
Proceedings Paper:
Barmpis, Konstantinos, Kolovos, Dimitrios orcid.org/0000-0002-1724-6563 and Hingorani, Justin (2018) Towards a framework for writing executable natural language rules. In: Modelling Foundations and Applications - 14th European Conference, ECMFA 2018, Held as Part of STAF 2018, Proceedings. 14th European Conference on Modelling Foundations and Applications, ECMFA 2018 Held as Part of STAF 2018, 26-28 Jun 2018 Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) . Springer , FRA , pp. 251-263.

https://doi.org/10.1007/978-3-319-92997-2_16

Towards a framework for writing executable natural language rules

[1]Konstantinos Barmpis, [1]Dimitrios Kolovos, and [2]Justin Hingorani

[1]Department of Computer Science,
University of York, United Kingdom
[2]JC Chapman LTD
Suite 4, 12 Jenner Av. London W3 6EQ
[1]{konstantinos.barmpis, dimitris.kolovos}@york.ac.uk
[2]justin@jcchapman.com

Abstract. The creation of domain-specific data validation rules is commonly performed by the relevant domain experts. Such experts are often not acquainted with the low-level technologies used to actually execute these rules and will hence document them in some informal form, such as in natural language. In order to execute these rules, they need to be transformed by technical experts into a relevant executable language, such as SQL. The technical experts in turn are often not familiar with the business logic these rules are depicting and will thusly have to collaborate with the business experts to gain insight into the semantics of the rules. This paper presents an approach for writing financial data validation rules in constrained natural language, that can then be automatically transformed and executed against the data they are referring to. In order to achieve this, we use the Xtext framework for creating the editor where business experts can create their rules that can then be transformed into executable constraints. We evaluate this approach in terms of its extensibility, coverage and verboseness with respect to the business rules sent to specific UK banks submitting data under one of the Bank of England's annual reviews.

1 Introduction

Organizations will commonly communicate their policies in natural language, be it internally to their staff and stakeholders or externally to interested parties. As natural language is inherently vague, for achieving consistency and amenability to computer-based processing it needs to be either written in or converted to a formal notation. A common approach is to introduce domain experts (in the domain the data these policies are written against is stored) to convert the natural language documents into the appropriate executable form. This introduces another level of risk as the domain experts will have to interpret these documents, introducing formal meaning to an inherently informal description. As such, the domain experts may have to consult the business experts themselves in order to gain a better understanding, leading to a large increase in both company resources used as well as error-prone cross-domain knowledge transfer.

Another approach is to write the policies themselves in a form more amenable to automation, but also retaining the ability for them to be written by business experts. In this paper we introduce a constraint natural language (CNL) for expressing constraints providing the benefits of machine-readable content whilst also being closely resemblant of natural language itself. This allows data validation rules to be written by non-technical stakeholders without the need for the technical experts; this allows each expert to focus on their respective fields, avoiding any additional risk.

The remainder of the paper is structured as follows: Section 2 discusses tools and methodologies of a CNL-based approach to business rules and Section 3 introduces the Open Rules Platform (ORP), a framework for writing validation rules in CNL. ORP is a commercial product from JC Chapman[1] that was produced as part of a knowledge transfer partnership (KTP) program[2] in collaboration with the University of York. Section 4 discusses the results obtained by using ORP in an industrial use-case and finally Section 5 concludes and mentions future lines of work.

2 Background and Related Work

The development and use of constrained natural language, also referred to as controlled natural language or controlled language, has been extensively investigated over the past decade. This section presents the main state-of-the-art practices and technologies and discusses the approach taken by the Open Rules Platform. As it is assumed that the reader is familiar with model-driven engineering practices like model transformation and domain-specific languages and editors, such information is omitted.

2.1 Constrained Natural Language

One of the most popular standards used to create such business rules in constrained natural language is the Semantics of Business Vocabulary and Business Rules (SBVR). This specification by the Object Management Group (OMG) covers two aspects: Vocabulary (natural language ontology) and Rules (elements dictating policy) [1]. Rules are composed of facts that rely on concepts which are made up of terms. Each term expresses a business concept and a fact can make assertions regarding this concept. Since SBVR does not use any specific language to express these concepts in a concrete fashion (it uses the notion of a "semantic formulation" to describe structure), it is left to the creator to decide the scope and expressiveness of any language conforming to this standard. As SBVR supports both formal and informal expressions, covering both aspects of the formalist vs naturalist approach to constrained natural language [2], it is down to the SBVR-based languages to decide which way they lean towards.

[1] `http://www.jcchapman.com/`
[2] `http://ktp.innovateuk.org/`

For example SBVR Structured English leans heavily towards the formal side whilst RuleSpeak [3]leans towards a more natural form of constrained natural language [1].

There are various tools with languages conforming to the SBVR standard, such as RuleCNL [3] and others [4–6], that vary in their expressiveness and ease of use. As good as SBVR is at introducing structure to constrained natural languages, its inherent complexity means that for smaller dialects the overhead of following the standard may overshadow its usefulness. As such, even though the Open Rules Platform is heavily inspired by SBVR as well as languages using it, it does not formally abide by the standard, instead deciding to keep a more minimalist language metamodel, more amenable to extension.

3 Executable Natural Language Rules

This section presents the architecture and design of the OR platform. Since the platform uses multiple tools and technologies, we briefly introduce them, focusing on how each technology contributes to the system. Finally we discuss the various metamodels used by the OR platform, in order to provide more insight into how the system behaves.

3.1 Architecture

Figure 1 shows an overview of the OR system and below we detail the technologies used:

- Eclipse Modeling Framework (EMF) [7]. This framework is used to facilitate the creation of meta-models that encode the abstract syntax of the developed CNL.
- Xtext [8]. This framework is used to define a textual concrete syntax for the CNL. It offers various (semi-)automatically generated artefacts such as a rich editor for writing statements in this syntax as well as an API for this functionality, which can be used to create web-based editors offering similar capabilities.
- Epsilon [9]. This framework is used to transform CNL models into executable representations (e.g. SQL, EVL). Epsilon comprises a family of languages for performing various model management operations, underpinned by a common model connectivity layer that can access various modeling technologies (like EMF, relational databases or spreadsheets).
- CNL/Mapping metamodels. These metamodels (defined in EMF) are provided to the CNL Xtext parser in order to provide the structure for generating rules models as well as a mapping model from the relevant CNL documents. More details on these artefacts is given in Section 3.2.

[3] `http://www.rulespeak.com/en/`

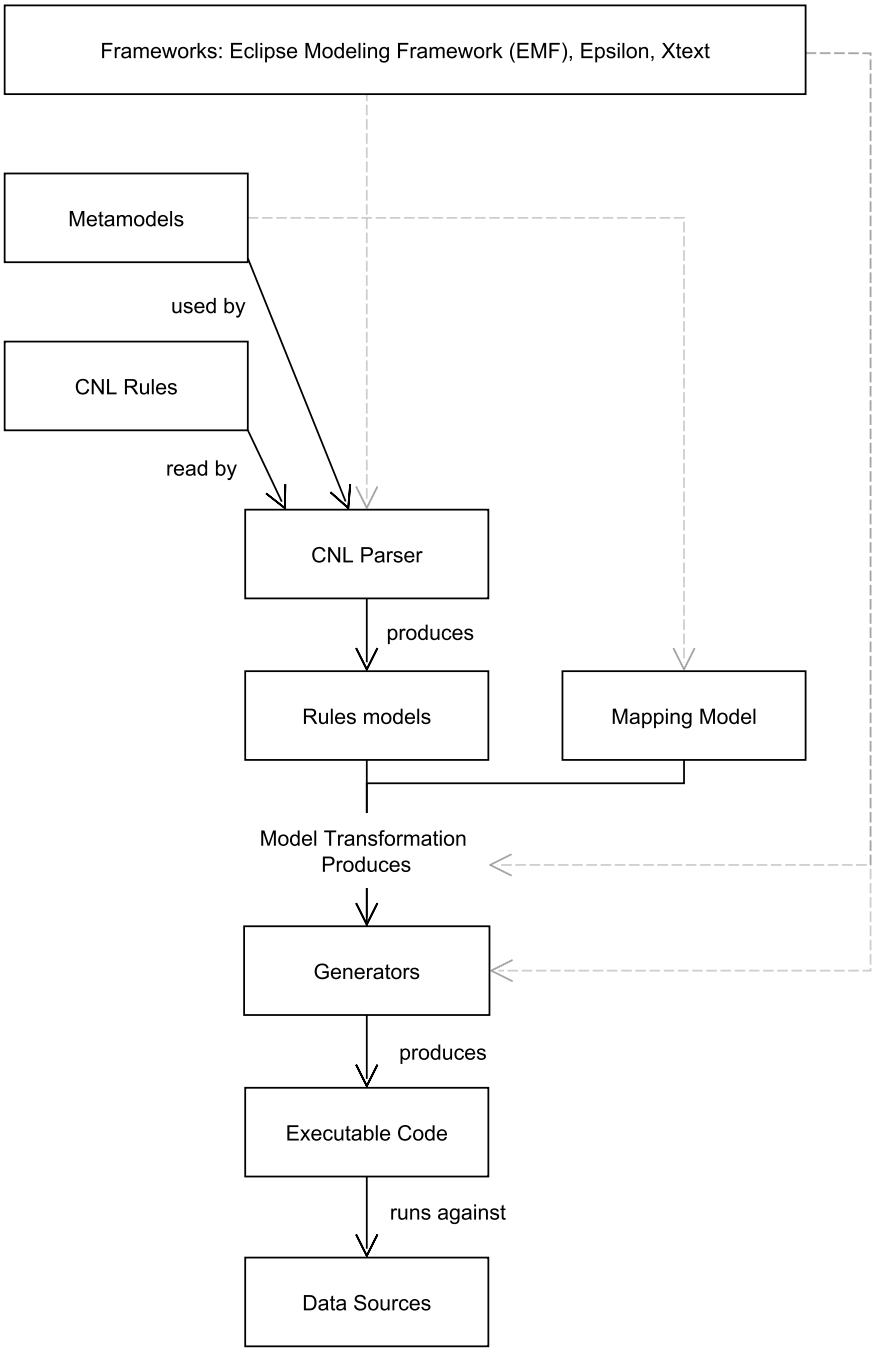

Fig. 1: Architecture of the Open Rules Platform

- Rules/rules domain/mapping CNL documents. These structured text documents, adhering to the CNL grammar, encode rules/constraints for a particular domain/metamodel of interest. The domain document will contain the terminology used by that domain, which can then be used by the CNL document to write rules for that domain. As such, changing to another domain is as simple as creating a new domain document, and will not affect any part of the CNL syntax other than offering new a new domain to write rules against.
- CNL parser. This component parses CNL rules expressed in the language's concrete syntax (aka CNL documents), into in-memory models that conform to the CNL metamodel, that will then be consumed by the transformation engine (Epsilon).
- Generators. Generators consume models and produce executables for a specific back-end and its configuration. These models can be either:
  - The rules/domain/mapping models themselves, whereby a model to text transformation is performed to produce executable code from the models.
  - Back-end specific models representing the technologies used to store the data (such as an SQL model that would conform to a metamodel of the Sequel language). Back-end metamodels will be used for a model to model transformation of the rules/domain/mapping models into a back-end specific model that can then be consumed by a generator to produce executable code. This approach has not been implemented but can have merit, as discussed in Section 5.
- Executables. These are specific to the runtime environment of the end-user such as a MySQL relational database with a specific runtime configuration.
- Back-ends. This is the actual data against which the generated rules will be run against.

3.2 Design

Two metamodels underpin the OR system: that of the CNL itself and that of the various configurations and mappings required to trace elements from the data itself to the rules written in CNL.

CNL Metamodel The CNL metamodel captures the abstract syntax of the language. As such it does not contain any information about *how* the CNL will look like but *what* types of elements it can contain. As such, it is the responsibility of the Xtext parser to convert CNL documents into models conforming to this metamodel, which can then be automatically consumed (in this case by Epsilon) to produce executables. Figure 2 shows a simplified version of the metamodel; important metaclasses are briefly introduced below:

- ConstrainedNaturalLanguageRules. Contains a list of validation rules and/or a list of CNL metadata. This root element allows any CNL document to either contain the rules, the metadata (domain the rules are written against) or both.

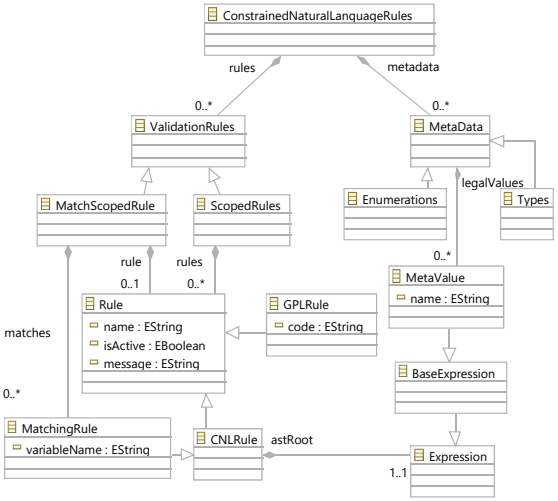

Fig. 2: Simplified CNL Metamodel

- ScopedRules. The first type of rule, where one or more rules are written against a single scope (domain element, found in the metadata document).
- MatchScopedRules. The second type of rule, where a single rule is written against sub-collections of data defined in the matches. These collections can be of the same or different scopes, allowing rules to link queries of multiple domain elements together into a single rule.
- CNLRule. A rule written against elements in the metadata domain, in constrained natural language. It contains a root element of the abstract syntax tree used to define it (after it is parsed into such a tree by the CNL parser).
- GPLRule. A rule written against elements in the metadata domain, in a programming language. It contains a String with the relevant syntax conforming to the language (in an unaltered state from the original CNL document). The responsibility of correctly defining this rule lies with the person writing the document as this rule is directly passed on to a relevant parser without change. This type of rule allows for arbitrarily complex expressions which would otherwise not be expressible in CNL to be written in the same document as the natural language rules themselves.
- Expression. The common supertype for all abstract syntax elements a rule can be made up of, such as comparison, arithmetic (summation, difference, multiplication and division), logical (and / or), unary and other simple binary expressions.
- MetaValue. Common supertype for defining domain metadata.

Mapping Metamodel As different back-ends can require slight variations of similar concepts (such as wrapping identifiers or escaping special characters), a set of configuration options allows abstracting the most commonly found ones from having to be hard-coded into the generators themselves. These include, for example, converting names to lowercase, specifying special characters that enclose identifiers or strings, other special characters that may need to be substituted, etc.

Since it is unlikely that the domain used to create the CNL rules will be a perfect match with the actual data it is validating, the mapping metamodel is also tasked to map various CNL-domain elements to their appropriate data-domain siblings:

- 1 to 1 mappings. Such mappings denote a single feature in the CNL is mapped onto a single feature in the back-end. This mapping is handled directly by the generator, without the need to introduce any new transformation or other overhead.
- 1 to n mappings. Such mappings denote that a single feature in the CNL needs to be mapped to a list of features in the back-end. The values from this list will be then aggregated (in the order provided by the list), using one of the AggregationOperations available in the mapping model, returning a single result.
- Logical Mappings. As different back-ends may use different symbols for denoting equality, negation or other logical and comparison operators, the technology-specific versions can be provided here, in case they differ from the defaults (for example if SQL uses = instead of == for equality).

These manipulations are used by any Epsilon Generation Language (EGL) generator that generates executables (such as the EVL or the SQL generators provided in the use-case presented in Section 4). Any time an identifier (type, feature, variable) is passed to the EGL generator (through the CNL model it is using as its source for code generation), the following may occur, for example:

- A simple mapping (from the mapping model) to replace the identifier with a new one.
- Replacing or removing special characters.
- Converting strings to lowercase.

Generators These components take a CNL model (after it has been mapped using the appropriate mapping model conforming to the metamodel presented above) and generate appropriate execution-level code to be run against the stored data. Such generators can be produced either using a model to model or a model to text transformation:

Using a model to text transformation is recommended when there are a lot of static text regions that need to be frequently repeated. This provides the freedom to create optimal execution-level code, but may end up being non-trivial to maintain, should the static regions end up becoming too verbose.

Using a model to model transformation is relevant when an appropriate metamodel and unparser of the execution language is available.

Generic EVL generator This generator produces code written in the Epsilon Validation Language. The generation mainly comprises transforming the rule abstract syntax tree into the appropriate expression in EVL. This generator can run against any data format supported by the Epsilon framework, such as EMF models, XML documents, Spreadsheets, Relational Databases, etc. Nevertheless, since it is a generic layer it may not be able to be fully optimized against all such technologies and alternative generators may need to be used for performance reasons.

Optimized native SQL generator Since SQL has a substantially different structure to EVL, a native generator provides full control over how a CNL model is converted to executable code to be run against a relational database. This allows tackling of issues such as type correctness by using the database metadata (instead of having to compare each relevant data item to a type), allows effective use of derived tables and merging etc. Preliminary tests have shown that for certain classes of validation rules this can greatly outperform a nave EVL generator, and that it can be up to an order of magnitude faster than writing inefficient SQL (further investigation onto this is required, as detailed in Section 5).

Web-based validation Xtext offers a generated web-based API for writing CNL rules on a web-client and executing a program on these rules on the server side, returning a document containing relevant execution information. Using one of the abovementioned generators, we can perform validation of the provided rules against data stored on the server. This would execute the following (on the server side, after receiving the CNL document from the client):

- The input CNL model is transformed using the relevant mapping model to an in-memory transformed CNL model.
- The transformed CNL model is used to generate the appropriate execution-level code to run against the stored data.

Currently the EVL generator is used to produce EVL code which will run against data stored on an Excel spreadsheet, but both the generator used and the data connector can be replaced if necessary with the appropriate ones. The EVL code is executed against the data, getting back violations for each of the rules. The violations are formatted into a report document which is then returned to the client, providing feedback on offending elements (if any).

4 Evaluation

In this section we present the empirical results obtained when evaluating the OR system against a domain-specific use-case.

4.1 Use-case

Annually the Bank of England (BoE) produces a large set of rules that specified UK banks follow to submit data as part of one of the BoE's annual reviews. These rules help ensure the submitted data is consistent with the BoE's expectations and cover a variety of different aspects such as retail risk, commercial risk, operational risk, etc. These rules are provided in a mixture of natural language and procedural statements and are written so that domain experts can understand them, hence are not amenable to machine consumption in any way. In order for these rules to be executed against the actual data the bank holds, they have to be understood by a domain expert and then a technical expert will have to write the appropriate low-level code representing these rules. This process requires stakeholders with different expertise to collaborate and can introduce further risk as the two interpretation steps need to be in line with one another.

4.2 Coverage

As a first criterion for evaluating the OR platform for this use-case, we classified BoE's rules into 8 categories and then analyzed the coverage of the OR platform with respect to the total number of rules. This was done to estimate the actual coverage of the OR system as it would not have been feasible within the scope of this project to convert all 3668 rules into CNL to execute them; as such random sampling of each category has been performed.

category	**description**	**count**
type/enum check	this rule only contains a single type or enumeration check	1556
comparison	this rule only contains a single comparison	315
comparison+logic	this rule only contains a combination of comparison and logical operators	40
using variables/functions	this rule requires comparison/logical operations across tables	1534
duplicate check	this rule requires all values of a field to be different	17
multi-key match	this rule requires multiple fields to be treated as a key to a search	99
enumeration sub-matching	this rule requires that the legal values of a field are determined by the current value of another field	27
complex rule	this rule is too complex to classify	80
		3668

Table 1: Classification of Business Rules

Table 1 presents these categories in more detail. Here, we can see that the large majority of rules fall under either simple type/enumeration checks or elab-

orate rules requiring the use of variables and functions (matching rules), often across different domain elements. from the remaining rules, the 80 complex rules are noteworthy as it was decided that attempting to further classify them or to convert them to CNL was not efficient. Instead, these rules are flagged as complex and meant to be executed through the use of GPL rules that are written in the target language used to execute against the data itself. Finally, the category of enumeration sub-matching is not yet supported, even though such a feature can be added in further iterations of the tool, as mentioned in Section 5.

As such, we achieve 97% coverage (as we don't currently offer CNL expressiveness for complex rules and enumeration sub-matching rules), whilst opening the possibility (through the use of GPL rules) for any rule to be written in the CNL document regardless, in order to ensure that a single document contains all the rules that need to be executed, regardless of whether they can be actually expressed in CNL.

4.3 Verboseness

The second criterion used to evaluate the OR platform is the verboseness of the rules, when written in CNL. Should the CNL form of the rules be disproportionate to the complexity of the rule (the size of the rule written in the execution language) then it may be unreasonable to expect them to be written by domain experts as it will become tedious to write extremely long CNL rules. As such, we compare the size in characters (ignoring whitespaces) of various rules written in CNL with the rule written in both EVL as well as SQL, as a representative sample of verboseness.

category	**cnl^1**	**cnl^2**	**evl^1**	**evl^2**	**sql^1**	**sql^2**
type/enum check	33	57	86	111	202	215
comparison	51	184	118	327	116	405
comparison+logic	131	139	209	243	139	154
using variables/functions	314	447	522	703	568	740
duplicate check	55	58	226	241	116	119
multi-key match	63	89	487	627	163	187

Table 2: Rule character count ignoring spaces

Table 2 shows the relevant character count for two representative rules written for each of the categories the tool supports. CNL written in the ORP framework is much less verbose than the SQL it would require to execute against data in relational databases (in this case a MySQL database) and less verbose than EVL constraints written in Epsilon. Considering both the EVL and SQL were generated by the tool and as such attempt to be as minimal as possible (as they do not care about human readability at all), we have gained confidence that

writing rules in CNL will require less effort than the same rule written by the relevant expert in EVL or SQL.

Below we see how the type/enum check constraint annotated as cnl[1] looks like:

```
1  in a Branch the country must be in Europe
```

Similarly for the comparison+logic cnl[1] constraint:

```
1  in a MortgageAgreement
2    when the beginningDate exists and the initialEndingDate exists
3    then the beginningDate must be before or by the initialEndingDate
```

4.4 Extensibility

The final criterion used to evaluate the OR platform is its extensibility. Since the system claims to offer domain-agnostic CNL capabilities for writing rules in any domain, we need to gain some confidence that this can be feasible. As such, extensibility can be broken down into three distinct categories:

- Language extensibility. Since the OR platform offers a constrained form of English for expressing rules, this category considers how easy it is to alter this constrained subset should a new type of (English) expression be required or should a new type of executable expression be required (such as the example of the enumeration sub-matching rules in the coverage example, which are not currently expressible in CNL).
  - Regarding extending the subset of English supported by the CNL, this would require adding the new expressions in the Xtext parser that reads the CNL document and creates the relevant model. Since adding new English phrases is unlikely to affect the model itself but rather only the parser, we believe that the OR platform is extensible in this regard as only one component of the system needs to be adapted to add this functionality.
  - Regarding the extending of the semantic expressions offered by the OR platform, this would require the extension of the CNL metamodel to include these new concepts, as well as the extension of the Xtext parser to include a way to express these rules in English. As the adaption of two different interconnected components is required to achieve this, we consider this to be a task of moderate difficulty for an extender of the tool.
- Domain extensibility. If rules need to be written in a different domain, a domain document will have to be created detailing the various concepts in that domain and their relevant features. These concepts will then be usable in the CNL document describing the rules written for that domain. Since the CNL document is not bound to a specific set of concepts but to another document which will describe these concepts, we believe it is natural to change from one domain to another without much effort.
- Execution technology extensibility. This category considers whether it is possible to change the data storage technology and still be able to execute CNL

rules. The execution layer of the OR platform is de-coupled from the language itself, as the data can be accessed either through the Epsilon model connectivity layer (whilst generating EVL rules), or through the use of a new generator that takes the rule document alongside the domain and mapping documents and produces executable code for the required storage technology. As such, we have gained confidence that the OR platform is extensible with respect to use of other data storage technologies.

Overall extending the OR platform for these three categories will require adding/changing only one component in most cases (with two components needing to be changed when new types of semantic expressions need to be added).

5 Conclusions and Further Work

Concluding, we have presented ORP and its CNL, aimed at offering executable validation rules written in natural language. We have evaluated this framework in a real-world case-study using a subset of the Bank of England's business rules and have obtained promising results in both the areas of coverage and verboseness, whilst qualitative evaluation of extensibility is also promising.

The tool can be extended to provide advanced features to cover even more types of rules, such as: n to 1 mappings; such mappings denote multiple fields in the CNL needing to be mapped onto a single field in the data schema (that needs to be disaggregated appropriately). Simple numerical disaggregations (such as each CNL field containing an equal (numerically) subdivision of the target field) can be performed within the CNL itself without the need for further information, but any complex expression-based mapping will need to be presented and incorporated into the mapping model. Reference resolution; to tackle data normalization, references need to be navigated using unique identifiers of elements. This navigation may require extra information such as naming conventions of the target object (such as using a foreign key with a different column name to the original, in a relational database, etc.).

Finally, investigating the applicability of using model-to-model transformations to various back-end technologies through an appropriate metamodel and unparser (for example using an SQL metamodel and an SQL unparser to convert a CNL model into an executable SQL model) can provide insight into this alternative approach.

Acknowledgments

This research was part supported by Innovate UK through its Knowledge Transfer Partnership (KTP) program and JC Chapman LTD.

References

1. Silvie Spreeuwenberg and Keri Anderson Healy. Sbvr's approach to controlled natural language. In Norbert E. Fuchs, editor, *Controlled Natural Language*, pages 155–169, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
2. Peter Clark, William R. Murray, Phil Harrison, and John Thompson. Naturalness vs. predictability: A key debate in controlled languages. In Norbert E. Fuchs, editor, *Controlled Natural Language*, pages 65–81, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
3. Paul Brillant Feuto Njonko, Sylviane Cardey, Peter Greenfield, and Walid El Abed. Rulecnl: A controlled natural language for business rule specifications. In Brian Davis, Kaarel Kaljurand, and Tobias Kuhn, editors, *Controlled Natural Language*, pages 66–77, Cham, 2014. Springer International Publishing.
4. G. Aiello, R. D. Bernardo, M. Maggio, D. D. Bona, and G. L. Re. Inferring business rules from natural language expressions. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pages 131–136, Nov 2014.
5. Paul Brillant Feuto Njonko and Walid El Abed. From natural language business requirements to executable models via sbvr. In *2012 International Conference on Systems and Informatics (ICSAI2012)*, pages 2453–2457, May 2012.
6. P. B. Feuto, S. Cardey, P. Greenfield, and W. E. Abed. Domain specific language based on the sbvr standard for expressing business rules. In *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pages 31–38, Sept 2013.
7. Marcelo Paternostro Dave Steinberg Frank Budinsky and Ed Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional, 2008.
8. Moritz Eysholdt and Heiko Behrens. Xtext: Implement your language faster than the quick and dirty way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, OOPSLA '10, pages 307–309, New York, NY, USA, 2010. ACM.
9. Kolovos, D.S., Rose, L., Garcia, A.D. and Paige, R.F. *The Epsilon Book*. 2008.