Specifying Conceptual Models Using Restricted Natural Language

Bayzid Ashik Hossain
Department of Computing
Sydney, NSW
Australia
bayzid-ashik.hossain@mq.edu.au

Rolf Schwitter
Department of Computing
Sydney, NSW
Australia
rolf.schwitter@mq.edu.au

Abstract

The key activity to design an information system is conceptual modelling which brings out and describes the general knowledge that is required to build a system. In this paper we propose a novel approach to conceptual modelling where the domain experts will be able to specify and construct a model using a restricted form of natural language. A restricted natural language is a subset of a natural language that has well-defined computational properties and therefore can be translated unambiguously into a formal notation. We will argue that a restricted natural language is suitable for writing precise and consistent specifications that lead to executable conceptual models. Using a restricted natural language will allow the domain experts to describe a scenario in the terminology of the application domain without the need to formally encode this scenario. The resulting textual specification can then be automatically translated into the language of the desired conceptual modelling framework.

1 Introduction

It is well-known that the quality of an information system application depends on its design. To guarantee accurateness, adaptability, productivity and clarity, information systems are best specified at the conceptual level using a language with names for individuals, concepts and relations that are easily understandable by domain experts (Bernus et al., 2013). Conceptual modelling is the most important part of requirements engineering and is the first phase towards designing an information system (Olivé, 2007). The conceptual design procedure generally includes data, process and behavioral perceptions, and the actual database management system (DBMS) that is used to implement the design of the information system (Bernus et al., 2013). The DBMS could be based on any of the available data models. Designing a database means constructing a formal model of the desired application domain which is often called the universe of discourse (UOD). Conceptual modelling involves different parties who sit together and define the UOD. The process of conceptual modelling starts with the collection of necessary information from the domain experts by the knowledge engineers. The knowledge engineers then use traditional modelling techniques to design the system based on the collected information.

To design the database, a clear understanding of the application domain and an unambiguous information representation scheme is necessary. Object role modelling (ORM) (Halpin, 2009) makes the database design process simple by using natural language for verbalization, as well as diagrams which can be populated with suitable examples and by adding the information in terms of simple facts. On the other hand, entity relationship modelling (ERM) (Richard, 1990; Frantiska, 2018) does this by considering the UOD in terms of entities, attributes and relationships. Object-oriented modelling techniques such as the unified modelling language (UML) (O´ Regan, 2017) provide a wide variety of functionality for specifying a data model at an implementation level which is suitable for the detailed design of an object oriented system. UML can be used for database design in general because its class diagrams provide a comprehensive entity-relationship notation that can be annotated with database constructs.

Alternatively, a Restricted Natural Language (RNL) (Kuhn, 2014) can be used by the domain experts to specify system requirements for conceptual modelling. A RNL can be defined as a subset of a natural language that is acquired by

Bayzid Ashik Hossain and Rolf Schwitter. 2018. Specifying Conceptual Models Using Restricted Natural Language. In *Proceedings of Australasian Language Technology Association Workshop*, pages 44–52.

constraining the grammar and vocabulary in order to reduce or remove its ambiguity and complexity. These RNLs are also known as controlled natural language (CNL) (Schwitter, 2010). RNLs fall into two categories: 1. those that improve the readability for human beings especially for non-native speakers, and 2. those that facilitate the automated translation into a formal target language. The main benefits of an RNL are: they are easy to understand by humans and easy to process by machines. In this paper, we show how an RNL can be used to write a specification for an information system and how this specification can be processed to generate a conceptual diagram. The grammar of our RNL specifies and restricts the form of the input sentences. The language processor translates RNL sentences into a version of description logic. Note that the conceptual modelling process usually starts from scratch and therefore cannot rely on existing data that would make this process immediately suitable for machine learning techniques.

2 Related Work

There has been a number of works on formalizing conceptual models for verification purposes (Berardi et al., 2005; Calvanese, 2013; Lutz, 2002). This verification process includes consistency and redundancy checking. These approaches first represent the domain of interest as a conceptual model and then formalize the conceptual model using a formal language. The formal representation can be used to reason about the domain of interest during the design phase and can also be used to extract information at run time through query answering.

Traditional conceptual modelling diagrams such as entity relationship diagrams and unified modelling language diagrams are easy to generate and easily understandable for the knowledge engineers. These modelling techniques are well established. The problems with these conventional modelling approaches are: they have no precise semantics and no verification support; they are not machine comprehensible and as a consequence automated reasoning on the conceptual diagrams is not possible. Previous approaches used logic to formally represent the diagrams and to overcome these problems. The description logic (DL) *ALCQI* is well suited to do reasoning with entity relationship diagrams (Lutz, 2002), UML class diagrams (Berardi et al., 2005) and ORM diagrams (Franconi et al., 2012). The DL *ALCQI* is an extension of the basic propositionally closed description logic *AL* and includes complex concept negation, qualified number restriction, and inverse role.

Table 1 shows the constructs of the *ALCQI* description logic with suitable examples. It is reported that finite model reasoning with *ALCQI* is decidable and ExpTime-complete[1]. Using logic to formally represent the conceptual diagrams introduces some problems too. For example, it is difficult to generate logical representations, in particular for domain experts; it is also difficult for them to understand these representations and no well established methodologies are available to represent the conceptual models formally. A solution to these problems is to use a RNL for the specification of conceptual models. There exist several ontology editing and authoring tools such as AceWiki (Kuhn, 2008), CLOnE (Funk et al., 2007), RoundTrip Ontology Authoring (Davis et al., 2008), Rabbit (Denaux et al., 2009), Owl Simplified English (Power, 2012) that already use RNL for the specification of ontologies; they translate a specification into a formal notation. There are also works on mapping formal notation into conceptual models (Brockmans et al., 2004; Bagui, 2009).

3 Proposed Approach

Several approaches have been proposed to use logic with conventional modelling techniques to verify the models and to get the semantics of the domains. These approaches allow machines to understand the models and thus support automated reasoning. To overcome the disadvantages associated with these approaches, we propose to use an RNL as a language for specifying conceptual models. The benefits of an RNL are: 1. the language is easy to write and understand for domain experts as it is a subset of a natural language, 2. the language gets its semantics via translation into a formal notation, and 3. the resulting formal notation can be used further to generate conceptual models.

Unlike previous approaches, we propose to write a specification of the conceptual model in RNL first and then translate this specification into description logic. Existing description logic rea-

[1] http://www.cs.man.ac.uk/∼ezolin/dl/

Construct	Syntax	Example
atomic concept	C	Student
atomic role	P	hasChild
atomic negation	$\neg C$	$\neg$ Student
conjunction	$C \sqcap D$	Student $\sqcap$ Teacher
(unqual.) exist. restriction	$\exists R$	$\exists$ hasChild
universal value restriction	$\forall R.C$	$\forall$ hasChild.Male
full negation	$\neg(C \sqcap D)$	$\neg$ (Student $\sqcap$ Teacher)
qual. cardinality restrictions	$\geq nR.C$	$\geq$ 2 hasChild.Female
inverse role	p^-	$\exists$hasChild$^-$.Teacher

Table 1: The DL *ALCQI*.

soners[2,3] can be used to check the consistency of the formal notation and after that desired conceptual models can be generated from this notation. Our approach is to derive the conceptual model from the specification whereas in conventional approaches knowledge engineers first draw the model and then use programs to translate the model into a formal notation (Fillottrani et al., 2012). Figure 1 shows the proposed system architecture for conceptual modelling.

3.1 Scenario

Let's consider an example scenario of a learning management system for a university stated below:

A Learning Management System (LMS) keeps track of the units the students do during their undergraduate or graduate studies at a particular university. The university offers a number of programs and each program consists of a number of units. Each program has a program name and a program id. Each unit has a unit code and a unit name. A student can take a number of units whereas a unit has a number of students. A student must study at least one unit and at most four units. Every student can enrol into exactly one program. The system stores a student id and a student name for each student.

We reconstruct this scenario in RNL and after that the language processor translates the RNL specification into description logic using a feature-based phrase structure grammar (Bird et al., 2009). Our RNL consists of function words and content words. Function words (e.g., determiners, quantifiers and operators) describe the structure of the RNL and their number is fixed. Content words (e.g, nouns and verbs) are domain specific and can be added to the lexicon during the writing process. The reconstruction process of this scenario in RNL is supported by a look-ahead text editor (Guy and Schwitter, 2017). The reconstructed scenario in RNL looks as follows:

(a) No student is a unit.

(b) No student is a program.

(c) Every student is enrolled in exactly one program.

(d) Every student studies at least one unit and at most four units.

(e) Every student has a student id and has a student name.

(f) No program is a unit and is a student.

(g) Every program is composed of a unit.

(h) Every program is enrolled by a student.

(i) Every program has a program id and has a program name.

(j) No unit is a student and is a program.

(k) Every unit is studied by a student.

(l) Every unit belongs to a program.

(m) Every unit has a unit code and has a unit name.

Additionally, we use the following terminological statements expressed in RNL:

(n) The verb studies is the inverse of the verb studied by.

[2]https://franz.com/agraph/racer/
[3]http://www.hermit-reasoner.com/

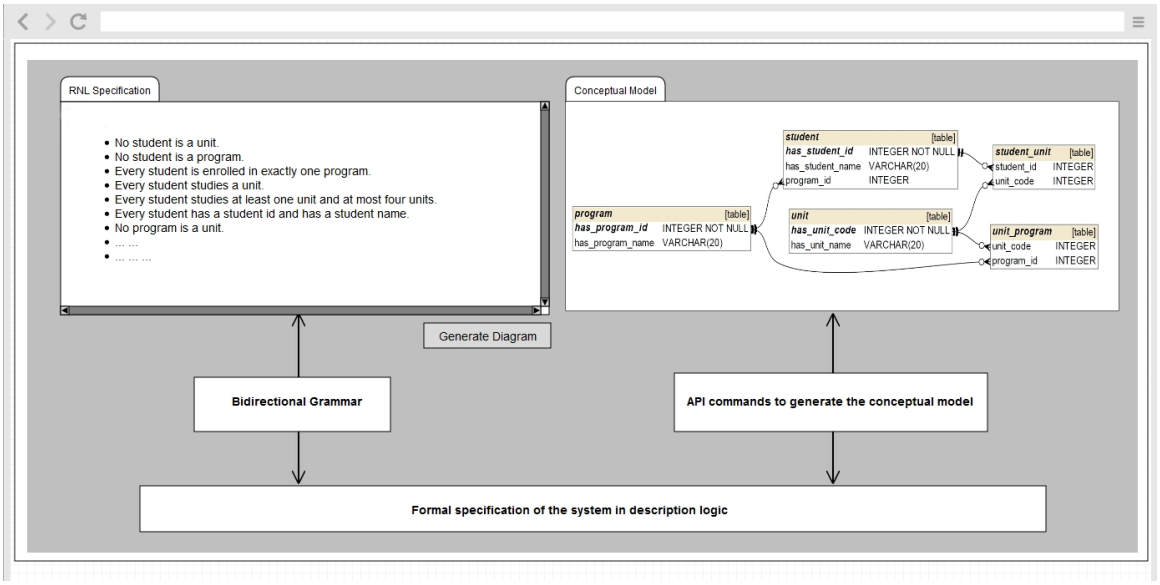

Figure 1: Proposed system architecture for conceptual modelling using restricted natural language.

(o) The verb composed of is the inverse of the verb belongs to.

3.2 Grammar

A feature-based phrase structure grammar has been built using the NLTK (Loper and Bird, 2002) toolkit to parse the above-mentioned specification. The resulting parse trees for these sentences are then translated by the language processor into their equivalent description logic statements. Below we show a scaffolding of the grammar rules with feature structures that we used for our case study:

```
S ->
  NP[NUM=?n, FNC=subj]
  VP[NUM=?n]
  FS

VP[NUM=?n] ->
  V[NUM=?n] NP[FNC=obj] |
  V[NUM=?n] Neg NP[NUM=?n, FNC=obj] |
  VP[NUM=?n] CC VP[NUM=?n]

NP[NUM=?n, FNC=subj] ->
  UQ[NUM=?n] N[NUM=?n] |
  NQ[NUM=?n] N[NUM=?n] |
  Det[NUM=?n] N[NUM=?n] |
  KP[NUM=?n] VB[NUM=?n] |
  KP[NUM=?n] VBN[NUM=?n]

NP[NUM=?n, FNC=obj] ->
  Det[NUM=?n] N[NUM=?n] |
  RB[NUM=?n] CD[NUM=?n] N[NUM=?n] |
  KP[NUM=?n] VB[NUM=?n] |
  KP[NUM=?n] VBN[NUM=?n]

V[NUM=?n] ->
  Copula[NUM=?n] |
  VB[NUM=?n] |
  Copula[NUM=?n] VBN[NUM=?n] |
  Copula[NUM=?n] JJ[NUM=?n]

VB[NUM=pl] -> "study" | ...
VB[NUM=sg] -> "studies" | ...
VBN -> "studied" "by" | ...
Copula[NUM=sg] -> "is"
Copula[NUM=pl] -> "are"

JJ -> "inverse" "of"
CC -> "and" | "or"

Det[NUM=sg] ->  "A" | "a" | ...
Det -> "The" | "the"

UQ[NUM=sg] -> "Every"
NQ -> "No"

Neg -> "not"

N[NUM=sg] -> "student" | ...
N[NUM=pl] -> "students" | ...

RB -> "exactly" | ...

CD[NUM=sg] -> "one"
CD[NUM=pl] -> "two" | ... | "four"

KP -> "The" "verb" | ...

FS -> "."
```

In order to translate the resulting syntax trees into the description logic representation, we have used the owl/xml syntax[4] of Web Ontology Language (OWL) as the formal target notation.

3.3 Case Study

The translation process starts by reconstructing the specification in RNL that follows the rules of the

[4]https://www.w3.org/TR/owl-xmlsyntax/

feature based grammar. While writing the specification in RNL, we tried to use the same vocabulary as in the natural language description.

The first two sentences of our specification use a negative quantifier in subject position and an indefinite determiner in object position:

(a) No student is a unit.

(b) No student is a program.

The translation of the sentence (a) into owl/xml notation results in the declaration of two atomic classes `student` and `unit` which are disjoint from each other.

```
<Declaration>
    <Class IRI="\#student"/>
</Declaration>
<Declaration>
    <Class IRI="\#unit"/>
</Declaration>
<DisjointClasses>
    <Class IRI="\#student"/>
    <Class IRI="\#unit"/>
</DisjointClasses>
```

Similarly, the translation of the sentence (b) results in the declaration of two disjoint atomic classes `student` and `program`. Both sentences (a) and (b) are related to expressing atomic negation in the DL *ALCQI* (see table 1).

```
<Declaration>
    <Class IRI="\#student"/>
</Declaration>
<Declaration>
    <Class IRI="\#program"/>
</Declaration>
<DisjointClasses>
    <Class IRI="\#student"/>
    <Class IRI="\#program"/>
</DisjointClasses>
```

The RNL that we have designed for this case study allows for verb phrase coordination; for example, the two above-mentioned sentences (a+b) can be combined in the following way:

(a+b) No student is a unit and is a program.

The translation of this sentence (a+b) results in the declaration of three atomic classes `student`, `unit` and `program` where `student` is disjoint from both `unit` and `program`.

```
<Declaration>
    <Class IRI="\#student"/>
</Declaration>
<Declaration>
    <Class IRI="\#unit"/>
</Declaration>
<Declaration>
    <Class IRI="\#program"/>
</Declaration>
<DisjointClasses>
    <Class IRI="\#student"/>
    <Class IRI="\#unit"/>
</DisjointClasses>
<DisjointClasses>
    <Class IRI="\#student"/>
    <Class IRI="\#program"/>
</DisjointClasses>
```

Now let us consider the following RNL sentences that use a universal quantifier in subject position and a quantifying expression in object position:

(c) Every student is enrolled in exactly one program.

(d) Every student studies at least one unit and at most four units.

The universally quantified sentence (c) which contains a cardinality quantifier in the object position is translated into an object property `enrolled_in` that has the class `student` as domain and the class `program` as range with an exact cardinality of `1`. This corresponds to a qualified cardinality restriction in the DL *ALCQI*.

```
<Declaration>
    <ObjectProperty IRI="#enrolled_in"/>
</Declaration>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#enrolled_in"/>
    <Class IRI="#student"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty IRI="#enrolled_in"/>
    <ObjectExactCardinality cardinality="1">
        <ObjectProperty IRI="#enrolled_in"/>
        <Class IRI="#program"/>
    </ObjectExactCardinality>
</ObjectPropertyRange>
```

The universally quantified sentence (d) which has a compound cardinality quantifier in object position is translated into the object property `study` that has the class `student` as domain and the class `unit` as range with a minimum cardinality of `1` and maximum cardinality of `4`. The translation of this sentence corresponds to a qualified cardinality restriction in the DL *ALCQI*.

```
<Declaration>
    <ObjectProperty IRI="#study"/>
</Declaration>
<ObjectPropertyDomain>
  <ObjectProperty IRI="#study"/>
  <Class IRI="#student"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
  <ObjectProperty IRI="#study"/>
   <ObjectMinCardinality cardinality="1">
     <ObjectProperty IRI="#study"/>
```

```
        <Class IRI="#unit"/>
      </ObjectMinCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
      <ObjectProperty IRI="#study"/>
      <ObjectMaxCardinality cardinality="4">
       <ObjectProperty IRI="#study"/>
       <Class IRI="#unit"/>
      </ObjectMaxCardinality>
    </ObjectPropertyRange>
```

The following RNL sentence has a universal quantifier in subject position and a coordinated verb phrase with indefinite noun phrases in object position:

(e) Every student has a student id and has a student name.

The translation of this sentence (e) results in two data properties for the class `student`. The first data property is `has_student_id` with a data type `integer` and the second data property is `has_student_name` with the data type `string`:

```
    <Declaration>
        <DataProperty IRI="\#has_student_id"/>
    </Declaration>
    <DataPropertyDomain>
        <DataProperty IRI="\#has_student_id"/>
        <Class IRI="#student"/>
    </DataPropertyDomain>
    <DataPropertyRange>
        <DataProperty IRI="\#has_student_id"/>
        <Datatype abbreviatedIRI="xsd:integer"/>
    </DataPropertyRange>
    <Declaration>
        <DataProperty IRI="\#has_student_name"/>
    </Declaration>
    <DataPropertyDomain>
        <DataProperty IRI="\#has_student_name"/>
        <Class IRI="#student"/>
    </DataPropertyDomain>
    <DataPropertyRange>
        <DataProperty IRI="\#has_student_name"/>
        <Datatype abbreviatedIRI="xsd:string"/>
    </DataPropertyRange>
```

(f) Every program is composed of a unit.

The universally quantified sentence (f) which contains an indefinite determiner in the object position is translated into the object property `composed_of` with the class `program` as domain and the class `unit` as range. This is corresponds to an unqualified existential restriction in the DL *ALCQI*.

```
    <Declaration>
        <ObjectProperty IRI="#composed_of"/>
    </Declaration>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="#composed_of"/>
        <Class IRI="#program"/>
    </ObjectPropertyDomain>
    <ObjectPropertyRange>
        <ObjectProperty IRI="#composed_of"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#composed_of"/>
            <Class IRI="#unit"/>
        </ObjectSomeValuesFrom>
    </ObjectPropertyRange>
```

The following two RNL sentences have a definite determiner both in subject position and object position and specify lexical knowledge for the language processor:

(g) The verb studies is the inverse of the verb studied by.

(h) The verb composed of is the inverse of the verb belongs to.

The translation of these two sentences results in the specification of inverse object properties. The translation of sentence (g) leads to the object properties `study` and `studied_by` which are inverse object properties. Similarly, the translation of sentence (h) states that the object properties `composed_of` and `belong_to` are also inverse object properties. These statements correspond to the inverse role construct in the DL *ALCQI*.

```
    <InverseObjectProperties>
        <ObjectProperty IRI="#study"/>
        <ObjectProperty IRI="#studied_by"/>
    </InverseObjectProperties>
    <InverseObjectProperties>
        <ObjectProperty IRI="#composed_of"/>
        <ObjectProperty IRI="#belong_to"/>
    </InverseObjectProperties>
```

The rest of the specification is similar to the examples that we have discussed above.

4 Reasoning

After generating the owl/xml notation for the RNL specification, we use Owlready (Lamy, 2017) that includes the description logic reasoner HermiT (Glimm et al., 2014) for consistency checking. Owlready is a Python library for ontology-oriented programming that allows to load OWL 2.0 ontologies and performs various reasoning tasks. For example, consistency checking of the specification can be performed on the class level. If a domain expert writes for example *"No student is a unit"* and later specifies that *"Every unit is a student"*, then the reasoner can detect this inconsistency and informs the domain expert about this conflict. The owl/xml notation below shows how

this inconsistency (*"owl:Nothing"*) is reported after running the reasoner.

```
<rdf:Description
    rdf:about="http://www.w3.org/2002/07/owl#Nothing">
  <owl:equivalentClass
    rdf:resource=
            "http://www.w3.org/2002/07/owl#Nothing"/>
  <owl:equivalentClass rdf:resource="#student"/>
  <owl:equivalentClass rdf:resource="#unit"/>
</rdf:Description>
```

This inconsistency can be highlighted directly in the RNL specification; that means the domain expert can fix the textual specification and does not have to worry about the underlying formal notation.

5 Conceptual Model Generation

In the next step, we extract necessary information such as a list of entities (classes), attributes (data properties) and relationships (object properties) from the owl/xml file to generate the conceptual model. This information is extracted by executing XPath (Berglund et al., 2003)[5] queries over the owl/xml notation and then it is used to build a database schema containing a number of tables representing the entities with associated attributes. Relationships among the entities are represented by using foreign keys in the tables. An SQL script is generated containing SQLite commands[6] for this database schema.

This SQL script is executed by using SQLite to generate the corresponding database for the specification. After that, we use SchemaCrawler[7] to generate the entity relationship diagram (see fig. 2) from the SQL script. SchemaCrawler is a free database schema discovery and comprehension tool that allows to generate diagrams from SQL code.

For mapping a description logic representation to an entity relationship diagram, we have used the approach described by Algorithm 1. All the classes in the OWL file become entities in the ER-diagram. Object properties are mapped into relations between the entities and data properties are mapped into attributes for these entities. The qualified cardinality restrictions of the object properties define relationship cardinalities in the diagram.

We understand conceptual modelling as a round tripping process. That means a domain expert can write the RNL specification first, then generate the conceptual model from the specification, and then a knowledge engineer might want to modify the conceptual model. These modifications will then be reflected on the level of the RNL by verbalising the formal notation. During this modification process the reasoner can be used to identify inconsistencies found in a given specification and to give appropriate feedback to the knowledge engineer on the graphical level or to the domain expert on the textual level.

Algorithm 1: Mapping description logic representation to SQLite commands for generating entity relationship diagrams.

Input: Logical notation in description logic
Output: SQLite script
$entity\_list$= extract_class(owl/xml file);
$data\_property\_list$=
 extract_data_property(owl/xml file);
$object\_property\_list$=
 extract_object_property(owl/xml file);
for *enity in* $entity\_list$ **do**
  create_table($entity$)
  for $data\_property$ *in* $data\_property\_list$ **do**
    add_data_property($entity$, $data\_property$)
    for $object\_property$ *in* $object\_property\_list$ **do**
      if $cardinality$ == *1* **then**
        add_data_property($entity$, $data\_property$)
      end
    end
  end
  for $object\_property$ *in* $object\_property\_list$ **do**
    create_relationship($entity$, $object\_property$)
  end
end

6 Discussion

The outcome of our experiment justifies the proposed approach for conceptual modelling. We have used a phase structure grammar to convert a RNL specification into description logic. This experiment shows that it is possible to generate formal representations from RNL specifications and

[5] https://www.w3schools.com/xml/xml_xpath.asp
[6] https://www.sqlite.org/index.html
[7] https://www.schemacrawler.com/

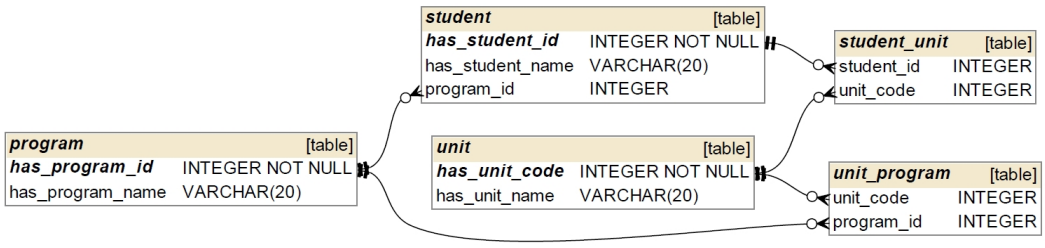

Figure 2: Entity relationship diagram generated from the formal representation using SchemaCrawler.

these formal representations can be mapped to different conceptual models. The proposed approach for conceptual modelling addresses two research challenges[8]: 1. providing the right set of modelling constructs at the right level of abstraction to enable successful communication among the stakeholders (i.e. domain experts, knowledge engineers, and application programmers); 2. preserving the ease of communication and enabling the generation of a database schema which is a part of the application software.

7 Future Work

We are planning to develop a fully-fledged restricted natural language for conceptual modelling of information systems. We want to use this language as a specification language that will help the domain experts to write the system requirements precisely. We are also planning to develop a conceptual modelling framework that will allow users to write specifications in RNL and will generate conceptual models from the specification. This tool will also facilitate the verbalization of the conceptual models and allow users to manipulate the models in a round tripping fashion (from specification to conceptual models and conceptual models to specifications). This approach has several advantages for the conceptual modelling process: Firstly, it will use a common formal representation to generate different conceptual models. Secondly, it will make the conceptual modelling process easy to understand by providing a framework to write specifications, generate visualizations, and verbalizations. Thirdly, it is machine-processable like other logical approaches and support verification; furthermore, verbalization will facilitate better understanding of the modelling process which is only available in limited forms in the current conceptual modelling frameworks.

[8] http://www.conceptualmodeling.org/Conceptual Modeling.html

8 Conclusion

In this paper we demonstrated that an RNL can serve as a high-level specification language for conceptual modelling, in particular for specifying entity-relationship models. We described an experiment that shows how we can support the proposed modelling approach. We translated a specification of a conceptual model written in RNL into an executable description logic program that is used to generate the entity-relationship model. Our RNL is supported by automatic consistency checking, and is therefore very suitable for formalizing and verifying conceptual models. The presented approach is not limited to a particular modeling framework and can be used apart from entity-relationship models also for object-oriented models and object role models. Our approach has the potential to bridge the gap between a seemingly informal specification and a formal representation in the domain of conceptual modelling.

References

Sikha Bagui. 2009. Mapping OWL to the entity relationship and extended entity relationship models. *International Journal of Knowledge and Web Intelligence* 1(1-2):125–149.

Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. 2005. Reasoning on uml class diagrams. *Artificial Intelligence* 168(1):70–118.

Anders Berglund, Scott Boag, Don Chamberlin, Mary F Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. 2003. XML Path Language (XPath). *World Wide Web Consortium (W3C)* https://www.w3.org/TR/xpath20/.

Peter Bernus, Kai Mertins, and Günter Schmidt. 2013. *Handbook on architectures of information systems*. Springer Science & Business Media.

Steven Bird, Ewan Klein, and Edward Loper. 2009. Publications received. *Computational Linguistics* 36:283–284.

Sara Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. 2004. Visual modeling of owl dl ontologies using uml. In *International Semantic Web Conference*. Springer, pages 198–213.

Diego Calvanese. 2013. *Description Logics for Conceptual Modeling Forms of reasoning on UML Class Diagrams*. EPCL Basic Training Camp 2012-2013.

Brian Davis, Ahmad Ali Iqbal, Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, and Siegfried Handschuh. 2008. Roundtrip ontology authoring. In *International Semantic Web Conference*. Springer, pages 50–65.

Ronald Denaux, Vania Dimitrova, Anthony G Cohn, Catherine Dolbear, and Glen Hart. 2009. Rabbit to owl: ontology authoring with a cnl-based tool. In *International Workshop on Controlled Natural Language*. Springer, pages 246–264.

Pablo R Fillottrani, Enrico Franconi, and Sergio Tessaris. 2012. The icom 3.0 intelligent conceptual modelling tool and methodology. *Semantic Web* 3(3):293–306.

Enrico Franconi, Alessandro Mosca, and Dmitry Solomakhin. 2012. Orm2: formalisation and encoding in owl2. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, pages 368–378.

Joseph Frantiska. 2018. Entity-relationship diagrams. In *Visualization Tools for Learning Environment Development*, Springer, pages 21–30.

Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, Brian Davis, and Siegfried Handschuh. 2007. Clone: Controlled language for ontology editing. In *The Semantic Web*, Springer, pages 142–155.

Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. 2014. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning* 53(3):245–269.

Stephen C Guy and Rolf Schwitter. 2017. The peng asp system: architecture, language and authoring tool. *Language Resources and Evaluation* 51(1):67–92.

Terry Halpin. 2009. Object-role modeling. In *Encyclopedia of Database Systems*, Springer, pages 1941–1946.

Tobias Kuhn. 2008. *Acewiki: A natural and expressive semantic wiki*. arXiv preprint arXiv:0807.4618.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics* 40(1):121–170.

Jean-Baptiste Lamy. 2017. Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine* 80:11–28.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*. Association for Computational Linguistics, pages 63–70.

Carsten Lutz. 2002. Reasoning about entity relationship diagrams with complex attribute dependencies. In *Proceedings of the International Workshop in Description Logics 2002 (DL2002), number 53 in CEUR-WS (http://ceur-ws.org)*. pages 185–194.

Gerard O´ Regan. 2017. Unified modelling language. In *Concise Guide to Software Engineering*, Springer, pages 225–238.

Antoni Olivé. 2007. *Conceptual Modeling of Information Systems*. Springer-Verlag, Berlin, Heidelberg.

Richard Power. 2012. Owl simplified english: a finite-state language for ontology editing. In *International Workshop on Controlled Natural Language*. Springer, pages 44–60.

Barker Richard. 1990. *CASE Method: Entity Relationship Modelling*. Addition-Wesley Publishing Company, ORACLE Corporation UK Limited.

Rolf Schwitter. 2010. Controlled natural languages for knowledge representation. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics, pages 1113–1121.