Kaunas University of Technology

Department of Information Systems

Creating UML&OCL Models from SBVR Business Vocabularies and Business Rules

VeTIS User Guide

Kaunas, 2009

Lithuanian State Science and Studies Foundation High Technology Development Program Project “Business Rule Solutions for Information Systems Development (VeTIS)" (Reg.No. B-07042)

Table of Contents

Vision: from Natural Language to UML / OCL Models

The vision of OMG "Semantics of Business Vocabulary and Business Rules" (SBVR) standard is expressing business knowledge in controlled natural language unambiguously understandable by human and computer systems. Such knowledge is captured by business experts / information system analysts who need tools that would allow storing SBVR specifications in MOF repositories for interchanging and linking them with other models based on MOF (e.g. UML&OCL models). SBVR is fully integrated into the OMG's Model-Driven Architecture (MDA) (Fig. 1).

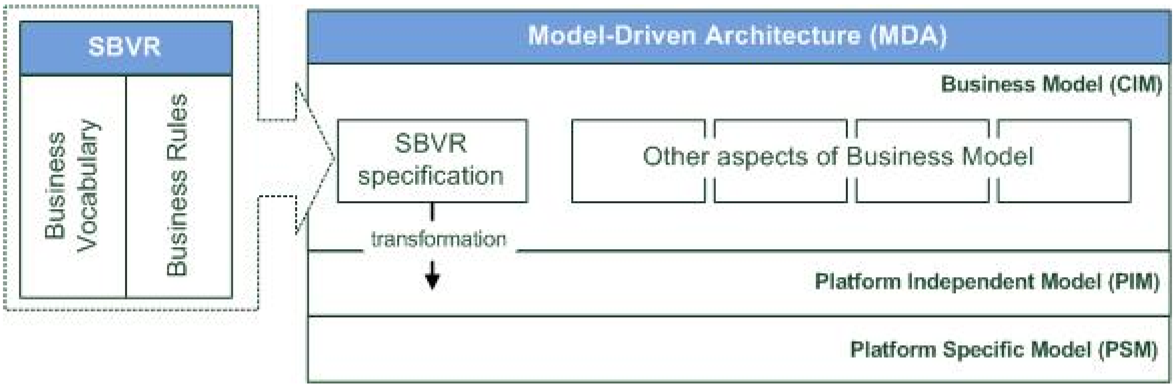

SBVR
Business Vocabulary
Business Rules
Model-Driven Architecture (MDA)
Business Model (CIM)
SBVR specification
Other aspects of Business Model
transformation
Platform Independent Model (PIM)
Platform Specific Model (PSM)

Fig. 1. SBVR position in MDA [2]

The purpose of VeTIS tool is to support transformation SBVR → UML&OCL. It was implemented on the Eclipse 3.4.1 platform on the basis of SBVR 1.0 and UML 2.1.2 metamodels (in XMI format), ATL transformation language 3.0.0 and ATL transformation engine 3.0.0. VeTIS user interface was adapted from SBeaVeR tool [3].

Brief Introduction to SBVR

“Semantics of Business Vocabulary and Business Rules” (SBVR) is an OMG specification for expressing **business knowledge** in the language that is primarily **understandable to the business people**. One can state that SBVR is aimed at helping business people understand models with no special knowledge in modeling notations or IT skills.

The most common way of expressing business vocabularies and business rules is by using textual specifications rather than some diagrams. While diagrams are helpful when one needs to see how business concepts are related, they are not well suited for defining vocabularies and expressing rules. **SBVR uses controlled natural language** (e.g. English) for business model specification.

SBVR realizes the core principle of the Business Rules Approach at the business level, so called **business rules “mantra”** that has been introduced by Business Rules Group in 1995. This principle can be abbreviated as follows:

> “Business rules are based on facts, and facts are based on terms”.

Following the business rules “mantra”, SBVR introduces its basic elements:

– a **noun concept** (individual concept, object type (i.e. general concept) or role);

– a **fact type,** which denotes some type of relationship between two or more noun concepts (e.g. “person owns account”), or a characteristic of the noun concept (e.g. “person is reliable”);

– a **business rule,** which is defined as a rule that is under business jurisdiction. Business rules are formulated as necessities, possibilities, permissions, prohibitions or obligations related to fact types. For example, the rule “It is necessary that a person owns at most 5 accounts” is based on the fact type “person owns account”.

SBVR organizes business knowledge into **vocabularies.** There are two meta-models defined in the form of vocabularies in the SBVR specification:

– **Vocabulary for Describing Business Vocabularies**. A business vocabulary is defined to contain “all the specialized terms and definitions of concepts that a given organization or community uses in their talking and writing in the course of doing business" [1];

– **Vocabulary for Describing Business Rules**, which is built on the Vocabulary for Describing Business Vocabularies and deals with the specification of the meaning of business rules.

What SBVR is and is NOT

– It is about and for the business, not information system (IS). However, SBVR defines business knowledge that can be used in information systems development;

– It is from the business perspective, not from the IS perspective;

– It uses natural language used by business people; it has no reference to any IS constructs and is independent of any IS design. However, SBVR propagates the use of structured, well defined natural language that can be transformed into IS design decisions;

– It is created and maintained by business people, and not IT people. However, IT people are able to understand SBVR specifications as well, therefore, they could manage SBVR business vocabularies and business rules if required.

VeTIS Tool – What, Where and How?

WHAT is VeTIS tool?

VeTIS tool is an SBVR-compliant plug-in for the CASE tool MagicDraw UML, but also it can be used as a standalone tool.

The user interface of VeTIS was developed on the basis of SBeaVeR an Eclipse plug-in, which was a part of the Digital Business Ecosystem project [2].

WHERE to use VeTIS tool?

VeTIS tool is used for the definition of Business Vocabularies and Business Rules using controlled natural language (a subset of English language) and for transformation of SBVR specifications into UML class diagrams with OCL constraints.

HOW to use VeTIS tool?

Basic scenario of working with the VeTIS tool is shown in Figure 2.

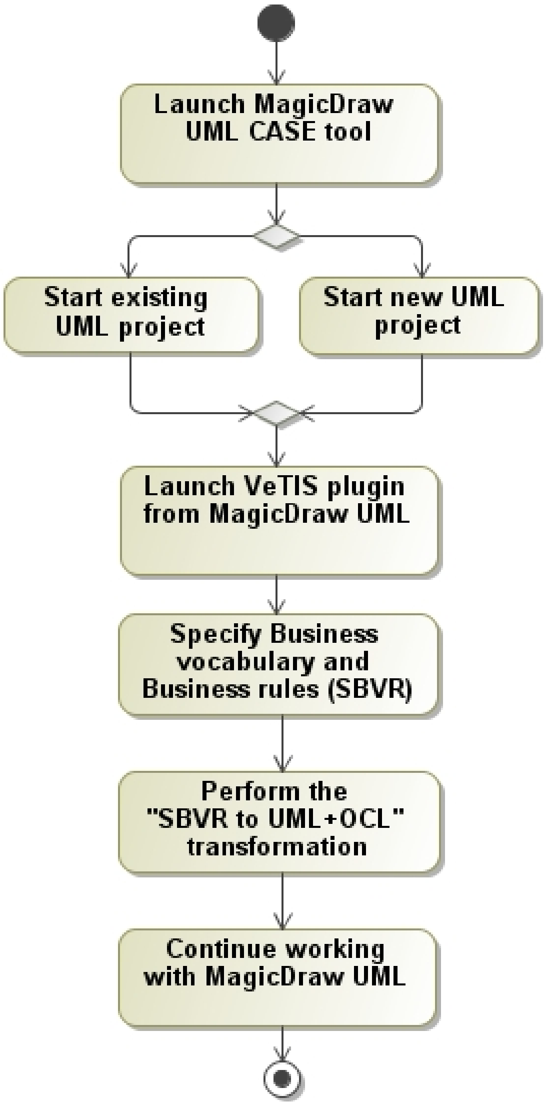

Launch MagicDraw UML CASE tool
Start existing UML project
Start new UML project
Launch VeTIS plugin from MagicDraw UML
Specify Business vocabulary and Business rules (SBVR)
Perform the "SBVR to UML+OCL" transformation
Continue working with MagicDraw UML

Fig. 2. Working with the VeTIS tool: basic scenario

One can launch the MagicDraw UML tool, launch VeTIS editor from MagicDraw UML menu and specify terms and fact types in Business Vocabulary. Having all necessary terms and fact types you will be able to define business rules in Business Rules Vocabulary and to transform the overall specification into UML class diagram with OCL constraints.

SBVR Expressions in Structured English

Font styling for the SBVR Structured English

Four types of font styles are used for specifying business terms, fact types and business rules in the SBVR-based Structured English:

Font	Description
term	The "term" font is used to represent **object types (general concepts)** and **roles**. Terms are defined in singular form using lower case letters. Examples: car, driver, loan, modal formulation, fact type etc.
Name	The "Name" font is used to represent **individual concepts** that usually are proper nouns. The first letter of a name is capitalized. One of the exceptions to the latter is the presentation of numerical values that are also shown in this style (e.g. 25). Examples: Lithuania, Kaunas, USA, 200 etc. **Note:** The "Name" font used by VeTIS tool slightly differs from the original SBVR notation and looks like this: Lithuania.
verb	The "verb" font is used to represent a **verb**, a **preposition**, or a **combination** of these two. Verbs are used in singular active or passive forms – these two are used as synonymous forms. For example, for the active form of an associative fact type "driver *drives* car" there is a synonymous passive form "car *is driven by* driver". Fact types, representing characteristics, are always used in passive form, e.g. "car *is damaged*". **Note:** In VeTIS tool, you should hyphen combinations of words playing the roles of verbs, terms or Names, e.g.: *is_driven_by*, approved_order, Lithuanian_Republic.
keyword	The "keyword" font represents **linguistic symbols** that are used to construct statements and definitions. Examples: each, It is obligatory that, greater than, " " etc. **Note**: combinations of words representing keywords usually have no hyphens.

Keywords and phrases for logical formulations

Keywords and phrases are used to express **logical formulations**. The letters "*n*" and "*m*" represent integers, and "*p*" and "*q*" – expressions of propositions.

Quantification	
each	universal quantification
at least one	existential quantification
at least *n*	at-least-n quantification
at most one	at-most-one quantification
at most *n*	at-most-n quantification
exactly one	exactly-one quantification
exactly *n*	exactly-n quantification
at least *n* and at most *m*	numeric range quantification
Logical operations	
not *p*	logical negation
p and *q*	conjunction
p or *q*	disjunction
Modal operations	
it is obligatory that *p*	obligation formulation
it is necessary that *p*	necessity formulation
it is possible that *p*	possibility formulation
it is permitted that *p*	permissibility formulation

Other keywords

Keyword	Description
the	1. Used with a general concept to make a reference to a previous use of the same concept; 2. Introduction of the name of an individual thing or of a definite description.
a, an	Universal or existential quantification
that	When used after a noun concept and before a fact type symbol, this keyword introduces some restriction on that noun concept.
not	Used within an expression to introduce a logical negation.

Phrases having predefined meaning in VeTIS

Phrase	Description
Is_greater_than *is_not_greater_than* *is_less_than* *is_not_less_than* *equals*	Used for comparisons in logical formulations
is_a *specializes* *generalizes* *is_category_of* *is_of_category*	Used for categorization fact types
Is_included_in *includes*	Used for partitive fact types
has *Is_property_of*	Used for *is_property_of* fact types

Defining Business Vocabulary with VeTIS Tool

What is Business Vocabulary?

According to the SBVR specification, a business vocabulary "contains all the specialized terms, names, and fact type forms of concepts that a given organization or community uses in their talking and writing in the course of doing business" [1].

Describing Business Vocabulary

A business vocabulary is described by the vocabulary's name and several optional fields that are presented below.

Field	Description
Vocabulary name	The "Name" font is used to present the vocabulary name.
Description:	The scope and purpose of the vocabulary is presented in the "Description:" field.
Source:	The "Source:" field presents a glossary or any other formally-defined document, which is independent of the vocabulary being described and is used as a basis for that vocabulary.
Speech Community:	The "Speech Community:" field is used to present the speech community that controls and is responsible for the vocabulary.
Language:	The "Language:" field is used to name the language that is the basis of the vocabulary. By default, English is assumed.
Included Vocabulary:	The "Included Vocabulary:" field indicates that another vocabulary is fully incorporated into the vocabulary being described. All the entries of the included vocabulary become a part of the vocabulary being described. **Note**. Currently this option is not supported in VeTIS.
Note:	The "Note:" field is used to present any other notes that do not fit under the other captions.

Business Vocabulary Entries

Business vocabulary has glossary-like entries that specify concepts having representations in the vocabulary. **Each entry is for a single concept**.

Each vocabulary entry starts with a primary presentation, which denotes a name of the concept. Additionally, a concept can be defined by other optional fields that are presented in the skeleton of a vocabulary entry below.

Field	Description
primary presentation	A primary representation of an entry can be a term, a name, or a fact type form (in SBVR, fact type form is the representation for a fact type).
Definition:	A definition is understood as an expression that defines the primary representation. A definition can be formal, partly formal or informal. Formal definitions are in SBVR-styled text, and informal definitions are unstyled.
Source:	The "Source:" field is used to indicate the source vocabulary or document for a concept. Keyword phrase "based on" can be used in this field to indicate that the definition of the concept is derived from the given source but now has some modifications.
Dictionary Basis:	This field presents a definition from a common dictionary that supports the use of the primary representation.
General Concept:	The "General Concept:" field can be used to present a concept that generalizes the entry concept.
Concept Type:	The "Concept Type:" field is used to specify the type of the entry concept when the type of the concept is not obvious from the primary representation (e.g. a Name indicates an Individual concept, a Term – object_type etc.).
Necessity: Possibility:	The "Necessity:" or "Possibility:" field is usually used to supplement a definition. A "Necessity" states something that is necessarily true – the necessary conditions for the existence of the concept. A "Possibility" states something that is possible and not prevented by definition. Typical keyword phrases "it is necessary that" and "it is possible that" can be omitted from a statement because it is implied by the field itself.
Reference Scheme:	The "Reference Scheme:" field shows structuring of things denoted by the concept. It defines the sufficient conditions for the existence of the concept.

Note:	Some explanatory notes that do not fit within other captions may be presented in the “Note:” field.
Example:	Some examples involving the entry concept may be presented in the “Example:” field.
Synonym:	A synonym is another designation of the same concept that can be substituted for the primary representation. Synonyms represent the same concept. Individual concepts and object types may have synonyms.
Synonymous Form:	A synonymous form is a fact type form for the same fact type. E.g. “customer *buys* book” and “customer *purchases* book” are two synonymous forms for the same fact type.
See:	The “See:” field is used when the primary representation is not the preferred representation for the entry concept. The “See:” field introduces the preferred representation.

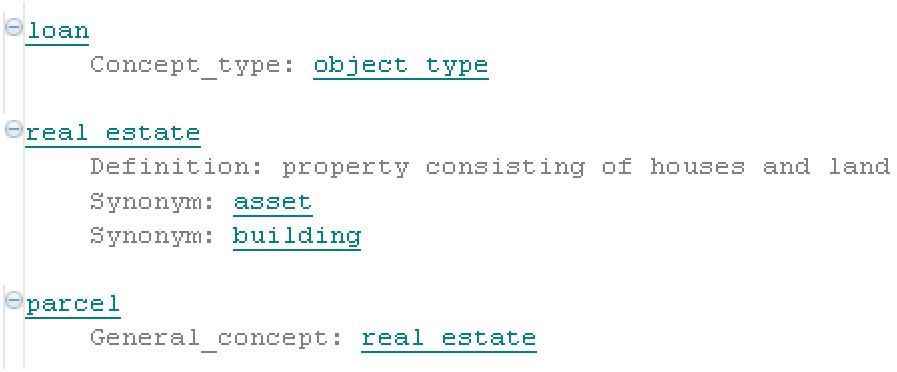
Defining Concepts in a Business Vocabulary

There are five basic **concept types** that can be presented in a Business vocabulary using VeTIS tool:

- **Object type** (or general concept)**;**
- **Individual concept**;
- **Fact type** (or verb concept);
- **Role**.

> An **object type** is a noun concept that classifies things on the basis of their common properties.

An object type is presented in the “Term” font and starts with a small letter. A vocabulary entry for an object type can be additionally specified using the description fields presented above. Some examples of the object type entries:

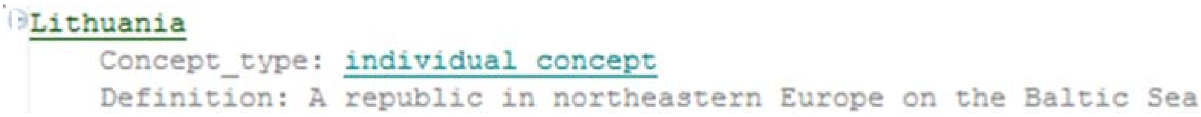


```
loan
    Concept_type: object type

real estate
    Definition: property consisting of houses and land
    Synonym: asset
    Synonym: building

parcel
    General_concept: real estate
```

Note that the field “Concept_type:” in the vocabulary entry “loan” is not necessary because the system interprets the concept as an object type by default (like in the “real_estate” case), if no other specifications are applied.

> An **individual concept** is a concept that corresponds to only one object (thing).

An individual concept is presented in the “Name” font and can be additionally specified using description fields presented above:

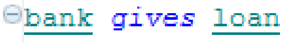


```
Lithuania
     Concept_type: individual concept
     Definition: A republic in northeastern Europe on the Baltic Sea
```

Note that the field “Concept_type:” in the vocabulary entry “Lithuania” is not necessary because the name of the concept starts with a capital letter, which indicates the concept being an individual concept.

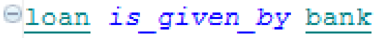
> A **fact type** is a concept that denotes some type of relationship between two or more noun concepts or a characteristic of the noun concept.

Following the definition, fact types are defined using the existing noun concepts (object types or individuals), which have already been defined in the business vocabulary. Verbs represent fact types creating relationships between these nouns or specifying their characteristics. Verbs are presented in the “*verb*” font. A simple example of the fact type:



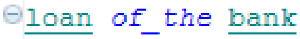
```
bank gives loan
```

The given example is a **sentential form** of the fact type. Sentential forms may be **active** or **passive** (the previous example presents a fact type in an active form). Fact types may also be in **noun forms**. The following examples illustrate fact type in its passive and noun forms:



```
loan is_given_by bank
```

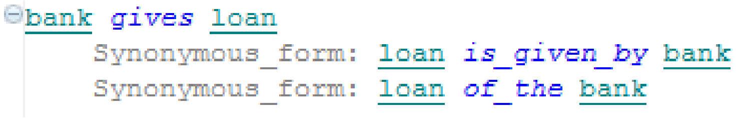
– passive form for the fact type “bank *gives* loan”;



```
loan of_the bank
```

– noun form for the same fact type (noun form is identified by the verb phrase “*of_the*”).

Note that all these forms are synonymous forms of one and the same fact type “bank *gives* loan”:



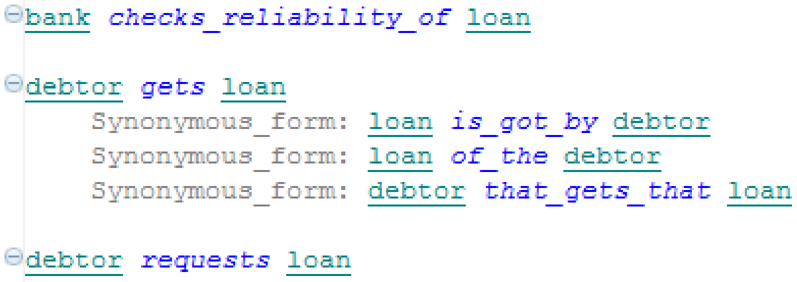
```
bank gives loan
    Synonymous_form: loan is_given_by bank
    Synonymous_form: loan of_the bank
```

Mostly, noun forms of fact types are used in business rules statements.

That is not all about the fact types. There's a **classification** for the fact types. Let us introduce it with some examples made with the VeTIS tool:

Associative fact type

An associative fact type is the most common fact type that has two or more roles involved (VeTIS supports only two roles for associative fact types). Usually, there are two noun concepts playing specific roles in the relationship defined by the associative fact type. Only in the case of a recursive relationship, there is one noun concept playing both roles.

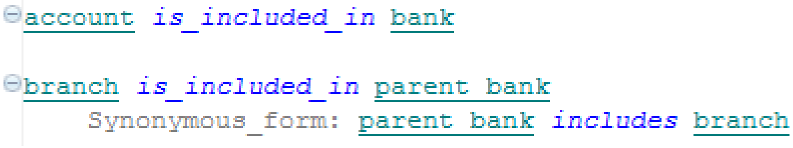


```
bank checks_reliability_of loan

debtor gets loan
    Synonymous_form: loan is_got_by debtor
    Synonymous_form: loan of_the debtor
    Synonymous_form: debtor that_gets_that loan

debtor requests loan
```

Associative fact type is identified by a verb or a verb phrase, which is not reserved for other types of fact types.

Partitive fact type

Partitive fact type is a binary fact type stating that one noun concept (all of its instances) is in the composition of a given whole, i.e. another noun concept.

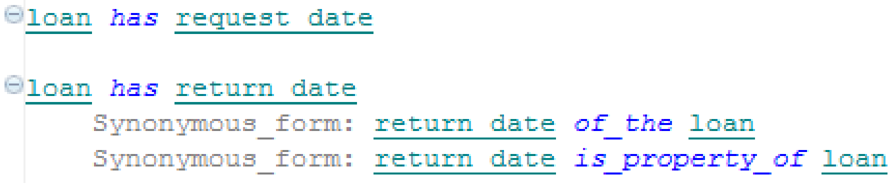


```
account is_included_in bank

branch is_included_in parent bank
    Synonymous_form: parent bank includes branch
```

Partitive fact types are identified by the verb phrases "*includes*" and "*is_included_in*" (for active and passive forms respectively).

Is_property_of fact type

Is_property_of fact type defines an essential quality of a given noun concept.



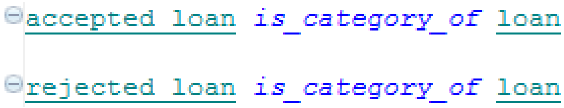
```
loan has request date

loan has return date
     Synonymous_form: return date of_the loan
     Synonymous_form: return date is_property_of loan
```

Is_property_of fact types are identified by the verb phrases "*has*" and "*is_property_of*" (for active and passive forms respectively).

Categorization fact type

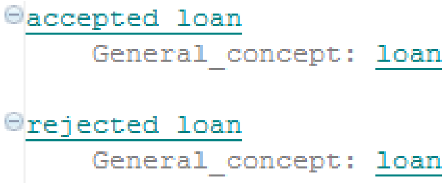
Categorization fact type is a fact type that represents relationship between the more general noun concept and another (more specific) noun concept, which is a category of the first concept.

With VeTIS tool, one can specify simple categorization fact types, for example:



```
accepted loan is_category_of loan

rejected loan is_category_of loan
```

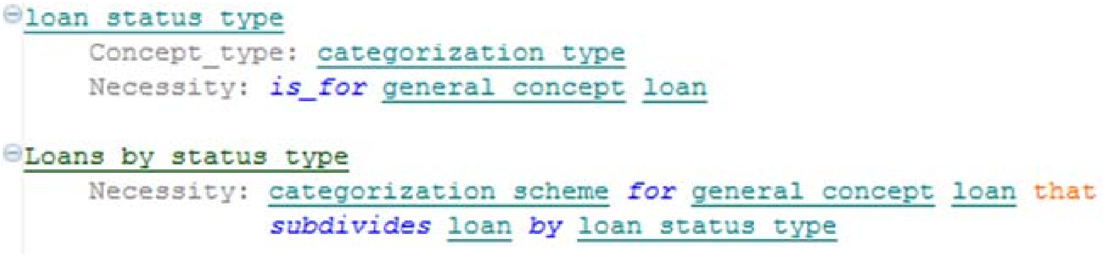
Note that such simple categorization fact types can also be represented in noun concept entries with additionally specified "General_concept:" fields indicating the more general concepts of those noun concepts:



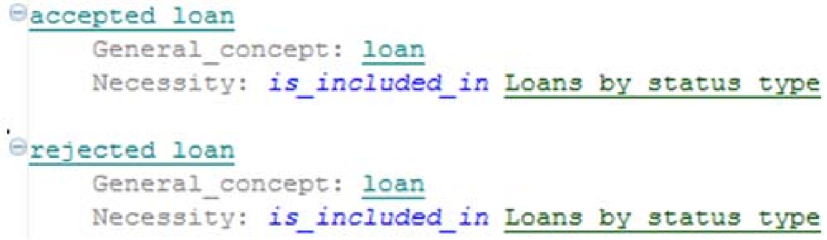
```
accepted loan
     General_concept: loan

rejected loan
     General_concept: loan
```

Simple categorization fact types are identified by the following pairs of verbs and verb phrases: "*specializes*" and "*generalizes*", "*is_category_of*" and "*is_of_category*", "*is_a*" and "*is_a*" (VeTIS interprets *is_a* as a relationship between a specific concept and the general concept, and creates the synonymous form with the verb phrase *is_a* between the general concept and the specific concept by default).

More complex structures involving categorization types and categorization schemes are also supported by VeTIS:



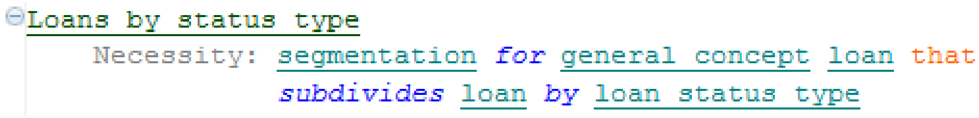
```
loan status type
     Concept_type: categorization type
     Necessity: is_for general concept loan

Loans by status type
     Necessity: categorization scheme for general concept loan that
                subdivides loan by loan status type
```



```
accepted loan
    General_concept: loan
    Necessity: is_included_in Loans by status type

rejected loan
    General_concept: loan
    Necessity: is_included_in Loans by status type
```

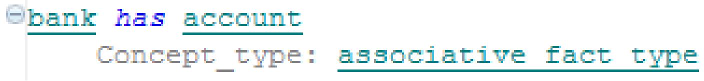
Categorization scheme is a set of categories that subdivides instances of a general concept into subsets specialized by some feature (categorization type). Note that the above-shown structure of the vocabulary entries for the categorization type, categorization scheme and specialized noun concepts is **predefined** and **mandatory**, if you want to specify the complete and correct information about a categorization scheme.

A specialized case of a categorization scheme is **segmentation**. Segmentation is a categorization scheme whose contained categories comprise complete set of categories (with respect to the general concept), and sets of objects belonging to these categories are disjoint. Vocabulary entry for segmentation has a structure analogous to categorization scheme:



```
Loans by status type
    Necessity: segmentation for general concept loan that
               subdivides loan by loan status type
```

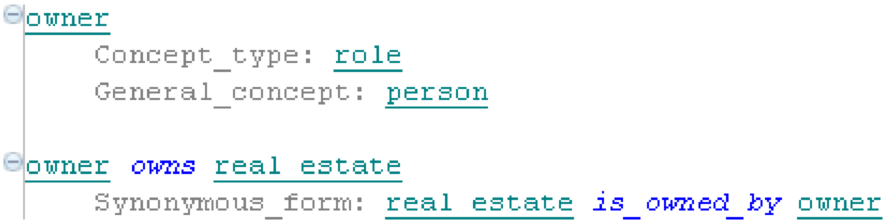
Changing the meaning of a predefined verb. If you want to use the verb phrase reserved for the some category of fact type in a fact type of a different type, specify that type of the fact type in the "Concept_type:" field. VeTIS will ignore the default meaning of the verb for this particular vocabulary entry. In the following example, fact type will be interpreted as an associative fact type even though the verb "*has*" is reserved for the is_property_of fact types:



```
bank has account
    Concept_type: associative fact type
```

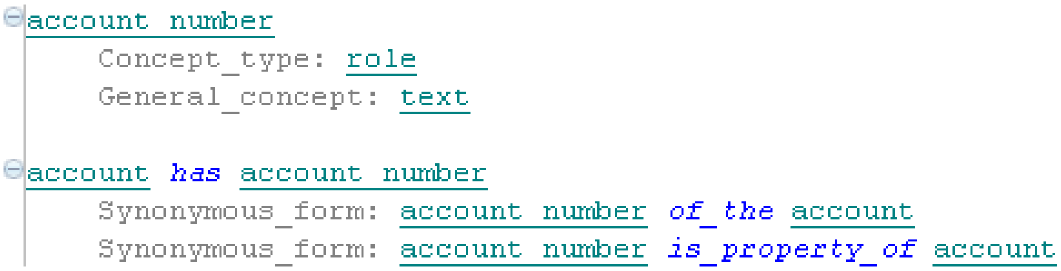
A **role** is a noun concept that corresponds to things based on their playing a part, assuming a function or being used in some particular situation.

A role is always understood with respect to actualities of a particular fact type or to other particular situations. For example, let us assume the role "owner" – this is the role played by a person in the particular fact type that is also specified in the business vocabulary:



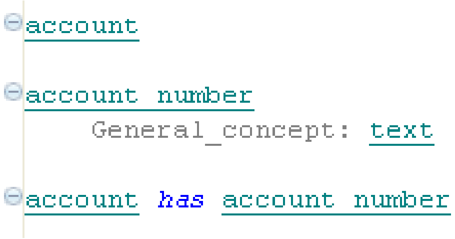
```
owner
     Concept_type: role
     General_concept: person

owner owns real estate
     Synonymous_form: real estate is_owned_by owner
```

A role is identified by the vocabulary entry, which has the field “Concept_type:” set to “role”, and the “General_concept:” field set to the object type, which plays that role in the corresponding fact type. Roles having predefined **elementary object types** text, integer and number as their general concepts are used in *is_property_of* fact types:



```
account number
     Concept_type: role
     General_concept: text

account has account number
     Synonymous_form: account number of_the account
     Synonymous_form: account number is_property_of account
```

Note. It is unnecessary to define “Concept type: role” for roles that are used in *is_property_of* fact types, i.e.:



```
account

account number
     General_concept: text

account has account number
```

Defining Business Rules with VeTIS Tool

According SBVR, business rules are rules under business jurisdiction. **Business rules** are constructed as closed modal formulations that have recursively embedded logical formulations and are based on **fact types**. Modal formulations are:

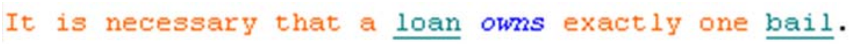
- Alethic modal formulations, i.e. necessities, possibilities or impossibilities used in structural business rule statements expressing structural business rules;
- Deontic modal formulations, i.e. obligations, permissions or prohibitions used in operative business rule statements expressing operative business rules.

Structural Business Rule: These are rules about how the business chooses to organize (i.e., “structure”) the things it deals with. Structural Rules are constraints and derivations, and supplement definitions.

Operative Business Rule: These are rules that govern the conduct of business activity (dynamic or action rules). In contrast to Structural Rules, Operative Rules can be *directly* violated by people involved in the affairs of the business.

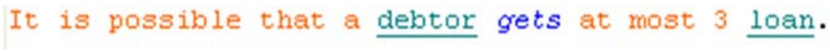
VeTIS supports four types of business rules (you can define impossibilities and prohibitions using remaining types of rules):

Necessities:



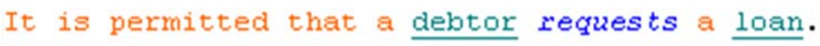
```
It is necessary that a loan owns exactly one bail.
```

Possibilities:



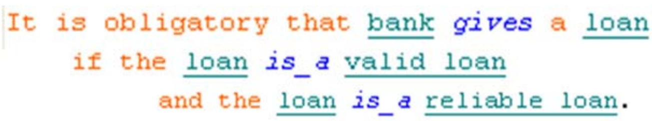
```
It is possible that a debtor gets at most 3 loan.
```

Permissions:



```
It is permitted that a debtor requests a loan.
```

Obligations:



```
It is obligatory that bank gives a loan
     if the loan is_a valid loan
            and the loan is_a reliable loan.
```

You can find more information about defining business rules in the upcoming sections. Keywords and phrases for logical formulations and predefined fact types are presented in the section “SBVR Expressions in Structured English”.

Transforming the Business Vocabulary into UML Class Model

One of the core features of the VeTIS tool is the transformation of the business vocabulary and business rules into the UML Class model.

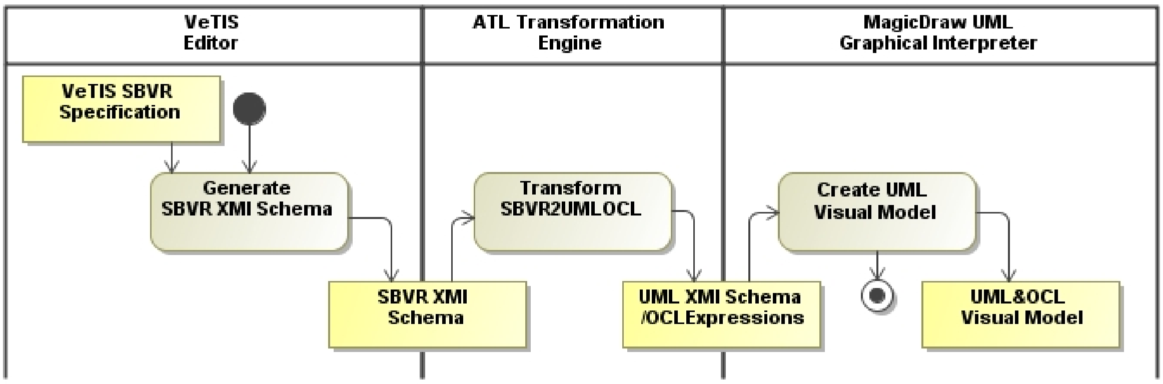
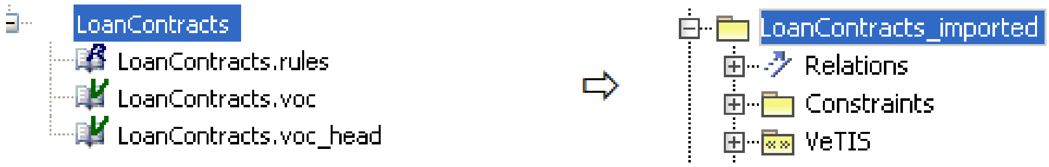


Fig. 3. Transforming SBVR specifications to UML&OCL models

Let us demonstrate basic transformation rules with some examples.

Transformation of the Vocabulary

Business Vocabulary along with Business Rules is transformed into a UML package, while vocabulary name is transformed into a package name concatenated with the phrase "_imported". Two elements are included in that newly created package: "VeTIS" profile that supports visualization of constraints in UML class diagrams, and "Constraints" package for holding OCL constraints obtained from SBVR. "VeTIS" profile includes single stereotype <<constrained>>, base metaclass of which is "Element", so it is applicable for classes, operations and other UML elements.

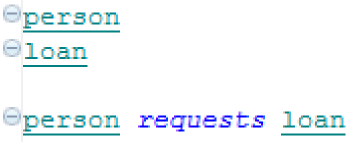


Transformation of an **object type**

Object type is transformed into a UML class holding the same name as the object type:

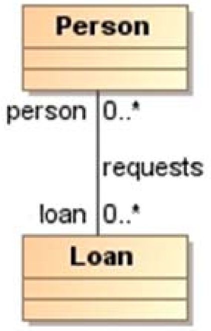
Transformation of an **associative fact type**

Associative fact type is transformed into an association relationship between two classes. Associative fact type involves two roles of object types whose names become names of the ends of properties corresponding to the ends of that association. A verb (or a verb phrase) used by the fact type becomes a name of that association. If there are no business rules constraining the number of occurrences of instances of a fact type, multiplicity bounds of the association are set to “0..*” by default.

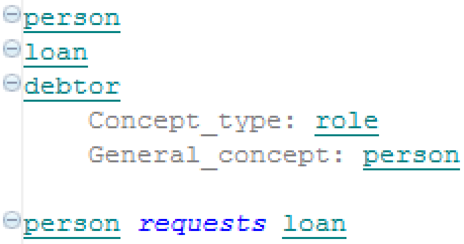


```
person
loan

person requests loan
```

⇨

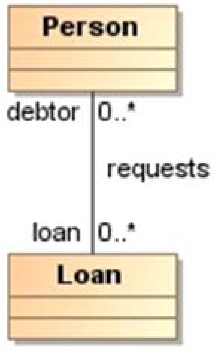


Object types may play particular roles in associative fact types. In such a case, the names of these roles map to the names of properties corresponding to the ends of the association:



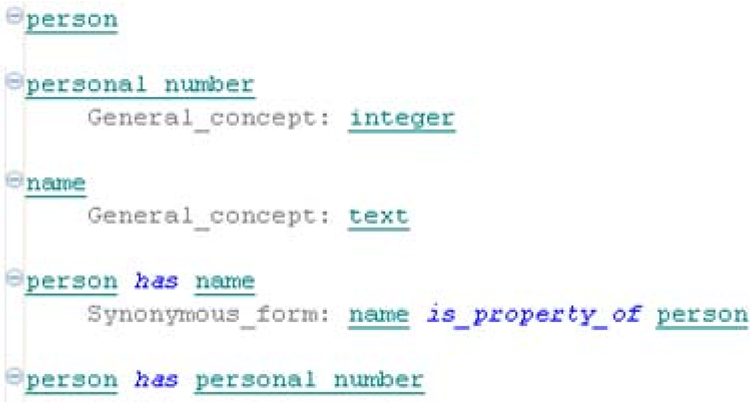
```
person
loan
debtor
     Concept_type: role
     General_concept: person

person requests loan
```

⇨

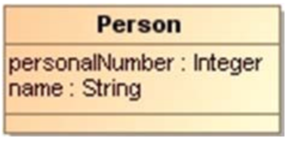


Transformation of **is_property_of fact type**

Is_property_of fact type is transformed into an attribute of a class. A general concept describing some property of another general concept is specified by a vocabulary entry having the “Concept_type:” field set to “role” and “General_concept:” – to some data type. A data type specifies what kind of data that property of a business object (general concept) represents:

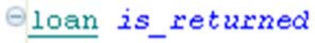


```
person

personal number
    General_concept: integer

name
    General_concept: text

person has name
    Synonymous_form: name is_property_of person

person has personal number
```

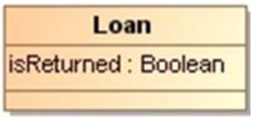


Transformation of a **characteristic fact type**

Characteristic (fact type having only one role) is transformed into an attribute of the Boolean type:

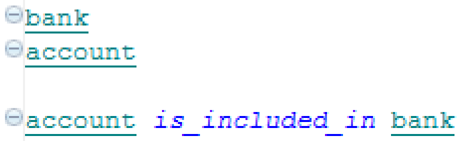


```
loan is_returned
```

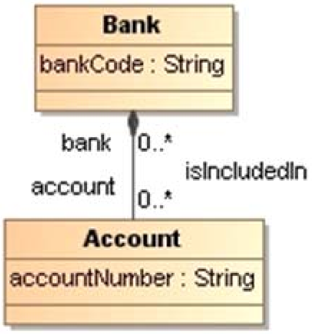


Transformation of a **partitive fact type**

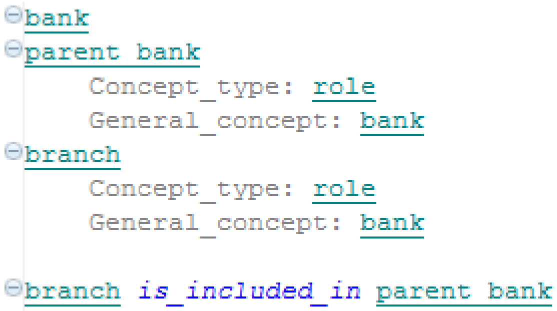
Partitive fact type is transformed into a composition relationship between two classes in an analogous way:



```
bank
account

account is_included_in bank
```

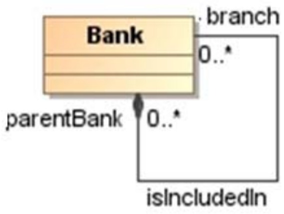


In the following example of a partitive fact type, the fact type holds two specific roles ("parent_bank" and "branch") of the single general concept "bank". This fact type is transformed into the recursive composition (note that both roles ranging over the same object type are possible for associative fact types as well):



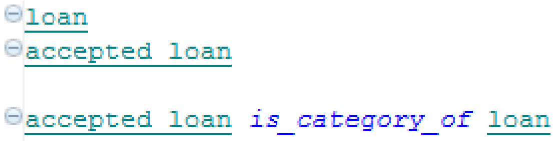
```
⊖bank
⊖parent bank
     Concept_type: role
     General_concept: bank
⊖branch
     Concept_type: role
     General_concept: bank

⊖branch is_included_in parent bank
```

⇨



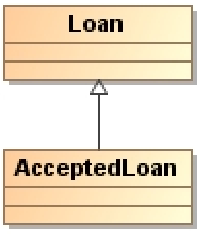
Transformation of a **categorization fact type**

Categorization fact type is transformed into a generalization relationship between two classes:

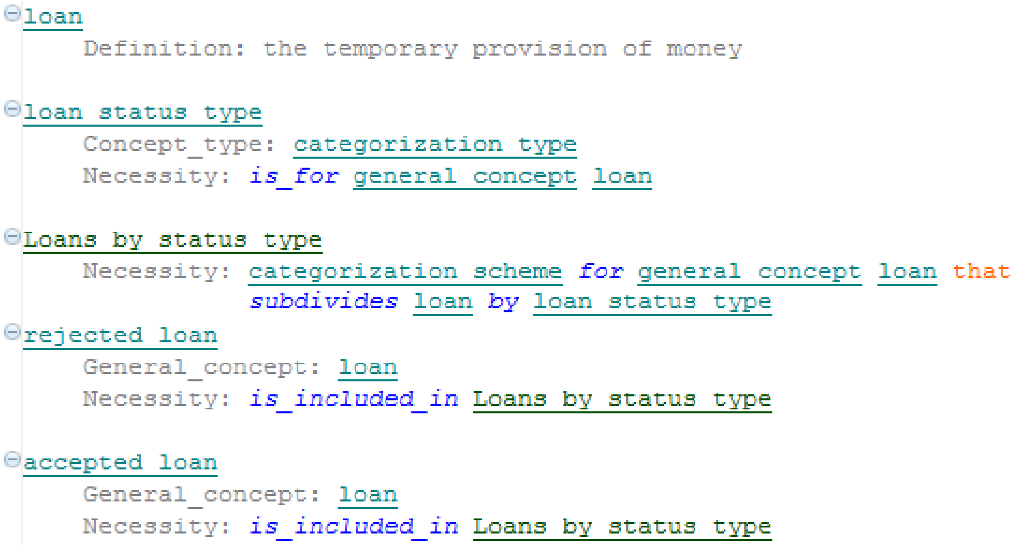


```
⊖loan
⊖accepted loan

⊖accepted loan is_category_of loan
```

⇨

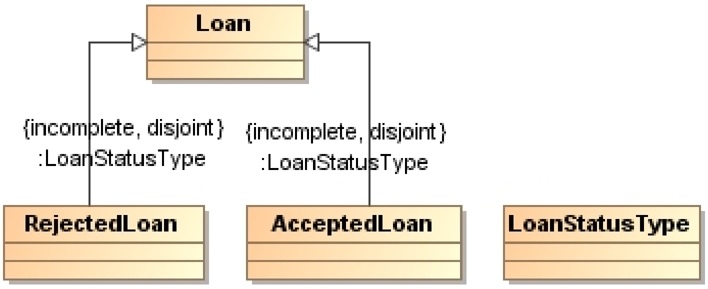


VeTIS makes transformations of more complex SBVR constructions involving categorization types and schemes as well. In the following example, a generalization set involving the superclass “Loan”, two subclasses “AcceptedLoan” and “RejectedLoan” as well as the class “LoanStatusType” representing a powertype is created in MagicDraw UML from the categorization scheme specified in a business vocabulary:

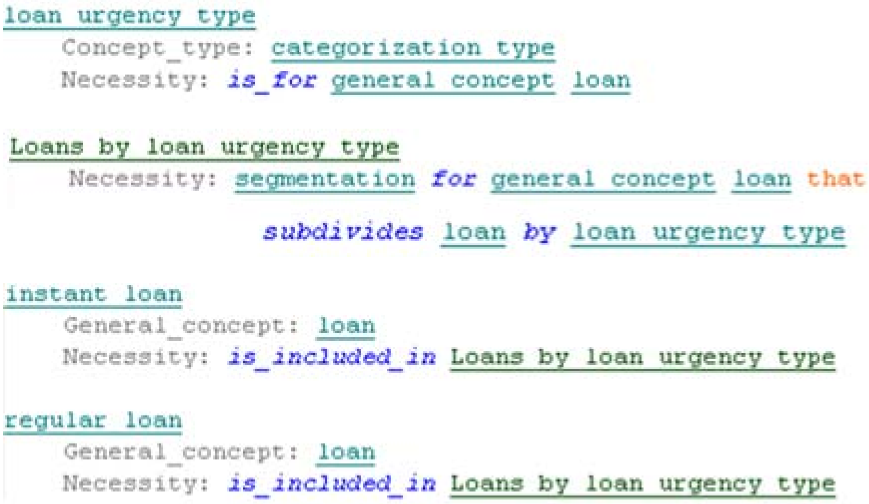


```
⊖loan
     Definition: the temporary provision of money

⊖loan status type
     Concept_type: categorization type
     Necessity: is_for general concept loan

⊖Loans by status type
     Necessity: categorization scheme for general concept loan that
                subdivides loan by loan status type
⊖rejected loan
     General_concept: loan
     Necessity: is_included_in Loans by status type

⊖accepted loan
     General_concept: loan
     Necessity: is_included_in Loans by status type
```

⇩

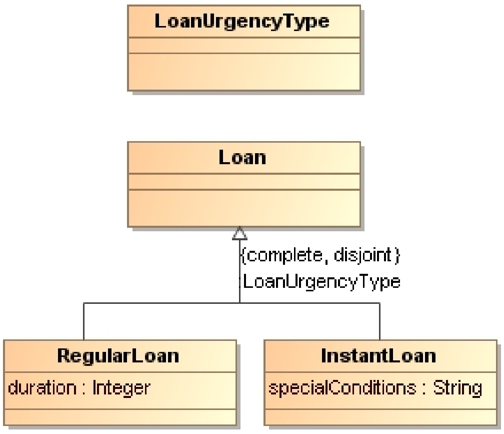


Note that in cases where segmentation is used instead of categorization scheme generalization relationships are generated with the {complete, disjoint} constraints instead of the {incomplete, disjoint} as shown in the previous example where the categorization scheme was used.



```
loan urgency type
    Concept_type: categorization type
    Necessity: is_for general concept loan

Loans by loan urgency type
    Necessity: segmentation for general concept loan that

                subdivides loan by loan urgency type

instant loan
    General_concept: loan
    Necessity: is_included_in Loans by loan urgency type

regular loan
    General_concept: loan
    Necessity: is_included_in Loans by loan urgency type
```

⇩

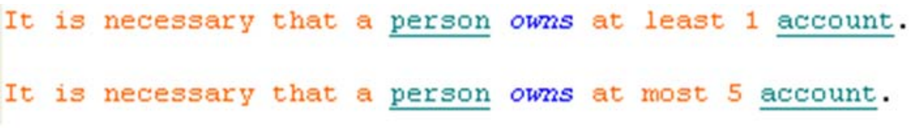


Transforming Business Rules into UML Class Model

Some types of business rules are transformed into UML class diagrams.

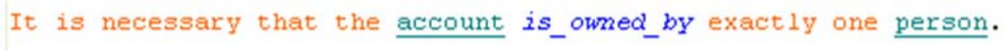
Transforming business rules into **multiplicity bounds**

Structural business rules formulated by alethic modal formulations having directly embedded quantifiers are transformed into multiplicity bounds of the corresponding associations or compositions:



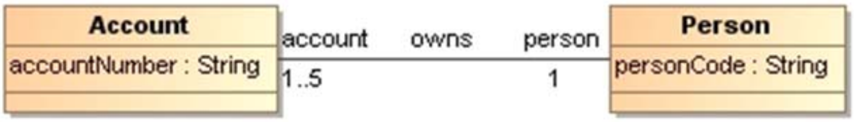
It is necessary that a person owns at least 1 account.

It is necessary that a person owns at most 5 account.

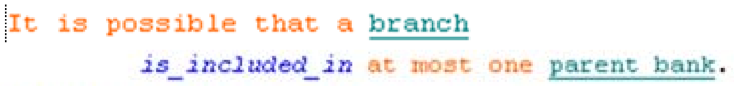


It is necessary that the account is_owned_by exactly one person.

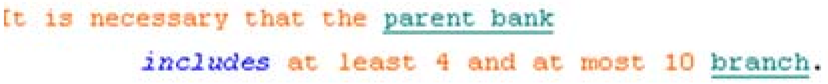
⇩



SBVR structural business rules can to define the overall variety of multiplicity bounds:

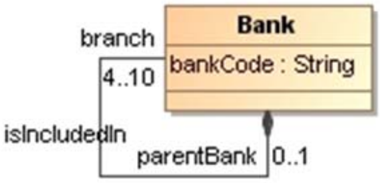


It is possible that a branch
is_included_in at most one parent bank.



It is necessary that the parent bank
includes at least 4 and at most 10 branch.

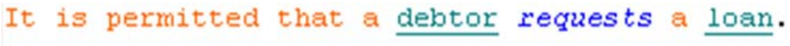
⇩



Transforming business rules into **operations**

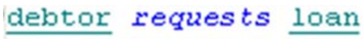
Operative business rules (obligations and permissions) formulated as closed deontic formulations are transformed into operations. Verbs denoting fact types on which these formulations are based are transformed into operation names, roles – into parameter names, object types playing these roles – into parameter types.

For example, a business rule:



It is permitted that a debtor requests a loan.

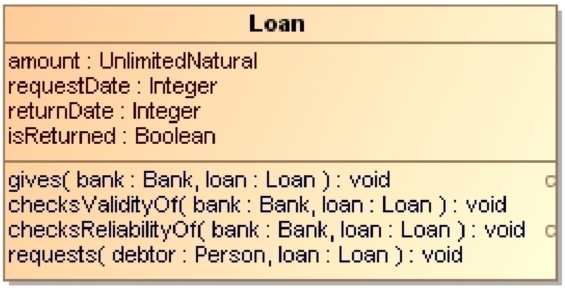
is based on fact type:



debtor requests loan

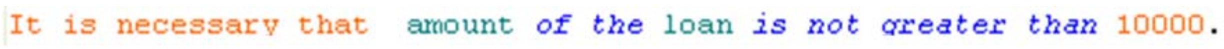
and it is transformed into the following operation:

```
Loan::requests(debtor:Person, loan:Loan):void
```


Loan
amount : UnlimitedNatural
requestDate : Integer
returnDate : Integer
isReturned : Boolean
gives(bank : Bank, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : void
checksReliabilityOf(bank : Bank, loan : Loan) : void
requests(debtor : Person, loan : Loan) : void

Transforming Business Rules into OCL Constraints

Transforming business rules into class **invariants expressing integrity constraints**

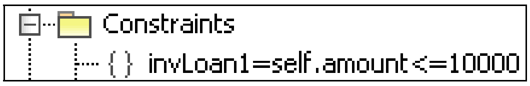


It is necessary that amount of the loan is not greater than 10000.

⇩

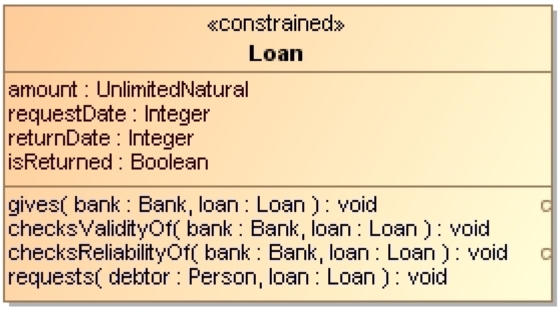
```
Context Loan
inv: self.amount<=1000
```

Constraints are created in the package “Constraints”.

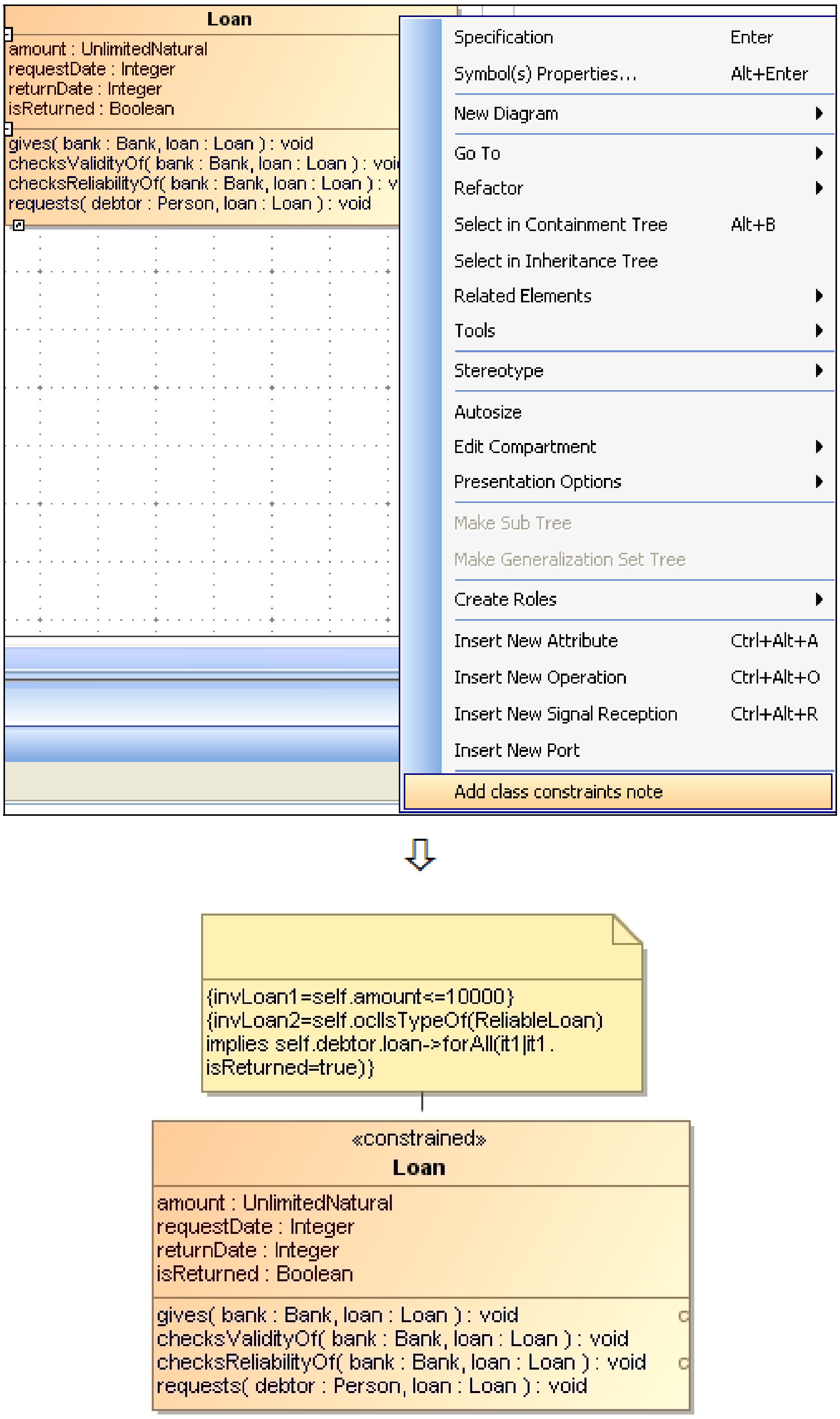


Constraints
{ } invLoan1=self.amount<=10000

Constraint names are unique. The name of a constraint consists of the part indicating the constraint type name (“inv” for invariants, “pre” for preconditions”), class or operation names and a sequence number. Stereotype <<constrained>> marks classes having related OCL constraints:


«constrained»
Loan
amount : UnlimitedNatural
requestDate : Integer
returnDate : Integer
isReturned : Boolean
gives(bank : Bank, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : void
checksReliabilityOf(bank : Bank, loan : Loan) : void
requests(debtor : Person, loan : Loan) : void

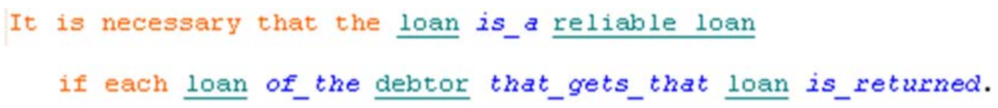
You can visualize constraints by clicking “Add class constraint note” on the right-click class menu:


Loan
amount : UnlimitedNatural
requestDate : Integer
returnDate : Integer
isReturned : Boolean
gives(bank : Bank, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : voi
checksReliabilityOf(bank : Bank, loan : Loan) : v
requests(debtor : Person, loan : Loan) : void
Specification Enter
Symbol(s) Properties... Alt+Enter
New Diagram
Go To
Refactor
Select in Containment Tree Alt+B
Select in Inheritance Tree
Related Elements
Tools
Stereotype
Autosize
Edit Compartment
Presentation Options
Make Sub Tree
Make Generalization Set Tree
Create Roles
Insert New Attribute Ctrl+Alt+A
Insert New Operation Ctrl+Alt+O
Insert New Signal Reception Ctrl+Alt+R
Insert New Port
Add class constraints note
«constrained»
Loan
amount : UnlimitedNatural
requestDate : Integer
returnDate : Integer
isReturned : Boolean
gives(bank : Bank, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : void
checksReliabilityOf(bank : Bank, loan : Loan) : void
requests(debtor : Person, loan : Loan) : void

```
{invLoan1=self.amount<=10000}
{invLoan2=self.oclIsTypeOf(ReliableLoan)
implies self.debtor.loan->forAll(it1|it1.
isReturned=true)}
```

Transforming business rules into **class invariants expressing derivation rules**

Structural business rules (necessities), formulated as alethic formulations with embedded implications, are transformed into OCL invariants expressing derivation rules:



It is necessary that the loan is_a reliable loan

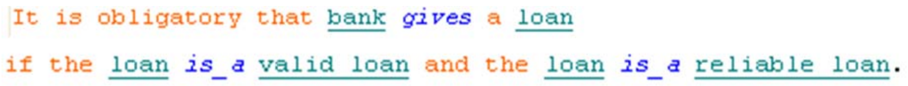
if each loan of_the debtor that_gets_that loan is_returned.

⇩

```
Context Loan
inv: self.oclIsTypeOf(ReliableLoan)implies
      self.debtor.loan◊forAll(it1|
            it1.isReturned=true)
```

Transforming business rules into operation **preconditions**

Operative business rules (obligations and permissions), formulated as deontic formulations with embedded implications, are transformed into operations and operation preconditions. Implication consequent is transformed into operation, antecedent – into precondition:



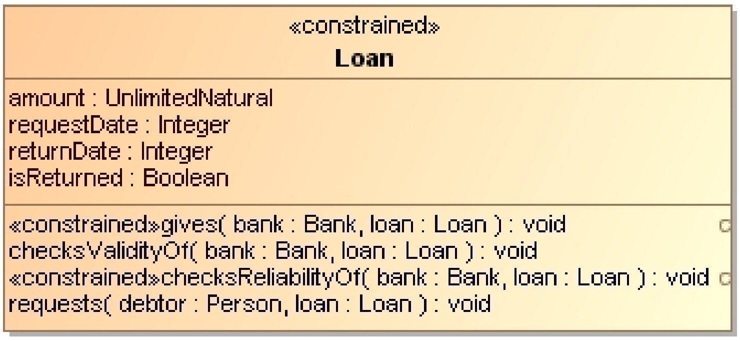
It is obligatory that bank gives a loan

if the loan is_a valid loan and the loan is_a reliable loan.

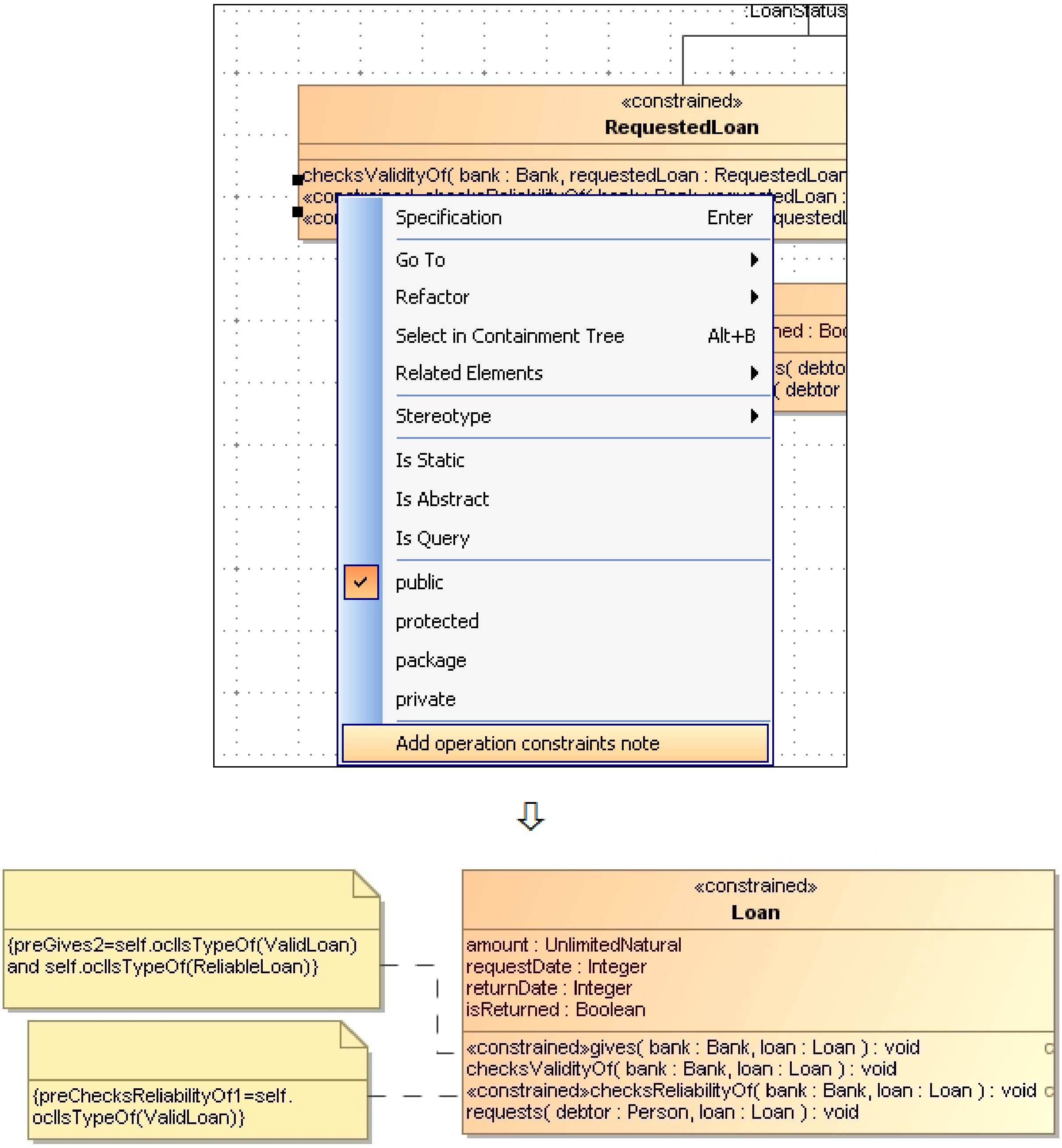
⇩

```
Context Loan::gives(ban:Bank,loan:Loan):OclVoid
   pre: self.oclIsTypeOf (ValidLoan) and
      self.oclIsTypeOf(ReliableLoan)
```

Stereotype <<constrained>> marks operations having related OCL preconditions.


«constrained»
Loan
amount : UnlimitedNatural
requestDate : Integer
returnDate : Integer
isReturned : Boolean
«constrained»gives(bank : Bank, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : void
«constrained»checksReliabilityOf(bank : Bank, loan : Loan) : void
requests(debtor : Person, loan : Loan) : void

You are able to visualize operation constraints by clicking the “Add operation constraints note” on the operation right-click menu:


:LoanStatus
«constrained»
RequestedLoan
checksValidityOf(bank : Bank, requestedLoan : RequestedLoan
Specification Enter
Go To
Refactor
Select in Containment Tree Alt+B
Related Elements
Stereotype
Is Static
Is Abstract
Is Query
public
protected
package
private
Add operation constraints note
⇩
{preGives2=self.oclIsTypeOf(ValidLoan) and self.oclIsTypeOf(ReliableLoan)}
{preChecksReliabilityOf1=self. oclIsTypeOf(ValidLoan)}
«constrained»
Loan
amount : UnlimitedNatural
requestDate : Integer
returnDate : Integer
isReturned : Boolean
«constrained»gives(bank : Bank, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : void
«constrained»checksReliabilityOf(bank : Bank, loan : Loan) : void
requests(debtor : Person, loan : Loan) : void

There are further possibilities to automatically obtain attribute default values, operation results and post conditions from SBVR definitions. Though mathematical calculations are difficult to express in SBVR, the standard is extensible. Therefore, such improvements depend on the willingness of developers.

Development Process Based on SBVR

The process of defining business vocabularies and business rules is not easy. It is difficult to formulate consistent and complete sets of concepts and propositions governing business. Furthermore, SBVR specifications are of declarative nature and they define just business constraints – constraints on structure and on activities, but not on control flows of these activities, i.e. business processes. Though it is possible to predefine sequences of activities by declarative constraints or even extend SBVR for specifying business processes, such a practice is not recommended by business rule methodologists. In contrast, they propose “separation of concerns” – keeping models of business processes and specifications of business rules separately, not intertwining them, because they are changing independently. Also, visual modelling is better suited for definition of business processes. Additionally, we argue that modelling of business processes helps define right and consistent business rules as well. Let’s take a look at how all this integrates into the software development process (Fig. 4).

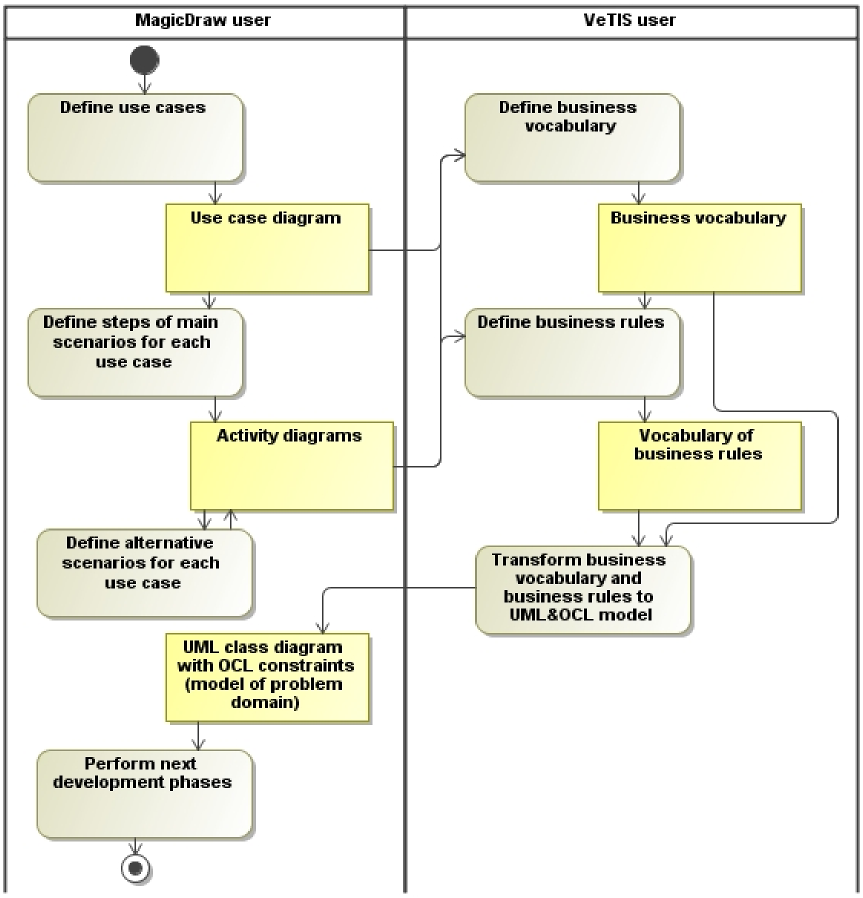

MagicDraw user
VeTIS user
Define use cases
Use case diagram
Define steps of main scenarios for each use case
Activity diagrams
Define alternative scenarios for each use case
UML class diagram with OCL constraints (model of problem domain)
Perform next development phases
Define business vocabulary
Business vocabulary
Define business rules
Vocabulary of business rules
Transform business vocabulary and business rules to UML&OCL model

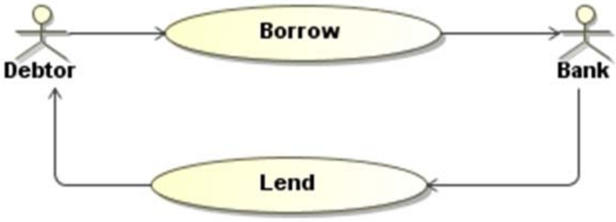
Fig. 4. Development process using SBVR business vocabularies and business rules

In the presented process, we recommend writing software requirements (i.e. use case specifications) in alignment with defining business processes (e.g. in the form of BPMN or UML activity diagrams), business vocabularies and business rules.

We will describe the process with the simplified Loan Contracts example. Let's assume we need to develop software for the information system in which persons (debtors) would be able to borrow money from a bank, which in turn could lend it. Arguably, the bank would be interested in checking validity and reliability of requested loans; then it should take obligations to give loans, if they meet defined conditions, and debtors should assume obligations to return received loans. We will go over the steps of the presented development process and explain it in detail.

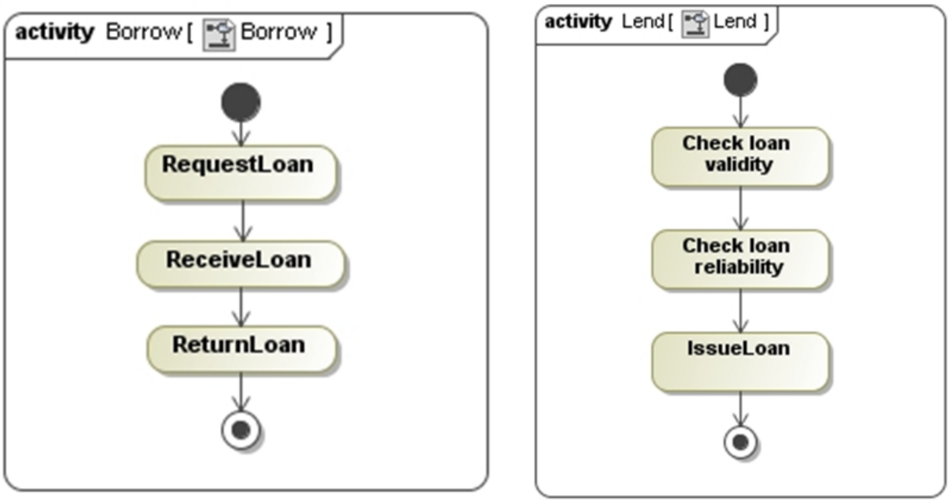
1st step. Define use cases

Use cases represent functional requirements for the system under development. We launch MagicDraw UML CASE tool and create the project with the following use case diagram:



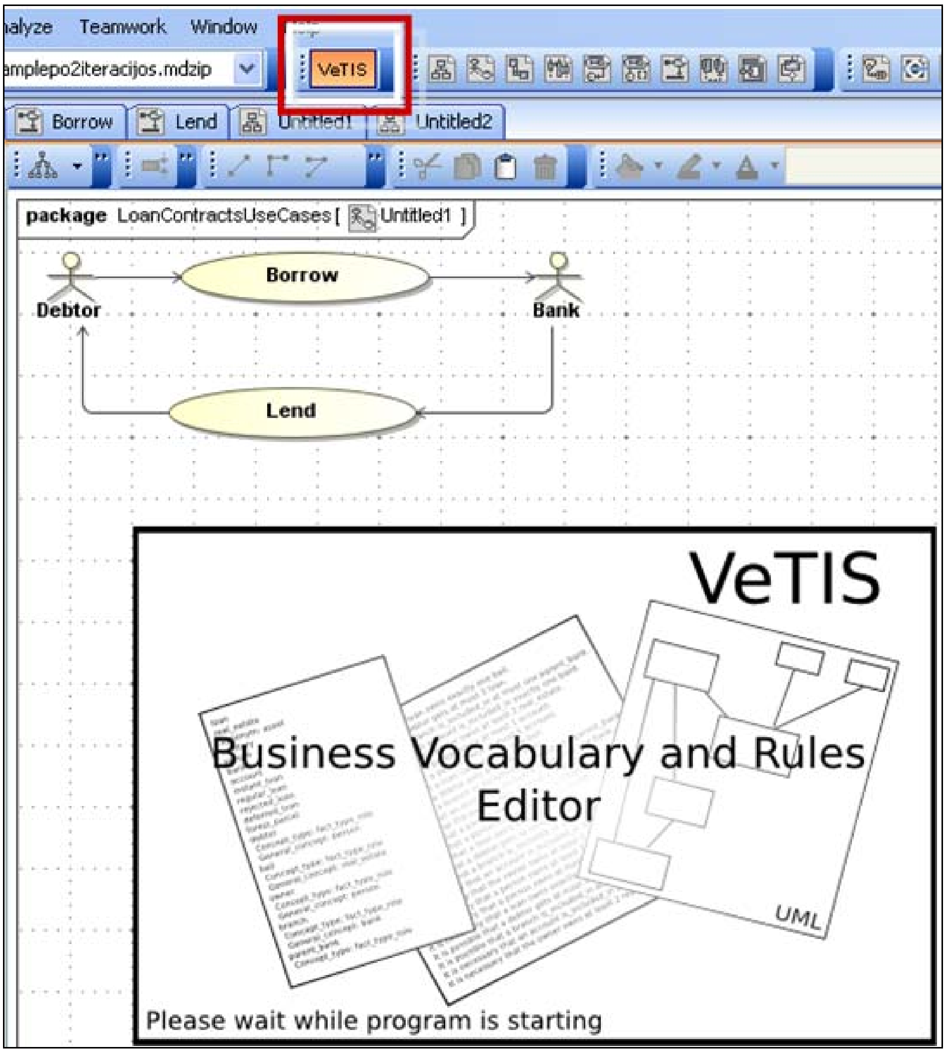
2nd step. Define steps of main scenarios of each use case

Initially, we define straightforward processes representing steps of main success scenarios. We draw a UML activity diagram for each use case and represent our desired business processes:



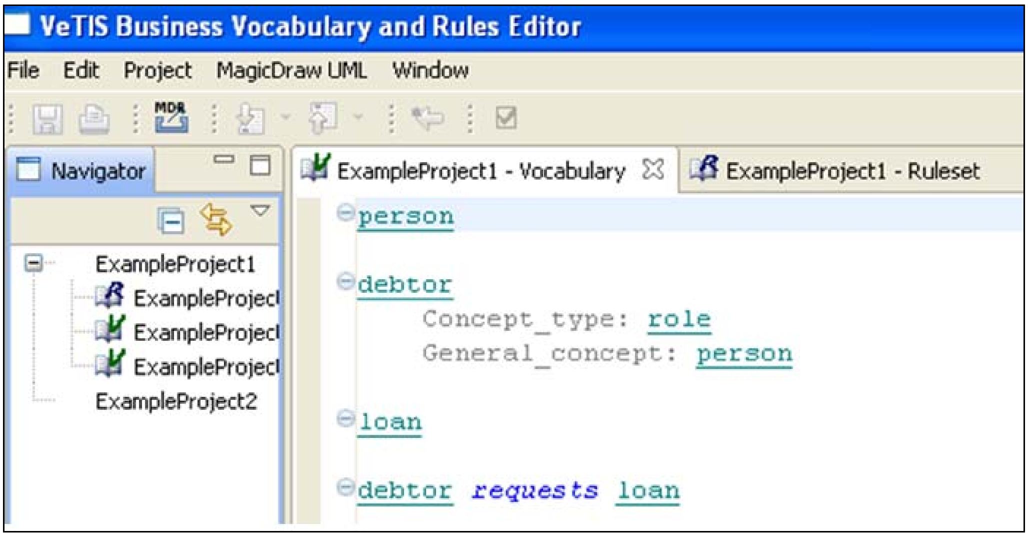
3rd step. Define business vocabulary

Now we are launching the VeTIS editor from the MagicDraw UML tool menu:



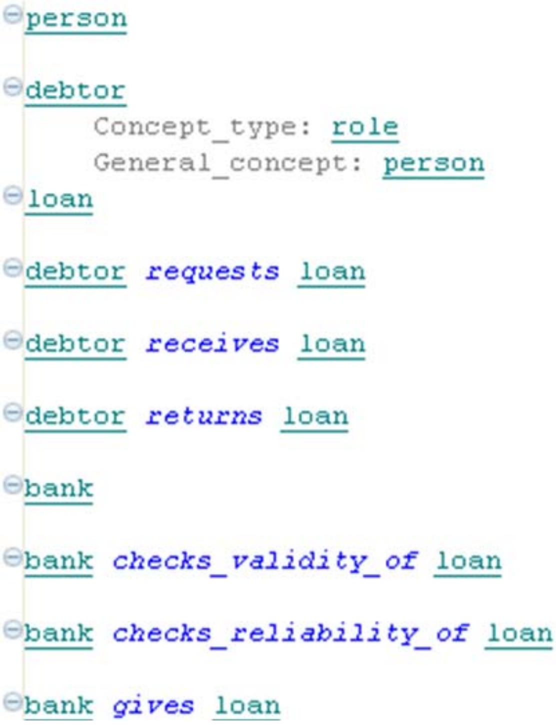
Please wait while program is starting

When VeTIS editor is opened, we create a new project (e.g. ExampleProject1) and define the use case scenario steps as fact types in the Business Vocabulary:



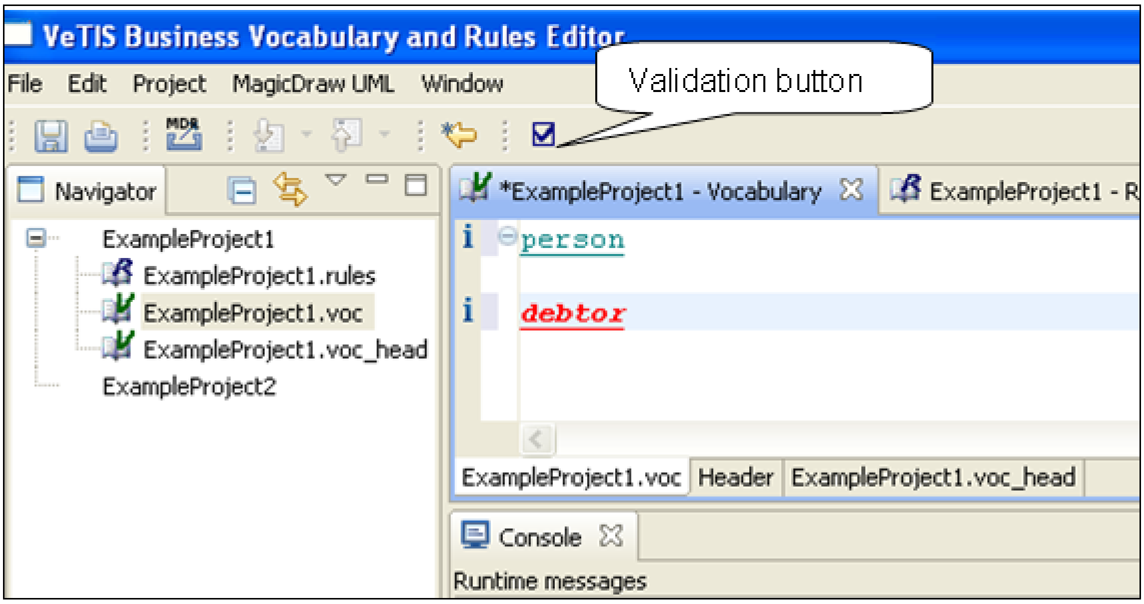
```
person

debtor
    Concept_type: role
    General_concept: person

loan

debtor requests loan
```

To be able to do this we have to define all terms (object types and roles), on which fact types are based.



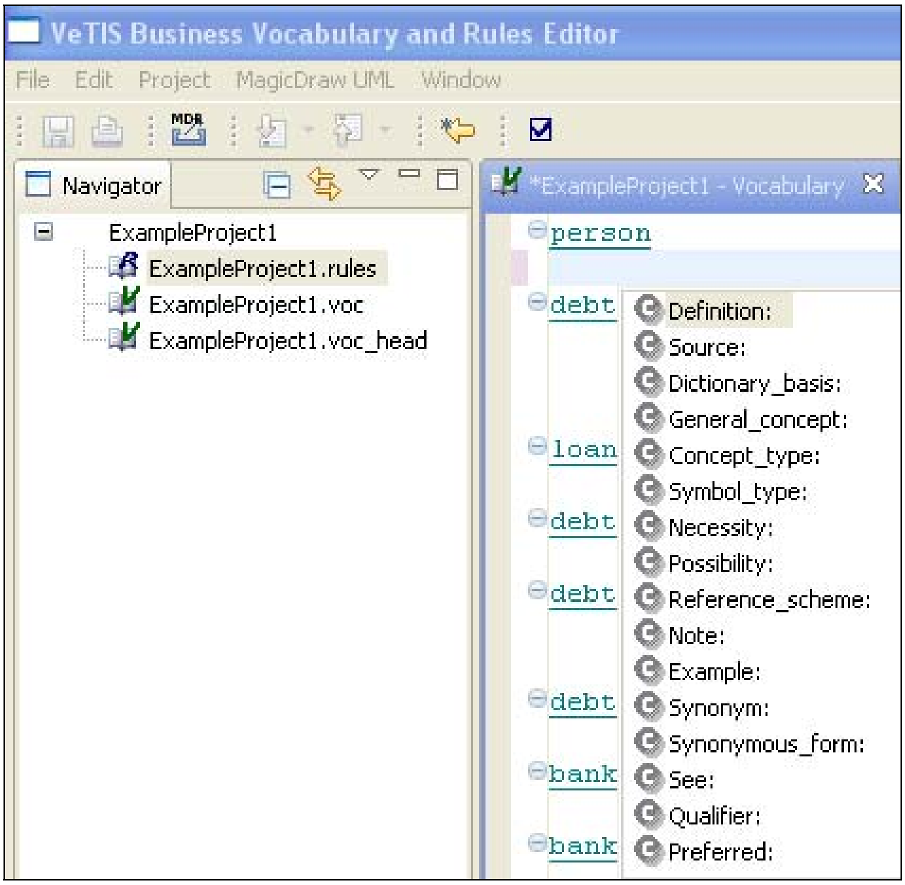
```
person

debtor
      Concept_type: role
      General_concept: person
loan

debtor requests loan

debtor receives loan

debtor returns loan

bank

bank checks_validity_of loan

bank checks_reliability_of loan

bank gives loan
```

You will see that validation is required before introducing a new concept. You can validate concepts by clicking the validation button:

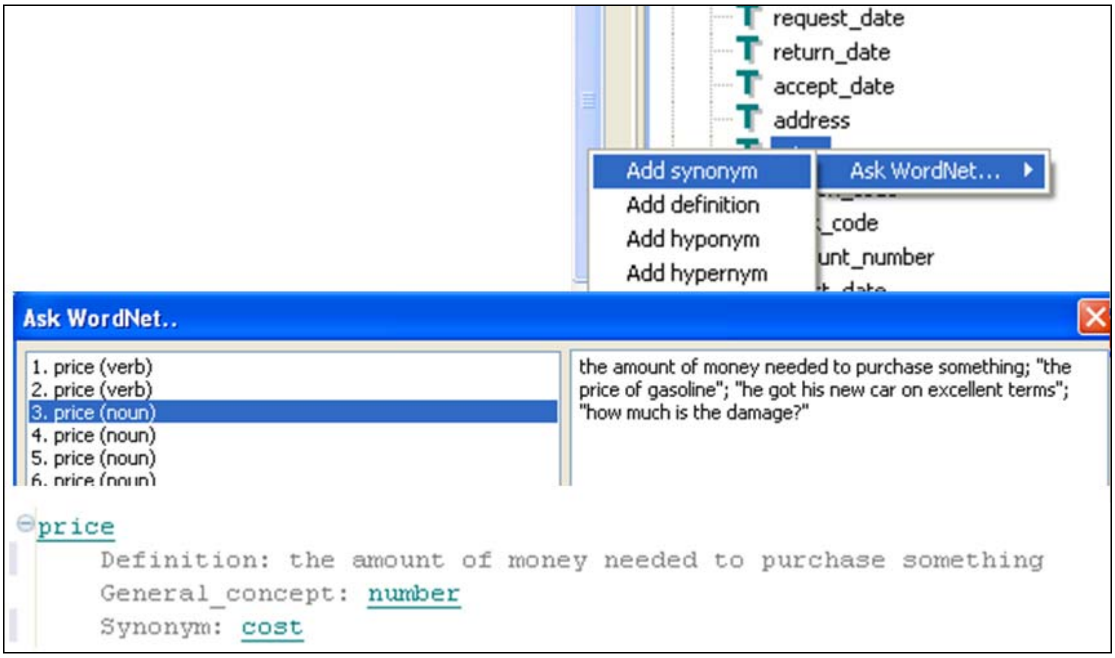


```
i person

i debtor
```

Each concept is defined by its name (the primary representation of a concept) and can have (optional) definitions (definitions that are required for the particular concepts have already been explained in the section “Defining Business Vocabulary”):



Sometimes you can borrow definitions or synonyms from WordNet by clicking “Ask WordNet” on the corresponding term displayed in the layout on the right side of the VeTIS editor window:

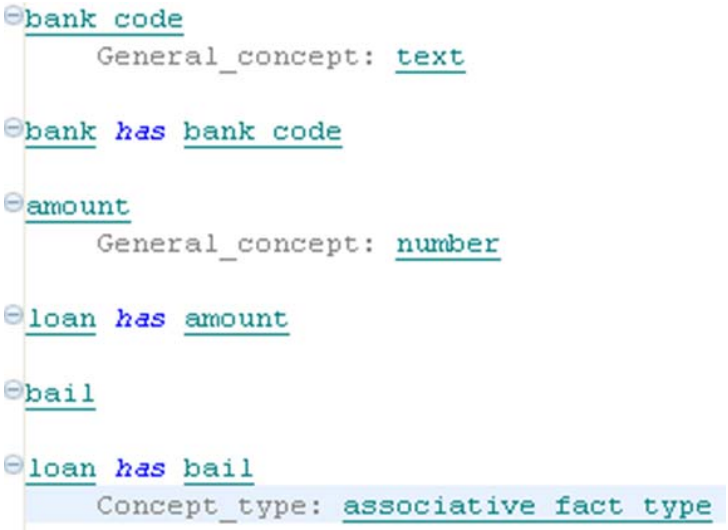


1. price (verb)
2. price (verb)
3. price (noun)
4. price (noun)
5. price (noun)
6. price (noun)

the amount of money needed to purchase something; "the price of gasoline"; "he got his new car on excellent terms"; "how much is the damage?"

```
price
    Definition: the amount of money needed to purchase something
    General_concept: number
    Synonym: cost
```

Also, we specify properties of the concepts – every bit of information that constitutes the required knowledge about concepts:

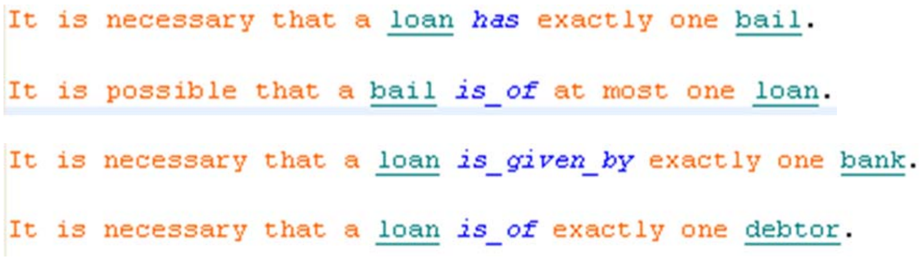


```
bank code
    General_concept: text

bank has bank code

amount
    General_concept: number

loan has amount

bail

loan has bail
    Concept_type: associative fact type
```

4th step. Define business rules

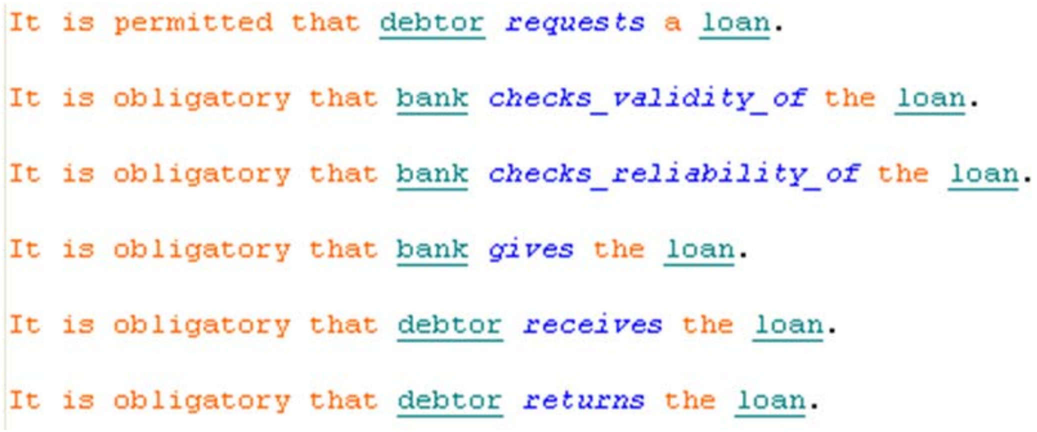
Now we define main rules related with the structure of business concepts and activities performed in that business. These are structural rules and operative rules.

Structural rules:



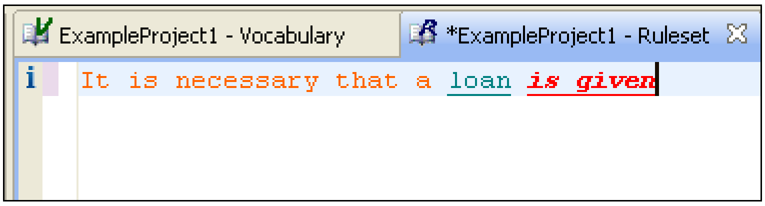
```
It is necessary that a loan has exactly one bail.

It is possible that a bail is_of at most one loan.

It is necessary that a loan is_given_by exactly one bank.

It is necessary that a loan is_of exactly one debtor.
```

Operative rules:



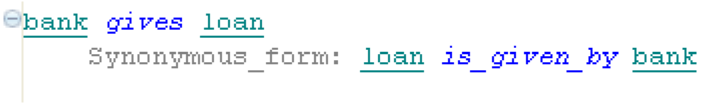
```
It is permitted that debtor requests a loan.

It is obligatory that bank checks_validity_of the loan.

It is obligatory that bank checks_reliability_of the loan.

It is obligatory that bank gives the loan.

It is obligatory that debtor receives the loan.

It is obligatory that debtor returns the loan.
```

When defining business rules, it is important to specify all necessary synonymous forms, i.e. the VeTIS editor does not allow insertion of unrecognized terms or fact type forms.



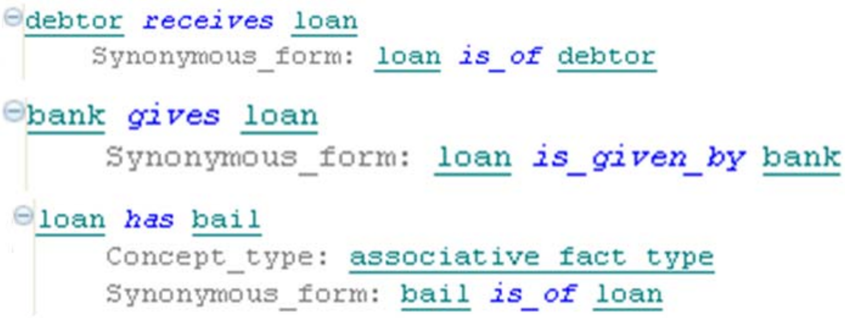
```
It is necessary that a loan is given
```

For example, it will “understand” fact type “loan is_given” only when you introduce that fact type form into the Business Vocabulary:



```
bank gives loan
    Synonymous_form: loan is_given_by bank
```

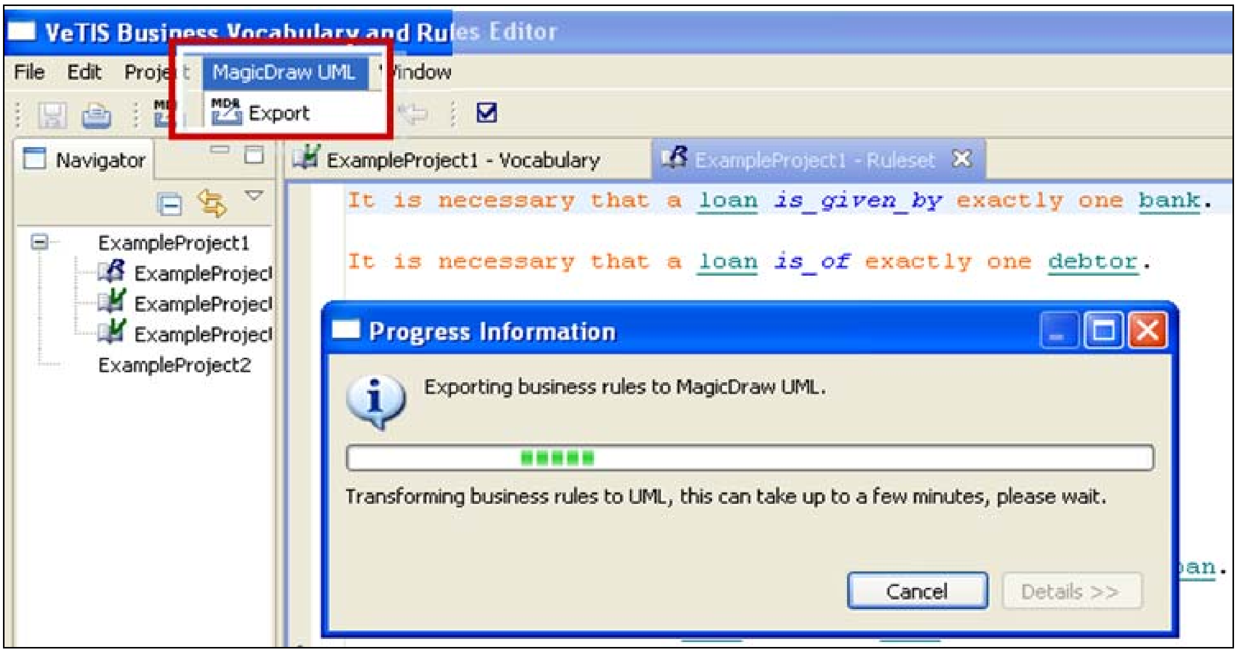
So after that we have required synonymous forms:



```
debtor receives loan
    Synonymous_form: loan is_of debtor
bank gives loan
    Synonymous_form: loan is_given_by bank
loan has bail
    Concept_type: associative fact type
    Synonymous_form: bail is_of loan
```

5th step. Transform business vocabulary and business rules into a UML&OCL model.

Now we can try to obtain our initial UML class model by saving the project and exporting it to the MagicDraw UML tool.

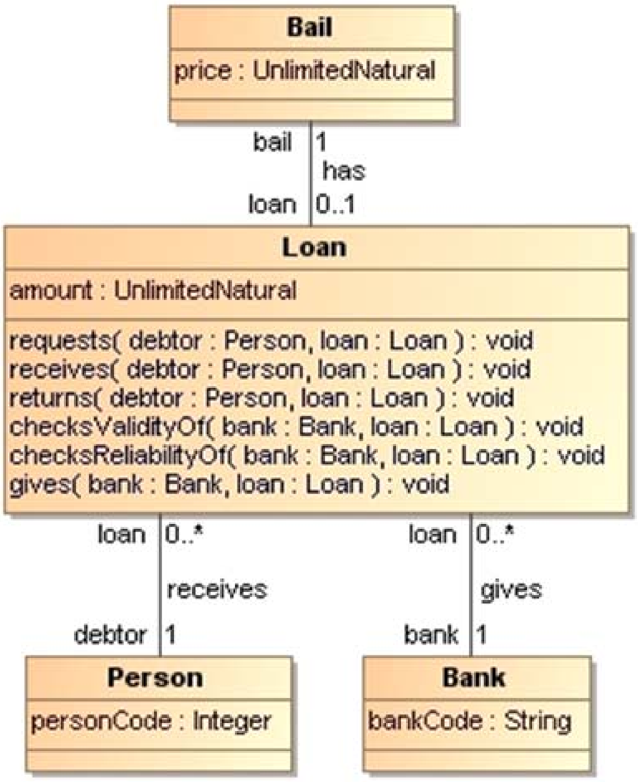


```
It is necessary that a loan is_given_by exactly one bank.

It is necessary that a loan is_of exactly one debtor.
```

Exporting business rules to MagicDraw UML.

Transforming business rules to UML, this can take up to a few minutes, please wait.

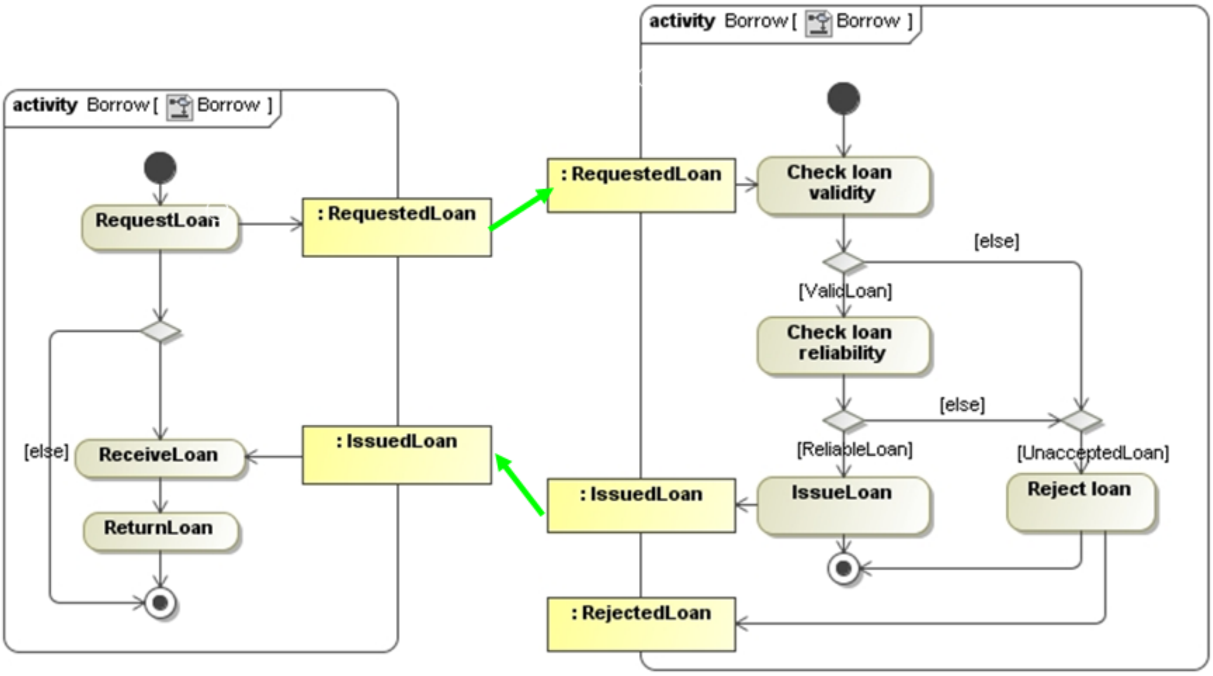
Our result is a simple UML class diagram:


Bail
price : UnlimitedNatural
bail
1
has
loan
0..1
Loan
amount : UnlimitedNatural
requests(debtor : Person, loan : Loan) : void
receives(debtor : Person, loan : Loan) : void
returns(debtor : Person, loan : Loan) : void
checksValidityOf(bank : Bank, loan : Loan) : void
checksReliabilityOf(bank : Bank, loan : Loan) : void
gives(bank : Bank, loan : Loan) : void
loan
0..*
receives
debtor
1
loan
0..*
gives
bank
1
Person
personCode : Integer
Bank
bankCode : String

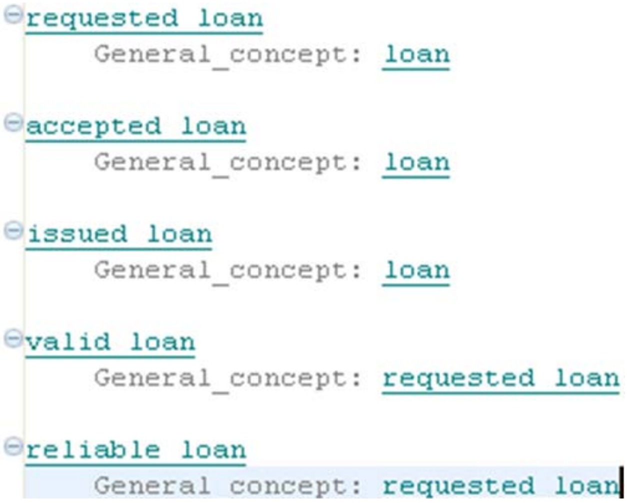
We will develop it in the following steps.

6th step. Define alternative scenarios for each use case

Now let's consider how the process will look in the cases when requested loan is not valid or unreliable. Refined activities representing use cases include specific concept categories corresponding to different states of object types: requested loan, issued loan, rejected loan, etc.

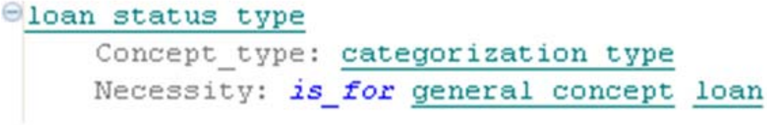


So we supplement business vocabulary adding these categories:

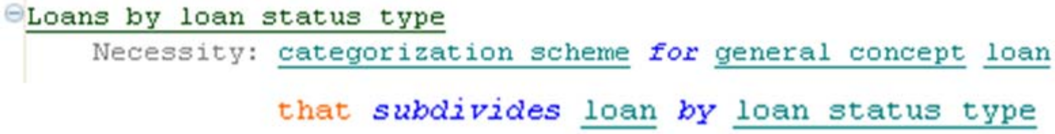


```
⊖requested loan
     General_concept: loan

⊖accepted loan
     General_concept: loan

⊖issued loan
     General_concept: loan

⊖valid loan
     General_concept: requested loan

⊖reliable loan
     General_concept: requested loan
```

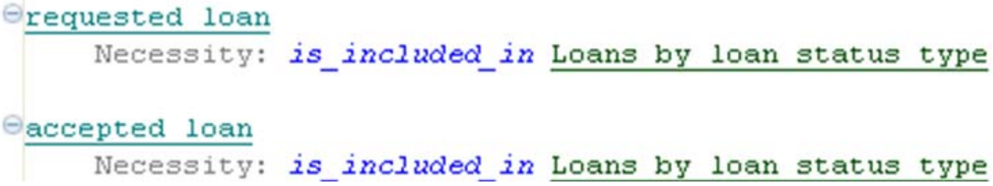
We can group the categories according to their categorization types:



```
⊖loan status type
     Concept_type: categorization type
     Necessity: is_for general concept loan
```

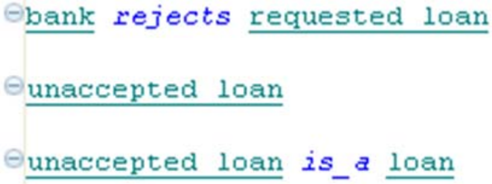


```
⊖Loans by loan status type
     Necessity: categorization scheme for general concept loan
                 that subdivides loan by loan status type
```



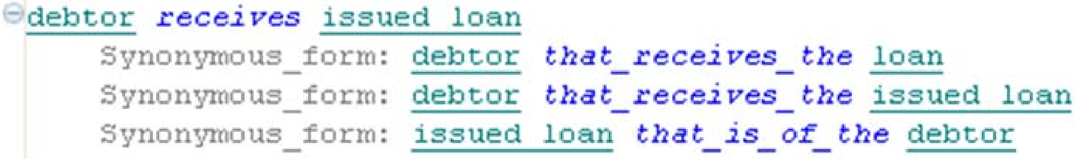
```
⊖requested loan
     Necessity: is_included_in Loans by loan status type

⊖accepted loan
     Necessity: is_included_in Loans by loan status type
```

Now we can supplement the business vocabulary with fact types corresponding to activities of alternative scenarios:

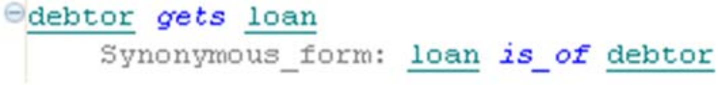


```
⊖bank rejects requested loan

⊖unaccepted loan

⊖unaccepted loan is_a loan
```

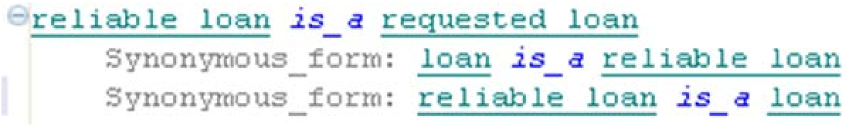
We add synonymous forms for describing business rules:



```
⊖debtor receives issued loan
     Synonymous_form: debtor that_receives_the loan
     Synonymous_form: debtor that_receives_the issued loan
     Synonymous_form: issued loan that_is_of_the debtor
```

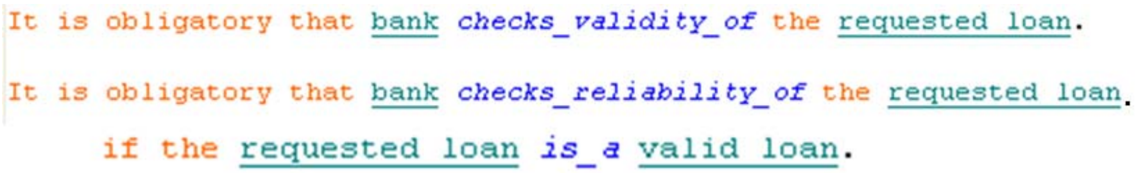


```
⊖debtor gets loan
     Synonymous_form: loan is_of debtor
```

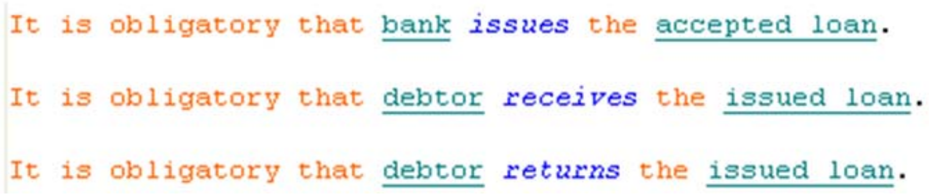


```
reliable loan is_a requested loan
    Synonymous_form: loan is_a reliable loan
    Synonymous_form: reliable loan is_a loan
```

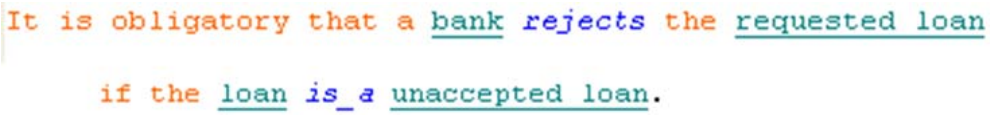
Now we can refine operative business rules regarding alternative scenarios:



```
It is obligatory that bank checks_validity_of the requested loan.

It is obligatory that bank checks_reliability_of the requested loan.
      if the requested loan is_a valid loan.
```

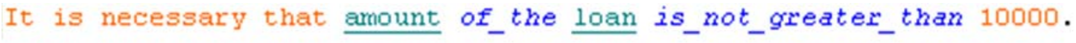


```
It is obligatory that bank issues the accepted loan.

It is obligatory that debtor receives the issued loan.

It is obligatory that debtor returns the issued loan.
```

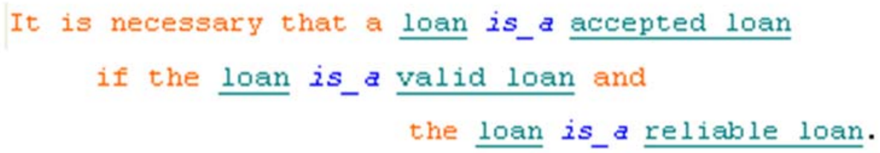


```
It is obligatory that a bank rejects the requested loan

      if the loan is_a unaccepted loan.
```

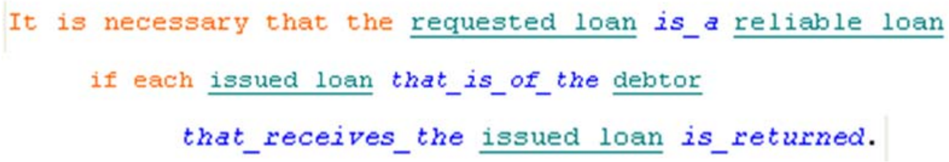
Also, we describe additional structural rules for derivation of categorized object types and all required constraints:



```
It is necessary that amount of_the loan is_not_greater_than 10000.
```



```
It is necessary that a loan is_a accepted loan
      if the loan is_a valid loan and
                       the loan is_a reliable loan.
```

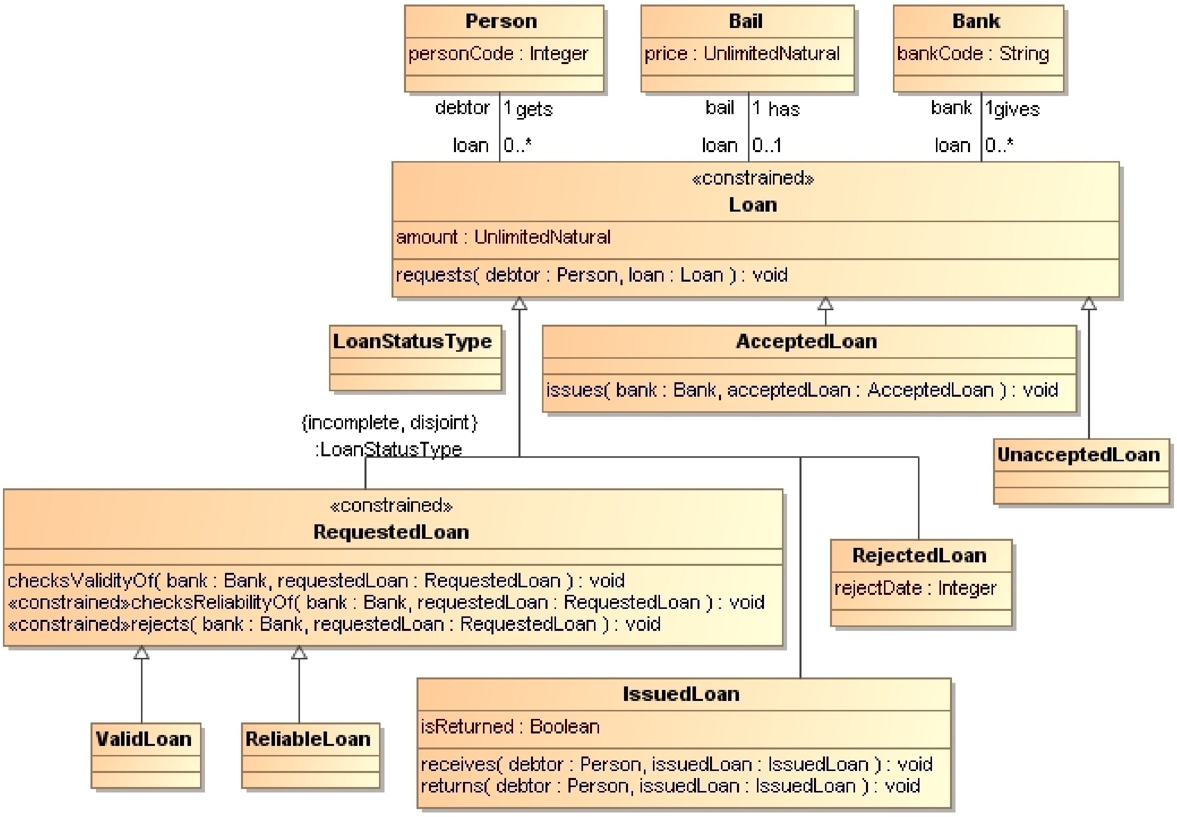


```
It is necessary that the requested loan is_a reliable loan
      if each issued loan that_is_of_the debtor
          that_receives_the issued loan is_returned.
```

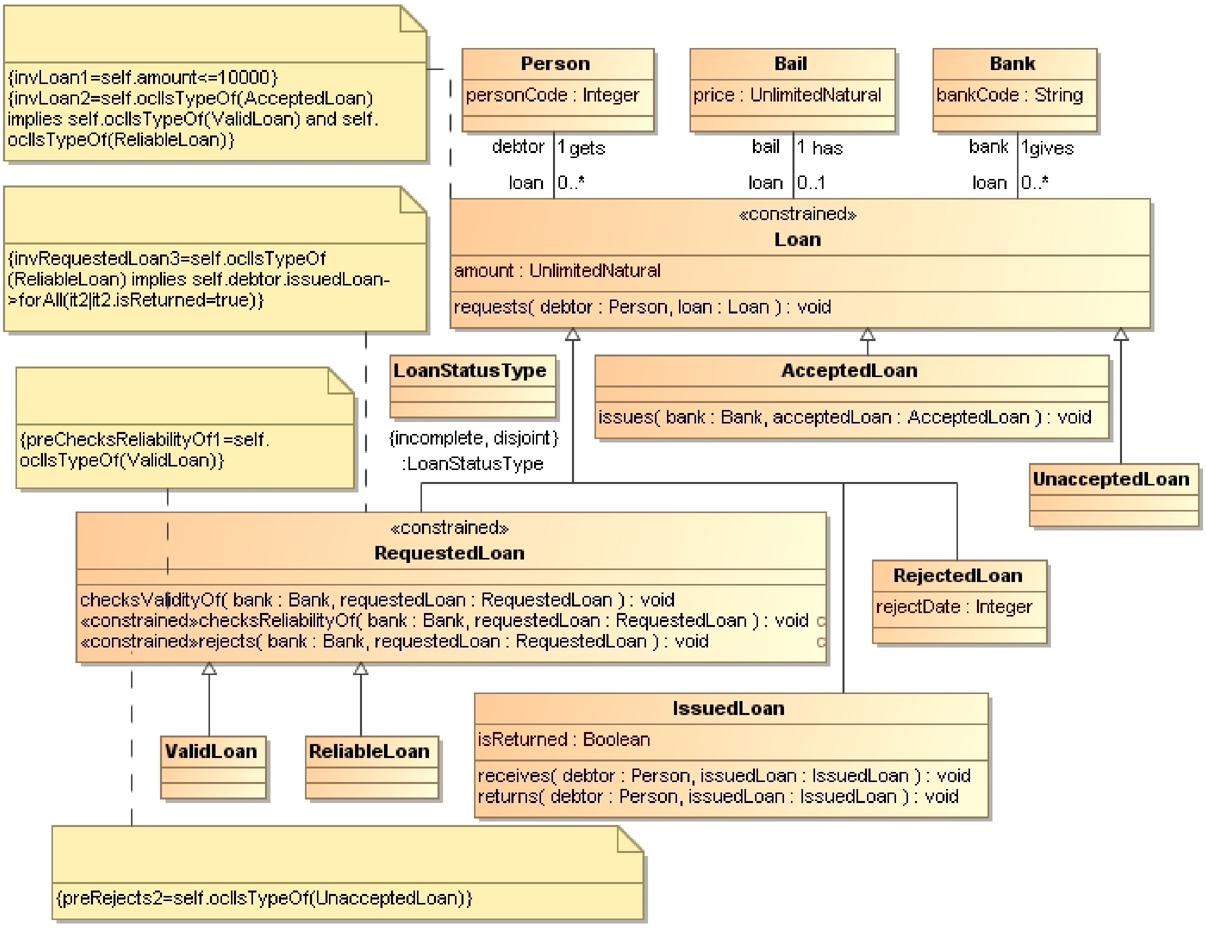
7th step. Transform repeatedly the business vocabulary and business rules into UML&OCL model.

After the next export to MagicDraw UML tool, we obtain the corresponding class diagram, shown on the next page (similar, but much bigger example from the same business domain is given in the appendixes A and B).

8th step. Make the further refinements to the business vocabulary and business rules or move to the next phase(s) of software development.

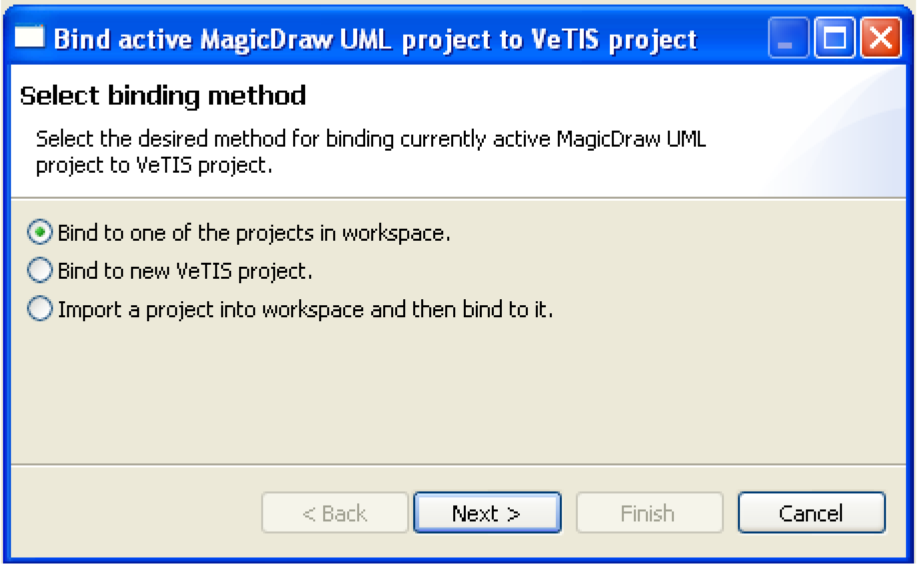

Person
personCode : Integer
Bail
price : UnlimitedNatural
Bank
bankCode : String
debtor
1 gets
loan
0..*
bail
1 has
loan
0..1
bank
1gives
loan
0..*
«constrained»
Loan
amount : UnlimitedNatural
requests(debtor : Person, loan : Loan) : void
LoanStatusType
AcceptedLoan
issues(bank : Bank, acceptedLoan : AcceptedLoan) : void
{incomplete, disjoint}
:LoanStatusType
UnacceptedLoan
«constrained»
RequestedLoan
checksValidityOf(bank : Bank, requestedLoan : RequestedLoan) : void
«constrained»checksReliabilityOf(bank : Bank, requestedLoan : RequestedLoan) : void
«constrained»rejects(bank : Bank, requestedLoan : RequestedLoan) : void
RejectedLoan
rejectDate : Integer
ValidLoan
ReliableLoan
IssuedLoan
isReturned : Boolean
receives(debtor : Person, issuedLoan : IssuedLoan) : void
returns(debtor : Person, issuedLoan : IssuedLoan) : void

The same class diagram with visualized OCL constraints:


{invLoan1=self.amount<=10000}
{invLoan2=self.oclIsTypeOf(AcceptedLoan)
implies self.oclIsTypeOf(ValidLoan) and self.
oclIsTypeOf(ReliableLoan)}
{invRequestedLoan3=self.oclIsTypeOf
(ReliableLoan) implies self.debtor.issuedLoan-
>forAll(it2|it2.isReturned=true)}
{preChecksReliabilityOf1=self.
oclIsTypeOf(ValidLoan)}
{preRejects2=self.oclIsTypeOf(UnacceptedLoan)}
Person
personCode : Integer
Bail
price : UnlimitedNatural
Bank
bankCode : String
debtor
1 gets
loan
0..*
bail
1 has
loan
0..1
bank
1gives
loan
0..*
«constrained»
Loan
amount : UnlimitedNatural
requests(debtor : Person, loan : Loan) : void
LoanStatusType
AcceptedLoan
issues(bank : Bank, acceptedLoan : AcceptedLoan) : void
{incomplete, disjoint}
:LoanStatusType
UnacceptedLoan
«constrained»
RequestedLoan
checksValidityOf(bank : Bank, requestedLoan : RequestedLoan) : void
«constrained»checksReliabilityOf(bank : Bank, requestedLoan : RequestedLoan) : void
«constrained»rejects(bank : Bank, requestedLoan : RequestedLoan) : void
RejectedLoan
rejectDate : Integer
ValidLoan
ReliableLoan
IssuedLoan
isReturned : Boolean
receives(debtor : Person, issuedLoan : IssuedLoan) : void
returns(debtor : Person, issuedLoan : IssuedLoan) : void

Working with the VeTIS Editor

VeTIS editor is implemented as a plug-in for the MagicDraw UML CASE tool. You can launch it from the MagicDraw UML tool by clicking on the menu item “VeTIS”. There are many options for binding VeTIS and MagicDraw UML projects: you can choose an existing VeTIS project from the VeTIS workspace, import it or create a new one:



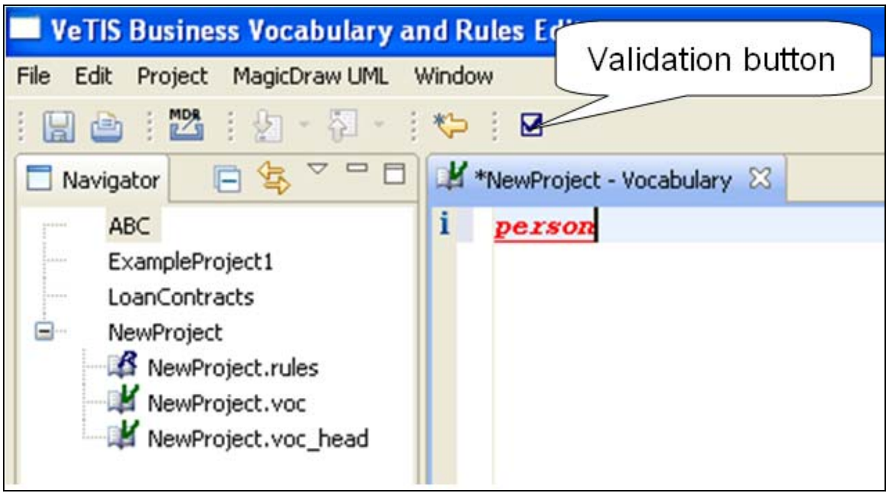
Bind active MagicDraw UML project to VeTIS project

Select binding method

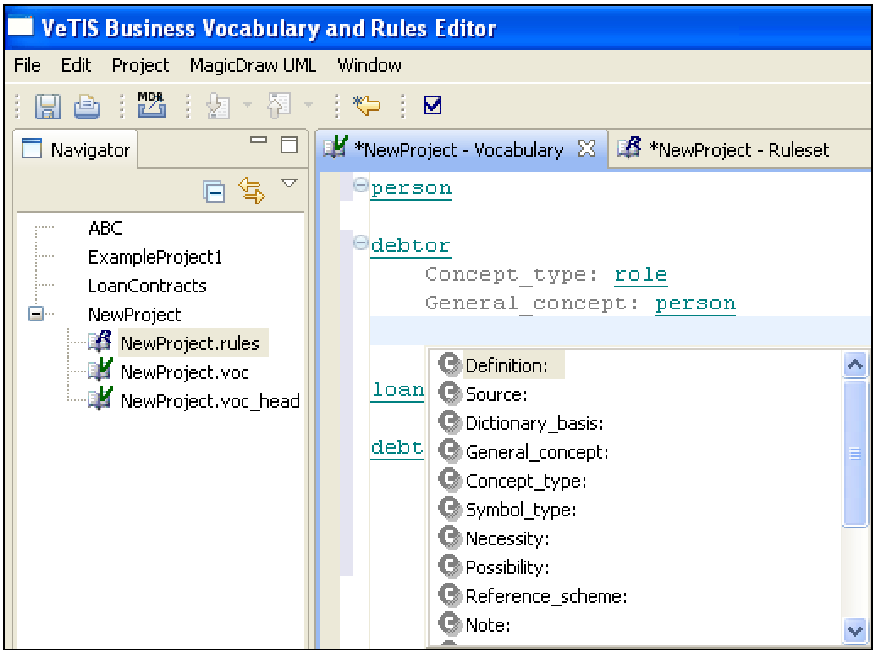
Select the desired method for binding currently active MagicDraw UML project to VeTIS project.

● Bind to one of the projects in workspace.
○ Bind to new VeTIS project.
○ Import a project into workspace and then bind to it.

After choosing “Bind to new VeTIS project” you will be asked to enter a name (e.g. NewProject) and to appoint a location for the project. In order to describe the business vocabulary in the VeTIS editor double-click on the vocabulary in the browser on the left (e.g. NewProject.voc) and enter terms and fact type forms in the Vocabulary tab in a VeTIS window. After inserting a new term, you have to validate it by clicking the validation button:



For entering definitions of concepts you can open the Content assistant. To do this enter a new line after the corresponding vocabulary entry (e.g. debtor) and click the Tab button on your keyboard (or directly call the Content assistant form the right-click menu, or press Ctrl+Space); choose the description item (e.g. “Concept_type:”) from the Content assistant; press Enter; type the corresponding text (e.g. “person”); press Enter.



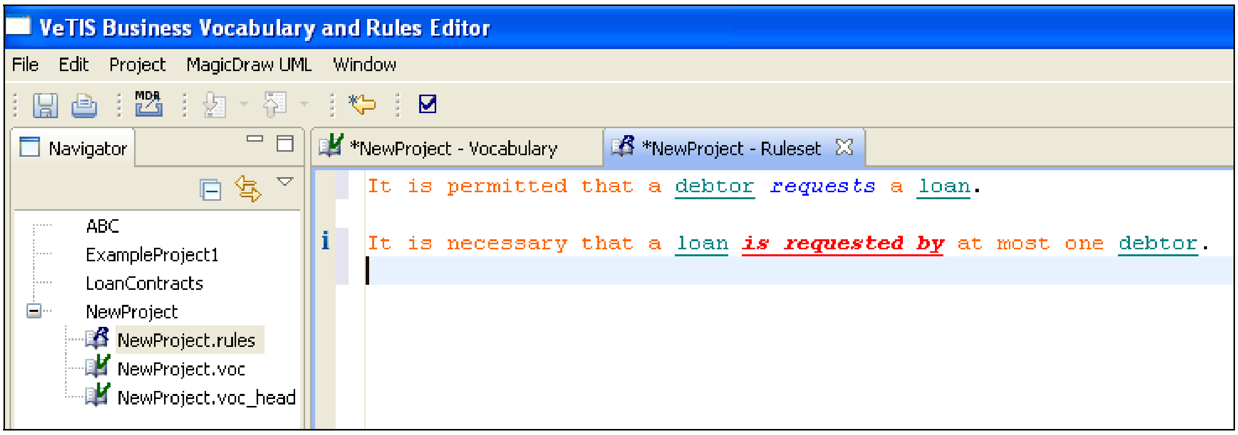
```
person

debtor
    Concept_type: role
    General_concept: person

loan

debt
```

As was already mentioned, you can insert definitions or synonyms from the Word Net by choosing “Ask WordNet” on the right-click menu of a corresponding term displayed in the layout on the right side of the VeTIS editor window:

To describe business rules in the VeTIS editor double-click on a business rules vocabulary in the browser on the left (e.g. NewProject.rules) and enter business rules in the RuleSet tab in the VeTIS window (it is possible to define several rule sets for the same vocabulary).

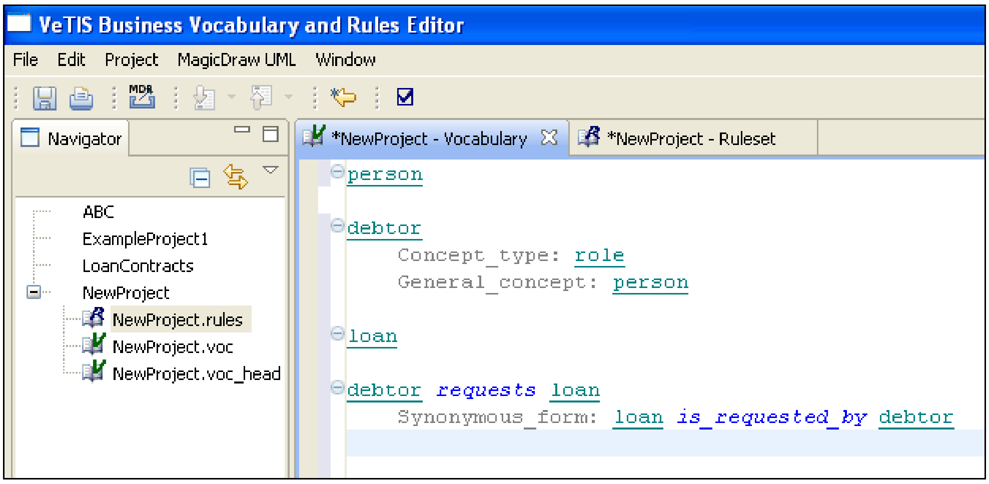
Finally, specify the name of the Business Vocabulary by double-clicking on a business vocabulary heading (e.g. NewProject.voc_head). Here you describe the language, the speech community and other (optional) fields.

For entering business rules, you will often need to have synonymous forms for fact types described in your business vocabulary (unknown fact type forms are marked in red):



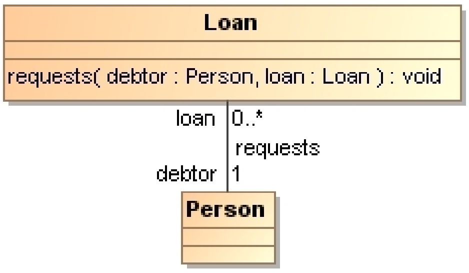
```
It is permitted that a debtor requests a loan.

It is necessary that a loan is requested by at most one debtor.
```

To avoid potential problems describe corresponding synonymous fact type forms:

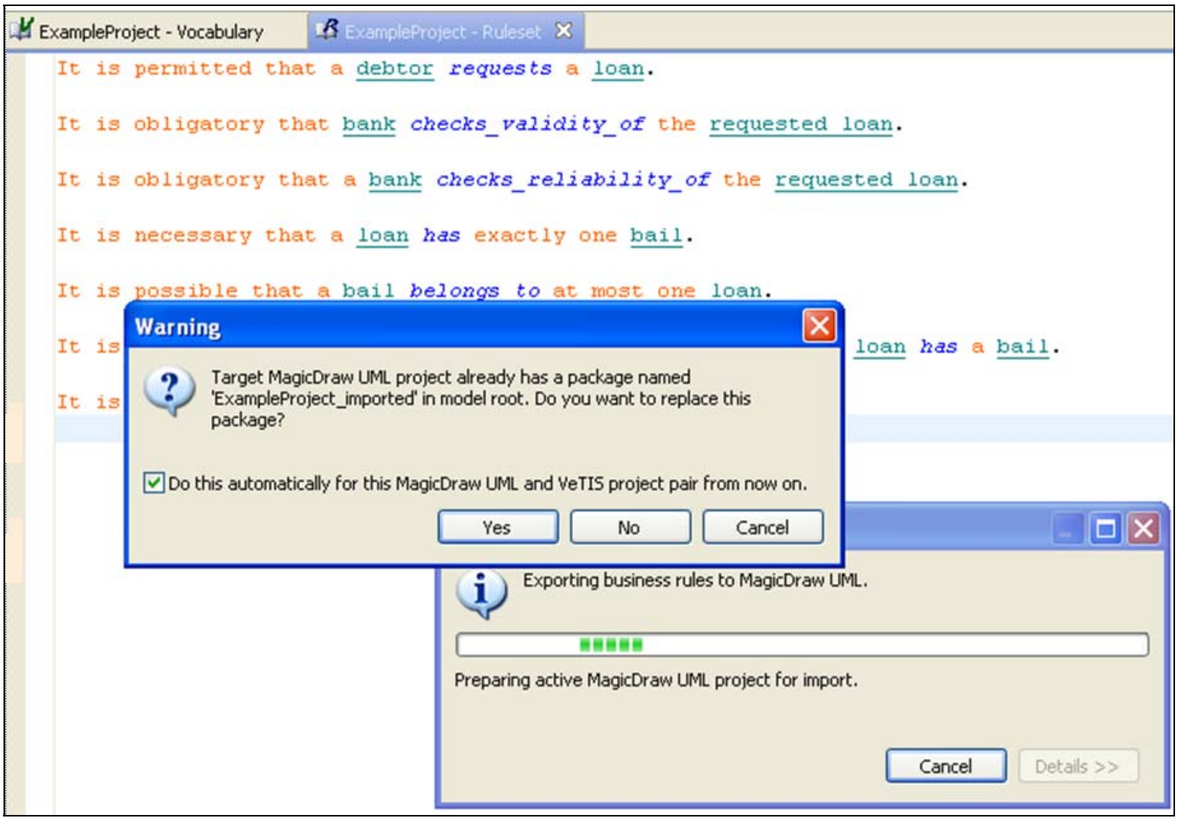


```
person

debtor
    Concept_type: role
    General_concept: person

loan

debtor requests loan
    Synonymous_form: loan is_requested_by debtor
```

Validate business rules, save the project and export it to MagicDraw UML by choosing MagicDraw UML / Export. There are many options for choosing a MagicDraw UML project to which the VeTIS project will be bound. By default, the VeTIS project will be bound to an active MagicDraw UML project, or a new MagicDraw UML project will be created, or you will be asked to specify it, if any of active VeTIS / MagicDraw UML projects were bound before. After the transformation, you will have a UML package with a reference to the transformed Business Vocabulary in it’s name (e.g. NewProject_imported), while the results of the transformation will be represented by the UML class model and class diagram:



Exporting the same (e.g. corrected or supplemented) VeTIS project once more you will get the following message:



It is permitted that a debtor requests a loan.

It is obligatory that bank checks_validity_of the requested loan.

It is obligatory that a bank checks_reliability_of the requested loan.

It is necessary that a loan has exactly one bail.

It is possible that a bail belongs to at most one loan.

It is loan has a bail.

It is

Warning

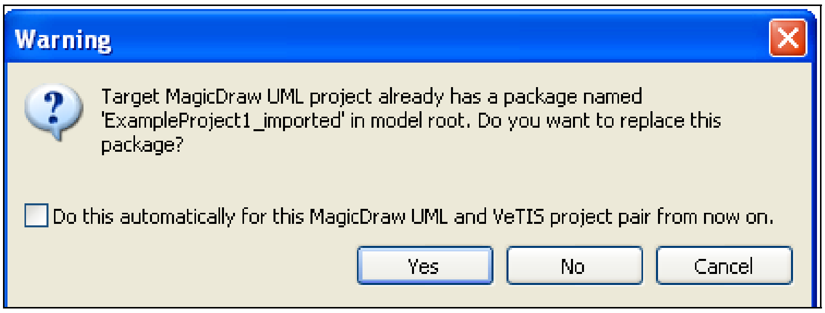
Target MagicDraw UML project already has a package named 'ExampleProject_imported' in model root. Do you want to replace this package?

☑ Do this automatically for this MagicDraw UML and VeTIS project pair from now on.

Exporting business rules to MagicDraw UML.

Preparing active MagicDraw UML project for import.

It is possible to bind a pair of VeTIS / MagicDraw UML projects by ticking the “Do this automatically…”. This way you will avoid messages asking for replacement of previous transformations results:



Warning

Target MagicDraw UML project already has a package named 'ExampleProject1_imported' in model root. Do you want to replace this package?

□ Do this automatically for this MagicDraw UML and VeTIS project pair from now on.

In case of generating a UML class model with OCL constraints, you will see what classes or / and operations have constraints as they will be marked by stereotype <<constrained>>. You can visualize constraints by choosing “Add class constraint note” or “Add operation constraint note” on the right-click class / operation menu as was already mentioned in the section “Transforming business rules into OCL constraints”.

VeTIS Installation and Configuration

You can install the VeTIS MagicDraw UML plug-in by simply placing the folder org.vetis.md in your ProgramFiles/MagicDraw UML …/plugins directory along with other MagicDraw UML plug-ins. The VeTIS plug-in can be used with MagicDraw UML versions 16.0 and up.

VeTIS tool has the configuration file (language/en.xml), which is used for the configuration of various settings (e.g. predefined phrases) of the tool. However, graphical user interface for the configuration file is still under development, therefore, it is not possible for end users to make changes to the VeTIS predefined phrases and other features at the moment.

Concluding Remarks for VeTIS Users

Thank you for reading this document (or at least it’s conclusions)! It may seem there are many limitations and drawbacks in the current VeTIS implementation. This is partially true. While almost all required Business Vocabularies definition capabilities are available, Business Rules description tools still lack some important features. We honestly recommend you to keep rules (as well as your business processes) as simple as possible – not only because of the immaturity of the VeTIS tool, but also for the sake of your design models, your software and your business.

Our future efforts are aimed at extending VeTIS capabilities by including more types of business rules; creating a user assistant for defining them; allowing configuration of the tool; reusing existing Business Vocabularies; and, most importantly, closer relation of the VeTIS capabilities with software development methodologies.

We are waiting for your comments, suggestions and any reflections whatever they would be. We will try to answer all messages and make improvements if possible.

References

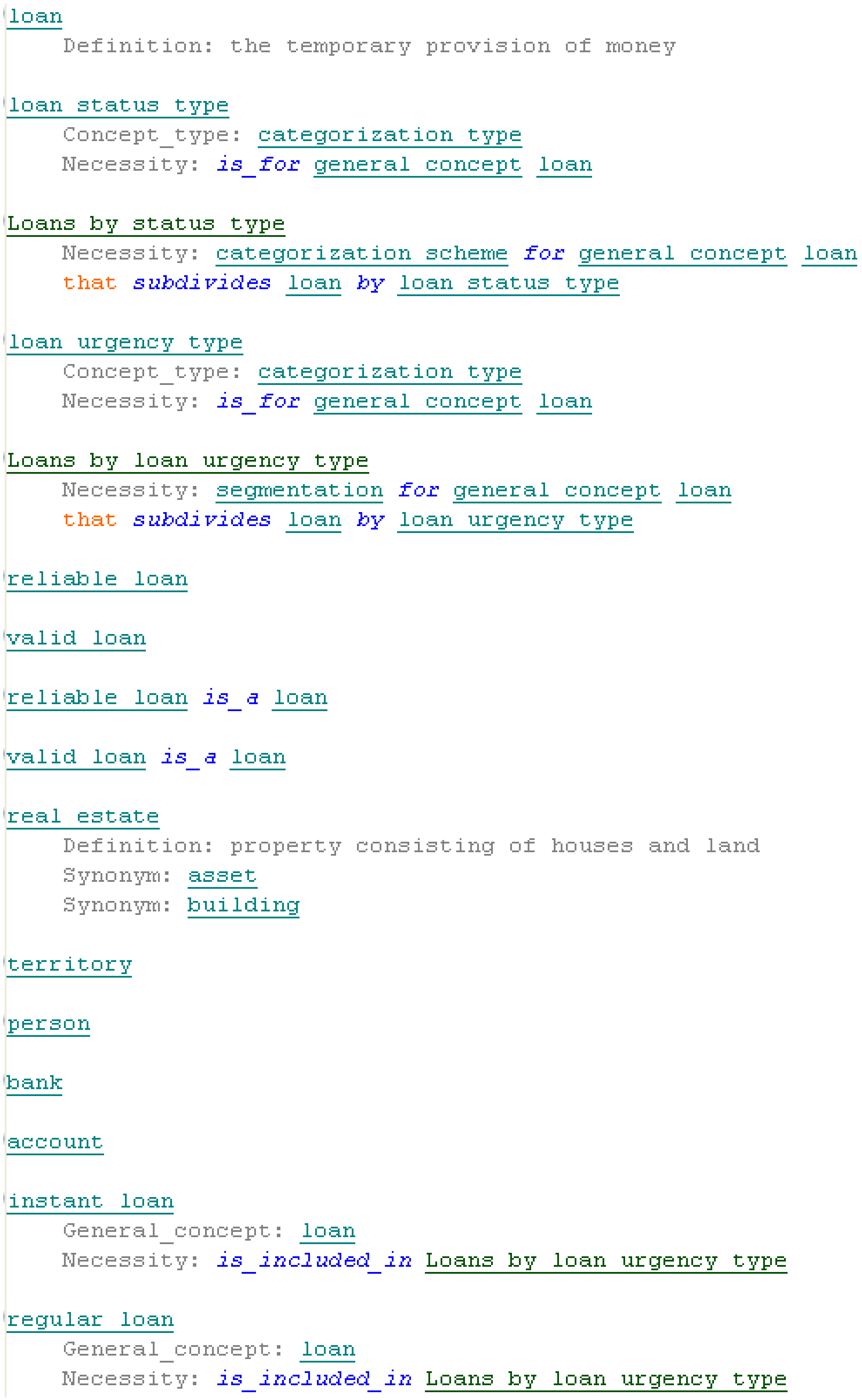
[1] Semantics of Business Vocabulary and Business Rules (SBVR). Version 1.0. December, 2008, available at http://www.omg.org/docs/formal/08-01- 02.pdf.

[2] Ceravolo, P., Fugazza, C., Leida, M. Modeling Semantics of Business Rules. 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007), 171–176.

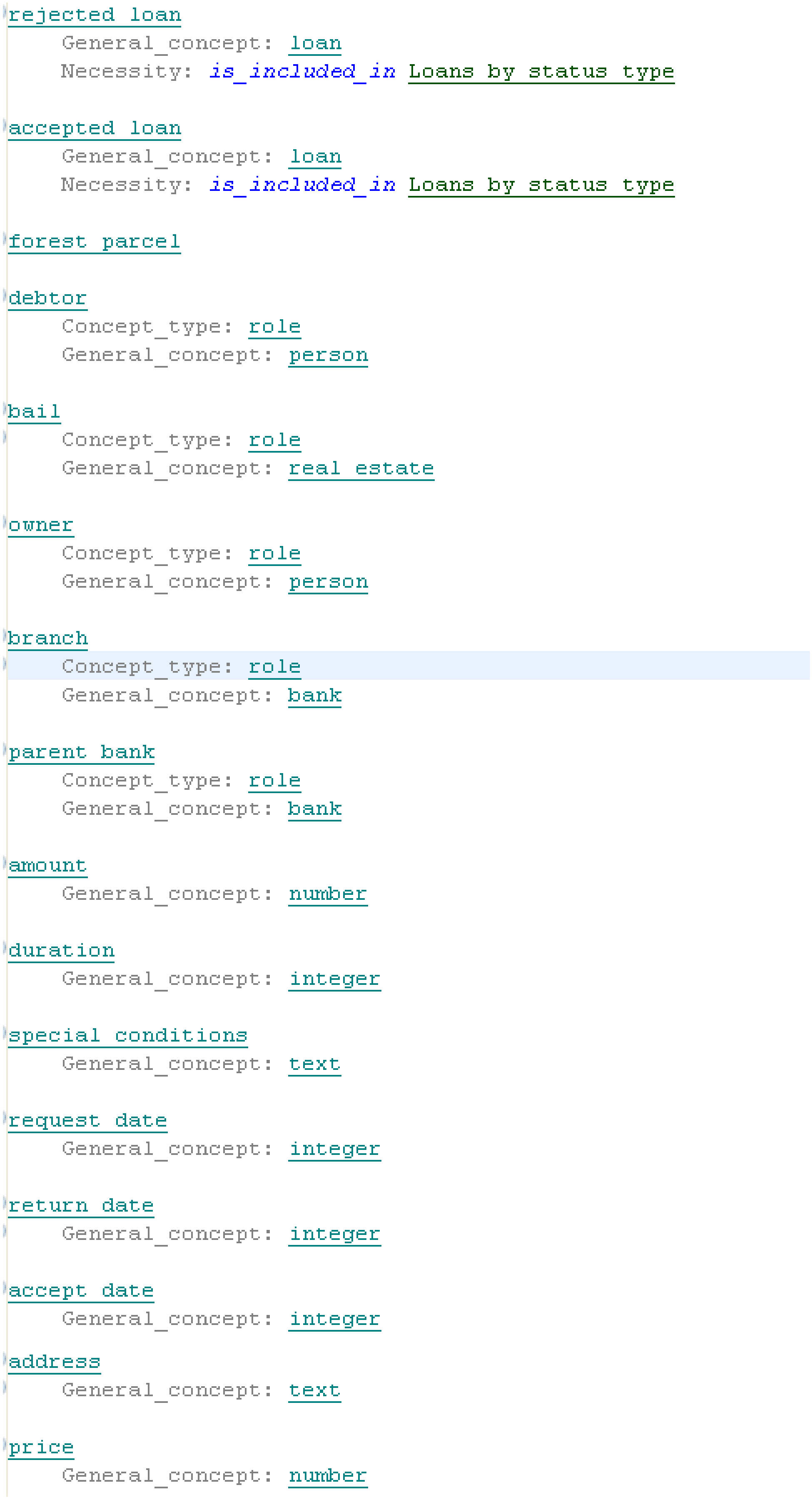
[3] SBeaVeR – Business Modeller. Eclipse plug-in, available at http://sbeaver.sourceforge.net

Appendix A. An Example of the SBVR Business Vocabulary

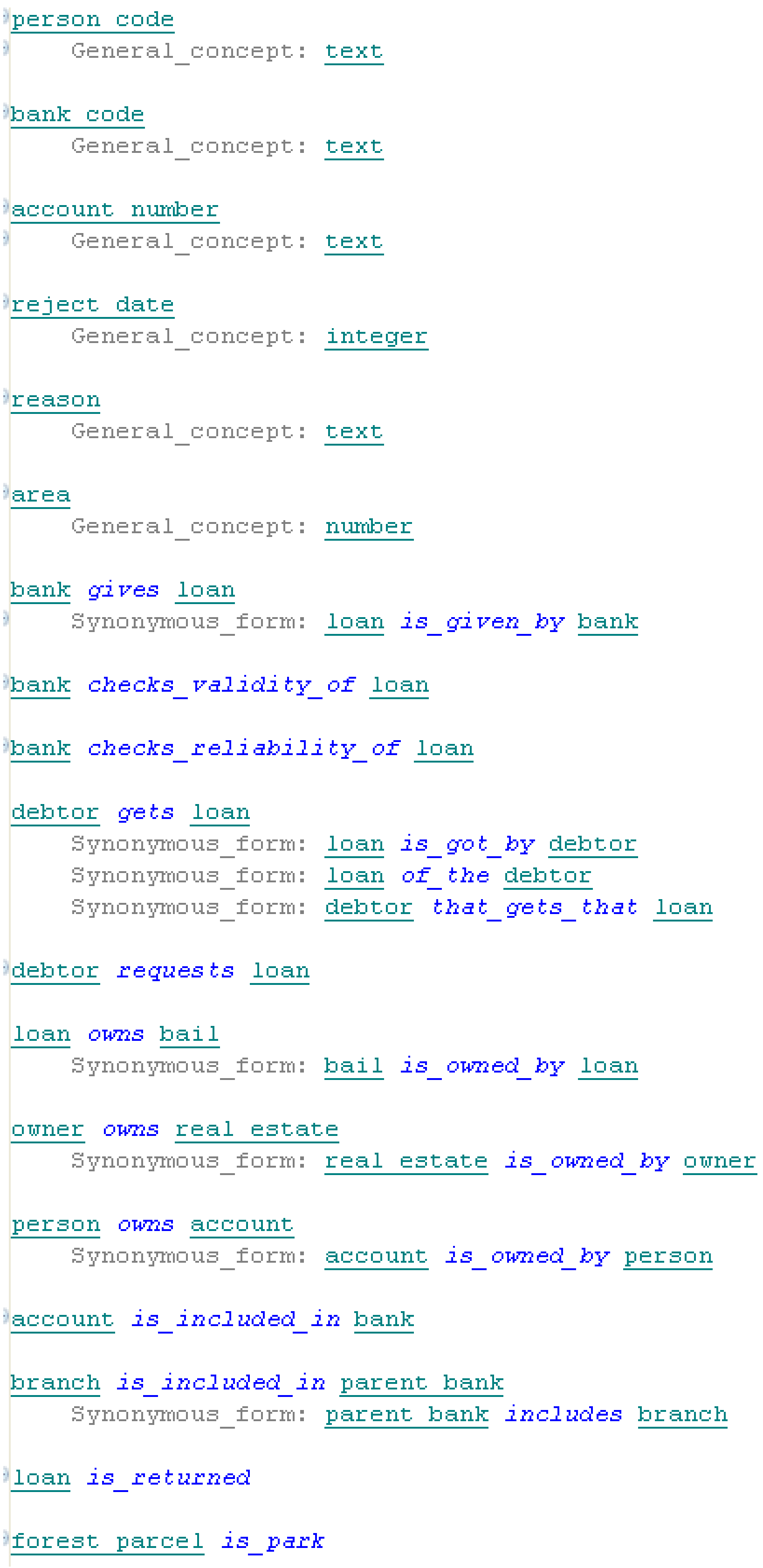
Business Vocabulary for Loan contracts domain:



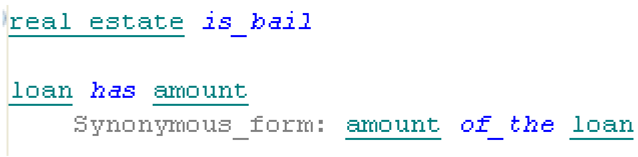
```
loan
    Definition: the temporary provision of money

loan status type
    Concept_type: categorization type
    Necessity: is_for general concept loan

Loans by status type
    Necessity: categorization scheme for general concept loan
    that subdivides loan by loan status type

loan urgency type
    Concept_type: categorization type
    Necessity: is_for general concept loan

Loans by loan urgency type
    Necessity: segmentation for general concept loan
    that subdivides loan by loan urgency type

reliable loan

valid loan

reliable loan is_a loan

valid loan is_a loan

real estate
    Definition: property consisting of houses and land
    Synonym: asset
    Synonym: building

territory

person

bank

account

instant loan
    General_concept: loan
    Necessity: is_included_in Loans by loan urgency type

regular loan
    General_concept: loan
    Necessity: is_included_in Loans by loan urgency type
```



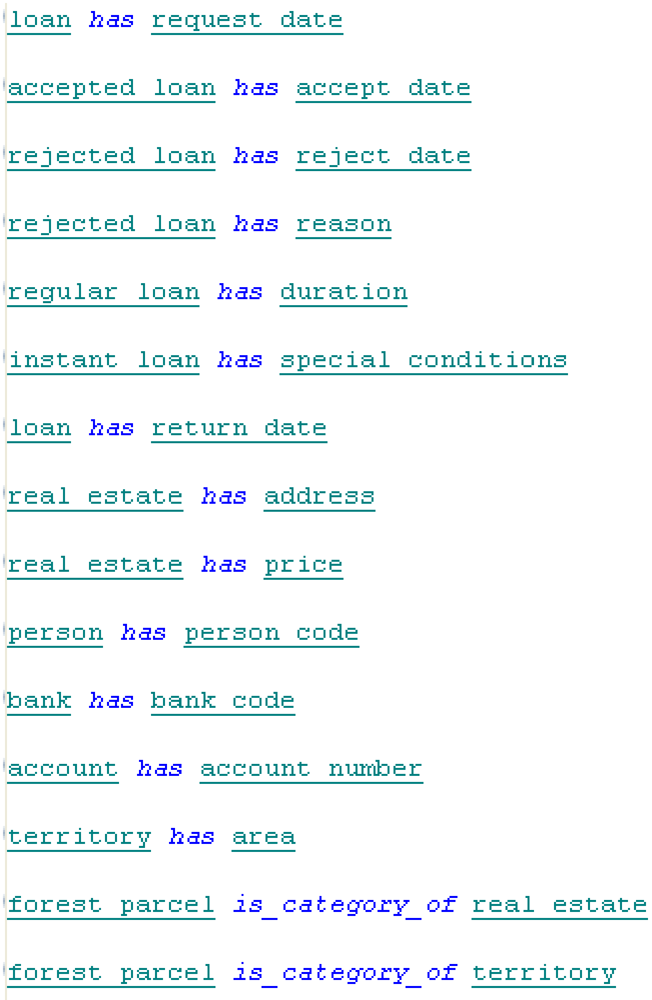
```
rejected loan
    General_concept: loan
    Necessity: is_included_in Loans by status type

accepted loan
    General_concept: loan
    Necessity: is_included_in Loans by status type

forest parcel

debtor
    Concept_type: role
    General_concept: person

bail
    Concept_type: role
    General_concept: real estate

owner
    Concept_type: role
    General_concept: person

branch
    Concept_type: role
    General_concept: bank

parent bank
    Concept_type: role
    General_concept: bank

amount
    General_concept: number

duration
    General_concept: integer

special conditions
    General_concept: text

request date
    General_concept: integer

return date
    General_concept: integer

accept date
    General_concept: integer
address
    General_concept: text

price
    General_concept: number
```



```
person code
     General_concept: text

bank code
     General_concept: text

account number
     General_concept: text

reject date
     General_concept: integer

reason
     General_concept: text

area
     General_concept: number

bank gives loan
     Synonymous_form: loan is_given_by bank

bank checks_validity_of loan

bank checks_reliability_of loan

debtor gets loan
     Synonymous_form: loan is_got_by debtor
     Synonymous_form: loan of_the debtor
     Synonymous_form: debtor that_gets_that loan

debtor requests loan

loan owns bail
     Synonymous_form: bail is_owned_by loan

owner owns real estate
     Synonymous_form: real estate is_owned_by owner

person owns account
     Synonymous_form: account is_owned_by person

account is_included_in bank

branch is_included_in parent bank
     Synonymous_form: parent bank includes branch

loan is_returned

forest parcel is_park
```



```
real estate is_bail

loan has amount
     Synonymous_form: amount of_the loan
```



loan has request date

accepted loan has accept date

rejected loan has reject date

rejected loan has reason

regular loan has duration

instant loan has special conditions

loan has return date

real estate has address

real estate has price

person has person code

bank has bank code

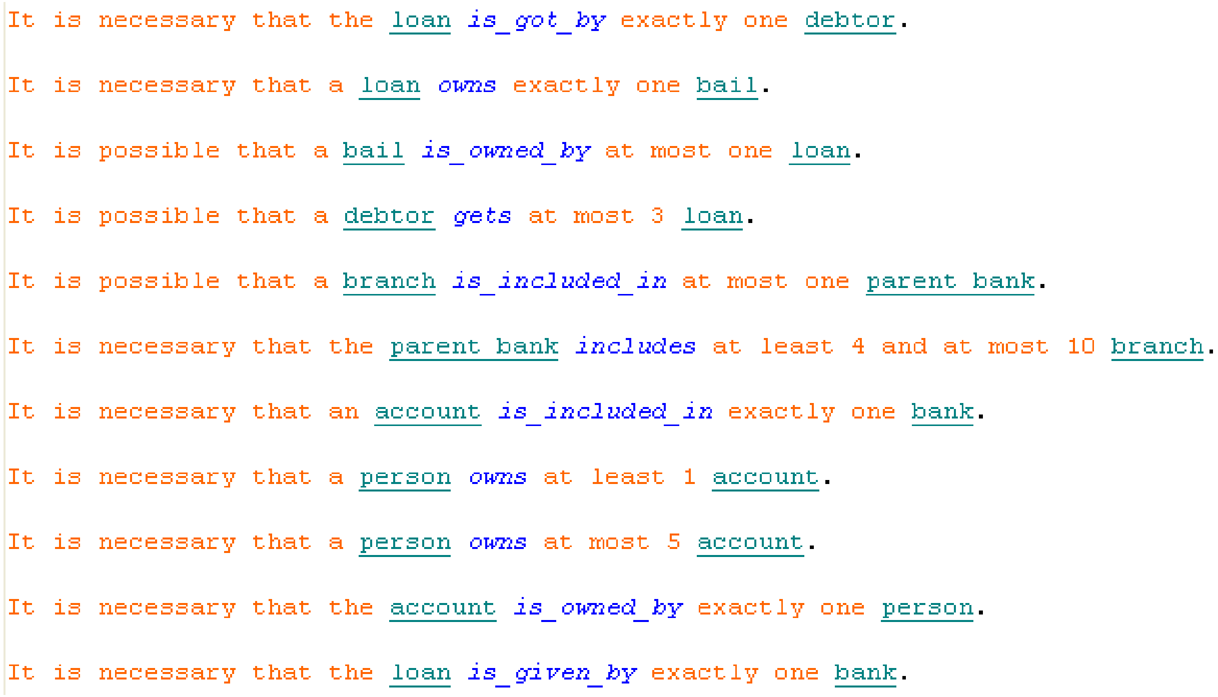
account has account number

territory has area

forest parcel is_category_of real estate

forest parcel is_category_of territory

Business rules:



It is necessary that the loan is_got_by exactly one debtor.

It is necessary that a loan owns exactly one bail.

It is possible that a bail is_owned_by at most one loan.

It is possible that a debtor gets at most 3 loan.

It is possible that a branch is_included_in at most one parent bank.

It is necessary that the parent bank includes at least 4 and at most 10 branch.

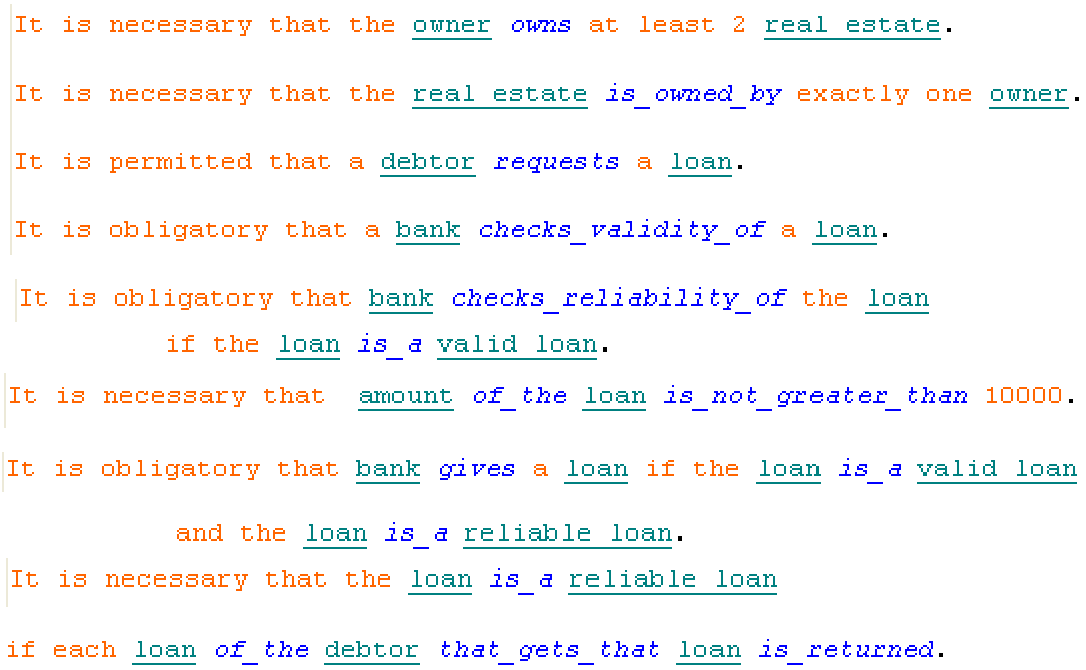
It is necessary that an account is_included_in exactly one bank.

It is necessary that a person owns at least 1 account.

It is necessary that a person owns at most 5 account.

It is necessary that the account is_owned_by exactly one person.

It is necessary that the loan is_given_by exactly one bank.



It is necessary that the owner owns at least 2 real estate.

It is necessary that the real estate is_owned_by exactly one owner.

It is permitted that a debtor requests a loan.

It is obligatory that a bank checks_validity_of a loan.

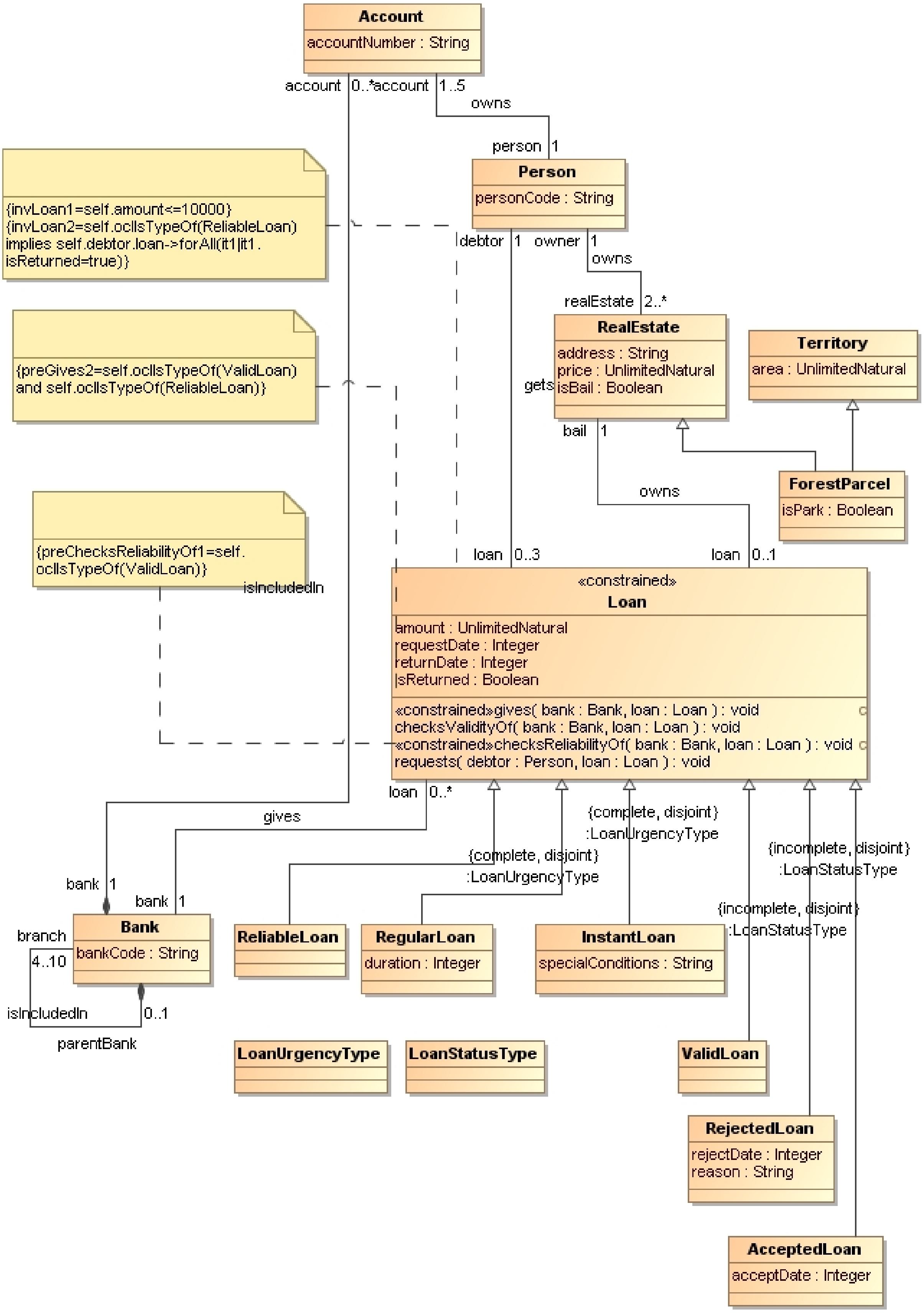
It is obligatory that bank checks_reliability_of the loan
if the loan is_a valid loan.

It is necessary that amount of_the loan is_not_greater_than 10000.

It is obligatory that bank gives a loan if the loan is_a valid loan
and the loan is_a reliable loan.

It is necessary that the loan is_a reliable loan
if each loan of_the debtor that_gets_that loan is_returned.

Appendix B. Example of the Generated UML Class Diagram



Appendix C. About the Developers of the VeTIS Tool

The VeTIS tool is the development of the **Information Systems Department** (ISD) at the Kaunas University of Technology (Lithuania). The tool is one of the results of the project “Business Rules Solutions for Information Systems Development (VeTIS)”, which was initiated by the Department and its partners from Vilnius Gediminas Technical University and IT company “No Magic Europe” in 2007.

The main goal of the VeTIS project was to improve the quality of business model-based development of information systems and the quality of information systems overall by delivering the novel business rules specification method and engineering solutions for this method. The VeTIS tool is one of such engineering solutions.

The main R&D areas of Information Systems Department at Kaunas University of Technology are:

- Model-driven development of information systems;
- Business rules approach;
- Requirements engineering;
- Conceptual data modeling and databases;
- Business process modeling;
- Enterprise modeling;
- Web services and service oriented architecture;
- Semantics of information systems.

Contact information:

Studentu 50-313a, LT-51368
Kaunas Lithuania
Phone no.: +370 (37) 453445
Fax no.: +370 (37) 300352
E-mail: lina.nemuraite@ktu.lt