Parsing SBVR-based Controlled Languages

Mathias Kleiner[1], Patrick Albert[2], and Jean Bézivin[1]

[1] INRIA - EMN, Atlanmod team, Nantes 44000, France,
[2] ILOG S.A, 9 rue de Verdun, 94253 Gentilly, France
mathias.kleiner@inria.fr,

Abstract. Conceptual schemas (CS) are core elements of information systems knowledge. A challenging issue in the management processes is to allow decision makers, such as business people, to directly define and refine their schemas using a pseudo-natural language. The recently published Semantics for Business Vocabulary and Rules (SBVR) is a good candidate for an intermediate layer: it offers an abstract syntax able to express a CS, as well as a concrete syntax based on structured English. In this article, we propose an original method for extracting a SBVR terminal model out of a controlled English text and then transform it into a UML class diagram. We describe a model-driven engineering approach in which constraint-programming based search is combined with model transformation. The use of an advanced resolution technique (configuration) as an operation on models allows for non-deterministic parsing and language flexibility. In addition to the theoretical results, preliminary experiments on a running example are provided.

Key words: controlled languages, parsing, SBVR, model-driven engineering, constraints, configuration, model search

1 Introduction

Conceptual schemas (CS) are widely used in industry as formal representations of information systems knowledge. A CS is often the central element on which relies a set of operations. The UML/OCL combination is the de-facto standard for specifying a CS. However, designing, maintaining and refining a CS currently requires important technical skills. Stakeholders thus rely on experts to model their requirements. A recent trend in software engineering (requirements engineering) is to discover ways to facilitate this communication, by allowing decision makers to express their needs in a comprehensive language which can then be transformed into a formal representation.

In the business context, the Object Management Group (OMG) has recently published the SBVR (Semantics for Business Vocabulary and Rules) recommendation. SBVR provides a metamodel for business concepts and statements which can be used to define a CS. The specification also proposes a structured English form for a model that is easily understood by business people. However parsing

[0] Work partially funded by the French ANR IdM++ project

the proposed structured English into a SBVR model is a difficult problem. In particular, the different interpretations that can be made for one sentence disqualifies the existing model driven engineering (MDE) tools (such as ATL[11] or QVT[19]) that are based on deterministic algorithms. If one wishes to keep this freedom in the language then the problem requires to *search for* a solution using advanced AI technics.

This article describes an automatic translation of SBVR structured English into a SBVR concrete model and a UML model of the described CS. The originality of our approach is that it combines constraint programming techniques with model transformation tools in a MDE framework. It is composed of three main operations. The first task is a syntactical and grammatical analysis of the text, which is directly related to the well-known and challenging field of (controlled) language parsers: we describe a SBVR parser based on an advanced constraint programming technique known as configuration. The second task is the transformation of the resulting model into a SBVR model. The third task is the transformation of the SBVR model into a UML model of the CS. The integration of constraint programming in MDE as an advanced transformation tool is an important and innovating contribution of this work.

Plan of article Section 2 briefly introduces each technology used in our approach. We also present an overview of the whole process and a running example. Section 3 introduces the controlled language parser. Section 4 shows how the resulting model is transformed into a SBVR model. Section 5 proposes a transformation from SBVR to UML. Validation, implementation and experiments are presented in Section 6. Finally, we discuss related work in Section 7 and conclude.

2 Context of the work

2.1 Brief introduction to SBVR

SBVR is an OMG standard [21] intended to be the basis of the description of business activities in natural languages. SBVR attempts to build the bridge between Business Users and software artifacts, enabling non-IT specialists to parameterize and evolve the business logic embedded into applications. SBVR standardizes a set of concepts enabling the definitions of business specific Controlled Languages (declarative languages, whose grammar and lexicon have been limited in order to eliminate part of the ambiguity. See [22] as a historical paper, or more recently [15]). Business Rules Systems such as for example ILOG JRules[12] or Drools[5] are popular controlled languages used to explicitely model the business logic in a growing number of applications

We will not describe SBVR exhaustively in this article. However a look at the Figures 1 and 2 might provide a feeling about the sophistication of the meta-model and about the approach that separates logical formulations from meanings.

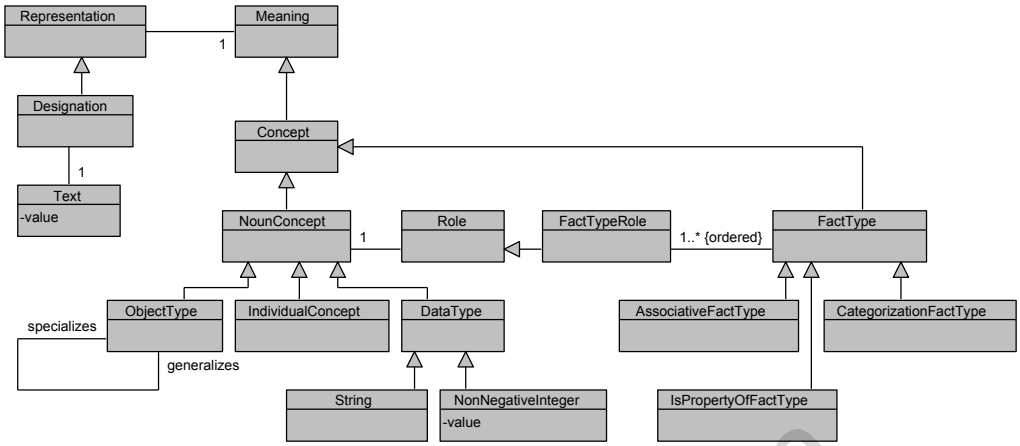

Fig. 1. Extract of the SBVR metamodel: meanings

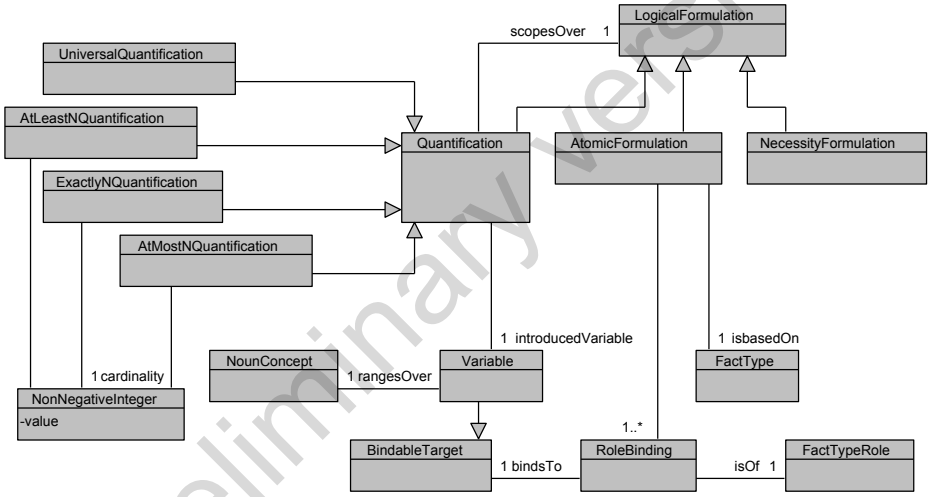

Fig. 2. Extract of the SBVR metamodel: logical formulations

2.2 Brief introduction to configuration

Configuring is the task of composing a complex system out of generic components [13]. Components, also called objects or model elements, are defined by their types, attributes, and known mutual relations. The acceptable systems are further constrained by the request: a set of problem-specific and/or user-specific requirements, represented by a fragment of the desired system (i.e. interconnected objects). From a knowledge representation perspective, configuration can be viewed as the problem of finding a graph (i.e. a set of connected objects)

obeying the restrictions of an object model under constraints. From a model driven approach, it can be viewed as the problem of finding a finite model that conforms to a metamodel. More precisely, we consider the process as a model transformation where the source model is the request and the target model is the solution. The configuration model acts as the target metamodel. A *relaxed* version of this metamodel acts as the source metamodel. This relaxed metamodel is obtained by the removal of all constraints: minimum cardinalities are set to zero, attributes are optionals and OCL constraints are removed. The request, which is a set of target model elements with incomplete knowledge (for instance linked elements and attribute values are usually undefined), is therefore conformant to the source metamodel. In this context, the configurator *searches* for a target model, completing the source and creating all necessary model elements so that the result (if any) is conformant to the target metamodel.

Various formalisms or technical approaches have been proposed to handle configuration problems: extensions of the Constraint Satisfaction Problem paradigm [18, 25, 20], knowledge-based approaches [24], logic programming [23], and object-oriented approaches [17, 14]. Configuration has traditionnaly been used with success in a number of industry applications such as manufacturing or software engineering. More recently, the expressive power of configuration formalisms has proven its usefulness for artificial intelligence tasks such as language parsing [7]. A deeper introduction to configuration can be found in [13].

In the sequel we propose to use Ilog JConfigurator [14] as a controlled language parser for SBVR. In this model-oriented approach, a configuration model (in our context, the target metamodel) is well-defined as a set of classes, relations and constraints. The UML/OCL language combination may be used to this purpose [8].

2.3 Brief introduction to MDE and model transformation

Model Driven Engineering is a research area that considers the main software artifacts as graphs. This comes from an industrial need to have a regular and homogeneous organization where different facets of a software system may be easily separated or combined.

In MDE, models are considered as the unifying concept. The MDE community has been using the concepts of terminal model, metamodel, and metametamodel for quite some time. A terminal model is a representation of a system. It captures some characteristics of the system and provides knowledge about it. MDE tools act on terminal models expressed in precise modeling languages. The abstract syntax of a modeling language, when expressed as a model, is called a metamodel. A language definition is given by an abstract syntax (a metamodel), one or more concrete syntaxes, and a definition of its semantics. The relation between a model expressed in a language and the metamodel of this language is called conformsTo. This should not be confused with the representationOf relation holding between a terminal model and the system it represents. Metamodels are in turn expressed in a modeling language called metamodeling language. Its

conceptual foundation is itself captured in a model called metametamodel. Terminal models, metamodels, and metametamodel form a three-level architecture with levels respectively named M1, M2, and M3. A formal definition of these concepts may be found in [10]. The principles of MDE may be implemented in several standards. For example, OMG proposes a standard metametamodel called Meta Object Facility (MOF).

The main way to automate MDE is by providing transformation facilities. The production of model Mb from model Ma by a transformation Mt is called a model transformation. In this work we use ATL (AtlanMod Transformation Language), a QVT-like model transformation language [11] allowing a declarative expression of a transformation by a set of rules.

2.4 Process overview

Figure 3 sketches the overall process in a model-driven engineering framework. The input is a text in the form of structured English as has been proposed in the SBVR specification [21]. The text is injected into a model thanks to a simple metamodel for sentences and words annotating the position of words in the text. A second simple transformation uses a lexicon to label each word with a set of possible syntactical categories. We have then defined a metamodel, called *Syntax*, where we adapted configuration grammars[7] to SBVR and the model-driven engineering context. The text (as labeled words) is fed into a constraint-based configurator using the relaxed version of Syntax. The result of the configuration process, acting as a syntax and grammar analysis, is a finite model that conforms to the Syntax metamodel. This model is then transformed with a usual model transformation tool (here, ATL) to a model conforming to the SBVR metamodel through a set of rules using the grammatical dependencies found during configuration. This SBVR terminal model may then be processed again with ATL to obtain a corresponding UML model.

2.5 Running example

An example will be used throughout the paper to illustrate the approach. The considered text is composed of three sentences defining a simple CS:
(1) Each company sells at least one product.
(2) Each product is sold by exactly one company.
(3) A software is a product.
This relatively simple example still embeds many important concepts. From the language parsing viewpoint it uses nouns, active and passive verbs, as well as different quantifiers. From the modelling viewpoint it shows the notions of classes, inheritance and relations with cardinalities. In the following Sections, we will show how each main task is applied on those sentences.

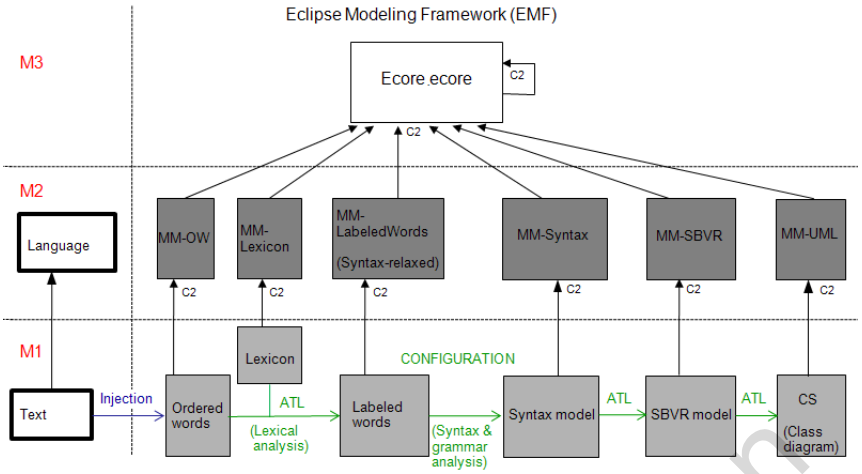

Fig. 3. Process overview

3 Parsing a controlled language for SBVR

Parsing natural languages is one of the major challenges of AI. Considering the difficulty of the task, many efforts have been focused on the more accessible field of controlled languages (CL) [16]. Among the existing approaches, [7] shows how property grammars [1] can be captured into a configuration model in order to parse a subset of French with a constraint-based configurator. The resulting parser does not inherit the deterministic behaviour of most CL parsers and is designed to be adapted to different grammars. We have modified and extended this method to the form of structured English proposed in the SBVR specification [21]. In our MDE approach, the proposed configuration model is defined as a metamodel called Syntax.

3.1 Syntax metamodel

A fragment of this metamodel (most classes and relations) is presented in Figure 4. Syntax captures three main informations from the input text: syntactical categories, grammatical dependencies and SBVR semantics.

Syntactical categorization In order to obtain a syntactic tree from a sentence, we have adapted the property grammar model from [7] to English. The main class of the model is *Cat*, which denotes a syntactical category. A category is *terminal* when it is directly associated to a single word. Such categories include *NCat* (noun), *VCat* (verb) or *DCat* (determiner). Those categories may be further specialized: a verb is either transitive (*TVCat*) or intransitive (*ITVCat*). The possible categories of a given word are obtained with the lexicon in the previous transformation. A category is *non-terminal* when it is composed of other

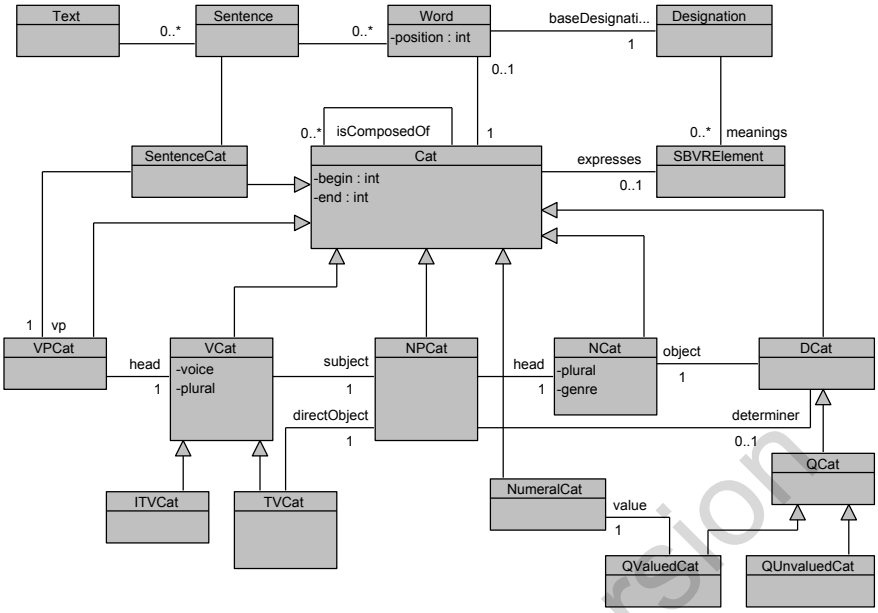

Fig. 4. Extract of the Syntax metamodel: syntax and grammar

categories. *SentenceCat* (sentence), *NPCat* (noun phrase), *VPCat* (verb phrase) are the main non-terminal categories. A set of constraints further defines the acceptable categorizations. Such constraints involve for instance the categories constituents, or relative positions in the sentence. Here are some example constraints specified in OCL:

– *Each verb phrase for which the heading verb is transitive is composed of at least one noun phrase.* This constraint applies to the relation *isComposedOf* of a category:

```
context VPCat
inv: head.oclIsTypeOf(TVCat)
     implies isComposedOf->
       exists( elt.oclIsTypeOf(NPCat) )
```

– *A verb phrase is always preceeded by a noun phrase.* The constraint applies to the attributes "begin" and "end" of categories, which are obtained from their associated word(s):

```
context SentenceCat
inv: isComposedOf->exists(
       elt.oclIsTypeOf(NPCat)
       and elt.end < vp.begin )
```

Grammatical dependencies We have extended the syntactic model so that grammatical dependencies appear as explicit relations between categories. Simi-

lary to the syntactical part, a set of constraints defines the acceptable constructions. Here are some example constraints specified in OCL:

– *The subject of an active verb occurs before the verb phrase*:

```
context VPCat
inv: (head.voice = 'active')
        implies head.subject.end < begin
```

– *The head of a verb's subject shares the same plural*:

```
context VCat
inv: plural = subject.head.plural
```

SBVR semantics We have extended the metamodel with the main concepts of the SBVR metamodel. SBVR semantics are assigned to syntactical categories through the "expresses" relation. Again, a set of constraints governs the possible assignments. Here are some examples:

– *A transitive verb expresses a fact type.*:

```
context TVCat
inv: not expresses.oclIsUndefined()
        and expresses.oclIsKindOf(FactType)
```

– *The head of a subject of a verb expresses either an object type or an individual concept*:

```
context VCat
inv: subject.head.expresses.
         oclIsKindOf(ObjectType)
        or subject.head.expresses.
         oclIsKindOf(IndividualConcept)
```

About SBVR concepts singularity A critical issue in assigning SBVR semantics to categories is the one of concepts singularity. More precisely, the same SBVR concept may be expressed in different sentences (or even in the same sentence). Consider for instance the first two sentences of our running example: the concepts "Company", "Product" and "To sell" are expressed multiple times. We obviously wish to avoid creating duplicate SBVR elements in the resulting model. A set of constraints forces the uniqueness of SBVR elements based on an equivalency statement. In the case of elements of class *ObjectType*, the disambiguation is done on the word's base designation. It can be formalized in OCL as follows:

```
inv: NCat.allInstances()->
        forAll(n1,n2 : NCat |
        (n1.word.baseDesignation =
         n2.word.baseDesignation)
        = (n1.expresses = n2.expresses))
```

Note that since the base designation is used, different forms of the same word are still recognized (i.e. "products" and "product" are matched). The same principle is applied to other SBVR elements such as fact types.

3.2 Parsing process and result

As explained previously, the input of the configuration process is a model of a relaxed version of the target metamodel. In our context, the input text is transformed into a set of interconnected configuration objects of type Text, Sentence, Words and Designations. For each word, the preceeding transformation, using the lexicon, has provided its properties (plural, voice, etc.), base designation (the base designation of the word "has" is "to have"), and candidate syntactical categories (the word "one" may be a noun, a numeral or an adjective).

The result of the configuration process is one (or more) terminal model(s) satisfying the configuration model constraints, when such a model exists. For instance, Figure 5 shows (a fragment of) the syntactical and grammatical part of a generated model for the sentence "Each company sells at least one product". It should be noted that this parsing process is not deterministic: due to language ambiguities, multiple solutions may be valid for the same request. For instance, consider the sentence "MyCode is a software". Without other sentences or lexicon information, it is not be possible to decide whether the *NounConcept* "MyCode" is an *ObjectType* (a specialization of Software) or an *IndividualConcept* (an instance of Software). Rather than arbitrary deciding on one model, search allows to generate all valid solutions which can be later compared, or even to optimize the target model based on some preferences. In this regard, our approach offers a high flexibility.

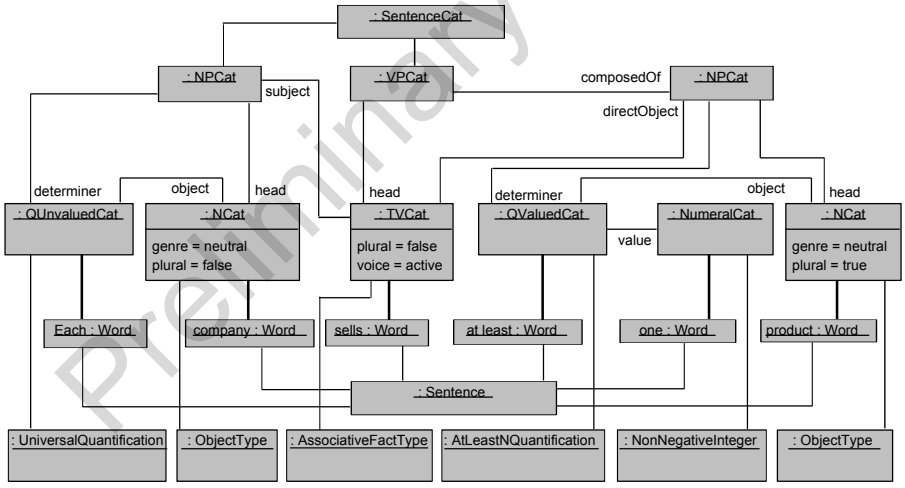

Fig. 5. Running example: fragment of a Syntax terminal model

4 Transforming the resulting model into a SBVR model

The model obtained during the parsing process exhibits the SBVR semantics expressed by (groups of) words. Using this information together with grammatical dependencies between elements, we are able to construct a complete SBVR model corresponding to the input text. This is achieved with model transformation using the ATL language [11]. Presenting each rule of the transformation in details is outside the scope of this article. However we propose an overview of its main principles.[1]

4.1 Mapping overview

A first straightforward mapping is obvious: each SBVRElement of the Syntax metamodel has its counterpart in the SBVR metamodel and therefore implies the creation of the target model element. However the relations between SBVR model elements are not exhibited in the source model and may require additional (intermediate) SBVR elements. A set of rules therefore allows to derive them from the grammatical relations of the source model.

As an example, consider the following rule that generates a (binary) AssociativeFactType and its roles from a transitive verb, its subject and direct object. It may be informally expressed as follows: "For an AssociativeFactType B expressed by a verb V in the source model, create an AssociativeFactType B' in the target model, with two roles R1 and R2, where R1's nounConcept is the target NounConcept of V's subject, and R2's nounConcept is the target NounConcept of V's direct object". The rule creates intermediate elements (the roles) and uses them to relate SBVR elements. Note that some of these elements (target NounConcepts of the subject and direct object) are created by a different rule.

Moreover, the transformation allows to create attribute values from a source information having a different datatype. Indeed, consider the word "one" in the first sentence of our running example. In the source model, the word is associated to the category "NumeralCat", expressing a non-negative integer in SBVR semantics. The rule that creates the target model element is able to assign a value to the attribute "value" of type "Integer", using the OCL construct "toInteger()" on the word's designation.

4.2 Transformation process and result

Once the transformation is complete, we obtain a model that conforms to SBVR, leaving aside the syntactical and grammatical information of the text. Figure 6 shows a fragment of the generated SBVR model for our running example, which corresponds to the sentence "Each company sells at least one product".

[1] Source code and documentation of all presented transformations have been submitted as a contribution to the Eclipse ATL project and are available on http://www.eclipse.org/m2m/atl/

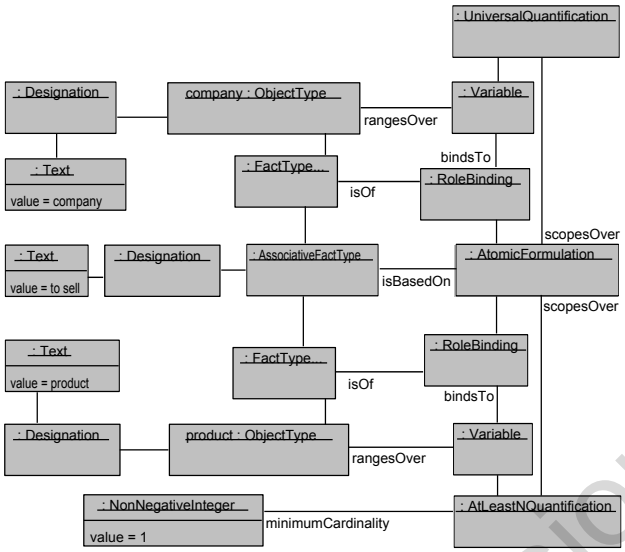

Fig. 6. Running example: fragment of the SBVR terminal model

5 Transforming the resulting model into a UML model

Once a valid SBVR model has been generated, it is possible to transform it into a corresponding UML (class diagram) model of the CS using ATL. Again, we do not detail each rule but present an overview of the mapping.

5.1 Mapping overview

Some examples of the mapped concepts are presented in Table 1 where the dotted notation is used to navigate classes attributes and relations. The correspondance between concepts is quite natural: an ObjectType becomes a Class, an AssociativeFactType becomes an Association, a CategorizationFactType denotes inheritance (Generalization in UML), an IsPropertyOfFactType refers to an attribute, an IndividualConcept becomes an InstanceSpecification, etc. Linked concepts and values are also quite explicit. However, most of the rules do not realize a straight one-to-one mapping but imply additional conditions, target elements from other rules, etc. For instance, consider the mapping for the SBVR concept *AtLeastNQuantification*. The *Property* for which *lowerValue* is assigned is the one obtained by transforming the *AssociativeFactType* that is target of the relation *AtLeastNQuantification.scopesOver.isBasedOn*. Ordered relations also play an important role: the first role of a categorization denotes the general class, whereas the second one refers to the specific class.

Table 1. Excerpt of the mapping from SBVR concepts to UML concepts

SBVR concept	UML Concept
ObjectType	Class
ObjectType.Designation.Text.value	Class.name
DataType	DataType
IndividualConcept	InstanceSpecification
AssociativeFactType	Association
AssociativeFactType.Designation.Text.value	Association.name
AssociativeFactType.FactTypeRole	Property (Association.memberEnd)
AssociativeFactType.FactTypeRole.nounConcept	Property.classifier
CategorizationFactType	Generalization
CategorizationFactType.FactTypeRole#1	Generalization.general
AtLeastNQuantification.minimumCardinality.value	Property.lowerValue
AtMostNQuantification.maximumCardinality.value	Property.upperValue

5.2 Transformation process and result

Once the transformation is complete, we obtain a UML specification of the CS. Figure 7 shows a UML terminal model obtained for our running example.

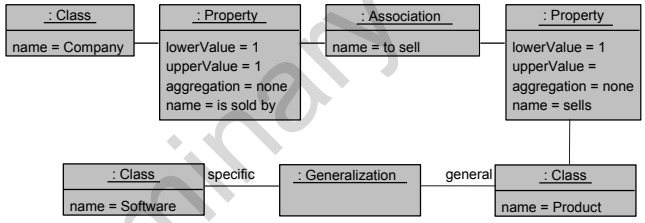

Fig. 7. Running example: fragment of the UML model

6 Implementation and experiments

6.1 Implementation

The proposed approach has been integrated into an Eclipse-based model-driven engineering framework. To those aims, each presented metamodel has been written using the KM3 metamodelling language [10], which offers an automatic conversion to the EMF's ECore format [6]. These ECore metamodels are source and target metamodels of the proposed ATL transformations. The configuration tool JConfigurator has its own modelling language and currently offers only XML inputs and outputs. Therefore the Syntax metamodel is also defined directly within

the tool as the configuration model. At runtime, the configuration request model is projected to XML in order to be parsed, and the XML representation of the solution is then injected into a model. This model (in XMI format) is passed over to the remaining ATL transformations.

6.2 Experiments

Experiments on the running example were conducted on a Core2Duo 3Ghz with 3GB of RAM. Table 2 shows the results. We first parsed each sentence separatly and then multiple sentences at once.

Parsing is efficient for separated sentences but the time required for the search

Table 2. Experiments on the running example (times in seconds)

`sentence(s)`	Text to Syntax			*Syntax to SBVR*	*SBVR to UML*
	Time	*Vars*	*Constraints*	*Time*	*Time*
(1)	0.26	527	1025	0.018	0.018
(2)	0.20	526	1022	0.018	0.016
(3)	0.19	475	885	0.015	0.018
(1)+(2)	0.38	973	2819	0.035	0.020
(1)+(2)+(3)	1.41	1328	5312	0.082	0.025

task quickly increases with the whole text. This is due to the size of the source model which directly impacts the search space of the configurator whereas the ATL transformations are able to handle bigger models. Splitting the tasks is positive for performance and could be investigated further so as to reduce the configuration model to the minimum required for syntactical analysis. We currently focused on a straightforward integration of the configurator with a configuration model covering the whole Syntax metamodel. Future steps are, on the one hand, to extract the combinatorial core of the metamodel, and on the other hand, to allow a further specified request through different relaxation levels of the source metamodel. It should however be emphasized that these are early experimental results. No optimizations have been applied to the configuration engine such as heuristics or symmetry breaking techniques, which are known to drastically reduce the computation times. Another alternative envisionned is to perform an incremental parsing of the text, sentence by sentence, using the ATL multiple source capabilities to unify the resulting SBVR models. The successful parsing of our non-trivial example however proves the feasability of the approach.

7 Related work

In [3], a procedure for performing the reverse transformation is described: from a UML/OCL description of a CS, the authors show how it can be transformed

into a SBVR terminal model using ATL, and then paraphrased with structured English text. Combining the two approaches is thus promising. Indeed, designing a CS often requires several discussions between stakeholders for refinements, and maintaining a CS leads to frequent evolutions. The combination would allow to switch from one representation to another automatically.

There has been previous research on using constraint programming technics in MDE, mostly about animation of relational specifications or model verification. [2] transforms UML/OCL specifications into a constraint satisfaction problem in order to check satisfiability. [4] proposes a similar method, although the underlying solver (Alloy [9]) is based on SAT. Both approaches inherit the limitations of the target formalism in which specifications must be translated. The configuration paradigm is expressive enough as self and thus avoids the translation phase. Moreover, to the best of our knowledge, this is the first time that a constraint-based search is embedded in MDE as a transformation tool.
With respect to the domain of natural or controlled language parsers, our approach differs from most existing methods (such as ACE[22]). Indeed these parsers do not accept ambiguous grammars (i.e. not context-free), whereas we are able to parameterize the level of accepted ambiguities, thus allowing to define a trade-off between language coverage and computation efficiency.

8 Conclusion and future work

We have described a method which allows to parse a CS specification expressed in structured English into a UML class diagram. To those aims, we proposed SBVR as an intermediate layer. The originality of our approach is the use of an advanced constraint-programming search technique (configuration) as a model transformation tool integrated in a MDE environment. Early experiments are provided as a proof-of-concept. Moreover, the proposed parser is flexible with respect to language coverage and disambiguation. There are many perspectives to this work. First, the metamodels can be extended to capture an increased portion of SBVR. The expressed meanings will then probably require to generate OCL constraints in addition to UML. Other target formalisms can also be considered such as OWL or Rule Systems. The experiments clearly show that there is a need for performance improvement in the search-based transformation. The leading direction is to reduce the search space by isolating the metamodel's combinatorial core, thus further decomposing the problem. Finally, the described configuration-based tool could benefit to other complex transformations that require searching for a target model instead of applying deterministic rules to the source model.

References

1. Philippe Blache and Jean-Marie Balfourier. Property grammars: a flexible constraint-based approach to parsing. In *IWPT*. Tsinghua University Press, 2001.

2. Jordi Cabot, Robert Clarisó, and Daniel Riera. Umltocsp: a tool for the formal verification of uml/ocl models using constraint programming. In *ASE*, pages 547–548. ACM, 2007.
3. Jordi Cabot, Raquel Pau, and Ruth Ravents. From uml/ocl to sbvr specifications: a challenging transformation. *Information Systems Elsevier*, 2009.
4. Trung T. Dinh-Trong, Sudipto Ghosh, and Robert B. France. A systematic approach to generate inputs to test uml design models. In *ISSRE*, pages 95–104. IEEE Computer Society, 2006.
5. *Drools : http://www.jboss.org/drools/*, 2009.
6. *EMF : http://www.eclipse.org/modeling/emf/*, 2009.
7. Mathieu Estratat and Laurent Henocque. Parsing languages with a configurator. In *Proceedings of the European Conference for Artificial Intelligence ECAI'2004*, pages 591–595, August 2004.
8. Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Configuration knowledge representation using uml/ocl. In *UML*, volume 2460 of *LNCS*, pages 49–62. Springer, 2002.
9. Daniel Jackson. Automating first-order relational logic. In *SIGSOFT FSE*, pages 130–139, 2000.
10. Frédéric Jouault and Jean Bézivin. Km3: A dsl for metamodel specification. In *FMOODS*, volume 4037 of *LNCS*, pages 171–185. Springer, 2006.
11. Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In *MoDELS Satellite Events*, volume 3844 of *LNCS*, pages 128–138. Springer, 2005.
12. *JRules : http://www.ilog.fr/products/jrules/*, 2009.
13. Ulrich Junker. *Configuration*, volume Handbook of Constraint Programming, chapter 26. Elsevier, 2006.
14. Ulrich Junker and Daniel Mailharro. The logic of (j)configurator : Combining constraint programming with a description logic. In *IJCAI'03*. Springer, 2003.
15. Kaarel Kaljurand. Ace view - an ontology and rule editor based on controlled english. In *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
16. R. I. Kittredge. *Sublanguages and controlled languages*. Oxford Press, 2003.
17. Daniel Mailharro. A classification and constraint-based framework for configuration. *AI in Engineering, Design and Manufacturing, (12)*, pages 383–397, 1998.
18. Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, pages 25–32, 1990.
19. Object Management Group. *Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.0*, 2008.
20. Daniel Sabin and Eugene C. Freuder. Composite constraint satisfaction. In *AI and Manufacturing Research Planning Workshop*, pages 153–161, 1996.
21. *Semantics of Business Vocabulary and Business Rules (SBVR) 1.0 specification: http://www.omg.org/spec/SBVR/1.0/*, 2008.
22. Rolf Schwitter and Norbert E. Fuchs. Attempto controlled english (ace) a seemingly informal bridgehead in formal territory. In *JICSLP*, page 536, 1996.
23. Timo Soininen, Ilkka Niemela, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring Symp. on Answer Set Programming*, pages 195–201, 2001.
24. Markus Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2):111–125, June 1997.
25. Markus Stumptner and Alois Haselböck. A generative constraint formalism for configuration problems. In *Advances in Artificial Intelligence: Proceedings of AI*IA'93*, pages 302–313. Springer, 1993.