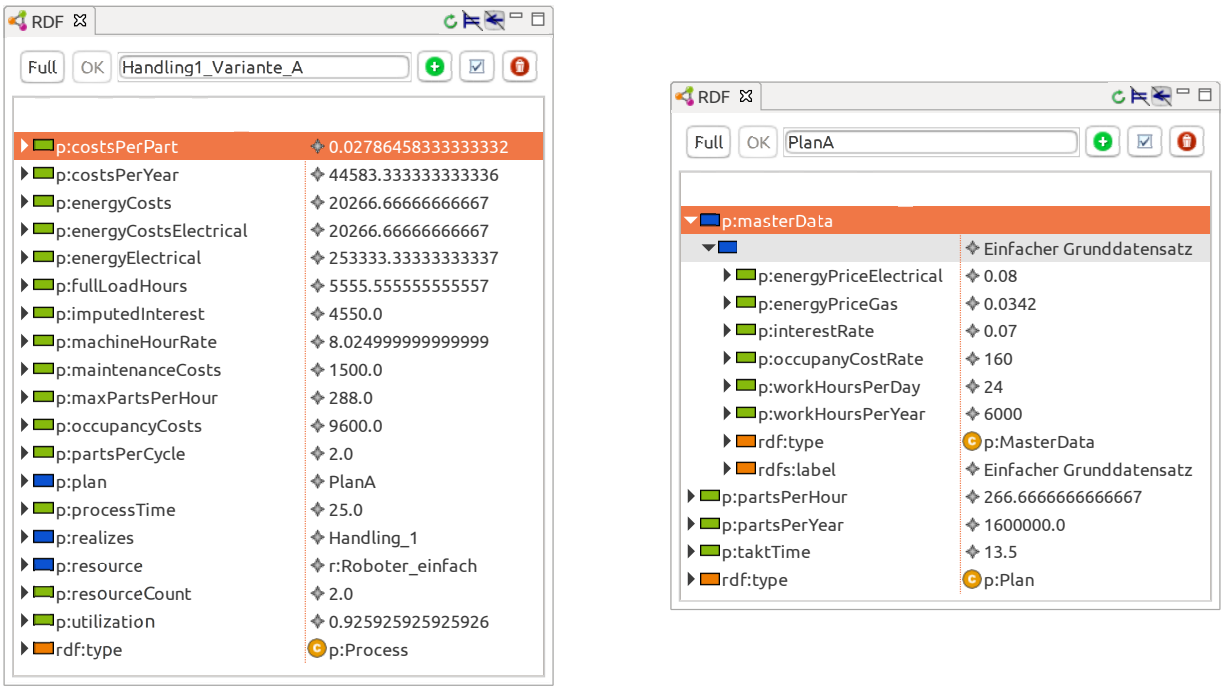

Fig. 18. RDF-representation of a handling process (left) as well as of a production program and master data (right) with calculated properties

sheet uses a fixed number of decimal places, whereas the RDF-based editor outputs the numbers without specific formatting.

8.4. Discussion of the case studies

The first case study explored how the methods and concepts developed in this work can be used to represent the available language set of OpenMath in RDF, including the associated metadata to enable catalog applications and structural search on mathematical expressions. Therefore, it can also serve as an example for a broader range of applications within the field of mathematical knowledge management, especially when Semantic Web technologies are considered.

The second case study explored the application of the Numerate Web concept to the modeling and evaluation of process chains. By implementing an ontology, the structure of process chains could be represented within a semantic model that provided the basis for a graphical editor and the definition of an extensible mathematical model. The models can be subdivided by an OWL-based module system into distributed resource catalogs and can be combined as needed for specific planning tasks. Finally, the case study demonstrated the integration of the models and calculations in a software system.

Especially the second case study demonstrates the potential of the presented Numerate Web concepts for future applications within engineering, economics, and related domains. It may help create distributed mathematical models where additional data can be integrated via linked data concepts as required.

9. Conclusions and future work

This work aimed to develop an approach to describe mathematical models by using structural models of (technical) systems for applications in engineering and related domains. Thus, concepts for multiscale modeling, RDF-based and textual representations for mathematical objects and rules, as well as algorithms for mathematical computations were presented and evaluated in two complementary case studies.

The developed causal modeling approach allows the representation of static system models comparable to typical spreadsheet software. Therefore, the same range of modeling tasks is supported as with spreadsheets. However, such a model has to define the direction of computation apriori, in that all equations must be formulated in resolved form.

Thereby the restriction arises that dynamic models with time-varying behavior cannot be calculated or can only be calculated by discretization and explicit representation of time-dependent variables within the structural model. In order to support dynamic models, future work may explore both the extension of the rule language defined in Section 7.2, among others, with respect to the definition of start conditions as well as the adaptation of the algorithms defined in Section 7.5. In this context, the representation of quantities in RDF must be revisited in order to take the time component into account and to model time series.

In addition, future work may include extending the rule language and computational algorithms to support acausal modeling approaches. For example, this could comprise concepts for representing systems of equations in an RDF model and the subsequent transformation into an equivalent Modelica model (or similar) to perform the computations. Such an extension could, for example, also allow the reverse mapping of Modelica models to an RDF representation, which is comparable to approaches like ModelicaXML [68] but based on Semantic Web technologies and graph databases. Thus, it would enable analysis, introspection, and transformation of complex systems of equations with SPARQL and other RDF tools.

Also, for exchanging physical quantities and units between OpenMath and RDF, further research is necessary. Questions are here the embedding of units as symbols in mathematical expressions while preserving compatibility between other OpenMath serializations and OpenMath-RDF, as well as a concept for the complete dimension checking of Numerate Web models.

For the practical application of the modeling approach, first of all, basic knowledge of OWL modeling and the POPCORN-LD language is necessary. Thus, the further simplification of the textual language and the development of intuitive authoring tools may be investigated. Also, the separation of semantic models and mathematical rules according to the roles and competencies of the respective modelers can be a subject of research. In this context, concepts for integrating CAD or PLM systems and the automated derivation of semantic models from existing system models are also interesting.

Further prerequisites for future applications are fast computation algorithms, which can be integrated into existing RDF databases or frameworks and combined with logical reasoning systems. These include methods for efficient tracing of dependencies for the partial recomputation when individual parameters change, support for efficient backward chaining, or the combination with RDFS and OWL inference.

In conclusion, the presented Numerate Web concepts can provide an essential basis for future applications in digital system models and mathematical knowledge management. Furthermore, they also offer room for further developments in subsequent research work.

References

[1] T. Tudorache, Employing Ontologies for an Improved Development Process in Collaborative Engineering, PhD thesis, Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik, 2006.

[2] J. Morbach, A. Yang and W. Marquardt, OntoCAPE – A large-scale ontology for chemical process engineering, *Engineering Applications of Artificial Intelligence* **20**(2) (2007), 147–161.

[3] P.W. Witherell, Semantic methods for intelligent distributed design environments, Dissertation, University of Massachusetts Amherst, 2009. http://proquest.umi.com/pqdlink?did=1917545091&Fmt=7&clientId=137359&RQT=309&VName=PQD.

[4] J. Morbach, A. Wiesner and W. Marquardt, OntoCAPE – A (re)usable ontology for computer-aided process engineering, *Computers & Chemical Engineering* **33**(10) (2009), 1546–1556, Selected Papers from the 18th European Symposium on Computer Aided Process Engineering (ESCAPE-18). doi:10.1016/j.compchemeng.2009.01.019. http://www.sciencedirect.com/science/article/B6TFT-4W7YXV0-1/2/ea404bde6a4995e395ca2373ef5694d9.

[5] S. Wanke, Neue Konzepte zur Verwaltung und Bereitstellung von Lösungen im Produktentwicklungsprozess : CPM/PDD-Lösungsmuster als Grundlage eines verhaltensbeschreibenden Lösungskataloges, PhD thesis, Universität des Saarlandes, 2010. http://scidok.sulb.uni-saarland.de/volltexte/2010/3159/.

[6] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, SPARQL 1.1 Query Language, *W3C Member Submission* (2004). hhttps://www.w3.org/Submission/SWRL/.

[7] S. Harris, A. Seaborne and E. Prud'hommeaux, SPARQL 1.1 Query Language, *W3C Recommendation* (2013). http://www.w3.org/TR/sparql11-query/.

[8] H. Knublauch, SPIN - SPARQL Syntax, *W3C Member Submission* (2011). https://www.w3.org/Submission/spin-sparql/.

[9] J. Qian, Z. Zhang, C. Shao, H. Gong and D. Liu, Assembly sequence planning method based on knowledge and ontostep, *Procedia CIRP* **97** (2021), 502–507. doi:10.1016/j.procir.2020.05.266. https://linkinghub.elsevier.com/retrieve/pii/S2212827120314980.

[10] C. Hildebrandt, M. Glawe, A.W. Müller and A. Fay, Reasoning on Engineering Knowledge: Applications and Desired Features, in: *The Semantic Web*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, eds, Springer International Publishing, Cham, 2017, pp. 65–78. ISBN 978-3-319-58451-5.

[11] A. Mohapatra and M. Genesereth, Aggregation in Datalog under set semantics, Technical Report, Tech. rep. 2012. url: http://logic.stanford.edu/reports/LG-2012-01.pdf, 2012.

[12] M.J. O'Connor and A.K. Das, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, in: *OWLED*, Vol. 529, 2009.

[13] Mathematical Markup Language (MathML) Version 3.0 2nd Edition, W3C, 2014. http://www.w3.org/TR/MathML3/.

[14] S. Buswell, O. Caprotti, D.P. Carlisle, M.C. Dewar, M. Gaetano, M. Kohlhase, J.H.D. Davenport, P.D.F. Ion and T. Wiesing, The OpenMath standard, Technical Report, The OpenMath Society, 2019, version 2.0 revision 2. https://openmath.org/standard/om20-2019-07-01/omstd20.pdf.

[15] M. Kohlhase, Using LaTeX as a Semantic Markup Format, *Mathematics in Computer Science* **2** (2008), 279–304.

[16] C. Lange, Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration, PhD thesis, Jacobs University Bremen, 2011.

[17] M. Kohlhase and F. Rabe, Semantics of OpenMath and MathML3, *Mathematics in Computer Science* **6**(3) (2012), 235–260. doi:10.1007/s11786-012-0113-x.

[18] C. Lange, Ontologies and languages for representing mathematical knowledge on the Semantic Web, *Semantic Web* **4**(2) (2013), 119–158. doi:10.3233/SW-2012-0059.

[19] M. Marchiori, The Mathematical Semantic Web, in: *Mathematical Knowledge Management*, Vol. 2594, A. Asperti, B. Buchberger and J.H. Davenport, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 216–223. ISBN 978-3-540-00568-1. http://www.springerlink.com/content/c087q2550g8k3867/.

[20] G. Schreiber and Y. Raimond, RDF 1.1 Primer, *W3C Working Group Note* (2014). http://www.w3.org/TR/rdf11-primer/.

[21] R. Cyganiak, D. Wood and M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, *W3C Recommendation* (2014). http://www.w3.org/TR/rdf11-concepts/.

[22] S. Ferré, An RDF Vocabulary for the Representation and Exploration of Expressions with an Illustration on Mathematical Search, 2012. https://hal.inria.fr/hal-00812197.

[23] D. Beckett, T. Berners-Lee, E. Prud'hommeaux and G. Carothers, RDF 1.1 Turtle, *W3C Recommendation* (2014). http://www.w3.org/TR/turtle/.

[24] E. Muñoz, E. Capón-García, J.M. Laínez-Aguirre, A. Espuña and L. Puigjaner, Using mathematical knowledge management to support integrated decision-making in the enterprise, *Computers & Chemical Engineering* **66** (2014), 139–150. doi:10.1016/j.compchemeng.2014.02.026. http://www.sciencedirect.com/science/article/pii/S0098135414000738.

[25] O. Nevzorova, N. Zhiltsov, A. Kirillovich and E. Lipachev, *OntoMath^{PRO}* Ontology: A Linked Data Hub for Mathematics, 2014.

[26] A.M. Elizarov, A.V. Kirillovich, E.K. Lipachev, O.A. Nevzorova, V.D. Solovyev and N.G. Zhiltsov, Mathematical knowledge representation: semantic models and formalisms, *Lobachevskii Journal of Mathematics* **35**(4) (2014), 348–354. doi:10.1134/S1995080214040143.

[27] A. Hogan, Linked Data & the Semantic Web Standards, in: *Linked Data Management*, A. Harth, K. Hose and R. Schenkel, eds, Chapman and Hall/CRC, 2014, pp. 3–48. http://www.crcnetbase.com/doi/abs/10.1201/b16859-3.

[28] A. Hoekstra, B. Chopard and P. Coveney, Multiscale modelling and simulation: a position paper, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **372**(2021) (2014), 20130377. doi:10.1098/rsta.2013.0377.

[29] A. Yang and W. Marquardt, An ontological conceptualization of multiscale models, *Computers & Chemical Engineering* **33**(4) (2009), 822–837. doi:10.1016/j.compchemeng.2008.11.015. http://linkinghub.elsevier.com/retrieve/pii/S0098135408002524.

[30] K. Wenzel and H. Reinhardt, Mathematical Computations for Linked Data Applications with OpenMath, in: *Joint Proceedings of the 24th OpenMath Workshop, the 7th Workshop on Mathematical User Interfaces (MathUI), and the Work in Progress Section of the Conference on Intelligent Computer Mathematics*, Vol. 921, CEUR Workshop Proceedings, Bremen, 2012, pp. 38–48. http://ceur-ws.org/Vol-921/openmath-01.pdf.

[31] K. Wenzel, *OpenMath-RDF: RDF Encodings for OpenMath Objects and Content Dictionaries*, 31st OpenMath Workshop, 2021. https://easychair.org/publications/preprint/XQwn.

[32] P. Horn and D. Roozemond, OpenMath in SCIEnce: SCSCP and POPCORN, in: *Intelligent Computer Mathematics*, Springer, 2009, pp. 474–479.

[33] C. Unicode Staff, *The Unicode standard: worldwide character encoding*, Addison-Wesley Longman Publishing Co., Inc., 1991.

[34] A. Phillips and M. Davis, Tags for identifying languages, Technical Report, BCP 47, RFC 4646, September, 2006.

[35] M. Horridge, N. Drummond, J. Goodwin, A.L. Rector, R. Stevens and H. Wang, The Manchester OWL syntax., in: *OWLed*, Vol. 216, 2006.

[36] M. Horridge and P.F. Patel-Schneider, OWL 2 Web Ontology Language Manchester Syntax, *W3C Working Group Note 5. September 2012* (2012). https://www.w3.org/2007/OWL/draft/owl2-manchester-syntax/.

[37] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi and P.F. Patel-Schneider (eds), *The Description Logic Handbook: Theory, Implementation and Applications*, 2 edition edn, Cambridge University Press, Cambridge, 2010. ISBN 9780521150118.

[38] D. Tomaszuk, RDF validation: a brief survey, in: *International Conference: Beyond Databases, Architectures and Structures*, Springer, 2017, pp. 344–355.

[39] E. Prud'hommeaux, J.E. Labra Gayo and H. Solbrig, Shape Expressions: An RDF Validation and Transformation Language, in: *Proceedings of the 10th International Conference on Semantic Systems*, SEM '14, ACM, New York, NY, USA, 2014, pp. 32–40. ISBN 978-1-4503-2927-9. doi:10.1145/2660517.2660523.
[40] A.G. Ryman, A. Le Hors and S. Speicher, OSLC Resource Shape: A language for defining constraints on Linked Data., *LDOW* **996** (2013).
[41] T. Bosch and K. Eckert, Towards description set profiles for RDF using SPARQL as intermediate language, in: *International Conference on Dublin Core and Metadata Applications*, 2014, pp. 129–137.
[42] C. Fürber and M. Hepp, Using SPARQL and SPIN for data quality management on the semantic web, in: *International Conference on Business Information Systems*, Springer, 2010, pp. 35–46.
[43] H. Knublauch, J.A. Hendler and K. Idehen, SPIN - Overview and Motivation, *W3C Member Submission* (2011). https://www.w3.org/Submission/spin-overview/.
[44] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), *W3C Recommendation* (2017). https://www.w3.org/TR/shacl/.
[45] M. Tiller, Block Diagrams vs. Acausal Modeling, in: *Introduction to Physical Modeling with Modelica*, Springer US, Boston, MA, 2001, pp. 255–264. ISBN 978-1-4615-1561-6. doi:10.1007/978-1-4615-1561-6_11.
[46] F.E. Cellier and E. Kofman, *Continuous system simulation*, Springer Science & Business Media, 2006.
[47] L.R. Petzold, Description of DASSL: a differential/algebraic system solver, Technical Report, Sandia National Labs., Livermore, CA (USA), 1982.
[48] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker and C.S. Woodward, SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers, *ACM Transactions on Mathematical Software (TOMS)* **31**(3) (2005), 363–396.
[49] M. Chein and M.-L. Mugnier, *Graph-based knowledge representation: computational foundations of conceptual graphs*, Springer Science & Business Media, 2008.
[50] C.L. Forgy, Rete: A fast algorithm for the many pattern/many object pattern match problem, in: *Readings in Artificial Intelligence and Databases*, Elsevier, 1989, pp. 547–559.
[51] A. Colmerauer and P. Roussel, The birth of Prolog, in: *History of programming languages—II*, ACM, 1996, pp. 331–367.
[52] J.H. Gallier, *Logic for computer science: foundations of automatic theorem proving*, Courier Dover Publications, 2015.
[53] B. Motik, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz et al., OWL 2 Web Ontology Language Profiles (Second Edition), *W3C recommendation* (2012). https://www.w3.org/TR/owl2-profiles/.
[54] K. Mehlhorn and P. Sanders, *Algorithms and data structures: the basic toolbox*, Springer, Berlin, 2008. ISBN 978-3-540-77977-3 978-3-540-77978-0.
[55] H. Rijgersberg, M. van Assem and J. Top, Ontology of Units of Measure and Related Concepts, *Semantic Web* **4**(1) (2012), 3–13.
[56] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist, Second Edition: Effective Modeling in RDFS and OWL*, 2nd edn, Morgan Kaufmann, 2011. ISBN 9780123859655.
[57] J.M. Keil and S. Schindler, Comparison and evaluation of ontologies for units of measurement, *Semantic Web* **10**(1) (2019), 33–51.
[58] K. Wenzel, Ontology-driven application architectures with KOMMA, in: *SWESE 2011 – 7th International Workshop on Semantic Web Enabled Software Engineering*, 2011.
[59] K. Wenzel, KOMMA: Modeling with RDF and linked data, Ludwigsburg, 2016.
[60] F. Guidi and C. Sacerdoti Coen, A Survey on Retrieval of Mathematical Knowledge, *Mathematics in Computer Science* **10**(4) (2016), 409–427. doi:10.1007/s11786-016-0274-0.
[61] K. Wenzel, *Pattern matching for mathematical expressions with OpenMath*, 31st OpenMath Workshop, 2021. https://easychair.org/publications/preprint/C6vL.
[62] A. Göschel, A. Sterzing and J. Schönherr, Balancing procedure for energy and material flows in sheet metal forming, *CIRP Journal of Manufacturing Science and Technology* **4**(2) (2011), 170–179. doi:10.1016/j.cirpj.2011.06.018. http://www.sciencedirect.com/science/article/pii/S1755581711000721.
[63] J. Schönherr, Achieving Energy Efficient Process Chains in Sheet Metal Forming, in: *Future Trends in Production Engineering*, G. Schuh, R. Neugebauer and E. Uhlmann, eds, Springer Berlin Heidelberg, 2013, pp. 331–341. ISBN 978-3-642-24490-2, 978-3-642-24491-9. http://link.springer.com/chapter/10.1007/978-3-642-24491-9_33.
[64] U. Götze, S. Zönnchen and J. Schönherr, Wirtschaftliche Bewertung von Prozesskettenvarianten am Beispiel von Strukturbauteilen, in: *Energetisch-wirtschaftliche Bilanzierung und Bewertung technischer Systeme – Erkenntnisse aus dem Spitzentechnologiecluster eniPROD*, Verlag Wissenschaftliche Scripten, 2013, pp. 375–396.
[65] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing?, *International Journal of Human-Computer Studies* **43**(56) (1995), 907–928. doi:10.1006/ijhc.1995.1081. http://www.sciencedirect.com/science/article/pii/S1071581985710816.
[66] M. Hepp, Goodrelations: An ontology for describing products and services offers on the web, in: *International conference on knowledge engineering and knowledge management*, Springer, 2008, pp. 329–346.
[67] A. Westerinen and R. Tauber, Integrating GoodRelations in a domain-specific ontology, *Applied Ontology* **12**(3–4) (2017), 323–340.
[68] A. Pop and P. Fritzson, ModelicaXML: A Modelica XML representation with applications, in: *Proceedings of the 3rd Modelica Conference*, Linköping, Sweden, 2003.