Abstract. The future vision of business information systems rely on semantic technologies as the Web Ontology Language OWL (currently, OWL2) that defines a meaning of business concepts and make them unambiguously understandable by human experts and software systems. Nevertheless, the spread of these technologies is observed mainly in the scientific papers and in some special domains, but not in everyday business operations. The reason for this is the fact that OWL is a language for information technology experts but not for business people – the actual managers of business policies and rules. From the other side, the Semantics of Business Vocabulary and Business Rules (SBVR), one of the recent OMG specifications, provides a means for describing semantic formulations of business concepts and business rules in a language that business people use. Currently some researchers ground their works on assumptions that formal logic-based SBVR formulations are transformable into OWL. However, an exhaustive study how these transformations could be built still is lacking. OWL2 presents new capabilities for linking two worlds. The goal of the paper is to introduce main concepts and problems in transforming SBVR into OWL2 by extending the relevant information from original SBVR specification and related works.

Keywords: Semantics of Business Vocabulary and Business Rules, Web Ontology Language, transformation, semantic technologies, SBVR, OWL2.

1 Introduction

Semantic technologies give a possibility for computer systems and human experts to understand and share semantics in a real time, thus presenting new capabilities for information processing. Unfortunately, these technologies are not mature yet to the certain level that we could notice significant benefits of them in everyday business operations. One of the problems for spreading these technologies is that they are based on sophisticated tools and languages as Web Ontology Language OWL, currently, OWL2 [10], [18] that are too difficult for business users. Therefore, ontology development is the main responsibility of information technology staff.

Object Management Group (OMG) has created the Semantics of Business Vocabulary and Business Rules (SBVR) metamodel [12] that provides opportunity to describe business concepts and business rules using so called Controlled Natural Language (CNL), which is understandable for business users. With the SBVR, domain experts are able to construct business vocabularies and business rules, or at least understand them for validation purposes. SBVR is based on formal logics and can be applied for computer processing, but it cannot be directly used in semantic technologies because these have their own languages as OWL2 or RDFS. Therefore, we are investigating a possibility of transforming SBVR into OWL2 for allowing business users to describe ontologies using the language similar to their everyday business language. That allows to prove consistency of company’s business vocabulary and rules by using OWL2 reasoners, and to gain other benefits.

Currently, some researchers ground their works on assumptions that formal logic-based SBVR formulations are transformable into OWL. However, an exhaustive study how these transformations could be built still is lacking. As we will see, some mere SBVR constructions are not easy comparable with OWL. Besides, existing works concern OWL, while OWL2 provides more capabilities for linking the both worlds. Therefore, the goal of the paper is to introduce main concepts in transforming SBVR into OWL2 by extending the relevant information from SBVR specification and related works. This task will gain the particularly importance when application of business vocabularies will become available for the wide range of business users.

The rest of the paper is structured as follows. In section 2, we analyze related works. Sections 3 and 4 shortly present main concepts of SBVR and OWL2. Section 5 is devoted to considering transformations from main SBVR concepts into OWL2. Section 5 presents conclusions and future works.

2 Related works

The vision of filling the gap between semantic technologies and business experts is to use CNL for authoring ontologies and then transforming them to Web Ontology Language. For developing such CNL, there are two slightly conflicting requirements: the need to see OWL in the form of natural language, and the need to make a straightforward mapping to and from OWL. Different scientific groups are developing their own CNLs,
but none of them is widely accepted. E.g. we can mention First-Version Controlled English Rule Language [19], Attempto Controlled English (ACE) [7], Rabbit and Sydney OWL Syntax (SOS) whose comparison in [13] ends with conclusion that none of these languages is suitable for semantic representations equally acceptable to business users and computing.

Semantics of Business Rules and Business Vocabulary (SBVR), adopted in 2008, was the first OMG specification created with intention to incorporate natural languages in modeling. SBVR is an integral part of the OMG’s Model Driven Architecture (MDA) [5] and uses SBVR structured English (SSE) as one of possible CNLs. SBVR is the most matured specification of semantics, but it also has limitations [15]: facts may be inferred, but SBVR does not standardize inference; no references to discourse (rules are stated in one sentence); no free variables in a logical formulas; necessity to introduce concepts before referencing to them; impossibility to express directives, no meaning for past tense or future tense etc. The lack of advanced SBVR editors capable parsing SBVR SSE style texts also is a problem. These circumstances prevent the wider usage of SBVR.

The possibility and necessity to fill the gap between SBVR and semantic technologies was noticed by many authors [2], [3], [8], [16]. For example, Demuth and Liebau [3] suggested a translation from SBVR to OWL and REWERSE Rule Markup Language (R2ML), however, no further research in this direction was made. Ceravolo et al. [2] analyzed possibility to translate SBVR specification into the OWL+SWRL knowledge base with intention to check business rules consistency. These researches have shown that SBVR2OWL transformations are desirable and would gain wide applicability, however, currently they are under construction.

SBVR can serve many purposes: ensuring semantic interoperability between distributed information systems [8]; engineering information systems [11], [14] or verbalizing software models [1]; often it is related with service oriented systems [9]. The largest advantages in implementing SBVR seem achieved in commercial Collibra tool suite for Business Semantics Management*. Collibra presents capabilities for authoring SBVR vocabularies and rules, generate ontologies and various models of information systems. It has inherited experience from several EU projects, e.g. OPAALS [9]. Related research has started in ONTORULE project**, which purpose is extraction of SBVR business rules from different sources including texts in natural language, to manage them and implement in software applications. The research group of Kaunas University of Technology needs SBVR2OWL2 transformation for the further development of the VeTIS tool [11] (also, you can see the paper by Sukys et al. in current IT'2011 proceedings). Here we faced with real problems; a part of them are under discussion in the literature, the another part is not mentioned at all. So we are trying to find concrete solutions that would allow widening the usage of SBVR for authoring OWL2 ontologies and making the bridge between user friendly structured languages and applications of semantic technologies.

3 Main concepts of SBVR

The peculiarity of SBVR metamodel is the explicit separation of meaning, representation and symbolization: the same meaning can have many representations and the same expression can represent different meanings (Figure 1).

![Figure 1. Subtypes of meaning in SBVR metamodel [12]](image)

SBVR meaning is defined as “what is meant by a word, sign, statement, or description; what someone intends to express or what someone understands”. Subtypes of meaning are concepts (object types, roles, fact types, fact type roles, individual concepts), questions and propositions. Meaning corresponds to thing that is understood as “anything perceivable or conceivable” [12], and every other concept implicitly specializes the concept „thing“ similarly to OWL2 where “Thing” is a supertype of all OWL2 classes. SBVR specification [12]

* http://www.collibra.com/
** http://ontorule-project.eu/
suggests matching of some main SBVR concepts to OWL concepts, but this list is incomplete. OWL directly covers only part of the SBVR as the ability to represent rules in the Web Ontology Language is very limited. The Semantic Web Rule Language (SWRL), combining OWL and RuleML, and Rule Interchange Format (RIF) are frequently mentioned as a solution for adding some rules to the Web Ontology Language.

4 Main concepts of OWL2

The Web Ontology Language OWL2 is a family of languages for authoring ontologies. The previous OWL specification included the definition of three sublanguages with different levels of expressiveness: OWL Lite, OWL DL and OWL Full. In the rest of the paper, when mentioning OWL2, we will assume mainly OWL2 DL, because OWL2 DL reasoning supports the maximum expressiveness, computational completeness and decidability. OWL DL has some restrictions, e.g. a class cannot also be an individual or a property what OWL Full allows. Main OWL2 concepts corresponding to the part of SBVR are presented in Figure 2 and Figure 3. OWL2 ontology consists of axioms. The main concept of the OWL2 ontology metamodel is OWL2 Class, which is a subclass of the ClassExpression (Figure 2).

![Figure 2. Subclasses of Axioms, Entities and Individuals in OWL2 metamodel [10]](image)

![Figure 3. Subclasses of OWL2 ClassExpressions in OWL2 metamodel [10]](image)

Ontology and OWL2 entities (Classes, ObjectProperties, DataProperties, Datatypes and NamedIndividuals) are identified by IRI. The SubClassOf axiom states that each instance of one class expression is also an instance of another class. The EquivalentClasses axiom states that all class expressions \( CE_i, 1 \leq i \leq n \) in EquivalentClasses \( (CE_1 \ldots CE_n) \) are semantically equivalent to each other. This axiom allows one to use each \( CE_i \) as a synonym for each \( CE_j \) — that is, \( CE_i \) can be replaced with \( CE_j \) without affecting the meaning of the
The `DisjointClasses` axiom states that several class expressions are pairwise disjoint (have no common instances). The `DisjointUnion` class expression defines a class as a disjoint union of several class expressions and thus allows expressing covering constraints. `ClassExpression` subclasses are presented in Figure 3. They define various restrictions on `ObjectPropertyExpressions` and propositional connections that are allowed between `ClassExpressions` or `Individuals`. These expressions mainly have their counterparts in SBVR.

5 Transformation of main SBVR concepts into OWL2

5.1 SBVR object type

SBVR object type is defined as a noun concept that classifies things on the basis of their common properties. It corresponds to OWL2 class where classes can be understood as sets of individuals. SBVR meaning can have many representations (Figure 4) that are treated as synonyms for noun concepts and synonymous forms for fact types. This contrasts with OWL2 where meaning and representation are not separated. SBVR metamodel has no explicit elements for synonyms and synonymous forms: both are implied by multiple representations of the same meaning. Unusually, SBVR business vocabulary entry introduces the primary representation of meaning, and this primary representation also is the preferred designation of that meaning [12]. Additionally, a meaning can have prohibited or not-preferred designations that are included in the vocabulary entry as synonyms or synonymous forms. If a vocabulary entry is not a preferred designation, then the preferred designation is referenced under the caption “See:” All synonyms (or synonymous forms) of the same meaning share the same definition.

SBVR object type having exactly one (preferred) designation, should be transformed into OWL2 `Class` and the expression value of that designation should be transformed into `Class IRI`. Otherwise, SBVR object type is transformed into several classes − synonyms that are defined by OWL2 `EquivalentClasses` axiom [10]. We create OWL2 `Class` for each synonymous noun concept and constrain these class expressions with `EquivalentClasses` axiom, where `Class CE1` corresponding to the preferred designation is supplemented with `Annotation (Comment)` “Preferred_designation” (Figure 5).

OWL2 does not distinguish preferred designation between synonyms − this possibility is essential going from ontology or SBVR vocabularies and rules towards software applications used in everyday enterprise activities. We propose supplementing `Classes CEi`, corresponding to prohibited or not preferred designations, with comments “Prohibited_designation” or “Not_preferred_designation”. SBVR definition, general concept, concept type and other items of SBVR Vocabulary entry are transformed into OWL2 concepts related with class commented as `Preferred_designation`. Fact types and formulations of business rules can be transformed into OWL2 concepts related with prohibited or not-preferred equivalent classes representing synonyms. The presented way of distinguishing preferred, prohibited and not preferred designations is also suitable for object and data properties.
5.2 SBVR fact type

SBVR fact type denotes some type of relationship between noun concepts (object types) or is a characteristic of the noun concept. We will consider associative fact type, “is_property_of” fact type, partitive fact type, categorization fact type, assortment fact type and characteristic.

The associative fact type is a fact type having one or more roles and representing a relation between noun concepts is the most common fact type. The associative fact type having two fact type roles is transformed into OWL2 ObjectProperty (Figure 6) where the first fact type role is transformed into the domain Class, and the second fact type role – into the range Class of the corresponding ObjectProperty. The fact symbol of the preferred designation of the fact type form is transformed into the IRI of OWL2 ObjectProperty.

![Figure 6. Transformation of the SBVR associative fact type with two fact type roles into OWL2 ObjectProperty](image)

Fact type role. In Figure 6 we assume that SBVR fact type role coincides with the object type which is playing that role (a journal plays the role of the journal in the fact type “journal is referred in database”). However, fact type roles often are separate concept types, e.g. “journal is published by publisher” where fact type role “publisher” ranges over the object type “organization”. Representing SBVR fact type roles is cumbersome in OWL2 because ontology roles rather mean SBVR associative fact types than SBVR fact type roles (or roles in UML). For example, we can simply formulate fact “A journal ITC is published by publisher KTU” having facts “ITC is journal”, “KTU is organization” in SBVR vocabulary or in UML (Figure 7) but it is not so easy in OWL2.

![Figure 7. Example of representing role „publisher" in SBVR and UML](image)

Representing fact type role as OWL Class CE1 being subclass of OWL2 Class CE2 that represents the corresponding object type is not a good solution [4]. We solve this problem by using OWL2 SubObjectPropertyOf and ObjectPropertyChain. SubObjectPropertyOf(ObjectPropertyChain (OPE1 ... OPEn)) axiom states that, if an individual x is connected by a sequence of object property expressions OPE1, ..., OPEn with an individual y, then x is also connected with y by the object property expression OPE. In our case, for representing role “publisher” we should transform SBVR vocabulary on Figure 7 into OWL ontology (Figure 8) having the following axioms:

```
SubObjectPropertyOf( ObjectPropertyChain (is_published_by o is_role_of ) publisher ),
SubObjectPropertyOf( is_published_by publisher )
```

![Figure 8. Example of representing role “publisher” in OWL2](image)

When we specify instances „KTU“ of type „Organization“, „ITC“ of type „Journal“ and object property „ITC is published by KTU“, ontology reasoner derives object property “publisher” between ITC and KTU. However, ITC property assertion “publisher KTU” not fits well with OWL2 style. Hoekstra [6] proposes more sophisticated solution for representing roles in OWL, but he uses OWL Punning that is restricted in OWL DL.

Is_property_of fact type defines an essential quality or a characteristic of a given noun concept and is identified by the verb phrase “has” (the passive form “is_property_of”). In the “is_property_of” fact type, the first fact type role ranges over an object type that is an elementary concept (number, integer or text). The “is_property_of” fact type is transformed into OWL DataProperty where the first fact type role is transformed
into the domain (OWL2 Class) and the second fact type role – into the range (OWL2 Datatype) of the corresponding DataProperty. The value of designation’s expression of the first fact type role is transformed into the value of DataProperty’s IRI.

\[
\text{journal has name} \\
\text{impact factor}
\]

Figure 9. SBVR is_property_of fact type transformation into OWL2 DataProperty

**Specialization (categorization) fact type** is a fact type that represents relationship between two object types (noun concepts): the more general one and more specific another. The more specific noun concept is a category of the first (more general) one. In consequence, SBVR categorization fact type could be transformed to the SubClassOf OWL2 concept (Figure 10).

\[
\text{organization} \\
\text{is in} \\
\text{city}
\]

Figure 10. Categorization fact type transformation into OWL SubClassOf hierarchy

**N-ary associative fact types** having more than two fact type roles (require additional consideration, because they cannot be directly represented in OWL2. There are some proposals for possible representations of n-ary associations in OWL2 on the base of reification when a new class is introduced for representing the relation instead of a property [17]. However, such a way seems unnatural in many cases. Hoekstra in [6] states that the disallowance of n-ary associations is rather an advantage than the drawback of the OWL2. He claims that n-ary associations always can be expressed by binary associations, and formulations of n-ary associations only demonstrate that the meaning of the corresponding associations is not well-understood. Similarly, in [11] it was assumed that SBVR associative fact types have at most two fact type roles. W3 Consortium suggests several solutions to solve or pass-thru this problem [17]: introducing a new class for a relation or using lists for arguments in a relation. These proposals are applicable to OWL2, UML or ER languages but they do not solve the problem when queries or business rules must be constructed using controlled natural language because such constructions sometimes will appear unnatural. Another possible solution is to represent some fact type roles of SBVR n-ary fact types as properties of OWL2 properties. OWL2 lets us declare properties and classes with the same name (it is called *Punning* in OWL2 specification). We will consider these possibilities more deeply in the future.

### 5.3 Transformation of SBVR quantifications

Almost all SBVR quantifications can be directly transformed into OWL restrictions. “universal quantification” \(\forall x\) means that the meaning, formulated by the logical formulation of \(x\), for each instance of \(x\) is true. It could be transformed into OWL restriction, owl:allValuesFrom (ObjectAllValuesFrom or DataAllValuesfrom). “existential quantification” \(\exists x\) means, that there is at least one \(x\), so this is a special case of the “at-least-n quantification, where \(n=1\). This quantification could be transformed into the OWL2 owl:minCardinality 1. Another possibility is to render existential quantification as owl:someValuesFrom restriction. Both cases are correct from the logical point of view.

\[
\text{organization} \\
\text{city} \\
\text{is in} \\
\text{city}
\]

Figure 11. Example of SBVR “exactly-n quantification“ transformation into OWL2

“exactly-n quantification” \(\exists^n x\) could be transformed into the OWL2 restriction owl:cardinality \(n\) (ObjectExactCardinality or DataExactCardinality elements of OWL2 metamodel). “exactly-one quantification” \(\exists^1 x\) is a special case of the exactly-n quantification, where \(n=1\) (Figure 11). Alternatively, it can be represented by FunctionalObjectProperty or FunctionalDataProperty axioms. At-least-n quantification \(\exists^n x\), where \(n=1, 2, 3, \ldots (n \geq 1)\) means that there is at least \(n\) objects or properties marked as \(x\). It could be expressed as OWL cardinality restriction: owl:minCardinality \(n\) (ObjectMinCardinality or DataMinCardinality) (Figure 12).
It is necessary that the organization employs at least one employee.

Figure 12. Example of SBVR “at-least-n quantification” transformation into OWL2

“at-most-n quantification” $\exists 0..n x$, where $n = 1, 2, 3, \ldots (n \geq 1)$ restricts the maximum number of instances or properties. It can be transformed to $\text{owl:maxCardinality } n$ (ObjectMaxCardinality or DataMaxCardinality from OWL metamodel). “at-most-one quantification” $\exists 0..1 x$ is a special case (subclass) of at-most-n quantification, where $n = 1$, so it is rendered as $\text{owl:maxCardinality } 1$.

5.4 Transformation of SBVR categorization schemas and segmentations into OWL2

In practice, object types often have several generalization hierarchies (generalization sets in UML) on the base of different criteria (power types in UML). In SBVR, these generalization sets are represented by categorization schemas (corresponding to {disjoint, incomplete} generalizations) and segmentations in SBVR (corresponding to {disjoint, complete} ones) (Figure 13). Unaccountably, multiple generalizations are insufficiently addressed in ontology related literature. SBVR categorization schema corresponds to OWL2 ObjectUnionOf. OWL2 DisjointUnion can cover SBVR segmentation (Figure 14). Several categorization schemas (segmentations) of the same OWL2 Class are defined by the EquivalentClasses axiom.

Figure 13. SBVR segmentation

- 237 -
6 Conclusions and future works

SBVR is the most matured semantic representation that can use SBVR Structured English or other CNL for ontology development. Although OMG specification declares that SBVR is compatible with OWL, an exhaustive study how transformations from SBVR into OWL could be built still is lacking. Currently, OWL is superseded by OWL2 that provides more capabilities for expressing semantics of business domain.

The analysis of the related research has shown that while SBVR2OWL transformation is considered as straightforward elsewhere, it is true only for main concepts of SBVR but even there are some primary questions unanswered yet. Solutions to some of these primary questions – how to represent SBVR synonyms and synonymous forms, preferred designations, categorization schemas and segmentations, some (but incomplete) approximation to fact type roles, – were presented in the current paper. The further work will be directed towards including a more comprehensive subset of SBVR concepts that are worth to represent in OWL2, and implementing the corresponding transformations for accelerating employment of SBVR and OWL2 in practise.

References
CONCEPTUAL DESIGN PATTERNS FOR RELATIONAL DATABASES

Nicola Vitacolonna
University of Udine, Department of Mathematics and Computer Science,
via delle Scienze 206, 33100 Udine, Italy, nicola.vitacolonna@uniud.it

Abstract. Database design patterns (DDPs) at the semantic level are introduced. A definition of conceptual DDPs is provided and the issues of their representation, their independence from the implementation details and from the logical data model, as well as some behavioural aspects, are discussed. We argue in favor of conceptual DDPs as a mechanism of abstraction and synthesis, and as a way to improve the quality of a database design. Moreover, DDPs can be supported by rigorous mathematical theories (e.g., functional dependencies and normalisation). Two examples of DDPs (Multiple Roles, Hierarchy) are analysed in depth, which will also show how the Extended Entity-Relationship model can be enhanced to capture more integrity constraints.

Keywords: Databases, Design Patterns, Modelling methods.

1 Introduction
The goal of conceptual database design is to achieve a thorough understanding of a database system by deriving an expressive, yet abstract, specification of the data and their related integrity constraints, typically in the form of a graphical diagram obtained incrementally by mixed strategies. Design patterns, on the other hand, have become a conventional approach to object-oriented (OO) software analysis and design [13]. In data modelling, however, the relational model has become a widespread and mathematically sound framework, so that design techniques tend to gravitate around it. For this reason, we believe, the topic of design patterns for database development has been mostly discussed by the OO software engineering community (see for example [1, 15, 21]) and has received less attention in the data modelling community [9, 16]. Object-oriented design patterns for databases, though, suffer from the so-called “impedance mismatch” between the object-oriented framework and the relational model.

In this paper, we propose database design patterns (DDPs) for conceptual data modelling of relational databases. To our knowledge, no systematic or extensive research has been carried out in this direction, and no catalogue of such patterns exists yet. Designing patterns for semantic data models has a more direct impact to relational database design than OO-based approaches; besides, the quality assessment of a design can benefit from the rigorous mathematical foundation provided by the relational model. Other advantages of DDPs, widely recognized in the software engineering community, can be summarised as follows: they embody common sense and experience; they avoid repeating mistakes or discovering the same solutions over and over; they establish a common language for exchanging design ideas. Therefore, DDPs can help improve the database development process and the quality of the artifacts.

On a different level, DDPs are valuable for teaching, both in our experience and as reported in the literature [19]. Students tend to provide different solutions, even for relatively simple problems, and they may find it difficult to compare them. After letting students try to design a database and after commenting on their mistakes or clever solutions, it is useful to synthesise the design processes in the form of a DDP.

In recent years, conceptual models have been relevant in the design of many bioinformatics databases [3, 5, 22]. In fact, biological databases have posed new challenges for designers [10], because biological data have features typically not found in many business-oriented data. What is clearly understood in one domain may not be as clear in a different domain: this is one more reason why a toolset like conceptual DDPs may be beneficial.

2 Related Work
Most of the research work that is related to design patterns in relational databases focuses on the logical and physical level. The problem of selecting a database structure that is robust against different query patterns and changing workload has been studied in [4] and a framework for “self-tuning” the logical schema for query optimisation has been proposed in [7]. How some classical object-oriented design patterns (such as Command or Memento) occur both in database design and database implementation has been discussed by [19]. Some other work has focused on design patterns at the level of application development or the architectural framework [15, 24]. More recently, pattern-based design with the goal of automatic code generation has been investigated [9]. Other logical design patterns on top of which object-relational structures can be applied have been proposed in [23].
With respect to the above, our proposal places itself at a more abstract, semantic, level, since we concur with the point of view according to which modelling constraints during conceptual analysis is advantageous [20]. For a similar approach, see for example, the “library loan” design pattern in [16].

Other examples of database design patterns occur in data warehouses: they have been extensively investigated and they deal with special types of databases, hence they are not covered in the present paper.

3 Conceptual DDPs

A conceptual DDP is a partial specification of a conceptual schema or, equivalently, an abstraction for a family of semantic models. A conceptual DDP describes and emphasises some semantic properties of the data that are common to many concrete cases. It is, therefore, a sort of meta-model. The idea is that a DDP should highlight the salient features of a data model and omit all the rest. Similarly to design patterns in architecture or object-oriented software engineering, a conceptual DDP embodies the core of a model for a data pattern that occurs often and in different domains. In fact, a good DDP should possess the following features:

- it should not be domain-specific;
- it should somehow encode some form of non-obvious expertise;
- it should not be too complex (e.g., a complete data architecture), to maximise the potential for reuse;
- it should describe in an abstract way the elements that form an appropriate design for a given problem.

Conceptual modelling typically makes use of graphical notations, such as EER (Extended Entity-Relationship) diagrams [6, 11], ORM (Object-Role Modelling) diagrams [14], UML [2], and others. UML does not (yet) provide a standard for data modelling—in fact, no such standard exists, so that the choice for the notation to use is somewhat arbitrary. In the following, we will use EER diagrams, as in our opinion they provide a good balance between expressiveness and simplicity. It is worth noting, however, that the choice of the semantic model is not essential and that other conceptual data models may be used for the purpose of devising patterns.

The notation we use for EER diagrams is summarised in Figure 1 and it is essentially an extension of the original proposal [6]. The minimum and maximum cardinality with which an entity participates to a relationship are shown as pairs \((m,M)\) on the edge connecting the entity type to the relationship type. According to the maximum cardinalities on the edges adjacent to a relationship type, we distinguish among one-one (1:1), one-many (1:N) or many-many (N:N) relationships. Other constructs (derived attributes, relationship types of degree greater than two, etc.) are omitted, as they are not used in this paper. The graphical syntax includes a construct for aggregation, which represents a relationship type that behaves at the same time as an entity type. Aggregation does not increase the expressive power of EER diagrams, as, in general, it is equivalent to a weak entity type that is identified by means of two other entity types (Figure 2). Nonetheless, from the point of view of conceptual modelling, it conveys a peculiar semantic nuance and it is a convenient shorthand.

EER diagrams are intuitive representations, but they take up much space. Therefore, in the following we will also use the following equivalent, though less appealing, textual notation:

- \(A(a_0, a_1, \ldots, a_k)\) denotes an entity type \(A\) with identifier \(a_0\) and additional attributes \(a_1, \ldots, a_k\).
- \(R(A(m_a, M_a), B(m_b, M_b))\) denotes a binary relationship type \(R\) that relates entities \(A\) and \(B\); \(A\) (resp., \(B\)) participates to \(R\) with minimum cardinality \(m_a\) and maximum cardinality \(M_a\) (resp., \(m_b\) and \(M_b\));
- \(A(a_0, \ldots, a_k) \rightarrow \{B_1, \ldots, B_l\}\) denotes that \(A\) is a specialisation of \(B_1, \ldots, B_l\) with additional attributes \(a_0, \ldots, a_k\); we will write \(A(a_0, \ldots, a_k) \Rightarrow B\) when \(A\) has only one parent.
- \(A(a_0, a_1, \ldots, a_k) \Rightarrow \{B_1, \ldots, B_l\}\) denotes that \(A\) is a weak entity type identified by \(B_1, \ldots, B_l\), with (optional) semi-identifier \(a_0\) and with additional attributes \(a_1, \ldots, a_k\); we will write \(A(a_0, \ldots, a_k) \Rightarrow B\) when \(A\) is identified by a single entity type \(B\).
- \((G)(A(m_a, M_a), B(m_b, M_b))\) denotes an aggregation of \(A\) and \(B\) with cardinalities defined as for relationships. An aggregation may also appear in the previous notations wherever an entity type occurs.

Arguments in parentheses may be omitted when they are clear from the context or irrelevant.

With abuse of terminology, in what follows we will use the term “entity” to equivalently denote an entity, an entity set or an entity type, and similarly for “relationship”, as the exact meaning will be clear from the context. Besides, from now on, by DDP we mean a conceptual DDP in the EER model. A DDP is therefore made of entities and relationships. Attributes are specified only if they are essential to the DDP; they are otherwise left implicit.

Cardinalities, too, are shown only when needed, otherwise they are replaced by a dash or omitted entirely. Other integrity constraints are also part of the definition of a DDP: when they cannot be represented in the diagram

\* Note that this is not the same as UML aggregation.
itself, they are specified in some other way. In the examples described later, such constraints are so intuitive that we do not think it necessary to define a formal language for their specification, although a language similar to predicate calculus may be defined for the purpose.

Together with the structural properties, conceptual DDPs should provide information about the most common transactions supported by the pattern. A DDP supports a transaction if that transaction can be executed without violating any integrity constraint. For simplicity, we consider only the following atomic conceptual transactions:

- \( \text{insertEntity}(A(v_0, \ldots, v_k)) \) - insert an entity whose attributes have the specified values;
- \( \text{insertRelationship}(R(A(v_0), B(w_0))) \) - insert a relationship between the entity \( A \) with identifier value \( v_0 \) and the entity \( B \) with identifier value \( w_0 \);
- \( \text{delete}(X(v)) \) - delete an entity or relationship \( X \) identified by \( v \) (optionally, in cascade).

In principle, a pattern should be designed so that the associated transactions not only respect all the integrity constraints, but they are also executed as efficiently as possible by an implementation. It is not possible, however, to provide such a guarantee without making strong assumptions about the underlying logical and physical levels. A good design pattern, however, typically results in a good logical design, which in turn facilitates physical optimisation. But we remark that, in the present work, we mainly focus on the foremost matter of the consistence of the integrity constraints.

Finally, DDPs may be broadly categorised into query-oriented or transaction-oriented, depending on whether they are more suitable for retrieving information (e.g., a star-schema) or for dynamic updates. Of course, some patterns may be both or provide different versions for both.

## 4 Assessing the quality of a DDP

Even with simple requirements, there are often several solutions for a modelling problem. How do we select a DDP out of many choices? Before answering this question, we should clarify the assumptions upon which DDPs rely. Since we are mainly concerned with the design of relational databases, the proposed DDPs are implicitly tailored towards a mapping into the relational model. This has some consequences. For example, some existence dependencies cannot be represented in a relational schema, and they must be dealt with in some other way (transactions, assertions, triggers, etc...). Hence, we consider it acceptable for a DDP not to capture some of those integrity constraints (if they exist, they must be specified outside the diagram, however). On the contrary, the inability to express some other kind of constraint or business rule that can be accurately represented in the relational model might be considered as a serious deficiency for a DDP.

We contend that such interdependency between DDPs and the logical model might be avoided, but it is convenient to retain, mainly for the verification of the design. That is, the analysis of the relational schema obtained by applying one or more instances of a DDP is one criterion to evaluate the quality of the DDP itself. This can be done by applying the well-known principles of good logical design, such as the analysis of the functional dependencies and normalisation theory (this kind of analysis may be carried out at the conceptual level as well). Nonetheless,
although it is possible to evaluate the normal form of the DDP itself, no DDP can guarantee a minimal normal form for its instances since DDPs are not complete specifications (a trivial counterexample is given by an instance of a DDP in which an entity has a multivalued or composite attribute: such an entity is not even in first normal form).

On the other hand, conceptual models often lead in a natural way to schemas that are at least in third normal form. It is useful, then, to take into account other parameters for their comparison.

The complexity needed to formulate important queries or execute crucial transactions (at the conceptual or logical level) is also a parameter that eventually influences the quality of a DDP (or our perception of it in relation to a given problem). It should be noted, however, that performance issues are not relevant at the conceptual level: it is not possible to predict how a given conceptual pattern will perform in practice, because that depends, among the rest, on the target physical model.

Another test for a DDP is its range of applicability, both in the sense of different domains and in the sense of different data models. Despite our focus on the relational model, we claim that DDPs might be used, for example, in the design of XML schemas [12].

5 Examples of Conceptual DDPs

In the following examples, we mainly evaluate the DDPs by their ability to faithfully represent the relevant integrity constraints and, to a minor extent, by their support for certain transactions or queries.

5.1 Multiple Roles

Consider the following scenarios:
1. during a congress, students may register to sessions, and they are allowed to participate only to the sessions to which they have subscribed.
2. In a university, each professor belongs to a faculty; besides, a professor in each faculty is the Dean of Faculty.
3. Within an organization there are several committees; each employee may belong to many committees at the same time, but an employee may be the chair of at most one committee at a time, to which the employee must belong (each committee has exactly one chair). An employee may also be an honorary member of one or more committees (a committee may have many honorary members); finally, each committee has a restricted council whose members are chosen among its honorary members, but an employee cannot belong to more than one council.

The above are all instances of a pattern that we call Multiple Roles, because an entity (e.g., a student) is somehow related to another entity (e.g., a session) with different roles, and each role is a subset of another, so that there is a partial ordering among the relationships (e.g., students register to many sessions, but they participate only to some of them).

Structurally, this pattern is very simple. What makes it relevant is the ordering by set-theoretic inclusion among the relationships. For comparison, examples of structurally similar situations that do not fall into this pattern, are the pair of relationships Has-Sender(Message, Contact) and Has-Recipient(Message, Contact), or the pair Takes-Off(Flight, Airport) and Lands(Flight, Airport), as, in such cases, the involved relationships are independent.

![Figure 3. A common pattern for multiple roles.](image)

The main difficulty in modelling the above scenarios is that it is not obvious how to capture both the cardinality requirements and the ordering. A straightforward way to represent any of the above examples is by using two entities A and B connected by two or more relationships \( R_1, \ldots, R_k \), as shown in Figure 3, with the additional integrity constraint \( R_1 \supseteq R_2 \supseteq \cdots \supseteq R_k \). Using this pattern, the first example above can be modelled as in Figure 4.

The problem of this design is that the most important requirement (“a student may participate only to the sessions to which that student has subscribed”) is not captured in the diagram, but it must be expressed by an external integrity constraint. Moreover, in general this kind of solution tends to generate cycles that typically
lead to problems with data manipulation operations. Another possibility is to use a single relationship type with an
attribute to discriminate among the different roles. But this solution is not viable if not in the simplest cases, whereas
design patterns aim at being as general as possible. The reader is invited to verify that this and other attempts (e.g.,
introducing a specialisation for subscribed students) most likely do not improve over the straightforward model
of Figure 3.

At this point, one may wonder whether a satisfying solution exists at all (at least, one that is representable
with EER diagrams). One way to design a better model consists in realising that the only way to represent set-theo-
retic inclusion of relationships is by turning them into entities by some form of reification. It turns out that
aggregation is exactly what is needed. Indeed, a better alternative is given in Figure 5. This is better because the
model captures all the requirements explicitly.

This example may be generalised to the Restricted Multiple Roles DDP of Figure 6. The restriction is that
the involved relationships must all be either 1:1 or 1:N or N:N. It is not possible to force a specialised relationship
to have cardinalities that are different from the parent. For example, in Figure 7 (ignore the dashed lines for the
moment), Multiple Roles is applied to the third scenario: note that there is no way to force a unique chair per
committee.

A shortcoming of the EER model becomes apparent by this analysis. Hence, we propose to enrich the
diagram by a dashed arrow, which we call weak specialisation. A dashed arrow is allowed only from a specialisation
of an aggregation to one of the entities associated to the aggregation. The interpretation is as follows: although a
specialised entity inherits all the attributes of its parent aggregation, it is actually identified by means of the entity
pointed to by the dashed arrow, which is only a part of the aggregation. An additional label, which can be 1 or
N, specifies whether the specialisation represents a 1:1 or 1:N relationship, respectively. We extend the textual
notation accordingly: $R \rightarrow_A S$ denotes that $R$ is a 1:1 weak specialisation of $S$ identified by $A$, and $R \rightarrow_A S$ denotes
that $R$ is a 1:N weak specialisation of $S$ identified by $A$. For example, in Figure 7, the dashed arrow from “is chair”
to Committee must be interpreted as follows: although “is chair” is a specialisation of the aggregation “e”, only
the committee is needed to identify a chair (because the committee functionally determines the chair). Besides, a
chair can be related to at most one committee, that is, “is chair” is a 1:1 relationship. The university scenario can be
modelled similarly: $Professor(pid), Faculty(fid), (Belongs-To)(Professor(1,1), Faculty(0,N)), Is-Dean \rightarrow Faculty$
$(Belongs-To)$.

---

*We talk about relationships, even if in Figure 6 they are depicted as entities, because they are specialisations of aggregations,
which have a twofold nature.
The of Figure 7 is mapped to the relational model as follows:

Employee(eid, …)
Committee(cid, …)
Belongs-To(employee, committee),
FK: employee → Employee, FK: committee → Committee
Is-Honorary(employee, committee),
FK: (employee, committee) → Belongs-To
Is-In-Council(employee, committee),
FK: (employee, committee) → Is-Honorary, Not null: committee
Is-Chair(committee, employee),
FK: (employee, committee) → Belongs-To, Not null: employee, Unique: employee

The schemas above and the ones that can be derived from an instance of the DDP of Figure 3 are both in Boyce-Codd normal form (assuming that this holds for the involved entities). Hence, they are both “good-quality” as far as normalisation is concerned, but the theory of normalisation is not sufficient to evaluate which schema is better, because functional dependencies capture only a part of the semantic constraints (for this example, inclusion dependency theory helps). Note also that there is no way to derive the above schema from an EER diagram without extending it as we have proposed, because the foreign keys introduced by EER diagrams are always contained in the primary keys. The above schema, instead, contains foreign keys that are proper supersets of the primary keys. The general form of the resulting Multiple Roles DDP is shown in Figure 8.

Figure 8. The Multiple Roles DDP.

Multiple Roles is an unusually common pattern indeed. A further application of this pattern will be met in the next section. Besides its inherent significance, Multiple Roles is interesting also because data that should be modelled with other patterns are often mistakenly modelled using this pattern (especially in the form of Figure 3).

5.2 Hierarchy

A very common requirement in data modelling is to represent data that is structured in hierarchical, or tree-like, form—the paradigmatic example being a hierarchy of employees within an organisation. The main problem with hierarchical information is that most models naturally give rise to recursive queries. A query as simple as “the list of bosses of employee X” (that is, the boss of X, the boss of the boss of X, and so on) may be impossible to write if the query language does not support recursion. For this reason, it makes sense to try to design a database so that recursive queries will not be needed.

Another issue with hierarchies in relational databases is the well-known fact that relation algebra and predicate calculus have the same expressive power as first-order logic in the finite, and first-order logic cannot express recursive queries as well as many other interesting properties, such as connectivity, acyclicity or tree-likeness of graphs [18]. Even SQL (without recursion), which has more expressive power than first-order logic, is essentially

* It is worth mentioning that the current SQL standard does support recursion, although existing DBMSs may not yet implement it. Certainly, many DBAs and database developers refrain from using SQL recursive features, because of legacy requirements, compatibility issues or simply because the adopted implementation lacks that feature.
subject to the same limitations [17]. The same weakness affects EER diagrams: there is no way to faithfully model a tree-like relationship. Therefore, we must assume that the needed integrity constraints are enforced externally (by stored procedures or at the application level). Notwithstanding, it makes sense to design a pattern that (albeit partially) supports a parent-child and/or an ancestor-descendant relationship.

The simplest, and in many cases, the most effective way to represent a hierarchical structure in the EER model is by means of a 1:N recursive relationship Link(Node, Node). If the ancestor-descendant relationship is needed then it must be modelled explicitly [8]. Since such relationship contains the parent-child relationship, we may apply Multiple Roles, which leads to the Hierarchy DDP of Figure 9.

![Hierarchy DDP](image)

**Figure 9. The Hierarchy DDP.**

Maintaining the transitive closure of arbitrary graphs is not trivial [8], but for tree-like structures the task is simplified. For the sake of simplicity, assume that only leaves are added to the hierarchy (which is a pretty common use case). The transaction that inserts node y as a new child of node x is as follows:

1. `insertEntity(Node(y));`
2. for each A-D relationship (z,x), for some Node z: `insertRelationship(A-D(z,y));`
3. `insertRelationship(A-D(x,y));`
4. `insertEntity(Parent(x,y));`

Deletion of leaf y is obtained simply by `delete(Node(y)) cascade`. Insertions and deletions require a number of operations bounded by the height of the tree, so this pattern is especially suitable for shallow trees (e.g., threads in a web forum) or balanced trees. In any case, it is a query-oriented DDP.

Hierarchy trivially supports the retrieval of the parent, children, ancestors and descendants of a node. If such lists should be provided in order, this can be accomplished by adding an attribute that specifies the level of a node (such value should be set upon the insertion of a new node in the transaction above). Also more complex queries are supported. For example, the least common ancestor of a pair of nodes x and y can be easily retrieved: 

\[
\{ z \mid Node(z) \land A-D(z,x) \land A-D(z,y) \land \neg\exists w (A-D(w,x) \land A-D(w,y) \land A-D(z,w)) \}
\]

For hierarchies that do not change frequently and are not very deep, the proposed Hierarchy DDP is a powerful and expressive pattern. Besides, it can be extended to deal with situations in which leaves are treated differently from internal nodes, as in evolutionary trees (such an extension is omitted for lack of space).

It is instructive to consider an alternative, “bad”, implementation, and compare it with the proposed DDP. Consider the following relational schema: Tree(id, left, right). Given two tuples (id1,l1,r1) and (id2,l2,r2), we stipulate that id1 is a descendant of id2 if and only if [l1,r1] is a proper subinterval of [l2,r2]. The left and right endpoints of each interval are computed as the discovery time and finishing time, respectively, in a depth-first visit. With this convention, a non-recursive query to retrieve all the ancestors or descendants of a given node is trivial to formulate. But we consider this a bad design, both structurally and behaviourally, because it violates one of the principles of the relational model, i.e., that the relationships are established by equality of values. In this case, tuples are related implicitly by an implicit rule (this is apparent if a conceptual model is derived: there are no relationships at all). Besides, many transactions are no more efficient than in the Hierarchy DDP: for example, inserting a leaf requires a number of operations that is always at least as large as (and often much larger than) in the transitive-closure model, because at least the discovery and/or finishing times of all the ancestors of the inserted leaf must be updated. We conclude that such “nested set” model is a good example of an anti-pattern.

6 Conclusion

We have suggested that in data modelling, too, there are best practices, common sense knowledge and expertise that can be concisely expressed in a semi-formal way in order to facilitate learning and communication of design concepts. DDPs can guide the design phase of an information system, but they can also be used to analyse existing systems both to recognise known patterns and to identify anti-patterns. Database refactoring techniques would greatly benefit from a well-developed set of DDPs. Besides, the “impedance-mismatch” between OO frameworks and the relational model might be reduced by designing persistence frameworks that map object models to
DDPs rather than directly to the relational model. We have exemplified our approach by describing two common DDPs in detail. By doing so, we have extended EER diagrams to capture even more meaning. Other DDPs we are working on are History (for information that varies in time, such as document revisions) and Access Control (for modelling permissions). We hope to see more DDPs appear in the future.

References


QUERYING ONTOLOGIES ON THE BASE OF SEMANTICS OF BUSINESS VOCABULARIES AND BUSINESS RULES

Algirdas Sukys, Lina Nemuraite, Edvinas Sinkevicius, Bronius Paradauskas

Kaunas University of Technology, Department of Information Systems, Studentu 50-308, Kaunas, Lithuania, sukys.algirdas@gmail.com, lina.nemuraite@ktu.lt, edvinas.sinkevicius@ktu.lt, bronius.paradauskas@ktu.lt

Abstract. Today information systems more and more often rely on ontologies that are able to represent meaningful concepts and complex relationships among them relevant for business models and their supporting software systems. However, ontology development and accessing to ontological data is only possible on the deep technological level that is not friendly for business experts. The goal of the paper is to present a possibility of querying OWL ontologies using semantic formulations of Semantics of Business Vocabulary and Business Rules (SBVR), expressed in SBVR Controlled English. We introduce the initial approach for specifying ontology queries as SBVR questions and transforming them to SPARQL.

Keywords: ontology, SBVR, OWL, SPARQL.

1 Introduction

Semantics of Business Vocabulary and Business Rules (SBVR) is the OMG created metamodel and specification that defines the vocabulary and rules for describing the business semantics – business concepts, business facts, and business rules using some kind of Controlled Natural Language [18]. The SBVR specification considers SBVR Structured English (SSE) as the concrete language for this purpose though other languages are possible. Formal SSE sentences cannot represent all possible constructions of natural English language because they are precise, based on predicate logic with a small extension in modal logic. Nevertheless, SSE language is understandable to business and information system experts as it looks like natural language, and is interpretable by machine in the same breath. The linguistic analysis is outside the scope of SBVR specification; however, SBVR supports linguistic analysis of text for business vocabularies and rules.

In our previous work, we were focused on generating UML & OCL models from SBVR specifications. We proposed the methodology for specifying information system requirements on the base of SBVR business vocabularies and business rules related with business process models, and implemented the prototype of tool VeTIS* supporting that methodology [3], [14]. VeTIS tool is capable to recognize SBVR concepts (object types, roles, fact types, fact type roles, individual concepts) and business rules (various kinds of semantic formulations) that make foundations for conceptualizing business and correspond to knowledge and metaknowledge level of knowledge triangle [16]. However, the complete specification of business conceptual schema and conceptual model needs to include business facts – instances of fact types (ground facts) and propositions (instances of complex semantic formulations based on several fact types) comprising the bottom knowledge level (i.e. fact level) [17]. Therefore, the business semantics editor should recognize instances of concepts and logical formulations needed for specifying ground and complex facts that make sense for various purposes:

- Validating conceptual schema, and conceptual model;
- Validating business facts;
- Representing business facts for different groups of people;
- Formulating SBVR questions about business and answers to them.

The last issue is the focus of the current paper, which presents the idea of transforming SBVR questions into SPARQL queries [9] that can be executed against Web Ontology Language (OWL) ontologies obtained from SBVR vocabularies and business rules. Consequently, it would be possible to describe business vocabularies and business rules, and querying software systems generated on the base of these vocabularies and rules using a single language and terminology.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 outlines the overall process of creating SBVR business vocabularies and business rules from SSE style texts, formulating SBVR questions, transforming them into SPARQL queries and executing them against ontologies generated from SBVR vocabularies and rules. Section 4 is devoted to formulating SBVR questions. Section 5 presents
transformation of SBVR questions into SPARQL and example of such transformation. Section 6 draws conclusions and outlines the future work.

2 Related works

The contributors of SBVR emphasize that SBVR specification is devoted for business people and for business purposes independently of information systems designs, i.e. for using business vocabularies and business rules as guidance for the business and different groups of people: workers, customers, suppliers, partners. OMG also proposes the XMI schema for the interchange of business vocabularies and business rules among organizations and between software tools. SBVR metamodel and XMI schema may be used for developing software tools for managing business vocabularies as well as for automating development of software for managing business on the base of business semantics, i.e. in the way different of previously existed approaches, e.g. [15], [19]. SBVR business vocabularies are transformable into UML & OCL [3], [14], and vice versa [2]; BPMN [1], RDB schemas [13], OWL [4], Web services [5], [7] etc. Besides automating development of software models and code [11], [12], SBVR Structured English may serve for creating semantic specifications of legacy information resources, integrating these resources, implementing contextualized and multilingual information systems, etc. The power of SBVR is nicely disclosed by the fact that SBVR specification itself is formally written in SSE [10].

Applying SBVR in practice, various limitations become obvious. For example, SBVR lacks the larger collection of data types and patterns for constructs needed for expressing arithmetic operations, data and time, past and future events and similar. SBVR specification is easy extensible, however, it would be desirable having standard constructs for most frequent cases. Spreeuwenberg and Anderson notice more deficiencies of SBVR: lack of inference; lack of references (rules should be stated in one sentence); necessity to introduce concepts before referencing to them; impossibility to express directives etc [22], [23]. Surely, SBVR should be extended in the future.

From the other side, until now only part of SBVR was used. For example, SBVR questions provide a nice capability of querying business models and their implementations but they attained a little attention in SBVR specification and related research. Kriechhammer in 2006 has noticed about possibilities of SBVR questions [9] for business people to query systems for business modeling without the support of programmers. SBVR questions are based on logical projections that are much more comfortable for business people than various query languages that are platform-specific and suitable for IT specialists having the experience of working with special tools.

Realizing the idea of querying business in SBVR requires a creation of a whole infrastructure including tools for authoring SBVR business vocabularies and rules, transforming them into various software models and code, including OWL, SQL, Web Services, business process execution languages and so on. Several EU projects are devoted for this purpose: OPAALS* (2006-2010, generating Web services and data models from SBVR specifications [13], [21]), ONTORULE** (2009-2012, aiming at integrating knowledge and technologies needed for extracting ontologies and business rules from various documents, including natural language texts; managing them and implementing in software systems). The commercial tool suite for Business Semantics Management Collibra*** presents capabilities for authoring SBVR vocabularies and rules, generate ontologies and various models of information systems.

None of the mentioned works addresses SBVR questions (except [9], but no further research in that direction was done). As interest in SBVR is growing and corresponding infrastructures are arising, we present the initial methodology for transforming SBVR questions into SPARQL.

3 Process of transforming SBVR Structured English questions into SPARQL

Process of transforming SBVR Structured English questions and executing them as SPARQL queries is presented in Figure 1. First of all, we need to define SBVR business vocabulary and business rules, and specify individual concepts and facts of the problem domain under consideration. Business concepts and rules are specified in SSE style text that consists of terms, verbs, names and keywords [18], and needs to be parsed for transforming them into SBVR schema. Our parser [14] is based on Antlr grammar and recognizes strict sentence structures. The development of a flexible parser is also a problem investigated in [8]. In a general case, an input into the parser may come from linguistic analysis tools.

In the next step, the SBVR model should be transformed into OWL ontology, which will be a data source to execute query. After SBVR model is defined, a question can be specified, using keywords, terms, verbs and names of the problem domain. The formulated question should be parsed to generate SBVR model, which

* http://www.opaals.eu/
** http://ontorule-project.eu/
*** http://www.collibra.com/
eventually can be transformed into SPARQL query. In real life applications, SPARQL queries will be executed on ontologies that may be stored in relational databases [24] and may be related with various software applications. These ontologies can specify business knowledge including business data, services and processes. The potential possibilities of applying SBVR questions are invaluable for business people but implementing such capabilities require many efforts from the research community.

Figure 1. Process of transforming and executing SBVR questions

In Figure 1, the dashed rectangular marks artifacts concerned in the current paper. Figure 1 also includes artifacts that are not considered here. Functionality of specifying business vocabularies and transforming them into SBVR schema was implemented in our first prototype [14]. Transforming SBVR schema into OWL is under development by other members of our research group (please look at the paper of Karpovic, Nemuraite in the current IT’2011 proceedings).

4 Modeling SBVR questions

The basic SBVR concept types that can be presented in a business vocabulary are object types (or general concepts), fact types (or verb concepts), and roles. For formulating SBVR questions, a vocabulary should be capable to represent individual concepts and facts. SBVR individuals and facts are essential elements for representing OWL ontology individuals and their relations.

SBVR defines individuals using logical formulations instantiation formulation, which relates an individual concept with the object type. The instantiation formulation means a classification of things that are individual concepts of some object type. It is also necessary to use the assortment fact type that is defined with respect to a given general concept and a given individual concept such that each instance of the fact type is an actuality that the instance of the individual concept is an instance of the general concept. We present a fragment of SBVR metamodel, specifying an individual concept, and an example of SBVR model of fact type “Ktu is university” in Figure 2 and Figure 3.

Figure 2. SBVR metamodel fragment specifying the individual concept
SBVR question is meant by closed logical formulation – question nominalization (projecting formulation). SBVR questions are based on fact types, defined in a business vocabulary. For example, question “What journals are referred in the database ‘Web of Science’?” is based on the fact type “journals are referred in the database”. Complex questions consider complex facts based on several fact types. Complex facts are constructed using keywords “and”, “that” or “which”. The example of a complex question is “What journals are referred in the database ‘Web of Science’ and have an impact factor, which is greater than 0, and are published by the university that is in city ‘Kaunas’.”.

Figure 3. SBVR model for specifying the individual concept

After specifying a question using SBVR Structured English, we parse it using Antlr grammar and generate SBVR model of the question. Model is based on Eclipse EMF SBVR 1.0 metamodel. We present a fragment of SBVR metamodel for formulating questions in Figure 4.

Figure 4. Fragment of SBVR metamodel for representing questions

For demonstrating how questions are formulated in SBVR let us take an example (Question1) – “What universities are in city ‘Kaunas’?”. SBVR model of this question is given in Figure 5. A core element of a question in SBVR model is the question, which is a meaning of interrogatory. A question is formulated by a closed projection, so the result of the projection answers the question. A question starts with the interrogative operator “what”, which means things that we want to see in the answer.

In the SBVR model, a closed projection that means a question is formulated by a question nominalization (projecting formulation). Projecting formulation has a projection on a variable that ranges over a concept that matches the operator. In our example we use variable1 which ranges over object type university. That means, we want to get a list of universities. Closed projection is constrained by the universal quantification, which has a variable2 that ranges over the object type university.

Universal quantification is restricted by a logical formulation aggregation formulation. The aggregation formulation is used to associate a variable with a set of things that satisfy some condition. It formulates natural language expressions of the form: “let \(<variable>\) be the set of all things \(t\) such that \(<some\ condition\ involving\ \(t\)>\)” [18] so that \(<variable>\) can then be used in other formulations regarding the set. In our
example aggregation formulation set restrictions for university objects. These restrictions are expressed by a projection, which has atomic formulation, based on a fact type university is_in city. Object type of the second fact type role of this fact type is connected with individual concept 'Kaunas'.

Figure 5. SBVR model of the question “What universities are_in city 'Kaunas'?"

5 Transforming SBVR questions into SPARQL queries

SBVR vocabulary should be represented as OWL ontology for transforming SBVR questions into SPARQL queries where SPARQL is a query language for ontologies. Ontologies are stored using Resource Description Framework (RDF) data model that is considered to be the most relevant standard for data representation and exchange on the Semantic Web [6]. RDF statements are expressed in the form of subject-predicate-object. A set of RDF statements compose directed labeled graph. SPARQL query contains a set of triple patterns called a basic graph pattern. Triple patterns are similar to RDF triples, except that each of this triple can be a variable [20]. Query statements are compared with RDF graph statements and the results of such comparison are returned for the user. SPARQL query consists of three main parts that are described in Table 1.

Table 1. Structure of SPARQL query

<table>
<thead>
<tr>
<th>Query part</th>
<th>Definition</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Defines four different kinds of SPARQL queries. All of them use the same graph matching approach, but they differ in result presentation.</td>
<td>select - the most common query which returns variables and their bindings; ask – returns answer, if query pattern meets RDF graph. Only yes or no values can be returned; describe – returns all of the relations of the certain resource; construct – returns a single RDF graph specified by a graph template.</td>
</tr>
<tr>
<td>solution modifiers</td>
<td>Enable to modify information output for user.</td>
<td>projection - choose certain variables to output; distinct – remove duplicate solutions; order – sort results by specified variable. Asc and desc keywords can be used to define sort order; limit – specify maximum rows in a result set; offset – specify number of the first row in the result output.</td>
</tr>
<tr>
<td>pattern matching part</td>
<td>„where“ part of the query to describe graph matching pattern. It consists of collection of triple patterns, which are compared with RDF graph.</td>
<td>optional – does not eliminate the solution, if optional part does not meet RDF graph; union – merge different result sets into one; filter – filter by data type values.</td>
</tr>
</tbody>
</table>
We will provide an example of transforming a SBVR question into SPARQL query under the process model, which is presented in Figure 1. First of all, we have to define a business vocabulary – terms, names, fact types and facts of the problem domain. We will not go deep into this process, as it is described in [14]. Here we will only present ontology, which is generated from business vocabulary to execute query in. Ontology is presented in Figure 6 as UML class diagram.

The Query2 we will analyze is designed to get names of universities, names and impact factors of the journals, that are referred in the database 'Web of Science', have impact factor greater than 0, and are published by university that is in city Kaunas. Written in SBVR Structured English, this query will look like this:

What are university_names of universities, journal_names of journals and journal_impact_factors of the journals that are_referred_in the database 'Web of Science', have an journal_impact_factor, which is_greater_than 0, and are_published_by the university that is_in city 'Kaunas'?

This query is based on facts and fact types:

- journals are_referred_in database 'Web of Science'
- journals have journal_impact_factor, which is_greater_than 0
- journals are_published_by university
- university is in city 'Kaunas'

When a query is based on several facts it is difficult to analyze it, so predefined keywords as “that”, “which”, “and” are used. These keywords help to recognize object types and individual concepts of the question. Model of Query2 is based on the same principles as Query1, presented in Figure 5, but is much more complex.

For this question nominalization we must apply a projection 1 on three variables that range over the object types “names of universities”, “names of journals”, and “journal impact factors”. The projection 1 has an auxiliary variable 4 that is ranged over the object type “journal”. The variable 4 is restricted by a projecting formulation that is constrained by an atomic formulation based on the fact type “journal is in city”. The projecting formulation 2 has a projection on a variable 5 that ranges over the object type “journal” and has an auxiliary variable that is restricted by the instantiation formulation “Kaunas is city”. The projection 1 is on a variable 5 that ranges over a concept “proposition”. The proposition is meant by a closed logical formulation – conjunction of projecting formulations “journals are_referred_in database 'Web of Science', journals have journal_impact_factor, which is_greater_than 0, journals are_published_by university that is_in city 'Kaunas'. These projecting formulations are formulated similarly as in Figure 5.

The first step of designing SPARQL query is to define solution modifiers – ?university_name, ?journal_name and ?journal_impact_factor. These variables are defined by query fact types and pattern matching part is generated from facts. Transformations of example query are presented in tables Table 2 and Table 3.
Table 2. Composing solution modifiers

<table>
<thead>
<tr>
<th>SBVR fact types</th>
<th>Solution modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>university_names of universities</td>
<td>?university_name</td>
</tr>
<tr>
<td>journal_names of journals</td>
<td>?journal_name</td>
</tr>
<tr>
<td>journal_impact_factors of journals</td>
<td>?journal_impact_factor</td>
</tr>
</tbody>
</table>

Table 3. Composing triple patterns

<table>
<thead>
<tr>
<th>SBVR facts</th>
<th>Triple patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>journal has journal_impact_factor which is_greater_than 0</td>
<td>FILTER(?journal_impact_factor &gt; &quot;0&quot;^^xsd:double)</td>
</tr>
<tr>
<td>journal is_published_by university</td>
<td>?journals journal:isPublishedBy ?university</td>
</tr>
<tr>
<td>university is_in city 'Kaunas'</td>
<td>?university journal:isIn ?city . ?city journals:name &quot;Kaunas&quot;^^xsd:string</td>
</tr>
</tbody>
</table>

After transforming SBVR into SPARQL, we get such SPARQL query:

```sparql
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX journals_ontology: <http://Data.JournalsOntology/>

select ?university_name ?journal_name ?journal_impact_factor
where {
  ?journals journals_ontology:isReferredIn ?database .
  FILTER (?journal_impact_factor > "0"^^xsd:float) .
  ?journals journals_ontology:isPublishedBy ?university .
  ?university journals_ontology:isIn ?city .
  ?city journals_ontology:name "Kaunas"^^xsd:string .
  ?university journals_ontology:name ?university_name .
  ?journals journals_ontology:name ?journal_name
}
```

Results of the Query2 are presented in Figure 7.

![Figure 7. Results of Query 2](image)

6 Conclusions and future works

Currently, the growing attention to SBVR seems most focused on engineering or verbalizing various software models and applications. Such attention is not paid to SBVR questions that are worth to be considered for solving some topical problems e.g. querying about business data and processes in a language understandable to business people.

Our research and implemented prototype in principle have shown the feasibility of transforming SBVR Structured English questions into SPARQL. Question transformations process starts from parsing it and analyzing related fact types and facts. The most difficult part is to recognize links among facts, so predefined keywords must be used. After fact types, facts and their links are discovered, a question in SBVR Structured English style text can be transformed into SBVR schema and later – into SPARQL solution modifiers and pattern matching parts.

Our future work will be concentrated on thorough analysis of various patterns for formulating SBVR questions and corresponding SPARQL patterns as well as integrating the obtained solution into the wider context where such questions could be executed for extracting answers from ontologies and underlying data sources.
References


