A FORMALIZED TRANSFORMATION PROCESS FOR GENERATING DESIGN MODELS FROM BUSINESS RULES

Mohammed Bonais  
*La Trobe University*, M.Bonais@students.latrobe.edu.au

Kinh Nguyen  
*La Trobe University*, Kinh.Nguyen@latrobe.edu.au

Eric Pardede  
*La Trobe University*, E.Pardede@latrobe.edu.au

Wenny Rahayu  
*La Trobe University*, W.Rahayu@latrobe.edu.au

Follow this and additional works at: http://aisel.aisnet.org/pacis2014

Recommended Citation

Bonais, Mohammed; Nguyen, Kinh; Pardede, Eric; and Rahayu, Wenny, "A FORMALIZED TRANSFORMATION PROCESS FOR GENERATING DESIGN MODELS FROM BUSINESS RULES" (2014). PACIS 2014 Proceedings. 21.  
http://aisel.aisnet.org/pacis2014/21

This material is brought to you by the Pacific Asia Conference on Information Systems (PACIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in PACIS 2014 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
A FORMALIZED TRANSFORMATION PROCESS FOR
GENERATING DESIGN MODELS FROM BUSINESS RULES

Mohammed Bonais, Department of Computer Science & Engineering, La Trobe University, Melbourne, Australia, Mabonais@students.latrobe.edu.au

Kinh Nguyen, Department of Computer Science & Engineering, La Trobe University, Melbourne, Australia, Kinh.Nguyen@latrobe.edu.au

Eric Pardede, Department of Computer Science & Engineering, La Trobe University, Melbourne, Australia, E.Pardede@latrobe.edu.au

Wenny Rahayu, Department of Computer Science & Engineering, La Trobe University, Melbourne, Australia, W.Rahayu@latrobe.edu.au

Abstract

Business rules play a critical role in building and maintaining effective and flexible information systems. In light of that critical role, the publication of the Semantic Business Vocabulary and Business Rules standard (SBVR), has been regarded a highly significant advance. Following that release, a number of research efforts have been made to convert SBVR to design models, most of which are structural models represented in UML. However, so far the proposed methodologies tend to be of an exploratory nature in the sense that they are not built on a rigorous foundation. Our aim is to identify a core subset of the SBVR features and show how those core SBVR features can be translated into an equivalent UML structural model. To do that on a sound foundation, we first provide formal models of the core SBVR and the target UML class model, completed with proofs of correctness, and describe how the mapping rules can be applied in a transformation process. Finally, to show the usefulness of our formal approach, we discuss how it is used as a crucial component in a larger project, which embraces a number of practical objectives.

Keywords: Business Vocabulary, Business Rules, Design Models, Formalization, Transformation, Mapping, SBVR, UML Class Diagram.
1 INTRODUCTION

Business rules derived from external sources such as regulations and internal sources such as policies are one of the most important assets of all organizations (Shao and Pound 1999). They are employed to define the business domain structure and control the business processes. All levels of information system development have to fully take into consideration the organization’s business rules.

Business rules are traditionally written in natural language as unstructured text to allow business people to review and validate them easily (Morgan 2002). The unstructured text makes the auto-transformation of business rules to other standards extremely difficult. The only viable option is to perform the transformation manually. This manual approach, besides being time-consuming, has several drawbacks: it is very hard to evaluate and ensure the correctness and consistency of both the input business rules and the output design models.

The publishing of the SBVR standard (Semantic of Business Vocabulary and Business Rules) by Object Management Group (OMG) allows domain experts to define business vocabulary and rules as structured text rather than unstructured text (OMG 2008) in a clear and unambiguous manner (Cerrie et al. 2009). This gives an opportunity to acquire business vocabulary and rules that can be machine-processed (Ross 2008). Thus, automated transformations from the SBVR specification to other models and standards can be performed (Bajwa et al. 2011). At the same time, business vocabulary and rules can be readable and understood by stakeholders for reviewing and validating (Linehan 2008). Consequently, the gap and misunderstanding between business people and IS developers can be decreased to the lowest level when the SBVR specification is adopted as a representation for defining business vocabulary and rules.

The Unified Modelling Language (UML) became a widely accepted standard for modelling the structure and behavior of IS projects. Though UML has a number of diagrams, the class diagram is considered as the most understandable, intuitive, and well-defined diagram. This is because it expresses the most important view of UML, which is the structure specification (Balaban et al. 2010). Furthermore, a study of the usage of UML diagrams in IS projects showed that class diagram is the most commonly used diagram. It found that most IS projects used the class diagram to sketch two-thirds or more of their design models (Dobing and Parsons 2006).

1.1 Motivations

After the publication of the SBVR standard, a number of researchers propose methodologies for mapping business vocabulary and rules represented in the SBVR specification to the UML class diagram (Raj et al. 2008; Kleiner et al. 2009; Nemuraite et al. 2010; Afreen and Bajwa 2011). These methodologies overcome the downsides for the manual mapping of business vocabulary and rules to the class diagram. However, none of these methodologies clearly demonstrates how the transformation is accomplished. More specifically, it is quite unclear what exactly features of the SBVR standard these methodologies are considering in the transformation process. In addition, it is also quite unclear what exactly the transformation rules are. For instance, Nemuraite et al. (2010) do not explain how super and sub classes in an inheritance are recognized (e.g. whether the feature of categorization type is taken into account or not). Thus, the correctness of the class diagram generated by those methodologies, which can be evaluated by an inspection of the obtained output, cannot be generally guaranteed.

The SBVR specification is a major breakthrough in defining and writing business vocabulary and rules. However, it still has few limitations that prevent its adoption by business people and IS developers. Some of these limitations are:

a) The SBVR is a large and substantially complicated standard (Linehan 2006), with a large number of elements and properties. In practice, when defining business vocabulary and rules for a specific business domain, it can be very hard to use and handle all these elements and properties consistently.

b) The SBVR standard does not clarify how a subset for a particular domain can be defined correctly and consistently. (Spreeuwenberg 2008).
1.2 Contributions

To overcome the complexity of the SBVR specification and the limitations of the existing transformation methodologies, we are engaged in a long-term research project to formulate the transformation process from the SBVR specification to the UML class diagram that represent data models on a secure foundation and formally defined basis that can ensure the correctness of the generated class diagram. In this research project, we:

a) Identified and formally defined a model for a subset of the SBVR standard that express the features of the structural design. This model clearly specifies the elements and properties of SBVR as well as identified the necessary constraints that can be employed to validate the SBVR model (thus, the input for the transformation process is well-defined and validated).
b) Specified a formal model for the class diagram. It includes all elements of the class diagram as well as the necessary constraints that can be used to validate the UML class model (thus, the output of the transformation process is well-defined and validated).
c) Constructed a SBVR repository based on the SBVR formal model.
d) Constructed a UML repository based on the UML formal model.
e) Formally specified the transformation process that includes a number of transformation rules and logically proofed the correctness of the generated UML elements.
f) As proof of concept, built a system that automates the validation of the formal models and the transformation process.
g) Evaluated the proposed formal models and transformation process by having sets of test cases, presenting case studies, and comparing the results of our transformation process with the results of the existing transformation methodologies by using precision and recall measures.

This paper, due to space limitation, covers only one essential part of the research project. This part focuses on identifying the features of the SBVR specification that express the basic features of structural design models, which are classes, attributes, and associations and providing a formal basis for formulating and implementing the transformation process from the core SBVR to the UML class diagram. The formal models and the transformation process are validated by applying a case study.

2 RELATED WORK

Linehan (2006) explores the notion of transforming SBVR automatically to the design models as UML diagrams and Object Constraint Language (OCL). Linehan’s work is explorative and it does not discuss or show how the SBVR specification can be mapped to the UML diagrams. After that, Raj et al. (2008) propose a methodology to transform business vocabulary and rules written into the SBVR Structured English to three UML diagrams. One of these diagrams is the class diagram. This methodology reuses some of the fundamental transformation rules provided in the official release of the SBVR specification (OMG 2008) and provides two additional rules that produce operations and cardinality. In this methodology, it is very difficult to generate precise elements as it only depends on the order of noun concepts in fact types to generate an attribute for a certain class. And this may result in mapping a noun concept improperly. Kleiner et al. (2009) propose a transformation methodology that creates a class diagram that is quite similar to the class diagram generated in (Raj et al. 2008). However, it initially maps the SBVR Structured English to the SBVR Semantic Formulation then the result of the initial transformation is mapped to the class diagram. Comparing with the methodology proposed in (Raj et al. 2008), this methodology produces a UML class diagram with additional minor details, which are the data types of attributes and the names of associations.

Another transformation is proposed as an editing and mapping tool called VeTIS (Nemuraite et al. 2010). This tool is used to edit SBVR vocabulary and rules and transform them to the UML class diagram. To the best of our knowledge, VeTIS tool provides the most sophisticated mapping methodology from the SBVR specification to the class diagram. However, this tool does not provide any details about the mapping rules, which makes it difficult to check the correctness of the generated model.
Afreen and Bajwa (2011) generate the class diagram in a different manner. They initially define a set of software requirements as SBVR business rules. Then, they extract SBVR vocabulary from the defined SBVR rules. Finally, they generate the class diagram from the extracted SBVR vocabulary based on some transformation rules. In their methodology, the super/sub classes for inheritances and container/contained classes for aggregations are only determined by the order of noun concepts in fact types. However, the order of noun concepts alone cannot guarantee the right type of the generated classes.

3 SBVR SPECIFICATION

3.1 Features of Core SBVR

To overwhelm the complexity of the SBVR specification, we identify a subset of SBVR features called Core SBVR. This subset includes all necessary elements and properties of SBVR that can express the features of the structural design models as well as give a correct, consistent and expressive SBVR vocabulary and rules. To determine which features of SBVR should be included in this subset, we use the UML class diagram, which is widely accepted diagram for representing structural design model, as a guidance for selecting SBVR features. Therefore, each feature of SBVR that is considered to be required to express a feature in the class diagram is added to the core SBVR subset. In figure 1, we provide a case study of a UML class diagram with its main features, which are classes, attributes, associations, aggregations, inheritances, enumerations, and generalization sets. The rest of this section demonstrates all SBVR features that can be equivalent for the main features of the UML class diagram.

![Figure 1](image-url)  
*Figure 1. A case study of a UML class diagram with its main features.*

The core SBVR subset, which concerns only about business vocabulary and rules that can express the features of structural design models, incorporates four main elements, which are noun concepts, fact types, categorization schemes, and business rules.

3.1.1 Noun Concept

A noun concept is the smallest and most basic element of the SBVR standard. It provides the meaning of a noun or noun phrase. A noun concept can be one of the followings:

a) Object type: It is a word or a group of words that classifies things (classes) on the basis of their shared attributes such as ‘person’ and ‘department’.

b) Role: It is a noun concept that is related to an object type based on its playing a part or being used in some specific situations. For instance, ‘manager’ can be a role for the object type ‘employee’.

c) Individual concept: It is an instance of an object type. i.e. ‘Male’ is an instance of ‘gender type’.

d) Categorization type: It categorizes an object type into subcategories. For example, the categorization type ‘employment type’ identifies the categorization criterion that categorizes the object type ‘employee’ into the instances ‘part time employee’ and ‘full time employee’. 
3.1.2 Fact Type (Verb Concept)

A fact type is a sentence that provides the meaning of a verb or verb phrase. It is composed of pre-defined noun concepts plus one verb or verb phrase. A fact type can be one of the followings:

a) Associative: It creates association between noun concepts. i.e. ‘department is managed by manager’.
b) Is-property-of: It defines an attribute of a noun concept. i.e. ‘ID is property of employee’.
c) Characteristic: It states a characteristic (Boolean attribute) of a noun concept. i.e. ‘person is smoker’.
d) Categorization: It creates inheritance relationships. i.e. ‘employee is a person’.
e) Partitive: It defines aggregation between noun concepts. i.e. ‘department includes section’.

3.1.3 Categorization Scheme

It is a concept that is used to categorize an object type based on a categorization type with the constraint ‘partial-disjoint’. Categorization schemes cannot be involved in building fact types or business rules. There is a restricted case of the categorization scheme, which is Segmentation. The segmentation is used when the categorization constraint is ‘total-disjoint’. i.e. ‘employee by type’

3.1.4 Business Rule

Business rules are built by combining pre-defined fact types with some keywords. For example, ‘It is necessary that each department is managed by exactly one manager’. The SBVR specification has four different kinds of keywords as follows:

a) Modal keywords: They add model operations to the business rule. i.e. ‘it is necessary’.
b) Quantification keywords: They express the quantity of noun concepts in business rules. i.e. ‘each’.
c) Logical keywords: They denote the logical operations in business rules. i.e. ‘if p then q’.
d) General keywords: they are used as supportive keywords in business rules. i.e. ‘the’.

As can be seen from the above background, the features of the core SBVR subset have the ability to fully and unambiguously express the features of the structural design models as business vocabulary and rules. To make it clearer, table 1 links each feature of the structural design model presented in the case study of the UML class diagram (figure 1) with its counterpart feature of the core SBVR subset.

<table>
<thead>
<tr>
<th>UML Feature</th>
<th>Equivalent SBVR Feature</th>
<th>SBVR Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Object type</td>
<td>Person</td>
</tr>
<tr>
<td></td>
<td>Categorization type instance</td>
<td>Full time employee</td>
</tr>
<tr>
<td>Attribute</td>
<td>Is-property-of fact type</td>
<td>Name is property of employee</td>
</tr>
<tr>
<td></td>
<td>Characteristic fact type</td>
<td>Person is smoker</td>
</tr>
<tr>
<td>Cardinality of attribute</td>
<td>Business rule</td>
<td>It is necessary that each department has exactly one name</td>
</tr>
<tr>
<td>Association</td>
<td>Associative fact type</td>
<td>Department is managed by manager</td>
</tr>
<tr>
<td>Cardinality of association end</td>
<td>Business rule</td>
<td>It is necessary that each department is managed by exactly one manager</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Partitive fact type</td>
<td>Department includes section</td>
</tr>
<tr>
<td>Cardinality of aggregation end</td>
<td>Business rule</td>
<td>It is necessary that a department includes at least two and at most twenty sections</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Categorization fact type</td>
<td>Employee is a person</td>
</tr>
<tr>
<td></td>
<td>Categorization type instance with object type</td>
<td>Employee Full time employee</td>
</tr>
<tr>
<td>Generalization set</td>
<td>Categorization scheme</td>
<td>Employee by type</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Object type</td>
<td>Gender type</td>
</tr>
<tr>
<td>Literal of enumeration</td>
<td>Individual concept</td>
<td>Male</td>
</tr>
</tbody>
</table>

Table 1. The linkage between the features of the structural design model and the features of the core SBVR subset.
3.2 Business Vocabulary and Rule Entries

The SBVR standard provides a guide for expressing vocabulary and rules in structured text. Each concept or rule has one and only one entry that has one representation plus several properties that provide further information about the concept or rule. Below an example for a noun concept entry is provided.

Manager
- **Concept Type:** Role
- **General Concept:** Person

The SBVR structured text uses four different font styles for various purposes:

- **Underlined:** It denotes noun concepts. e.g. ‘employee’.
- **Double underlined:** It describes individual concepts. e.g. ‘Adam’ or ‘Male’.
- **Italic:** It designates verbs. e.g. ‘has’.
- **Regular:** It denotes all keywords in business rules. e.g. ‘exactly’.

4 FORMAL MODELS OF SBVR AND UML

To establish a rigorous basis for the transformation process, this section provides formal models for the core SBVR and the UML class diagram. Each model, SBVR and UML, have a set of constraints. A valid model must have the structure specified and satisfy all constraints. Due to lack of space, in this paper, the formal models and transformation process only include the features that are related to the basic structural design model, which is composed of classes, attributes, and associations and the full version of the formal models and transformation are presented in www.SBVR2UML.info.

4.1 SBVR Formal Model

The SBVR formal model is divided into two main parts, which are SBVR vocabulary and SBVR rules.

4.1.1 SBVR Vocabulary

**Definition 1.** SBVR vocabulary is a tuple $\text{SBVRVocab} = (\text{NounConcepts, FactTypes})$.

**Definition 1.1.** $\text{NounConcepts}$ is a set of noun concepts. A noun concept $nc \in \text{NounConcepts}$ is a tuple $nc = (\text{representation, conceptType, generalConcept})$, where:

- **representation** is the primary representation of the noun concept.
- **conceptType** is the concept type. It can be one of $\text{objectType, role, or individualConcept}$.
- **generalConcept** is the general concept and it must be an existing $\text{objectType}$.

**Constraint 1.1.1.** The representations of noun concepts have to be unique. That is,

$$\forall nc_1, nc_2: \text{NounConcepts} \cdot nc_1 \neq nc_2 \Rightarrow nc_1.\text{representation} \neq nc_2.\text{representation}$$

// $nc_1, nc_2$ are unique noun concepts

**Constraint 1.1.2.** The representations and concept types of noun concepts must be specified. That is,

$$\forall nc: \text{NounConcept} \cdot nc.\text{representation} \neq \perp \land nc.\text{conceptType} \neq \perp$$

// the representation and conceptType of $nc$ are not unspecified

**Constraint 1.1.3.** The general concepts of noun concepts have to be an existing $\text{objectTypes}$. That is,

$$\forall nc: \text{NounConcepts} \cdot nc.\text{generalConcept} = \{x: \text{NounConcepts} \cdot x.\text{conceptType} = \text{‘objectType’}\}$$

// the generalConcept of $nc$ is an existing $\text{objectType}$

For example, $\text{manager}$ is a noun concept $nc \in \text{NounConcepts}$ and it can be specified as:

- $nc.\text{representation} = \text{manager}$.
- $nc.\text{conceptType} = \text{role}$.
- $nc.\text{generalConcept} = \text{employee}$.
In the following definitions, due to lack of space, the constraints that are used in the proof of correctness (provided in the next section) are presented formally and the other constraints are described in text.

**Definition 1.2.** FacTypes is a set of fact types. A fact type \( f \in \text{FacTypes} \) is a tuple \( f = (\text{representation}, \text{conceptType}, \text{noun}_1, \text{noun}_2, \text{role}_1, \text{role}_2, \text{verb}) \), where:

a) \( \text{representation} \) is the primary representation of the fact type.

b) \( \text{conceptType} \) is the concept type. It can be associative, is-property-of, characteristic, categorization, or partitive.

c) \( \text{noun}_1 \) is the first noun concept (\( \text{objectType} \)) that is included in the fact type’s representation. However, if the first noun concept is \( \text{role} \), \( \text{noun}_1 \) must be the general concept of the role.

d) \( \text{noun}_2 \) is the second noun concept that is involved in the fact type’s representation. Similar to the \( \text{noun}_1 \), if the second noun concept is \( \text{role} \), it has to be the general concept of the role.

e) \( \text{role}_1 \) is the first noun concept in the representation of the fact type if the noun concept is \( \text{role} \). Otherwise, it obtains the same value of \( \text{noun}_1 \) if the noun concept is \( \text{objectType} \).

f) \( \text{role}_2 \) is the second noun concept that is included in the representation if the noun concept is \( \text{role} \). Otherwise, it gains the identical value of \( \text{noun}_2 \) when the noun concept is \( \text{objectType} \).

g) \( \text{verb} \) is the actual verb that is used in the representation of the fact type.

The properties \( \text{noun}_1, \text{noun}_2, \text{role}_1, \text{role}_2 \) and \( \text{verb} \) are directly extracted from the representation of the fact types. All these properties are added to the fact types to reduce the complexity and increase the correctness level of the transformation process.

Fact types have to strictly follow a set of constraints, which are:

a) The representations of fact types must be unique.

b) The representations and concept types of fact types have to be specified.

c) The representations of fact types must include existing \( \text{objectTypes} \) or \( \text{roles} \).

d) The first noun concept and the verb in the representation of fact types must be specified.

For instance, \( \text{department is managed by manager} \) is a fact type \( f \in \text{FacTypes} \) and it can be specified as:

a) \( f.\text{representation} = \text{“department is managed by manager”} \)

b) \( f.\text{conceptType} = \text{associative} \)

c) \( f.\text{noun}_1 = \text{“department”} \)

d) \( f.\text{noun}_2 = \text{“employee”} \)

e) \( f.\text{role}_1 = \text{“department”} \)

f) \( f.\text{role}_2 = \text{“manager”} \)

g) \( f.\text{verb} = \text{“is managed by”} \)

**4.1.2 SBVR Business Rules**

**Definition 2.** SBVR BusinessRules is a set of business rules. A business rule \( br \in \text{BusinessRules} \) is a tuple \( br = (\text{statement}, \text{factType}, \text{modal}, \text{quantifier}_1, \text{quantifier}_2) \), where:

a) \( \text{statement} \) is the statement of the business rule. It is built by combining fact types and keywords.

   i. \( \text{factType} \) is the fact type that is used to build the business rule statement.

   ii. \( \text{modal} \) is the modal keyword that is included in the business rule statement.

   iii. \( \text{quantifier}_1 \) is the first quantification keyword that is included in the business rule statement. It always belongs to the first noun concept of the fact type.

   iv. \( \text{quantifier}_2 \) is the second quantification keyword that is employed in the business rule statement. It always belongs to the second noun concept of the fact type.

Business rules must firmly satisfy the following constraints:

a) The statements of business rules must be unique.

b) The statements of business rules must be specified.

c) The statements of business rules must include existing fact types.

d) A model keyword must be included in each business rules.
The properties factType, modal, quantifier₁, and quantifier₂ is added to the SBVR rules to reduce the complexity of the transformation. All these properties are extracted from the business rules’ statements.

For example, It is necessary that each department is managed by exactly one manager is a business rule br ∈ BusinessRules and it can be specified as:

a) br.name = "It is necessary that each department is managed by exactly one manager".

b) br.modal = "It is necessary".

c) br.quantifier₁ = "each".

d) br.quantifier₂ = "exactly one".

4.2 UML Formal Model

As mentioned earlier, this paper covers the class diagram that represents the basic features of the structural design model. In addition, due to shortage of space, features such as inheritance are omitted from the UML formal model described below.

Definition 3. UML class diagram is a tuple classDiagram = (Classes, Associations).

Definition 3.1. Classes is a set of classes. A class c ∈ Classes is a tuple c = (name, attributes), where:

a) name is the name of the class.

b) attributes is a set of attributes for the class. An attribute att ∈ attributes is a tuple att = (name, dataType), where:

i. name is the name of the attribute.

ii. dataType is the attribute’s data type. It can be one of the basic data types. e.g., Text.

Constraint 3.1.1. The names of classes must be unique. That is,

∀ c₁, c₂: Classes • c₁ ≠ c₂ ⇒ c₁.name ≠ c₂.name

∀ c₁ and c₂ are unique classes

Constraint 3.1.2. The names of the attributes within a class must be unique. That is,

∀ c: Classes • ∀ att₁, att₂: c.attributes • att₁ ≠ att₂ ⇒ att₁.name ≠ att₂.name

∀ att₁ and att₂ are unique attributes in the class c

Also, classes have to follow some additional constraints, which are:

a) The names of classes have to be specified.

b) The names of the attributes of a class have to be specified.

Definition 3.2. Associations is a set of association (For simplicity, in this paper, only binary associations are considered). An association asso ∈ Associations is a tuple asso = (name, end₁, end₂), where:

a) name is the name of the association.

b) end₁ is the first end in the association. end₁ is a tuple end₁ = (class, classRole, card), where:

i. class is the associated class with the end.

ii. classRole is the role of the associated class with the end₁ in the association.

iii. card is the cardinality of the end₁. It is a range (min, max).

c) end₂ is the second end in the association. end₂ is also a tuple and it has the same elements as end₁.

Constraint 3.2.1: The names of associations do not have to be unique. However, associations between the same two classes must have different names. That is,

∀ asso₁, asso₂: Associations • {asso₁ • asso₁.end₁.class, asso₁.end₂.class} =
{asso₂ • asso₂.end₁.class, asso₂.end₂.class} ∧ asso₁ ≠ asso₂ ⇒ asso₁.name ≠ asso₂.name

∀ asso₁ and asso₂ between the same classes have different names

In addition, associations have to satisfy some other constraints, which are:

a) The associated classes with end₁ and end₂ have to be specified.

b) The roles of end₁ and end₂ within an association have to be unique.
5 TRANSFORMATION PROCESS

The transformation process is based on a set of transformation rules. Each rule takes into account one or more elements of the SBVR model and translate them into elements of the UML model. When a rule is applied it changes the current in-progress UML model to a new UML model. Each rule specifies the pre-conditions (which have to be satisfied for the rule to be applied) and the post-conditions (which specified the result of the application; i.e. the changes made to the UML model).

The transformation rules are to be interpreted as follows. Given a valid SBVR model and a valid partial UML model, which contains a subset of the features expressed in the SBVR model. Then, if the pre-conditions of a rule are satisfied, then when the transformation process applies the rule, it transforms the UML model to a new one, which remains to be valid. The proof of correctness is essentially to show that the new UML model is valid. This means we need to show that all the constraints expressed in the formal UML model must be satisfied.

5.1 Generate Classes from Object Types

Rule 1. A noun concept of ObjectType is mapped to a new class.

Args:
nc: NounConcepts // nc is a noun concept

Pre-conditions:
nc.conceptType = 'objectType' // nc is object type
nc.representation ∉ {c: Classes • c.name} // there is no class with name nc.representation

Post-conditions:
∃ newClass: Classes' • // generate a new class newClass
newClass = (nc.representation, ∅) // let the name of newClass equals to nc.representation
Classes' = Classes ∪ {newClass} // add the new class to Classes

Proof of Correctness

With the exception of constraint 3.1.1, all other constraints are satisfied because they are constraints on sets of elements that are not changed by rule 1. As for constraint 3.1.1, consider two classes c₁ and c₂ in Classes' such that c₁ ≠ c₂. Then, there are two cases:

Case 1: c₁, c₂ ∈ Classes, and then c₁.name ≠ c₂.name due to the fact that Constraint 3.1.1 is valid for the classDiagram.

Case 2: Either c₁ or c₂ does not belong to Classes. WLOG, let c₁ ∈ Classes and c₂ ∉ Classes, then,
c₁.name ∈ {x: Classes • x.name} and
c₂.name = nc.representation ∉ {x: Classes • x.name}
It follows that c₁.name ≠ c₂.name

Therefore, the constraint 3.1.1 remains valid after any execution of rule 1.

5.2 Generate Attributes from Is-property-of Fact Types

Rule 2. An is-property-of fact type is transformed to a new attribute of an existing class.

Args:
f: FactTypes // f is a fact type

Pre-conditions:
f.conceptType = 'is-property-of' // f is is-property-of fact type
f.noun₁ ∈ {nc: NounConcepts • nc.conceptType = 'objectType' • nc.representation} // noun₁ is the representation of existing object type
Postconditions:

\[ \exists c: \text{Classes} \land c.\text{name} = f.\text{originalNoun} \land f.\text{role}_2 \notin \{\text{att}: c.\text{attributes} \land \text{att}.\text{name}\} \]

// class c does not have an attribute whose name equals \(role_2\)

Proof of Correctness

After the execution of rule 2, all constraints are not affected except constraint 3.1.2 as this constraint is applied on the attributes of classes. Considering two attributes \(\text{att}_1\) and \(\text{att}_2\) for the class \(c\) such that \(\text{att}_1 \neq \text{att}_2\). Then, there are two cases:

Case 1: \(\text{att}_1\), \(\text{att}_2\) \(\in c.\text{attributes}\), and then \(\text{att}_1.\text{name} \neq \text{att}_2.\text{name}\) due to the fact that constraint 3.1.2 is valid for the classDiagram.

Case 2: Either \(\text{att}_1\) or \(\text{att}_2\) does not belong to \(c.\text{attributes}\). WLOG, let \(\text{att}_1 \in c.\text{attributes}\) and \(\text{c}_2 \notin c.\text{attributes}\), then,

\[\begin{align*}
\text{att}_1.\text{name} & \in \{x: \text{Classes} \land x.\text{attributes} \land x.\text{name}\} \\
\text{att}_2.\text{name} & = f.\text{role}_2 \notin \{x: \text{Classes} \land x.\text{attributes} \land x.\text{name}\}
\end{align*}\]

It follows that \(\text{att}_1.\text{name} \neq \text{att}_2.\text{name}\)

As a confirmation of the fulfilment of constraint 3.1.2, rule 2 applies the pre-condition \((f.\text{role}_2 \notin \{\text{att}: c.\text{attributes} \land \text{att}.\text{name}\})\), which verifies whether there is an attribute whose name equals the name of the attribute being created in the same class or not. Thus, constraint 3.1.2 is still valid after applying rule 2.

5.3 Generate Associations from Associative Fact Types

Rule 3. An associative fact type is mapped to a new association between two existing classes.

Args:

\(f: \text{FactTypes}\)  
// \(f\) is a fact type

Pre-conditions:

\(f.\text{conceptType} = \text{‘associative’}\)  
// \(f\) is associative fact type
\(f.\text{noun}_1 \in \{\text{nc}_1: \text{NounConcepts} \land \text{nc}_1.\text{conceptType} = \text{‘objectType’} \land \text{nc}_1.\text{representation}\}\)
\(f.\text{noun}_2 \in \{\text{nc}_2: \text{NounConcepts} \land \text{nc}_2.\text{conceptType} = \text{‘objectType’} \land \text{nc}_2.\text{representation}\}\)  
// \(f\) is between two noun concepts of object types
\(f.\text{noun}_1 \in \{\text{c}_1: \text{Classes} \land \text{c}_1.\text{name}\}\)
\(f.\text{noun}_2 \in \{\text{c}_2: \text{Classes} \land \text{c}_2.\text{name}\}\)  
// there are two classes \(c_1\) and \(c_2\). The name of \(c_1\) equals \(\text{noun}_1\) and the name of \(c_2\) equals to \(\text{noun}_2\)

\(\neg \exists \text{asso}: \text{Associations} \land \text{asso}.\text{name} = f.\text{verb} \land \text{asso}.\text{end}_1.\text{class.name} = f.\text{noun}_1 \land \text{asso}.\text{end}_2.\text{class.name} = f.\text{noun}_2\)  
// there is no association with the name \(f.\text{verb}\) between 
// the classes whose names equals \(f.\text{noun}_1\) and \(f.\text{noun}_2\)

Post-conditions:

\(\exists \text{newAsso}: \text{Associations’}\)  
// generate a new association \(\text{newAsso}\)
\(\text{newAsso} = (f.\text{verb}, \{\text{getClass}(f.\text{noun}_1), f.\text{role}_1, \bot\}, \{\text{getClass}(f.\text{noun}_2), f.\text{role}_2, \bot\})\)  
// let the name of \(\text{newAsso}\) equals \(f.\text{verb}\), the class whose 
// name equals \(f.\text{noun}_1\) is the class of \(\text{end}_1\), and the class 
// whose name equals \(f.\text{noun}_2\) is the class of \(\text{end}_2\)
\(\text{Associations’} = \text{Associations} \cup \{\text{newAsso}\}\)  
// add the new association to \(\text{Associations}\)
Proof of Correctness

After performing rule 3, all constraints are not changed excluding constraint 3.2.1 as it is related to associations. To satisfy this constraint, rule 3 applies a pre-condition (last pre-condition) that is used to check if there is an association with the same name and between the same classes of the association being created or not. If there is an equivalent association, rule 3 does not generate any new association. If not, a new association is created. Thus, constraint 3.2.1 remains valid after the application of rule 3.

5.4 Generate Cardinalities for Associations

Rule 4. The second quantifier in a business rule is mapped to a cardinality for an end of an association.

Args:

<table>
<thead>
<tr>
<th>br: BusinessRules</th>
<th>// br is a business rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>f: FactTypes</td>
<td>// f is a fact type</td>
</tr>
</tbody>
</table>

Pre-conditions:

- br.modal = 'necessary' // br has necessity or impossibility statement
- br.factType = f.representation // f is included in br
- f.conceptType = 'associative' // f is an associative fact type
- ∃ asso: Associations • asso.name = f.verb ∧ asso.end1.className = f.noun1 ∧ asso.end2.className = f.noun2 // there is an association with the name f.verb between the two classes whose names equal noun1 and originalNoun2

Post-conditions:

- ∃ asso: Associations, updateAsso: Associations' • asso.name = f.verb ∧ asso.end1.className = f.noun1 ∧ asso.end2.className = f.noun2 ∧ updateAsso = updateEnd2Card(asso, br.expression.quantifier2)∧ Associations' = Associations \ {asso} \ U {updateAsso} // generate a new cardinality for end2

Proof of Correctness

The only difference between the Class Diagram before and after the application of rule 4 is the updated cardinality for one end of an existing association. The expression for the updated second end of the association is (a.name, a.end1, (a.end2.class, a.end2.classRole, a.end2.card)), which is valid because card is a valid range in the SBVR model.

To ensure that all elements of the core SBVR are completely transformed, the transformation rules need to be applied in the same order demonstrated in this section. Thus, the transformation process firstly applies rule 1 for all object types then it applies rule 2 for every is-property-of fact types etc.

6 SBVR CASE STUDY

Figure 2 presents a case study that contains three sets of SBVR entries. All these sets, which are used as inputs for the transformation process, are written using the structured text style. The transformation process starts with rule 1 to transforms all object types to classes. However, object type whose general concept is 'basic data type' are not mapped to classes; instead, they are used as sources for the data types of attributes that are generated by rule 2. The next step is performing rule 2 on all is-property-of fact types to add attributes with their data types to the classes that are created by rule 1.

After generating all possible classes with their attributes, the transformation process executes rule 3 on all associative fact types to create association between the generated classes. In the last step, rule 4 generates cardinality of association ends from the quantifiers that are included in business rules. The applications of the rules for the transformation process map the SBVR model to a UML model, which can be graphically represented by the notations of the UML class diagram as shown in figure 3.
In other words, the approach that we adopt here has a very desirable advantage that is anyone familiar with design models can easily measure the capability of the selected SBVR features to express the features of the structural design models.

However, it is extremely easy to be ‘vague’ about such a subset of the SBVR specification. We address this problem by providing a formal model to capture all features of the core SBVR. Next, we show how
the core SBVR can be mapped to the UML class diagram. Again, to eliminate ambiguity, we provide a formal model for the class diagram, which clearly captures the features of the class diagram we are concerned with. The formal models, which include both the structures and constraints, allow us to formally formulate the transformation process and prove the correctness of the generated UML elements.

In addition, the formal approach that we adopt for our project provide some practical advantages as we use it as a key guidance to build our system that automates the validation of the formal models and the transformation process. Consequently, we translate the formal models of SBVR and UML as well as the transformation process directly into the components of our system. For example, two relational schemas of data repositories that store the elements of SBVR and UML are entirely derived from the structure of the formal models. Additionally, the constraints included in the formal models are translated into an engine called Validation Engine, this engine is used for validating each elements before it is stored into the right data repository. Moreover, the transformation process is converted to another engine called Transformation Engine. The transformation engine has a set of methods. Each method is derived from a formal transformation-rule and used to check the pre-conditions and ensure the post-conditions of that rule as well as generate the proper UML element. On top of the methods of the transformation engine, we implement a master method that manages the execution of the other methods and invoke them in the required order.

With the automated system, business people need only to write their business vocabulary and rule as structured text by using a template or user interfaces that impose the structure specified in the formal model of the core SBVR. When business vocabulary or rules are entered, the system validates them against their relevant constraints and transforms all valid SBVR elements to valid UML elements that compose the structural design model.

Last but not least, we evaluate our research project into three additional evaluation methods. Firstly, we employ sets of test cases. Each set is developed for a specific SBVR element and includes a number of test cases that examine all valid and invalid possibilities for that element. Secondly, we again use a case study (with large size) defined in the SBVR specification and map it to the UML class diagram manually by using our transformation process as well as the other existing transformation methodologies. Then, we compared the results of our transformation with the results of the existing methodologies by using precision and recall measures. Finally, we reuse the large size case study to generate the UML class diagram by the developed prototype. Then, we compared the results of the developed prototype with the results of our manual mapping.

8 CONCLUSION AND FUTURE WORK

This paper has illustrated a core part of a long-term research project: to identify a subset of the SBVR specification that expresses the main features of the structural design model, and to provide a formal framework to denote the defined subset of the SBVR specification and to transform them into the structural design models in the form of the UML class diagrams. The defined subset of the SBVR specification can be beneficial for business people to define their business vocabulary and rules that express structural design models fully and unambiguously. The proposed transformation process can be very useful for information system developers to automate the transformation of the SBVR specification into the structural design models. Therefore, the formal model of the SBVR subset along with the transformation process can play a significant role to decrease the communication gap between business people and information system developers in designing and modelling the structure specification. And this can significantly improve the quality and robustness of information systems.

In the future, we plan to investigate the issue of managing changes that would be made on the identified subset of the SBVR specification (by business people) and how these changes can be traced and propagated correctly to the affected elements of the UML class diagram. We expect that our formalized transformation process can play a critical role for the managing changes task.
References


