Semantic Round-Tripping in Conceptual Modelling using Restricted Natural Language

MACQUARIE University
SYDNEY · AUSTRALIA

Bayzid Ashik Hossain
Supervisor: Rolf Schwitter

Department of Computing
Macquarie University
Sydney, Australia

31st Australasian Database Conference, 2020

Overview

Introduction

The key activity to design an information system is Conceptual Modelling which brings out and describes the general knowledge that is required to build the system.

Traditionally, this activity involves domain experts and knowledge engineers to work together and build a conceptual model of the system.

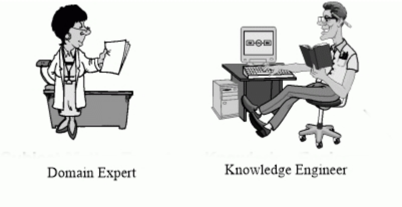

Figure: Conceptual Modelling

Popular Conceptual Modelling Techniques

There are a number of conceptual modeling techniques available for designing an information system.

- Entity Relationship Modelling (ERM) represents an information system in terms of entities, attributes and relationships among the entities.
- Object-oriented modelling such as Unified Modelling Language (UML) is a graphical modelling framework for software system design.
- Object Role Modelling (ORM) uses a fact-oriented approach for modeling the information system.

Motivation

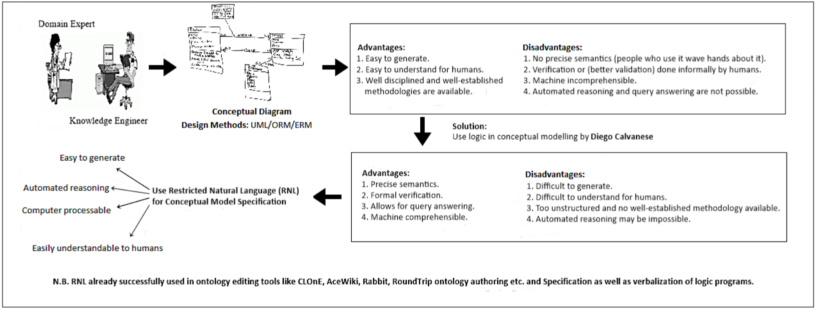

Figure: Problem space diagram

What is a Restricted Natural Language?

Restricted Natural Languages (RNL) are subsets of natural languages that are obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity. These restricted languages are also known as controlled natural languages (CNL).

There are two types of RNL:

- those that improve readability for human readers (e.g. non-native speakers) and
- those that enable reliable automatic semantic analysis of the language.

Related Works

The idea of using a RNL in conceptual modelling is not new. However, existing RNL based approaches do not use description logic for knowledge representation and they have not considered the idea of semantic round tripping.

- On the Systematic Analysis of Natural Language Requirements with CIRCE (Ambriola, et al 2006)
- A Natural Language-Based CASE Tool for Object-Oriented Analysis (Harmain, et al 2003)
- From Natural Language to Object Oriented Requirements Using the Natural Language Processing System LOLITA (Mich 1996)
- Software Development Process from Natural Language Specification (Saeki, et al 1989)

Related Works (cont.)

There has been a number of works on formalizing conceptual models for verification purposes:

- Description Logics for Conceptual Modeling (Calvanese, 2013)
- ORM2: Formalization and Encoding in OWL2 (Franconi, 2012)
- Reasoning on UML class diagrams (Berardi, et al 2005)
- Reasoning about Entity Relationship Diagrams with Complex Attribute Dependencies (Lutz, 2002)

These approaches first represent the domain of interest as a conceptual model and then formalize the conceptual model using a formal language.

The DL $\mathcal{ALCQI}$

The description logic (DL) $\mathcal{ALCQI}$ is well suited to do reasoning with Entity Relationship diagrams, UML class diagrams and ORM diagrams.

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: The DL $\mathcal{ALCQI}$.

Related Works (cont.)

There exists work on ontology editing and authoring using restricted natural language.

- Specifying and Verbalising Answer Set Programs in Controlled Natural Language (Rolf Schwitter, 2018)
- OWL Simplified English (Richard Power, 2012)
- Rabbit (Ronald Denaux, et al. 2009)
- ACE View (Kaarel Kaljurand, 2008)
- Round trip ontology authoring (Brian Davis, et al. 2008)
- Clone: Controlled language for ontology editing (Adam Funk, et al. 2007)
- Lite natural language (Raffaella Bernardi, et al. 2007)

Related Works (cont.)

There exists works on mapping ontologies to relational databases (SQL schema) and the other way around.

- Storing OWL ontologies in SQL relational databases (Astrova, et al. 2007)
- Rules for mapping SQL relational databases to OWL ontologies (Astrova, et al. 2009)
- Mapping owl to the entity relationship and extended entity relationship models (Sikha Bagui, 2009)

Proposed System Architecture

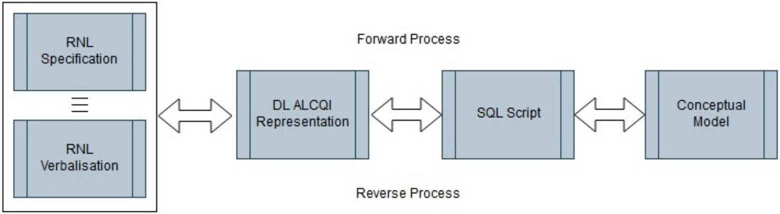

Figure: Proposed system architecture for conceptual modelling using RNL.

Proposed Approach

Unlike previous approaches, we propose the following:

Forward Process:

- Write the specification of a conceptual model in RNL first.
- Translate RNL specification into DL $\mathcal{ALCQI}$ representation.
  - (1) Translate the specification into an intermediate description logic representation (IDL) using a bi-directional grammar.
  - (2) We take this IDL and generate corresponding OWL/XML syntax.
- Existing description logic reasoners (Racer, HermiT) can be used to check the consistency of this OWL/XML syntax.
- Map this OWL/XML syntax into an SQL Script.
- After that generate desired conceptual models (i.e., ER diagram) from the SQL Script.

Proposed Approach (cont.)

Reverse Process:

- Build the SQL Script from the conceptual model.
- Generate OWL/XML syntax from the SQL Script.
- Existing description logic reasoners (Racer, HermiT) can be used to check the consistency of the formal notation.
- Then translate this OWL/XML syntax into intermediate description logic representation (IDL).
- Use the bi-directional grammar to translate the IDL into RNL verbalisation.

Example Scenario

A Learning Management System (LMS) keeps track of the units 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.

Reconstructing the Scenario in RNL

Entity Types

1. Program is an entity type.
2. Student is an entity type.
3. Unit is an entity type.

Data Types

4. Program id is of integer data type.
5. Program name is of string data type.
6. Student id is of integer data type.
7. Student name is of string data type.
8. Unit code is of integer data type.
9. Unit name is of string data type.

Reconstructing the Scenario in RNL (cont.)

Fact Types

10. Student is enrolled in program.
11. Program is enrolled by student.
12. Program is composed of unit.
13. Unit belongs to program.
14. Student studies unit.
15. Unit is studied by student.

Reconstructing the Scenario in RNL (cont.)

Constraints

16. Every student is enrolled in exactly 1 program.
17. Every program is enrolled by 1 or more students.
18. Every program is composed of 1 or more units.
19. Every unit belongs to 1 or more programs.
20. Every student studies at least 1 and at most 4 units.
21. Every unit is studied by 1 or more students.
22. Every program has exactly 1 program id and has exactly 1 program name.
23. Every student who has exactly 1 student id has exactly 1 student name.
24. Every unit has exactly 1 unit code and has exactly 1 unit name.

RNL Specification to DL ALCQI Representation

1. Translating RNL into IDL:
We use a bi-directional definite clause grammar (DCG) to process the RNL specification, to build the lexicon from the type system (entity, data and fact types) and then generate corresponding IDL representation from the constraints.

The advantage of using a DCG is that it implements a logic program that allows us to build a bidirectional grammar where only the pre-terminal rules need to be duplicated.

Input:
20. Every student studies at least 1 and at most 4 units.
Output:
forall(A, student(A)=> exists(B, unit(B)& min(1):study(A, B):max(4)))

A sample DCG grammar rule

```
%---------------------------------------------------------
% A sample grammar rule that process entity declarations
%---------------------------------------------------------

s([mode:M, type:entity, sem:Sem]) -->
  np([mode:M, num:sg, type:entity, sem:Sem]),
  [is, an, entity, type, '.'].

np([mode:M, num:N, type:entity, sem:Sem]) -->
   noun([mode:M, num:N, type:entity, sem:Sem]).

noun([mode:proc, num:N, type:entity, sem:[T1|T2]-[[T|T1]|T2]]) -->
   lexical_rule([cat:noun, num:N, type:entity, sem:T]).

noun([mode:gen, num:N, type:entity, sem:[[T|T1]|T2]-[T1|T2]]) -->
   { lexicon([cat:noun, wform:WForm, num:N, type:entity, arg:_X, sem:T]) },
   WForm.
```

RNL Specification to DL ALCQI Representation (cont.)

2. Translating IDL into OWL/XML syntax:

Input:

forall(A, student(A)=>exists(B, unit(B)& min(1):study(A, B):max(4)))

Output:

```
<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>
```

Reasoning

After generating the OWL/XML notation for the RNL specification, we use Owlready that includes the description logic reasoner *HermiT* for consistency checking. For example consider the inconsistent sentences below:

(a) No student is a tutor.

(b) Every PhD student is a student.

(c) Every PhD student is a tutor.

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="#PhD_student"/>
   <owl:equivalentClass  rdf:resource="#tutor"/>
</rdf:Description>
```

OWL/XML syntax to SQL Script

- In the next step, we extract necessary information such as a list of classes, data properties and object properties from the OWL/XML notation by executing XPath queries.
- We use this extracted information to build an SQL Script that is executed to create an SQL schema to generate an ER-diagram.
- All the classes in the OWL/XML 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 cardinality constraints of the object properties define relationship cardinalities in the diagram.

Sample SQL Script

```
Create Table et_student(
dp_student_id integer Not Null Primary Key,
dp_student_name varchar(20) Not Null,
enrolled_in_enrolled_by_program_id integer,
Foreign Key (enrolled_in_enrolled_by_program_id)
    references et_program(dp_program_id));

Create Table op_student_unit(
1_4_study_student_id integer,
-- Here m stands for undefined many (1 or more) relationship
1_m_studied_by_unit_code integer,
Foreign Key (1_4_study_student_id) references et_student(dp_student_id),
Foreign Key (1_m_studied_by_unit_code) references et_unit(dp_unit_code));
```

Figure: The excerpt of the SQL script that shows the example code that follows particular a naming convention for verbalisation.

Conceptual Model Generation

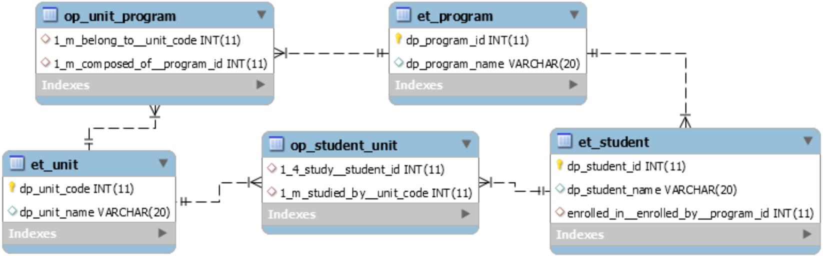

Figure: Entity relationship diagram generated from MySQL Workbench via reverse engineering.

Conceptual Model to SQL Script

We generate an SQL script from the conceptual model in the MySQL workbench via forward engineering and start the reverse process in conceptual modelling to generate the verbalisation.

Database Schema to DL $\mathcal{ALCQI}$ Representation

The SQL script generated from the conceptual model serves as a starting point for the verbalisation process. The verbalisation process is done in two steps:

- we create the database schema from the SQL script and then translate it into OWL/XML syntax; and
- we generate the intermediate DL representation from the translated OWL/XML syntax.

To generate the OWL/XML syntax from the database schema, we look up the following information by querying the database: 1. list of classes, 2. list of data properties for each class, and 3. list of object properties with domain, range and cardinality information.

DL ALCQI Representation to RNL verbalisation

Next, we feed the intermediate DL representation to the bi-directional DCG which translates this intermediate representation into an RNL specification.

This is the last stage of the reverse process because the DCG verbalises the conceptual model and completes the round-tripping process.

Evaluation

To evaluate the round-tripping process, we execute two types of queries that collect the defined content words and used constraints from the DL representation of the specification and verbalisation:

(1) What are the entity type names/data property names/fact type names? and
(2) What are the domain, range and cardinality constraints for the fact type names?

					Specification S1	Verbalisation S2
Entity names	Student, Unit, Program.				√	√
Data property names	Student id, Student name, Unit code, Unit name, Program id, Program name.				√	√
Fact type names with domain, range and cardinality constrains	**Name**	**Domain**	**Range**	**Cardinality Constrains**		
	enrolled in	student	program	exactly 1	√	√
	enrolled by	program	student	1 or more	√	√
	study	student	unit	at least 1 and at most 4	√	√
	studied by	unit	student	1 or more	√	√
	belongs to	unit	program	1 or more	√	√
	composed of	program	unit	1 or more	√	√

Figure: Comparison of specification and verbalisation

Future Work

- Currently working on the grammar to develop a fully-fledged RNL for conceptual modelling that covers *ERM, ORM^{zero}, and UML Class Diagrams.*
- Developing a conceptual modelling framework that will allow users to write specifications in RNL and will generate conceptual models from the specification.
  - To facilitate the verbalization of the conceptual models.
  - To allow users to manipulate the models in a round tripping fashion (from specifications to conceptual models and conceptual models to specifications).

Conclusion

We proposed a novel approach to conceptual modelling where the domain experts will be able to specify, construct and verbalise a model using a restricted form of natural language. This approach has several advantages:

- It uses a common formal representation to generate different conceptual models.
- It will make the conceptual modelling process easy to understand by providing a framework to write specifications, generate visualizations, and verbalizations.
- It is machine-processable like other logical approaches and supports verification.

Acknowledgement

I would like express my gratitude to CORE for the Student Travel Award.

Thank You

Questions???