CNL^{ER}: A Controlled Natural Language for Specifying and Verbalising Entity Relationship Models

Bayzid Ashik Hossain, Gayathri Rajan and Rolf Schwitter
Department of Computing
Macquarie University, Sydney, Australia
{bayzid-ashik.hossain|rolf.schwitter}@mq.edu.au
gayathri.rajan@students.mq.edu.au

Abstract

The first step towards designing an information system is conceptual modelling where domain experts and knowledge engineers identify the necessary information together to build an information system. Entity relationship modelling is one of the most popular conceptual modelling techniques that represents an information system in terms of entities, attributes and relationships. Entity relationship models are constructed graphically but are often difficult to understand by domain experts. To overcome this problem, we suggest to verbalise these models in a controlled natural language. In this paper, we present CNL^{ER}, a controlled natural language for specifying and verbalising entity relationship (ER) models that not only solves the verbalisation problem for these models but also provides the benefits of automatic verification and validation, and semantic round-tripping which makes the communication process transparent between the domain experts and the knowledge engineers.

1 Introduction

An information system is a piece of software that has integrated components for organizing and analyzing data to aid decision making in an organization (Laudon and Laudon, 2015). One of the major roles of an information system is to accumulate data, turn it into information and later transform that information into organizational knowledge (Bourgeois, 2014). To be successful an information system always depends on a good design and conceptual modelling is the first step in the design process (Olivé, 2007). Information systems are best specified on the conceptual level using a language with names for individuals, concepts, and relations that occur in the application domain. Such a language is easy to understand by the domain experts and enhances correctness, compatibility, productivity and clarity in information system design (Halpin, 1998). Conceptual modelling involves different parties (e.g., domain experts and knowledge engineers) who brainstorm together to identify the necessary information for building the system (Hossain and Schwitter, 2018). After identifying the required information, knowledge engineers build a conceptual model of the information system by using conceptual modelling techniques such as entity relationship modelling (ERM) (Richard, 1990; Frantiska, 2018), object oriented modelling (UML) (O'Regan, 2017), or object role modelling (ORM) (Halpin, 2009).

One of the problems with these models is that they are constructed graphically and as a result they are often hard to understand for domain experts (Jarrar et al., 2006). Another problem with these conceptual modelling techniques is that they have no formal semantics; therefore, they are not machine comprehensible, do not support automatic verification and validation nor automatic reasoning (Calvanese, 2013).

To overcome these problems previous works used logic in parallel with traditional conceptual modelling techniques (Lutz, 2002; Berardi et al., 2005; Franconi et al., 2012). There are tools (Fillottrani et al., 2012; Lembo et al., 2016b,a) that allow knowledge engineers to draw the conceptual model and then translate the model constructs into a logical representation. This logical representation is then used to verify and validate the model. Using logic with traditional conceptual modelling techniques also introduces some problems like the difficulty to generate logical representations. Furthermore, it is not easy to understand these logical representations for domain experts since no well established methodology is available to make this process transparent (Calvanese, 2013).

Recent research on conceptual modelling showed that using a controlled natural language

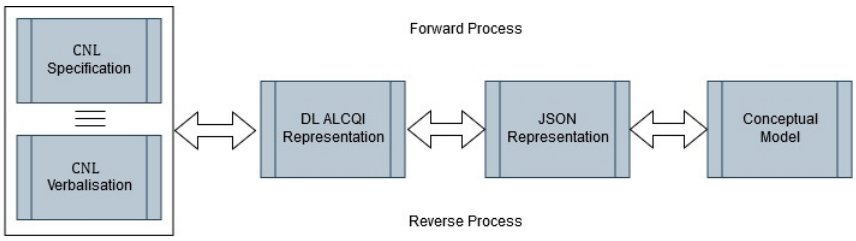

Figure 1: A CNL based conceptual modelling framework

(CNL) for specification and verbalisation can overcome the problems introduced by logic in the conceptual modelling process (Hossain and Schwitter, 2018). A CNL can be defined as a subset of natural language that is obtained by constraining the grammar and vocabulary in order to eliminate the ambiguity as well as the complexity of the language. A CNL can be designed in such a way that it has well defined computational properties and thus can be translated unambiguously into a formal representation (Schwitter, 2010). Using a CNL in conceptual modelling helps the domain experts to understand the conceptual models through specification and verbalisation, allows the machine to understand the models as the CNL can be translated into a formal representation, and therefore supports automated reasoning and question answering.

2 Motivation

The idea of using natural language for conceptual modelling is not new but previous approaches (Saeki et al., 1989; Mich, 1996; Harmain and Gaizauskas, 2003; Ambriola and Gervasi, 2006; Ibrahim and Ahmad, 2010) did not constrain the natural language enough and did not use logic to formally represent the conceptual models. Furthermore, the idea of semantic round-tripping from a specification to a conceptual model and from a conceptual model to a specification (verbalisation) is novel in this context. A recent survey (Störrle, 2017) on conceptual models showed that there are three modes of conceptual modelling: 1. informal modelling for cognition and communication; 2. semi-formal modelling for planning and documentation; and 3. formal modelling for generation and contracts. In the software industry 70-79% of the modelling is done informally (Störrle, 2017).

We want to use CNL in the conceptual modelling process to overcome the problems that occur in traditional conceptual modelling approaches. We want to bridge the gap between an informal and formal conceptual model. We also want to offer verbalisation for ERM. ERM is frequently used in the industry and has no verbalisation support. Existing tools that support creating ERM models do not provide the facility of writing specifications for conceptual models and therefore semantic round-tripping is not possible. We have developed a CNL-based conceptual modelling framework [Fig. 3] that supports the following points:

1. Writing textual specifications for the conceptual modelling process.
2. Description logic based common formal representation (DL *ALCQI*) for different conceptual models.
3. Generating a conceptual model from a written specification and the other way around.
4. Verification and validation of the written specification.

In this paper we present CNL^{ER}, a controlled natural language that is specially designed to specify and verbalize ERM constructs.

3 ERM Constructs

An ERM represents an information system in terms of entities, attributes and relationships (Song and Chen, 2009). ERM is mainly used to design relational databases and to do the planning and requirement analysis of an information system. The outcome of an ERM process is a graphical model often known as ER diagram (ERD). The basic components of an ERD are entities, attributes, and relationships [Fig. 2].

An *entity* is a real world object having independent existence (e.g., person, place, organisation) (Song and Chen, 2009). An entity is also known as a class or a concept. There are two types

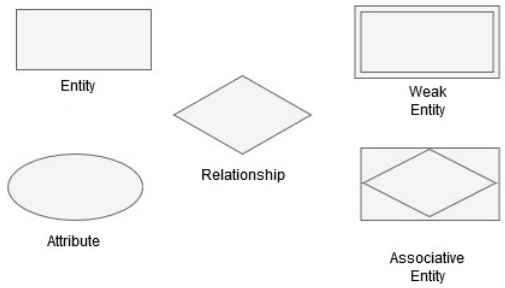

Figure 2: ERM constructs

of entities: (1) strong entities that have key attributes to uniquely identify each instance of an entity (e.g., student has student id as a key attribute); and (2) weak entities that do not have any key attributes and they depend on other strong entities to get identified (e.g., a room can not exist without a building, so “room" is a weak entity whereas “building" is a strong entity).

An *attribute* indicates a property or characteristic of an entity (Li and Chen, 2009). For example, if a student is an entity then “student name", “student id", and “phone number" would be the attributes for that student. Attributes help us to differentiate between entities. An attribute can be single valued (e.g., “student name") or multivalued (e.g., “skills"). A single valued attribute or a collection of single valued attributes that identify an instance of an entity uniquely is known as A key attribute or a primary key.

A *relationship* depicts the association between or among the entities. For example, if “student" and “program" are entities then in the fact “student is enrolled in program", the expression “is enrolled in" is the relationship between “student" and “program". Every relationship has a cardinality which defines the number of occurrences (minimum and maximum) of one entity that is related to a single occurrence of the other entity (Song and Chen, 2009). Based on the form of cardinality, we can divide a relationship in ERM into three types: one-to-one, one-to-many and many-to-many. Sometimes a many-to-many relationship acts as an entity itself which is known as an associative entity. An associative entity can have attributes that represent the properties of the corresponding relationship (Li and Chen, 2009). For example, if “student" and “course" are entities then the facts “every student studies 1 or more courses" and “every course is studied by 1 or more students" indicate that “student" and “course” have a many-to-many relationship. So the relationship between these two entities can act as an associative entity. We can consider “study details" as an associative entity that can have “study start date" and “study end date" as attributes.

Entity Declaration
1. Student is an entity type.
2. Department is an entity type.
3. Course is an entity type.
4. Teacher is an entity type.
5. Enrolment is an entity type.
6. Section is an entity type.
Attribute Declaration
7. Student id is of integer data type.
8. Student name is of string data type.
9. Department number is of integer data type.
10. Department name is of string data type.
11. Teacher id is of integer data type.
12. Teacher name is of string data type.
13. Course id is of integer data type.
14. Course name is of string data type.
15. Enrolment semester is of integer data type.
16. Enrolment grade is of integer data type.
17. Section id is of integer data type.
18. Section name is of string data type.
Relationship Declaration
19. Student belongs to department.
20. Department contains student.
21. Teacher works in department.
22. Department employs teacher.
23. Course is offered by department.
24. Department offers course.
25. Course is offered in sections.
26. Teacher teaches students in course.
27. Enrolment associates "Teacher teaches students in course".
Constraint Declaration
28. Every student belongs to exactly 1 department.
29. Every department contains 1 or more students.
30. Every teacher works in exactly 1 department.
31. Every department employs 1 or more teachers.
32. Every course is offered by exactly 1 department.
33. Every department offers 1 or more courses.
34. Every enrolment includes exactly 1 teacher.
35. Every enrolment includes exactly 1 student.
36. Every enrolment includes exactly 1 course.
37. Every student owns exactly 1 student id and owns exactly 1 student name.
38. Every teacher owns exactly 1 teacher id and owns exactly 1 teacher name.
39. Every course owns exactly 1 course id and owns exactly 1 course name.
40. Every department owns exactly 1 department number and owns exactly 1 department name.
41. Every enrolment consists of exactly 1 enrolment semester and consists of exactly 1 enrolment grade.
42. Every section is dependent of exactly 1 course.

Table 1: Extended example scenario from the ER paper (Frantiska, 2018) expressed using CNL^{ER}

Listing 1: Grammar rules for entity declaration

```
% Input:  Student is an entity.
% Output: entity(A, student)

s([mode:M, type:entity, sem:L1-L2]) -->
  np([mode:M, num:sg, type:entity, pos:subj, sem:L1-L2]),
  [is, an, entity, type], ['.'].

np([mode:M, num:N, type:T, pos:P, sem:L1-L2]) -->
  noun([mode:M, num:N, type:T, pos:P, sem:L1-L2]).

noun([mode:proc, num:N, type:entity, pos:P, sem:[L1|L2]-[[L0|L1]|L2]]) -->
  lexical_rule([cat:noun, num:N, type:entity, pos:P, sem:L0]).

noun([mode:gen, num:N, type:entity, pos:P, sem:[[L0|L1]|L2]-[L1|L2]]) -->
  {lexicon([cat:noun, wform:WForm, num:N, type:entity, pos:P, arg:_X, sem:L0])},
  WForm.
```

4 CNL^{ER} for ERM Constructs

In this section we discuss how to express ERM constructs using CNL^{ER}. For this purpose, we have taken an example scenario from the paper (Frantiska, 2018) and extended it by adding a weak entity and modifying the associative entity. CNL^{ER} has a distinct sentence format to declare an entity in ERM. For example, to define the fact that a student is an entity, CNL^{ER} has the following sentence pattern with a noun (*student*) in subject position, followed by a copula (*is*) and the key phrase *an entity type* in object position (e.g., see sentences 1-6 in table [1]).

CNL^{ER} also has a particular sentence pattern to declare attributes in ERM. For example, to specify an attribute of type integer, CNL^{ER} uses a sentence that contains a previously declared entity name (e.g., *student*) followed by an attribute name (e.g., *id*); this forms a data property name (*student id*) which is followed by a copula (*is*), and the key phrase (e.g., *of integer data type*). For example, sentences 7-18 in Table [1] show the attribute declarations for the example scenario.

To declare a relationship, we use a declared entity type name (e.g., *student*) in subject position and a declared entity type name (e.g., *department*) in object position with a relationship name (e.g., *belongs to*) in between. (e.g., see sentences 19-27 in table [1] for relationship declarations).

To define cardinality constraints over the relationships, CNL^{ER} uses a quantifying expression followed by entities and attributes: a quantifier (*every*) in subject position and either a quantifier (*0 or more, 1 or more*) or a cardinality constraint (*at least, at most, exactly*) in object position. For example, to define a one-to-one relationship cardinality between the entities “student" and “department", sentence (28) in table [1] is used, and to define a one-to-many relationship cardinality, sentence (29) in table [1] is used.

In order to define a many-to-many relationship in CNL^{ER}, we have to write two sentences that express the relationship between the entities in both direction. For example, to express a may-to-many relationship between “student" and “course", we have to write the following two sentences.

- *Every* student studies *1 or more* courses.
- *Every* course is studied by *1 or more* students.

To declare an associative entity in CNL^{ER} (e.g., “enrolment"), we have to declare the entity first. After that the entity needs to be linked with a many-to-many relationship using a predefined word “associates". The sentence 27 in table [1] declares an associative entity that links the ternary relationship among a teacher, a student and a unit. The relationship “includes" in the sentences 34, 35 and 36 of table [1] is predefined and can only be used together with an associative entity. An associative entity can have attributes like other entities. For example, sentence 41 in table [1] specifies attributes for the associative entity “enrolment".

To declare a weak entity in CNL^{ER}, we have to declare both strong and weak entities first as entities (e.g., “course" and “section" in table [1]). After that we need to specify that the weak entity is dependent of the strong entity. For example, to

declare that a section is a weak entity, sentence 42 specifies that “section" is a dependent of the “course" entity by using a predefined relationship “dependent of" in table [1].

5 Grammar

We use a definite clause grammar (DCG) (Pereira and Shieber, 2002) to process and translate a CNL^{ER} specification. The key advantage of using a DCG is that it implements a logic program that allows us to build a bi-directional grammar. The grammar translates the CNL^{ER} sentences into a corresponding internal description logic (DL) representation. This internal DL representation is further processed to generate an ERD.

A specification in CNL^{ER} consists of function words and content words. Function words (quantifiers, cardinality constraints and operators) describe the structure of the sentences and the number of these function words is fixed. In contrast, content words (nouns and verbs) are domain specific and are added to the lexicon during the writing process when they are declared. It is important to note that the bi-directional DCG contains grammar rules that translate every CNL^{ER} sentence into the internal DL representation and vice versa; this enable semantic round-tripping in our conceptual modelling framework. In this paper, we discuss the DCG rules that process some of the core ERM constructs. Below we discuss the grammar rules that process entity, attribute, relationship, constraint and associative entity declarations.

5.1 Entity Declaration

Listing [1] shows the grammar rules for an entity declaration. The first grammar rule states that a declarative sentence (*s*) consists of a noun phrase (*np*) and a key phrase (*“is an entity type"*), followed by a full stop (.). The grammar rule contains additional arguments that implement feature structures in the form of *attribute:value* pairs whereas the value can be a simple term or a complex term (for example in the form of a difference list: *[Head|Tail]-Tail*). The feature structure *mode:M* specifies whether the rule is used in the processing or generating mode and the feature structure *type:entity* specifies the type of the ERM construct. The feature structure *sem:L1-L2* is used to build up the semantic representation for an entity. The grammar rule for the noun phrase (*np*) contains a feature structure (*num:N*) that deals with number agreement (singular or plural), a feature structure (*arg:X*) that defines the argument of an entity, and one (*pos:subj*) that specifies the position of the noun. Finally, the feature structure *cat:noun* specifies a noun and *wform:WForm* specifies a word form consisting of potentially multiple elements. To extract an entity from the input list, a lexical rule *lexical_rule/1* is used that generates lexical entries with their corresponding singular and plural forms (e.g., student and students) for that entity with the help of a morphological component.

5.2 Attribute Declaration

To process a sentence that is used to declare an attribute, we use similar grammar rules as shown in listing [1] but with a different key phrase (e.g., *of integer data type*). To process an attribute and its data type, a lexical rule is used that extracts the attribute from the input list by excluding the copula and the key phrase and by identifying the type (e.g., integer, string, date) from the key phrase. After that the lexical rule is used to insert the attribute and its type information into the lexicon.

5.3 Relationship Declaration

Listing [2] shows the grammar rules for a relationship declaration. The first grammar rule states again that a declarative sentence (*s*) consists of a noun phrase (*np*) and a verb phrase (*vp*), followed by a full stop (.). The feature structures for the mode declaration and the semantic representation are similar to the entity declaration whereas the feature structure *type:fact* specifies the ERM construct relationship.

The second grammar rule states that a verb phrase (*vp*) consists of a verb (*verb*) and a noun phrase (*np*). In the case of processing, the grammar rule for the verb consists of a lexical rule (*lexical_rule/1*) that extracts a relationship from the input list by identifying the noun phrases in the subject and object position. After that, the lexical rule adds the extracted relationship to the lexicon and adds the semantic representation (*L0*) for that relationship to the outgoing part (*[[L0|L1]|L2]*) of the difference list. In the case of generating, the grammar rule for a verb removes the semantic representation (*L0*) from the incoming part (*[[L0|L1]|L2]*) of the difference list and tries to find this representation in the lexicon in order to return the corresponding lexical entry.

Listing 2: Grammar rules for relationship declaration

```
% Input:  Student is enrolled in program.
% Output: [entity(A, student), relation(A, B, is_enrolled_in), entity(B, program)]

s([mode:M, type:fact, sem:L1-L3]) -->
  np([mode:M, num:N, type:fact, pos:subj, arg:X, sem:L1-L2]),
  vp([mode:M, num:N, type:fact, arg:X, sem:L2-L3]),
  ['.'].

vp([mode:M, num:N, type:fact, arg:X, sem:L1-L3])-->
  verb([mode:M, num:N, type:fact, arg:X, arg:Y, sem:L1-L2]),
  np([mode:M, num:_N, type:fact, pos:obj, arg:Y, sem:L2-L3]).

np([mode:M, num:N, type:T, pos:P, arg:X, sem:L1-L2) -->
  noun([mode:M, num:N, type:T, pos:P, arg:X, sem:L1-L2]).

noun([mode:proc, num:N, type:fact, pos:P, arg:X, sem:[L1|L2]-[[L0|L1]|L2]]) -->
  {lexicon([cat:noun, wform:WForm, num:N, type:entity, pos:P, arg:X, sem:L0])},
  WForm.

noun([mode:gen, num:N, type:fact, pos:P, arg:X, sem:[[L0|L1]|L2]-[L1|L2]]) -->
  {lexicon([cat:noun, wform:WForm, num:N, type:entity, pos:P, arg:X, sem:L0])},
  WForm.

verb([mode:proc, num:N, type:fact, arg:X, arg:Y, sem:[L1|L2]-[[L0|L1]|L2]]) -->
  lexical_rule([cat:verb, num:N, arg:X, arg:Y, sem:L0]).

verb([mode:gen, num:N, type:fact, arg:X, arg:Y, sem:[[L0|L1]|L2]-[L1|L2]])-->
  {lexicon([cat:verb, wform:WForm, num:N, type:brel, arg:X, arg:Y, sem:L0])},
  WForm.
```

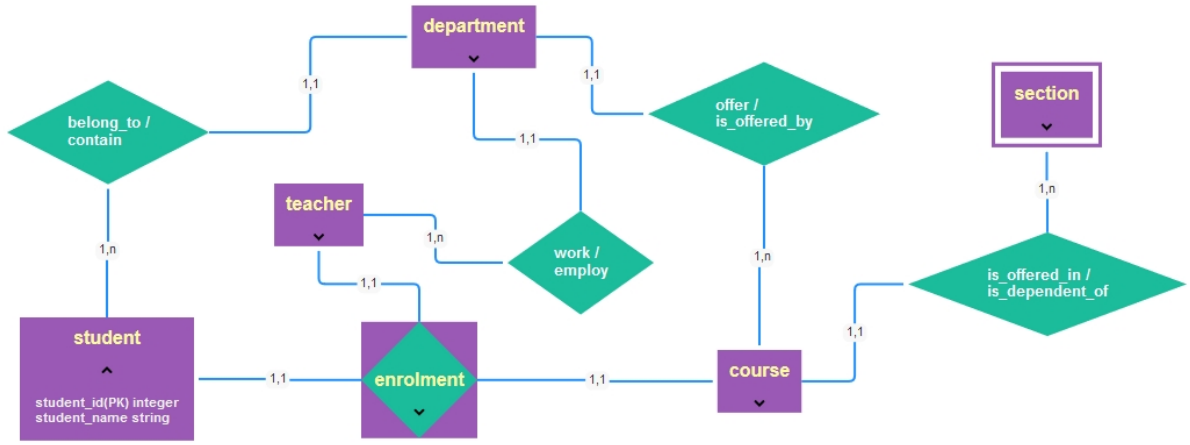

Figure 3: ERD of the example scenario (Frantiska, 2018) generated by the proposed CNL based conceptual modelling framework.

5.4 Constraint Declaration

To process a sentence that declares a cardinality constraint over a relationship, we use a grammar rule that is quite similar to the grammar rule in listing [2] with an additional quantifier (*qnt*) in subject position and a constraint (*cst*) in the object position. Note that the grammar rules for a quantifier and a cardinality constraint play an important role because they provide the relevant structure for the internal representation. For example, the grammar rule for the universal quantifier (*every*) results in a pattern of the form *sem:forall(X, Res ==> Sco)* that takes a restrictor (*Res*) that contains the information derived from the noun phrase in sub-

Listing 3: Grammar rules for associative entity declaration

```
% Fact Type "Association"
% Input:   Enrolment associates "student is enrolled in program".
% Output:  [entity(A, enrollment), associates(A, B), entity(C, student),
%           B#relation(C, D, is_enrolled_in), entity(D, program)]

s([mode:M, type:fact_ob, sem:L1-L4]) -->
  np([mode:M, num:N, type:fact, pos:subj, arg:X, sem:L1-L2]),
  verb([mode:M, wform:[associates], num:N, type:fact_ob, arg:X, arg:R, sem:L2-L3]),
  ['"'],
  s([mode:M, type:ob_fact, rel:R, sem:L3-L4]),
  ['"', '.'].

np([mode:M, num:N, type:T, pos:P, arg:X, sem:L1-L2) -->
  noun([mode:M, num:N, type:T, pos:P, arg:X, sem:L1-L2]).

noun([mode:proc, num:N, type:fact, pos:P, arg:X, sem:[L1|L2]-[[L0|L1]|L2]]) -->
  {lexicon([cat:noun, wform:WForm, num:N, type:entity, pos:P, arg:X, sem:L0])},
  WForm.

noun([mode:gen, num:N, type:fact, pos:P, arg:X, sem:[[L0|L1]|L2]-[L1|L2]]) -->
  {lexicon([cat:noun, wform:WForm, num:N, type:entity, pos:P, arg:X, sem:L0])},
  WForm.

verb([mode:proc, wform:WForm, num:N, type:fact_ob, arg:X, arg:Y,
      sem:[L1|L2]-[[L0|L1]|L2]]) -->
  {lexicon([cat:verb, wform:WForm, num:N, type:ob_rel, arg:X, arg:Y, sem:L0])},
  WForm.

verb([mode:gen, wform:WForm, num:N, type:fact_ob, arg:X, arg:Y,
      sem:[[L0|L1]|L2]-[L1|L2]]) -->
  {lexicon([cat:verb, wform:WForm, num:N, type:ob_rel, arg:X, arg:Y, sem:L0])},
  WForm.
```

ject position and the scope (*Sco*) that contains the information derived from the verb phrase and returns a pattern for an implication. In the case of a cardinality constraint that pattern might have the following form: *exists(X, Res & min(L) : Sco : max(U))*. Note also that the restrictor and scope are built up in these grammar rules with the help of specific feature structures.

5.5 Associative Entity Declaration

Listing [3] shows an excerpt of the grammar rules that are used to process and generate sentences for an associative entity. The first rule reuses the rule for a relationship and uses a key verb word (*associates*) to process an associative entity declaration. The first rule states that a sentence consists of a noun phrase (*np*) in subject position, a verb (*verb*) and a sentence in object position that refers to a particular relationship in the lexicon.

The grammar rule for *np* is the same as we used for the relationship declaration (see listing [2]) and for the ease of understanding we keep it in listing [3]. The feature structure *rel:R* in the sentence rule in object position states that this is a "reified" relationship available in the lexicon and unlike the relationship rule stated in listing [2], it does not need to be processed by the lexical rule (*lexical_rule/1*).

This relationship could be a binary relationship or a ternary relationship. This grammar rule is used to link an entity to a relationship that makes the entity an associative entity.

6 Evaluation

We took an example scenario from the ER paper (Frantiska, 2018), extended the scenario and expressed the scenario in CNL^{ER} (see table [1]). After that we translated the CNL^{ER} specification into an internal DL representation and translated the resulting ERM constructs into a RuleML (Boley et al., 2010) JSON[1] representation (see figure 4). This JSON representation is used by our conceptual modelling tool for building an interactive diagram. The implementation of our conceptual modelling tool is based on the GoJS 2.0 frame-

[1] https://wiki.ruleml.org/index.php/RuleML_in_JSON

```
{"And":
    {"Atom":
        [{
            "Rel": "entity",
            "Ind": "department",
            "Var": "X"
        },
        {
            "Rel": "relation",
            "Ind": "employ",
            "Var": ["X","Y"]
        },
        {
            "Rel": "entity",
            "Ind": "teacher",
            "Var": "Y"
        }]
    }
}
```


Figure 4: RuleML JSON representation of the sentence (22) from table [1].

work[2].

The JSON representation is parsed to identify the entities, attributes and relationships for the resulting diagram. Unique entity names and relationship names are identified and added to a node array. Relationships are assessed individually to identify the links and added to a link array. These constructs are then translated into the internal GoJS representation that is used by the GoJS engine to render the diagram in a web browser (see figure 3). New entities, attributes and relationships can be added to a diagram and existing components can be modified or deleted by the user. Building a new diagram from scratch is also possible, since we designed the graphical editor for conceptual modelling as a standalone tool. When a diagram is saved, its internal GoJS representation is produced from which the internal JSON representation can be derived to generate the CNL^{ER} specification again.

To evaluate the controlled natural language CNL^{ER}, we then checked the expressiveness of the language in terms of ERM constructs. We compared the constructed diagram with the textual CNL^{ER} specification of the scenario and found that the diagram correctly represented all corresponding entities, attributes and relationships. Furthermore, it is possible to generate a semantically equivalent CNL^{ER} specification from the diagram in a round-tripping fashion.

[2]https://gojs.net/latest/index.html

7 Discussion

In this paper we presented CNL^{ER}, a controlled natural language that can be used to specify and verbalise ER models. Our main objective is to develop a controlled natural language based universal conceptual modelling framework where a domain expert can actively participate in the conceptual modelling process with a knowledge engineer. CNL^{ER} is a controlled natural language that can express ERM constructs in natural language and the language processor can translate it into a formal language (e.g., any serialization of DL *ALCQI* (Lutz, 2002; Berardi et al., 2005; Franconi et al., 2012)). This formal representation can be further used for verification and validation. After that the formal representation can be processed to generate an ERD.

The goal of this work is to improve the current conceptual modelling process and enable domain domain experts to express their knowledge in a well defined subset of natural language that can be used as high-level interface language to construct conceptual models. Future work will investigate the scalability of CNL^{ER} by extending the coverage of the language so that also UML class diagrams (Calvanese, 2013) and ORM diagrams (Franconi et al., 2012) can be expressed in the same grammatical framework. These extensions involve parametrising and modularising the existing grammar to support additional modelling constructs.

8 Conclusion

CNL^{ER} is a high-level specification and verbalisation language for ER models that supports a controlled natural language based conceptual modelling approach. A textual specification of a conceptual model in CNL^{ER} can be translated via an internal representation into a JSON representation that is then used to generate an ER diagram. The translation works also in the other direction and supports the verbalisation of ER diagrams in CNL^{ER}. Because of these properties, CNL^{ER} has the potential to bridge the communication gap between a domain expert and a knowledge engineer in the domain of entity relationship modelling and makes the modelling process at the same time formal in a seemingly informal way.

References

Vincenzo Ambriola and Vincenzo Gervasi. 2006. On the systematic analysis of natural language requirements with circe. *Automated Software Engineering*, 13(1):107–167.

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

Harold Boley, Adrian Paschke, and Omair Shafiq. 2010. Ruleml 1.0: the overarching specification of web rules. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 162–178. Springer.

David Bourgeois. 2014. *Information systems for business and beyond*. The Saylor Foundation.

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

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"*, pages 368–378. Springer.

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

Terry Halpin. 1998. *ORM/NIAM Object-Role Modeling*. Springer Berlin Heidelberg, Berlin, Heidelberg.

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

HM Harmain and Robert Gaizauskas. 2003. Cm-builder: A natural language-based case tool for object-oriented analysis. *Automated Software Engineering*, 10(2):157–181.

Bayzid Ashik Hossain and Rolf Schwitter. 2018. Specifying conceptual models using restricted natural language. In *Proceedings of the Australasian Language Technology Association Workshop 2018*, pages 44–52, Dunedin, New Zealand.

Mohd Ibrahim and Rodina Ahmad. 2010. Class diagram extraction from textual requirements using natural language processing (nlp) techniques. In *2010 Second International Conference on Computer Research and Development*, pages 200–204. IEEE.

Mustafa Jarrar, C Maria, and Keet Paolo Dongilli. 2006. Multilingual verbalization of orm conceptual models and axiomatized ontologies.

Kenneth C. Laudon and Jane P. Laudon. 2015. *Management Information Systems: Managing the Digital Firm Plus MyMISLab with Pearson eText – Access Card Package*, 14th edition. Prentice Hall Press, Upper Saddle River, NJ, USA.

Domenico Lembo, Daniele Pantaleone, Valerio Santarelli, and Domenico Fabio Savo. 2016a. Easy owl drawing with the graphol visual ontology language. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Domenico Lembo, Daniele Pantaleone, Valerio Santarelli, and Domenico Fabio Savo. 2016b. Eddy: A graphical editor for owl 2 ontologies. In *IJCAI*, pages 4252–4253.

Qing Li and Yu-Liu Chen. 2009. *Entity-Relationship Diagram*, pages 125–139. Springer Berlin Heidelberg, Berlin, Heidelberg.

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.

Luisa Mich. 1996. Nl-oops: from natural language to object oriented requirements using the natural language processing system lolita. *Natural language engineering*, 2(2):161–187.

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

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

Fernando CN Pereira and Stuart M Shieber. 2002. *Prolog and natural-language analysis*. Microtome Publishing.

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

Motoshi Saeki, Hisayuki Horai, and Hajime Enomoto. 1989. Software development process from natural language specification. In *11th International Conference on Software Engineering*, pages 64–73. IEEE.

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

Il-Yeol Song and Peter P. Chen. 2009. *Entity Relationship Model*, pages 1003–1009. Springer US, Boston, MA.

Harald Störrle. 2017. How are conceptual models used in industrial software development?: A descriptive survey. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 160–169. ACM.