Natural Language Specification and Violation Reporting of Business Rules over ER-modeled Databases

Mika Cohen
FOI, Stockholm, Sweden
mika.cohen@foi.se

Michael Minock
KTH Royal Institute of Technology, Stockholm
minock@kth.se

Daniel Oskarsson
FOI, Stockholm, Sweden
daniel.oskarsson@foi.se

Björn Pelzer
FOI,Stockholm, Sweden
bjorn.pelzer@foi.se

ABSTRACT

This paper presents our work on adapting and extending *natural language interface (NLI) to database* technology to support the specification and violation reporting of business rules. The resulting system allows non-technical users to author and manage a rulebook in controlled natural language – serving as a single point of definition that can be compiled into SQL to generate violation reports. To achieve this we represent business rules in tuple calculus, handle negation in our query re-writing algorithms and add support for natural language reflexives (e.g. 'its', 'themselves', etc.). Our results show a large class of business rules can be captured with these extensions. Although our approach is general, we present it applied to compliance checking of regulations over a materiel capability development information system at the Swedish Defence Materiel Administration. At EDBT we will also demonstrate this work over a more generic package delivery domain. While there has been recent effort in pursuing *Semantics for Business Vocabulary and Business Rules* (SBVR) in the semantic web and description logic communities, to our knowledge ours is the first attempt to provide this capability for ER-modeled relational databases.

Keywords

SBVR, Business Rules, NLIs to databases

1. INTRODUCTION

Large organizations typically maintain a wide range of information systems each with their own interfaces and schemas. Much effort in recent years has been expended to aggregate overall information systems into federated databases (or *enterprise information systems*) so that overall activities can be monitored, analyzed and, if required, remedied. While an individual department can often ensure that its data sources conform to its own rules, violations of organization wide rules often occur when all the sources are federated. For example while it is not be permitted for a clerk to enter an order with no purchase item specified, other compliance level 'constraints', such as ensuring that those who ordered an item have authority to use it, are difficult to enforce in legacy systems. Even at the local level, there may be reasons not to enforce constraints too strictly, such as to monitor states that are not outright errors but nonetheless deviate from expectations in noteworthy ways.

An approach to relaxing constraint enforcement to accommodate these cases is to check for compliance *retroactively*. Operational staff are given the rights to enter data, and then, during *compliance checking*, rules are checked against the federated database, looking for violations of regulations, incompleteness in data sources, breakdowns in operations, etc. With such violation reports, errors may either be corrected, or, based on discretion, tolerated. This temporal decoupling of compliance checking from data entry also allows for a wider spectrum of rule specificity, enabling vaguer, high-level business rules in addition to specific data-level constraints. Rules will then originate from stakeholders at various levels across the organization who build up *rulebooks* in natural language, which are subsequently mapped to low-level, machine-executable implementations, such as SQL.

The translation of natural language rules to executable form can be tedious and costly, and since it is typically done by technical staff that are separate from the rule authors (both organizationally and in terms of skills), the translation involves interpretation – a source of possible errors, especially in light of the ambiguity of natural language. Also, keeping parallel representations (natural language and executable code) of the rulebook introduces a burden of management to maintain consistency over time. Finally, if the natural language formulations are overly verbose, stakeholders in more remote parts of the organization will find it difficult and burdensome to understand the rules.

Based on these shortcomings, recent efforts have focused on *Semantics for Business Vocabulary and Business Rules* (SBVR) [8]. SBVR argues that the specification of business rules should be based on a clear conceptual model of the domain. Moreover the business rules and the conceptual domain must be based on a controlled natural language syntax, so as to reduce or eliminate ambiguity, increasing the probability that all the stakeholders will understand the rules. Finally there is the potential that natural language rules may be automatically mapped to executable form.

10.5441/002/edbt.2015.53

While SBVR is now an OMG standard, implementation work is still in its infancy (see [1] for references to current efforts). Most, if not all, of the tools are focused on realizing SBVR over description logics and semantic web technology (for example [9, 3]). Approaches mapping all the way to SQL are promising, but still quite preliminary[7]. Our work seeks to seat SBVR[1] in classical ER-modeled relational databases. Given the dearth of existing tools, to achieve this, we decided to adapt and extend an existing natural language interfaces to database system, C-PHRASE[6] `https://code.google.com/p/c-phrase/`.

In section 2 we introduce the domain in which we are applying our work work – The SKTS at FMV, the Swedish Defence Materiel Administration. Section 2 gives an ER model and several prototypical business rules drawn from our corpus, which we can interpret, paraphrase and report the violations of. Section 3 discusses the extensions of C-PHRASE that were required to achieve this. Beyond representing rules, we were required to extend our treatment of negation in paraphrases and were forced to extend the system to support reflexives. Section 4 gives our demonstration plan for EDBT. Finally section 5 discusses the broader relevance and future directions of our work.

2. THE SKTS AT FMV

Let us consider the simple real-world example, the SKTS[2], which is a description of the organization of technical systems (i.e. equipment of non-trivial technical complexity) by the Swedish Armed Forces and managed by the Swedish Defence Materiel Administration FMV (*Försvarets materielverk*). SKTS describes the relations between systems as well as their deployment, their life-cycle phases and the associated decision making processes. Our non-classified extract of SKTS has the same schema as the full version. See Figure 1 for a simplified ER model of the SKTS.

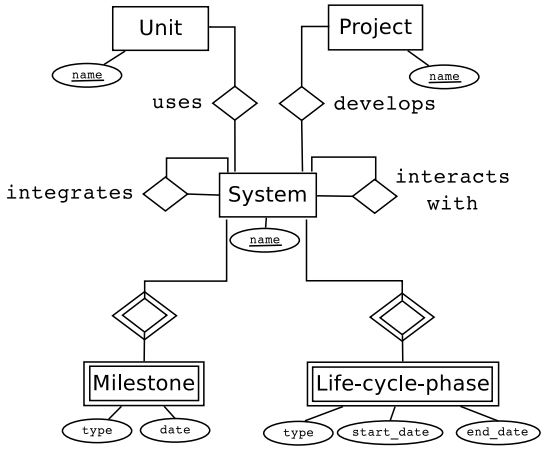

Figure 1: Simplified ER model of SKTS

The central entity in SKTS is the *system*, representing about 1,600 types of technical systems, ranging from ammunition and tools to vehicles and buildings. In our simplified schema a system type is identified by a single attribute *name*. Two special relationships between systems are:

- system type S_1 *integrates* system type S_2: S_2 is a component of S_1, like a tank integrating an engine;
- system type S_1 *interacts with* system type S_2: S_1 cooperates with S_2 (without integration), like artillery interacting with a forward observer vehicle to obtain target data;

The *unit* and *project* entities represent the actors employing technical systems, i.e. military units during active use and research projects during the earlier life-cycle phases. Systems are assigned to such users via the respective relationships *uses* and *develops*.

The usage of systems is constrained by weak entity sets life-cycle phases and milestones. A *life-cycle phase* represents an interval, and thus it has attributes specifying its *start date* and *end date*, and its *type* attribute is one of *concept*, *development*, *production*, *use*, *maintenance* or *retirement*. A system can have multiple life-cycle phases associated with it – ideally one of each type. A *milestone* represents an event like a deadline, so it has only a single *date* attribute. Numerous values are possible for its *type* attribute, since up to three different milestones govern each life-cycle phase of a system: a *deadline* by which the FMV must have decided on the start and end dates of the phase, a *planned decision* date when the FMV intends to make this decision, and a *decision* date when the decision is actually made.

The FMV wants to apply SBVR-formulated business rules to an SKTS-database to ensure that its regulations are not violated within the organization. Five prototypical rules drawn from our initial corpus of hundreds of rules are:

1 "It is forbidden that a system has a use phase that begins before its use decision."

2 "It is obligatory that a system that is in a use phase be used by some unit."

3 "It is obligatory that a system that interacts with another system with a use phase must also have a use phase."

4 "Every system that is assigned to some unit must be currently operational."

5 "It is obligatory that a life-cycle phase which is not a concept phase and which has a start date no later than the current date be subsequent to some other life-cycle phase."

Applying such rules to SKTS should produce a detailed list of violations, allowing an analyst to identify the best method to handle each problem – be it a simple database flaw or an actual breach within the organization, like a unit employing a system not cleared for use.

3. EXTENDING C-PHRASE TO SBVR

We have extended the C-PHRASE NLI to database system to enable users to state business rules of the form above (as well as many other rules and alternative phrasings), receive paraphrases of such business rules (in case of ambiguity),

[1]Technically it is SBVR-SE (SBVR Structured English), a subset of SBVR, that we seek to support.

[2]*Systemkarta tekniska system*, Swedish for "system map (of) technical systems"

and receive reports of the violations of such rules that give an indication of how an instance violated the rule. C-PHRASE is discussed elsewhere [6], but in short it uses a semantic grammar to map user utterances to (an extended) tuple calculus. From this representation either a natural language paraphrase or SQL may be generated. C-Phrase uses a theorem prover to evaluate when a natural language utterance has mapped to more than one semantically distinct query. In such a case of ambiguity, the system paraphrases all semantically distinct interpretations back to the user so that they may select the proper interpretation. The paraphrasing technique also heavily uses a theorem prover in finding an equivalent re-writing of the paraphrased query using a lexicon of elementary query expressions paired with associated words and phrases. Once a single interpretation of the user's utterance is determined, the tuple calculus expression is evaluated over the database and answers are reported back to the user.

For C-PHRASE to handle business rules, several extensions were required. First, it was necessary to extend the tuple calculus representation to represent rules. This was only necessary in the case of positive rules. *Negative rules* expressed as "it is prohibited that A", may simply be represented as the tuple expressions for A, leveraging C-PHRASE's existing semantic analysis mechanism. For example, the first example rule above is represented in the tuple query expression[3]: $\{x|System(x) \wedge (\exists y_1)(Life\text{-}Cycle\text{-}Phase(y_1) \wedge x.name = y_1.system \wedge y_1.type = \text{'use phase'} \wedge (\exists y_2)(milestone(y_2) \wedge y_1.start\text{-}date < y_2.date \wedge (\exists y_3)\ (System(y_3) \wedge y2.system = y_3.name \wedge \mathbf{x = y_3} \wedge y_3.type = \text{'use start decision'})))\}$. For *positive rules*, expressed as "it is necessary that A are B", we introduced a **:consequent** marker, which specifies which part of a tuple calculus expression is the right hand side of a rule. For example the third rule above is represented as: $\{x|System(x)(\exists y_1)(\exists y_2)(Interacts\text{-}With(y_1) \wedge System(y_2) \wedge x.name = y_1.interacts\text{-}with \wedge y_1.system = y_2.name \wedge (\exists y_3)(Life\text{-}Cycle\text{-}Phase(y_3) \wedge y_2.name = y_3.system \wedge y_3.type = \text{'use phase'}))\langle: \textbf{consequent}\ System(x) \wedge (\exists y_4)(Life\text{-}Cycle\text{-}Phase(y_4) \wedge x.name = y_4.system \wedge y_4.type = \text{'use phase'})\rangle\}$. Finally, C-PHRASE's grammar was extended to recognize a large set of variations of "it is prohibited that" (or "it is obligatory that"), to insert the **:consequent** marker in the positive case, and then to branch to the proper *rule handler* with the resulting tuple calculus expression.

Although our semantic grammar avoids the pathological ambiguity of more linguistically-oriented approaches (e.g. "time flies like an arrow"), we do still confront ambiguity. For example in rule specification 5 above, there is a hidden ambiguity in 'subsequent'. 'Subsequent' in this domain may mean immediately subsequent. Or subsequent with a possible time gap. There are thus two semantically distinct rules that rule 5 above maps to. Thus the user must pick between them. And to pick, the user must understand the nuances in these formulas. And for that, among other reasons, we paraphrase rules back to the user in natural language. At the end of this interaction, we will have one single semantically meaningful rule to pass to the rule handler.

The rule handler converts the rule into one or more *violation queries* which identify violation cases for the rule. This occurs in a two step process. First, a *core violation query* is calculated. Then this core violation query is extended to the (full)*violation query*, which includes supporting information indicating why matching answers to the core violation query are rule violators. In the negative case, the core violation query is simply the query representing the rule. In the positive case, the core violation query is the query with the consequent negated. In cases where the consequent is a conjunction, we apply De Morgan's law, followed by a simple rewriting to generate a set of core violation queries. Extending core violation queries to full violation queries is a capability inherited from C-PHRASE's answer strategy mechanism. In short, the theorem prover finds the most specific answer strategy that subsumes the core violation query and then augments the core violation query with the associated answer value bindings. For example, assume that the lexicon contains the answer strategy $\langle\{x|System(x) \wedge x.name = c_1 \wedge (\exists y_1)(Life\text{-}Cycle\text{-}Phase(y_1) \wedge y_1.system = x.name \wedge y_1.type = \text{'use phase'} \wedge y_1.start\text{-}date = c_2 \wedge y_1.start\text{-}date = c_3)\}$:*"The system c_1 has a use phase starting c_2 and ending c_3"*$\rangle$. Then a core violation query that is subsumed by this answer strategy, and by no more specific answer strategy, will be augmented with the additional bindings, and an answer fitting the template will be generated (The reported answers in Figure 2 apply this answer template). The translation of the full violation query to SQL is trivial. The report presented to the user consists of a paraphrase of the rule, followed by a paraphrase of the core violation query, followed by the answers to the full violation query. The report for rule 2 above is shown in Figure 2.

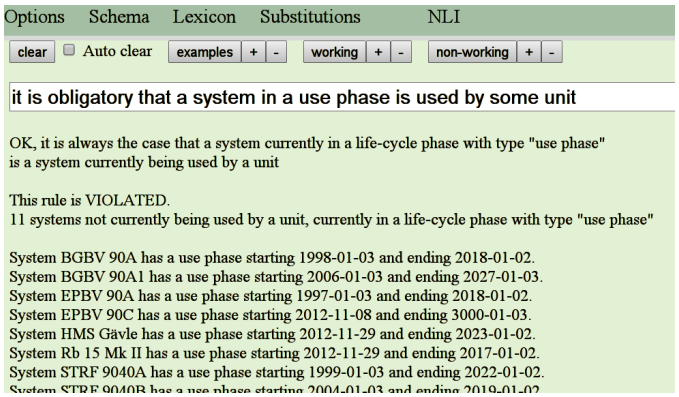

Figure 2: Report for rule 2.

Given that paraphrases of rules or violation queries necessarily contain negation of existential quantification, and given that our techniques of generating paraphrases involves finding equivalent re-writings of queries using elementary logical expressions, we were forced to confront a difficult non-conjunctive query rewriting problem. As a simple example, consider that the expression $\{x|System(x) \wedge \neg(\exists y_1)(\exists y_2)(Interacts\text{-}with(y_1) \wedge Sytem(y_2) \wedge y_1.system = x.name \wedge y1.interacts\text{-}with = y_2.name \wedge y_2.name = \text{"HMS Gavle"})\}$ should be rewritten using the three lexicon entries $\langle\{x|System(x)\}$:*"systems"*$\rangle$, $\langle\{x|System(x) \wedge \neg(\exists y_1)(\exists y_2)(Interacts\text{-}with(y_1) \wedge Sytem(y_2) \wedge y_1.system = x.name \wedge y1.interacts\text{-}with = y_2.name \wedge \varphi(y_2))\}$:*"does not interact with ref($\varphi(y_2)$)"*$\rangle$ and $\langle\{x|System(x) \wedge x.name = c_1\}$:"c_1"$\rangle$ to yield an equivalent rewrite, and in turn the

[3]Queries are defined over the schema that is the standard translation of the ER model in figure 1.

templates may be used to generate the paraphrase *"systems not interacting with HMS Gavle"*. Although we have not yet formalized our approach to the point where we can prove its completeness, we have considerably extended our query rewriting algorithms to handle negation.

A final extension to C-PHRASE that should be noted is support for reflexives. Normally in NLIs to databases one does not confront reflexives. A contrived example could be, "give the systems interacting with *themselves*". In business rules however, we have witnessed a fair number of cases in our corpus that require this. For example the *'its'* in the first rule above is a reflexive. Our approach to reflexives is similar to that for **:consequent**. We insert a marker **:reflexive** in at parse time when reflexives are recognized. This leads to a tuple calculus expression which is later resolved, finding bindings of the reflexive variables with the same entity type. The semantic representation of rule 1 presented above shows the results of this with the tuple equality condition $\mathbf{x} = \mathbf{y_3}$. In cases of multiple possibilities, all possibilities are passed on for interactive ambiguity resolution. The paraphrasing mechanism was trivially extended to generate reflexives.

4. DEMONSTRATION

Our demonstration plan at EDBT is to first show the live operation of the system in interactive mode, where a user states business rules over the SKTS schema above, receives paraphrases of rules, and then receives a detailed report on the violations of the rules over our declassified database instance. We will demonstrate all the rules in section 3 as well as additional rules in our corpus. We will also present rules that still give us problems due to either linguistic or conceptual complexity. In addition to the live demonstration, we will generate a series of videos that illustrates how we configured C-PHRASE to handle this task. Moreover we will demonstrate the same technology over a prototypical use case in the package delivery domain. All configuration files and source code for our demonstration will be open sourced to let others verify, replicate and build on our results.

5. RELEVANCE AND CONCLUSIONS

It has been noted many times [2, 6, 4, 5] that natural language interfaces to critical systems must include a paraphrasing mechanism which communicates back to the user how their utterance is interpreted. Without such a mechanism how could anyone ever rely upon such a system? Without such a capability how would one let users resolve ambiguity? This is particularly true with the interpretation and reporting of business rules which, arguably, are more complex than information seeking queries. Given that paraphrasing is critical, and that even very simple business rules may be reported in either positive or negative forms, interpretation and paraphrasing mechanism must be extended to handle negation over existential quantifiers; NLI systems restricted to conjunctive query representations, will simply not suffice. Our first contribution in this work is to demonstrate that such negation may be built into NLI systems to adequately handle this requirement. In addition we observed in our work the importance of supporting reflexives in the specification of business rules; others attempting to achieve the same will likewise need to cover reflexives. Finally we argue that the use of a theorem prover is critical to determining semantic equivalence and greatly simplifies the proper reporting of violations of business rules.

There are still types of business rules in our corpus that we do not cover. For example cardinality type queries are not yet covered. Nor do we yet support offsets in time expressions. So the rule, *"It is prohibited that a project develops a system that interacts with at least three other systems whose retirement phases begin less than two years after the start date of its use phase."* is currently beyond our grasp and will require both syntactic and semantic extensions. These extensions are currently being explored. Moreover, we are investigating the building up of complex rules, and rules with exceptions, as multiple step interactions. In some domains, sufficiently detailed rules can probably not be specified as single shot sentences.

While it is necessary to configure C-PHRASE over the ER modeled database, one side benefit is that one gets an NLI access interface as an added bonus, justifying in part the additional cost. Moreover, because a C-PHRASE configuration can be tailored by database administrators with limited linguistic training, the configuration may be extended to capture domain dependent idiosyncratic phrasings. As database administration staff build out the natural language interface over the ER modeled database, we envision stakeholders engaging in the process of proposing and organizing business rules that may be shared, understood and most importantly executed across the organization. Our work takes a step toward realizing this vision.

6. REFERENCES

[1] I. S. Bajwa, M. G. Lee, and B. Bordbar. SBVR business rules generation from natural language specification. In *AI for Business Agility, AAAI Spring Symposium*, 2011.

[2] E. Codd. Seven steps to rendezvous with the casual user. In *IFIP Working Conference Data Base Management*, pages 179–200, 1974.

[3] C. Fürber and M. Hepp. Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 2011 EDBT/ICDT Workshop on Linked Web Data Management, Uppsala, Sweden, March 25, 2011*, pages 1–8, 2011.

[4] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 333–344, 2010.

[5] F. Li and H. V. Jagadish. Nalir: an interactive natural language interface for querying relational databases. In *Proc. of SIGMOD 2014*, pages 709–712, 2014.

[6] M. Minock. A STEP towards realizing Codd's vision of rendezvous with the casual user. In *Proc. of Very Large Data Bases (VLDB)*, pages 1358–1361, 2007.

[7] S. Moschoyiannis, A. Marinos, and P. Krause. Generating SQL queries from SBVR rules. In *Semantic Web Rules*, volume 6403 of *LNCS*, pages 128–143. Springer, 2010.

[8] OMG. *Semantics of Business Vocabulary and Rules (SBVR) (version 1.2)*. November 2013.

[9] D. Solomakhin, E. Franconi, and A. Mosca. Logic-based reasoning support for SBVR. In *Proc. of the 26th Italian Conference on Computational Logic*, pages 311–325, 2011.