I1-D2
First-Version Controlled English Rule Language

Project number:	IST2004506779
Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Eindhoven/I1-D2/D/PU/b1
Responsible editor(s):	Gerd Wagner
Reviewer(s):	Piero Bonatti
Contributing participants:	Eindhoven, Zurich, LibRT
Contributing workpackages:	I1
Contractual date of delivery:	28. February 2005
Actual date of delivery:	13. April 2005

Abstract

This report discusses several approaches to Controlled English and the problem of verbal representations of rules and suggests a method for verbalizing the visual and formal representations of vocabularies and rules obtained by UML class models and OWL ontologies. The method, called *I1 Controlled English (I1CE),* is based on defining verbalization patterns for all important kinds of fact types (predicates), and combining them with the help of keywords expressing quantifiers, logical connectives and other sentence construction operators. An important issue is the quality of the verbalization results. It is discussed that the simple mapping techniques based on fact type verbalization patterns lead to results that are acceptable in many important cases but which could be further improved by applying more advanced techniques such as the DRS-based treatment of anaphora employed in the Attempto system.

Keyword List

vocabularies and rules, fact types, verbalization, Controlled English

Project co-founded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme

First-Version Controlled English Rule Language

Gerd Wagner[1], Sergey Lukichev[2], Norbert E. Fuchs[3], and Silvie Spreeuwenberg[4]

[1] Institute of Informatics, Brandenburg University of Technology at Cottbus
(formerly Eindhoven University of Technology)
Email: G.Wagner@tu-cottbus.de

[2] Institute of Informatics, Brandenburg University of Technology Cottbus
Email: slukichev@informatik.tu-cottbus.de

[3] Department of Informatics, University of Zurich
Email: fuchs@ifi.unizh.ch

[4] LibRT
Email: info@librt.com

Abstract

This report discusses several approaches to Controlled English and the problem of verbal representations of rules and suggests a method for verbalizing the visual and formal representations of vocabularies and rules obtained by UML class models and OWL ontologies. The method, called *I1 Controlled English (I1CE),* is based on defining verbalization patterns for all important kinds of fact types (predicates), and combining them with the help of keywords expressing quantifiers, logical connectives and other sentence construction operators. An important issue is the quality of the verbalization results. It is discussed that the simple mapping techniques based on fact type verbalization patterns lead to results that are acceptable in many important cases but which could be further improved by applying more advanced techniques such as the DRS-based treatment of anaphora employed in the Attempto system.

Keyword List

vocabularies and rules, fact types, verbalization, Controlled English

Contents

1. Introduction

A Controlled English (CE) vocabulary and rule language can be used for two purposes:

- *capturing* vocabularies and rules, and
- *verbalizing* vocabularies and rules.

The second purpose, *verbalizing* visually and/or formally represented vocabularies and rules, is important to be able to present rules to domain experts for validation. This is an essential building block in any rule-based methodology. Being able to *capture* rules in English is a desirable, but more advanced, feature of such a methodology. In this report we will only be concerned with developing a CE for the purpose of verbalization. We may consider the issue of rule capture in the future.

We want to be able to verbalize various kinds of expressions and statements from visual (UML) and formal (OCL, RDF, OWL) vocabulary and rule languages.

The mapping from the visual or formal representation to the CE proposed by I1 is defined case-wise for each type of atomic sentence in the representation language. In general, we will distinguish between different kinds of basic logical statements:

1 association fact
   - binary association fact

2 attribution fact
   - constrained attribution fact
   - data constraint

3 classification fact

4 equality/inequality fact

5 aggregation fact

6 generalization statement

These concepts are explained in the next section.

2. Foundational Concepts

In this chapter, we discuss the *foundational* concepts (or *meta-concepts*) being used in this report and the terms we are using to designate them. These concepts, and their canonical designations, are described in a ***foundational vocabulary***, which is also called a *foundational* (or 'upper level') *ontology*. They define a range of top-level domain-independent ontological categories, which form a general foundation for more elaborated domain-specific vocabularies. Our foundational vocabulary is based on the Unified Foundational Ontology (UFO) proposed in [GW05a, GW05b].

Our analysis is focused on four languages for expressing vocabularies and rules:

1. SBVR – "Semantics of Business Vocabularies and Rules", the main submission to the OMG BSBR CFP [SBVR]
2. UML – the *Unified Modeling Language* of the OMG [UML]
3. RDF – the *Resource Description Framework* of the W3C [RDFCAS]
4. OWL – the *Web Ontology Language* of the W3C [OWL]

All these languages come with their own foundational vocabulary, employing different (or the same) designations for the same (or different) concepts. The I1 foundational vocabulary helps to understand the differences and overlaps among these terminologies.

For simplicity, we will not always be consistent in distinguishing the conceptual from the terminological level; we will, for instance, often say "rule" instead of "rule expression", "fact" instead of "fact statement", and "fact type" instead of "fact type expression".

2.1. Things, Sets, Entities and Individuals

A ***thing*** is 'anything perceivable or conceivable', according to (ISO, 2000) where the designation 'object' is used instead of 'thing'. This includes concrete entities and also abstract things such as sets A ***set*** is a *thing* that has other *things* as *members* (in the sense of set theory).

An ***entity*** is a *thing* that is not a *set*; neither the set-theoretic membership relation nor the subset relation can unfold the internal structure of an entity. An ***individual*** is an *entity* that does not have any *instances*, i.e., that is not an *entity type*. A ***data value*** is a member of a *datatype*, which is a particular kind of named *set*.

REWERSE-I1	***UML***	***SBVR***	***RDF***	***OWL***
thing	n.a.	Thing	n.a.	n.a.
entity	object		resource (an instance of rdfs:Resource)	n.a.
individual		individual concept		individual (an instance of owl:Thing)
data value	data value		literal (an instance of rdfs:Literal)	data value

2.2. Entity Types and Datatypes

An ***entity type*** is an *entity* that has an *extension* (the set of entities that are instances of it) and an *intension*, which includes an applicability criterion for determining if an entity is an instance of it. A ***basic entity type*** is an entity type whose instances are individuals. A ***datatype*** is a *set* whose members are *data values*.

REWERSE-I1	***UML***	***SBVR***	***RDF***	***OWL***
entity type	type / class	object type / general concept	class (an instance of rdfs:Class)	n.a.
basic (1st order) entity type				class (an instance of owl:Class, which is a subclass of rdfs:Class)
datatype	datatype		datatype (an instance of rdfs:Datatype)	

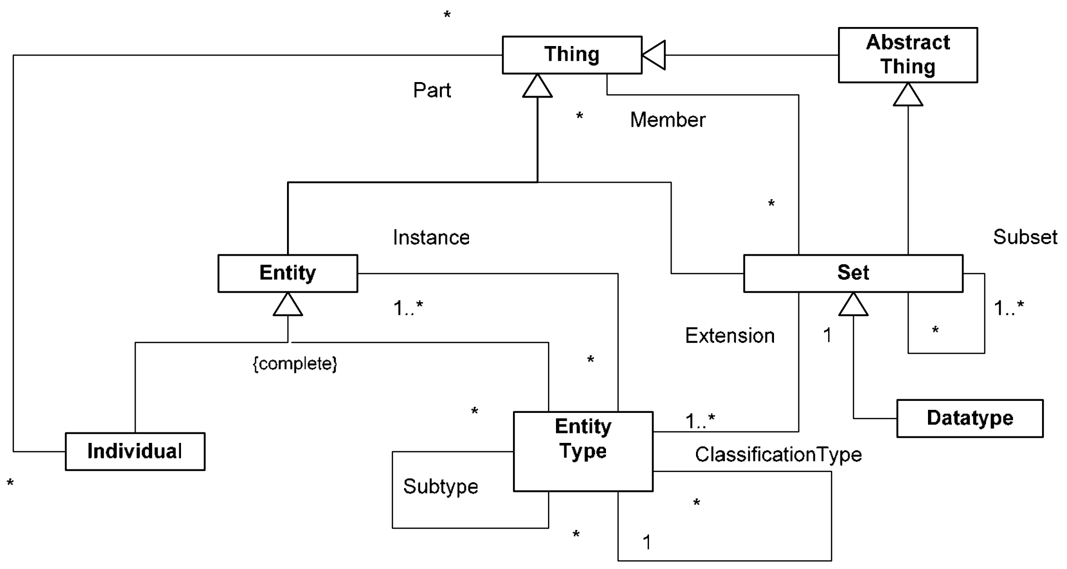


Figure 1: The foundational vocabulary adopted by I1

2.3. Facts and Statements

We distinguish between 5 different kinds of facts. In addition to the basic fact kinds of classification facts, association facts and attribution facts, we also consider categorization facts and aggregation facts.

REWERSE-I1	*UML*	*SBVR*	*RDF*	*OWL*
association fact	Link	associative fact	n.a.	n.a.
binary association fact	binary link	binary associative fact	triple, statement	individual-valued property fact
attribution fact	object-attribute-value triple	is-property-of fact		data-valued property fact
classification fact	instanceOf dependency	assortment fact	rdf:type statement	classification fact
categorization fact	n.a.	categorization fact	n.a.	n.a.
aggregation fact	aggregation link	partitive fact	n.a.	n.a.
generalization statement	Generalization	specialization fact	subclassOf statement	subclass axiom

2.4. Fact Types

A fact type corresponds to a predicates in predicate logic. But while there is no further distinction between different kinds of predicates in standard logic, we distinguish between four different kinds of fact types, as depicted in the following table.

REWERSE-I1	*UML*	*SBVR*	*RDF*	*OWL*
association fact type	association	associative fact type	n.a.	n.a.
binary fact type	binary association	associative binary fact type	property (an instance of rdf:Property)	individual-valued property (an instance of owl:ObjectProperty)
attribution fact type	attribute	is-property-of fact type		data-valued property (an instance of owl:DatatypeProperty)
classification fact type	n.a.	assortment fact type	n.a.	n.a.
categorization fact type	n.a.	categorization fact type	n.a.	n.a.
aggregation fact type	aggregation	partitive fact type	n.a.	n.a.

3. SBVR

The Semantics of Business Vocabulary and Business Rules (SBVR) is the approach to developing and structuring business vocabulary suited for business people to express business rules. [SBVR]

It is taken to be within the business model layer of OMG's Model Driven Architecture and implies that SBVR is targeted at business rules and that business models describe businesses and not the IT systems that support them.

3.1. Business vocabulary

A business vocabulary contains 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. SVBR is the result of the integration of this ISO standard, formal logics, linguistics, and practical experience from BRT members who are foremost practitioners in the field of Business Vocabulary for Business Rules.

SBVR-based business vocabulary strengths the semantics of ordinary business glossaries of terms and their definitions.

3.2. Business Rule

The SBVR follows a common-sense definition of 'business rule':

Business Rule: *rule that is under business jurisdiction*

'Under business jurisdiction' is taken to mean that the business can enact, revise and discontinue their business rules as they see fit.

All business rules need to be *actionable*. This means that a person who knows about a business rule could observe a relevant situation (including his or her own behavior) and decide directly whether or not the business was complying with the rule.

Just because business rules are actionable, this does *not* imply they are always automatable. Many business rules, especially operative business rules, are *not* automatable in IT systems.

3.3. Rules and Formal Logic

An important driver in the SBVR's treatment of 'rule' is consistency with formal logics. It is recommended that the best treatment for the SBVR's interpretation of rules would be as modal propositions claiming either *obligation* or *necessity* modalities.

Consequently, in SBVR, a Rule is "an element of guidance that introduces an obligation or a necessity". The two fundamental categories of Rule are:

- **Structural Rule** (necessities): These are rules about how the business chooses to organize (i.e., 'structure') the things it deals with. Structural Rules supplement definitions. For example (from EU-Rent): Necessity: A Customer has at least one of the following:
    - a Rental Reservation.
    - an in-progress Rental.
    - a Rental completed in the past 5 years.

- **Operative Rules** (obligations): These are rules that govern the conduct of business activity. In contrast to Structural Rules, Operative Rules are ones that can be *directly* violated by people involved in the affairs of the business. For example (from EU-Rent):

  Obligation: A Customer who appears intoxicated or drugged must not be given possession of a Rental Car.

3.4. Rules, Fact Types and Concepts expressed by Terms

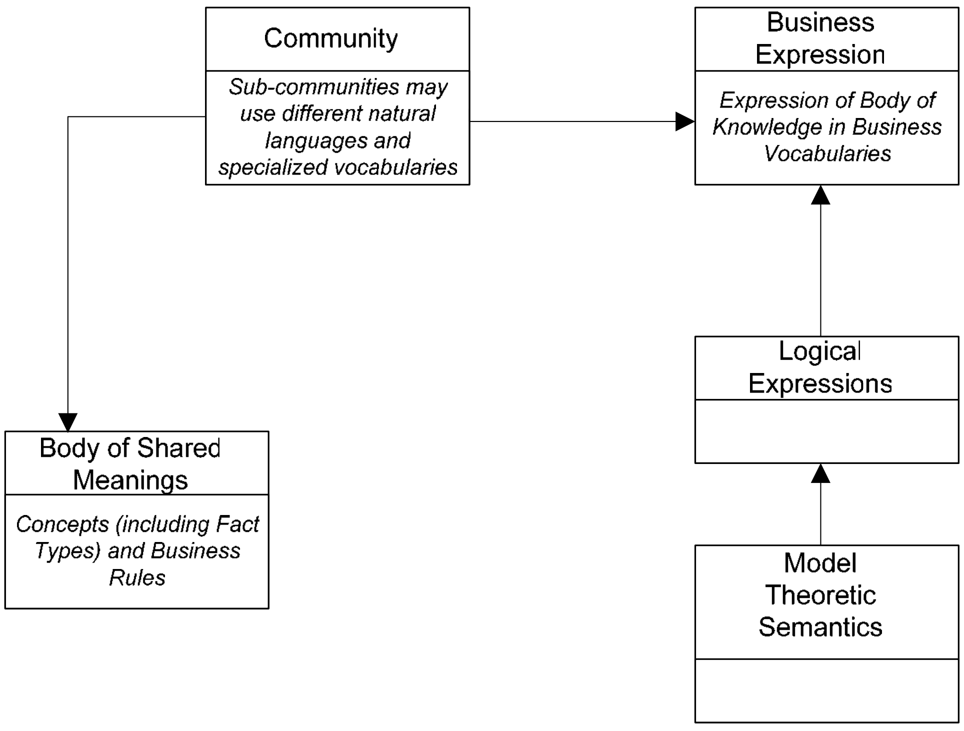
Rules in SBVR approach are always constructed by applying modal operators to fact types. For example, the rule “A Rental must not have more than three Additional Drivers” is based on the fact type “Rental has Additional Driver”.

By this means, SBVR realizes a core principle of the Business Rules Approach, which is that “Business rules build on fact types, and fact types build on concepts as expressed by terms.” This notion is well-documented in published material by foremost industry experts over the past 10 years beginning with the Business Rules Group’s seminal paper in 1995.

One important consequence of the SBVR’s approach in this regard is that concepts (including fact types) are *distinct* from rules, which are in a separate Compliance Point. This design permits SBVR’s support for concepts (including fact types) to be optionally used on its own for building business vocabularies.

3.5. Overview of SBVR

SBVR can be viewed as having five major aspects, as illustrated below:


Community
Sub-communities may use different natural languages and specialized vocabularies
Business Expression
Expression of Body of Knowledge in Business Vocabularies
Logical Expressions
Body of Shared Meanings
Concepts (including Fact Types) and Business Rules
Model Theoretic Semantics

3.5.1. Community

The basis for business vocabulary is Community. At the business level, communities of primary importance are enterprises for which business rules are being established and expressed. However, other communities - the industry in which an enterprise operates, partner enterprises, standards groups, regulatory authorities, etc. - also need to be recognized.

An important aspect of Community is that sub-communities within an enterprise may need its Body of Shared Meanings (starting with fundamental concepts) to be expressed in different vocabularies, ranging from specialized jargon to different natural languages. In SBVR, such sub-communities are called “Speech Communities”.

3.5.2. 'Body of Shared Meanings'

A community has a 'body of shared meanings', set of concepts (including fact types) and business rules for which there is a shared understanding in a given semantic community.

The structure of the 'Body of Shared Meanings' is defined by associating abstract concepts, fact types and business rules, not by associating statements in any given language.

3.5.3. 'Logical Formulation'

'Logical Formulation' means logical statements in formal logic language. The purpose is to formalize business statements.

'Logical formulation' supports the mapping of a 'Body of Shared Meanings' to Vocabularies used by Communities.

3.5.4. Business Expression

The concepts and business rules in a body of shared meanings need to be expressed in vocabularies acceptable to, and usable by, Speech Communities that share their meaning. These vocabularies may be in different natural languages, in artificial languages such as the UML, or in specialized subsets of natural languages, as used by, for example, engineers or lawyers.

SBVR supports mapping of business meaning to concrete language by associating elements of the Body of Shared Meanings with signifiers, e.g. terms such as “Customer”, “Car”, “Branch” for concepts, and fact symbols (often verb phrases) such as “rents”, “is located at” for fact types. Logical formulations provide the structure, and signifiers are placed in logical formulations to provide the expression.

3.5.5. Logical Expressions

SBVR has a sound theoretical foundation of formal logic, underpinning the structures of “Bodies of Shared Meanings”. The base is first-order predicate logic (although there is no restriction on extension into higher-order logics), with some limited extensions into modal logic – notably some deontic forms, for expressing obligation and prohibition, and alethic forms for expressing necessities.

3.6. SBVR Structured English

The SBVR Structured English is just one of possibly many notations that can map to the SBVR Metamodel.

There are any number of ways that BSBR vocabulary and other English vocabularies described using SBVR can be combined with common English words and structures to express definitions and statements. However expressed, the semantics of definitions and rules can be formally represented in terms of the SBVR vocabulary and, particularly, in terms of logical formulations (the SBVR conceptualization of formal logic).

The SBVR Structured English is not meant to offer all of the variety of common English, but rather, it uses a small number of English structures and common words to provide a simple and straightforward mapping.

All formal definitions and rules that are part of 'SBVR in terms of itself' are stated using the SBVR Structured English. These statements can then be interpreted automatically in order to create MOF and/or XMI representations.

The description of the SBVR Structured English is divided into sections.

- Expressions in SBVR Structured English
- Describing a Vocabulary
- Vocabulary Entries
- Specifying a Rule Set
- Rule and Clarification Entries

3.6.1. Expressions in SBVR Structured English

Examples of BSBR approach at the end of this section contain numerous amounts of statements.

Note that these fonts are also used for individual designations in the context of ordinary, unformalized statements in order to note that defined concepts are being used.

There are five markup font styles with formal meaning:

<u>term</u>	The 'term' font is used for a designation for a type, one that is part of a vocabulary being used or defined (e.g., person, rental car, fact type). If a designation is mentioned (where the designation is itself the subject of a statement) it appears in this style within single quote marks (e.g., 'modality')
concept	The 'concept' font is applied to a term in the special case where the term is used to name the represented concept rather than to refer to things denoted by the term. This is a reference to the concept itself.
<u>Name</u>	The 'name' font is used for a designation of an individual concept — a name. Names tend to be proper nouns (e.g., <u>California</u>).
verb	The 'verb' font is used for designations for fact types — usually a verb, preposition or combination thereof. Such a designation is defined in the context of a form of expression.

keyword	The ‘keyword’ font is used for linguistic symbols used to construct statements – the words that can be combined with other designations to form statements and definitions (e.g., ‘each’ and ‘it is required that’).

The SBVR Structured English uses designations and forms of expressions exactly as they are defined in a vocabulary. Plural forms are not used to avoid linguistic difficulties. For example, a formal statement would say “each concept” rather than “all concepts.” Both the active form and the passive form of a verb need to be defined in a vocabulary if both are used.

3.6.2. Key words and phrases for “logical formulations”

Key words and phrases are shown below for expressing each kind of “logical formulation”. The letters ‘*n*’ and ‘*m*’ represent use of a literal whole number. The letters ‘*p*’ and ‘*q*’ represent expressions of propositions.

3.6.2.1. Quantification

Universal quantification (each), existential quantification (some, at least one), at-least-n quantification (at least *n*), at-most-one quantification (at most one), at-most-n quantification (at most *n*), exactly-one quantification (exactly one), exactly-n quantification (exactly *n*), numeric range quantification (at least *n* and at most *m*), at-least-2 quantification (more than one).

3.6.2.2. Logical Operations

Logical negation (it is not the case that *p*), conjunction (*p* and *q*), disjunction (*p* or *q*), exclusive disjunction (*p* or *q* but not both), implication (if *p* then *q*, *q* if *p*), equivalence (*p* if and only if *q*), nand formulation (not both *p* and *q*), nor formulation (neither *p* nor *q*), whether-or-not formulation (*p* whether or not *q*)

Where a subject is repeated when using ‘and’ or ‘or’, the repeated subject can be elided.

The keyword ‘not’ is used within an expression before the verb “is” as a way of introducing a logical negation. Also, the key words “does not” are used before other verbs (modified to be infinitive) to introduce a logical negation.

3.6.2.3. Modal Operations

There are two styles of SBVR Structured English:

1. Prefixed Rule Keyword Style
2. Embedded (Mixfix) Rule Keyword Style

The Prefix Style introduces rules by prefixing a statement with keywords that convey a modality

Operative Business Rules and Clarifications	Structural Rules and Clarifications*
It is obligatory that	It is necessary that
It is prohibited that	It is impossible that
It is permitted that	It is possible that

The Embedded Style features the use of rule keywords embedded (usually in front of verbs) within rules statements of appropriate kind. The following key words are used within expressions having a verb (often modified to be infinitive) to form verb complexes that add a modal operation.

Operative Business Rules and Clarifications	Structural Rules and Clarifications*
… must …	… always …
… must not …	… never …
… may …	… sometimes …

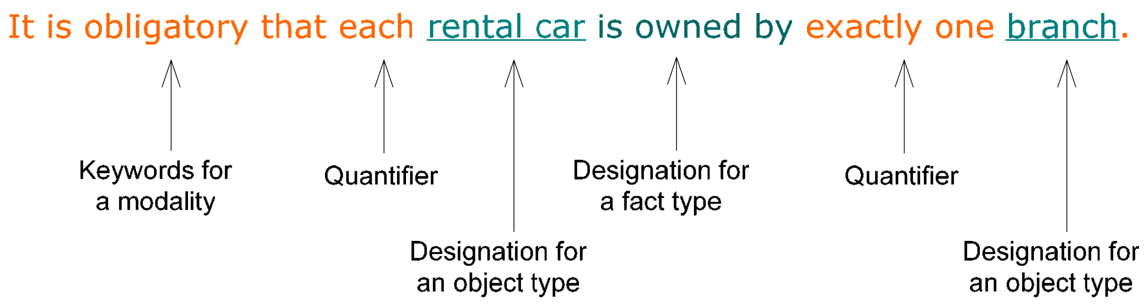
3.6.3. Other keywords

the	1. used with a designation to make a pronominal reference to a previous use of the same designation. This is formally a binding to a variable of a quantification. 2. introduction of a name of an individual thing or of a definite description
a, an	universal or existential quantification, depending on context based on English rules
that	1. when preceding a designation for a type or role, this is a binding to a variable (as with 'the') 2. when after a designation for a type or role and before a designation for a fact type, this is used to introduce a restriction on things denoted by the previous designation based on facts about them

3.6.4. Example

It is obligatory that each rental car is owned by exactly one branch.

The example above includes three key words or phrases, two designations for types and one for a fact type (from a form of expression), as illustrated below.


It is obligatory that each rental car is owned by exactly one branch.
Keywords for a modality
Quantifier
Designation for an object type
Designation for a fact type
Quantifier
Designation for an object type

3.7. Describing a Vocabulary

A vocabulary is described in a document section having glossary-like entries for concepts having representations in the vocabulary. The introduction to a vocabulary description includes the vocabulary's name and can further include any of the several kinds of details: vocabulary name, description, source, speech community, language, included vocabulary, and note.

3.8. Vocabulary Entries

Each entry is for a single concept, called the entry concept. It starts with a representation of the concept, either a designation or a form of expression.

Any of several kinds of captioned details can be listed under the representation. A skeleton of a vocabulary entry includes the following captions: designation of form of expression, definition, source, dictionary basis, general concept, concept type, symbol type, necessity, possibility, reference scheme, note, example, synonym, synonymous form, see, qualifier.

The explanation of the use of some captions is given below.

3.8.1. Designation or Form of Expression

The designation or form of expression, called the ‘primary representation’ with respect to each entry, can be for any concept type. The primary representation for a fact type is a form of expression.

3.8.2. Definition

A definition is shown as an expression that can be logically substituted for the primary representation. It is not a sentence, so it does not end in a period.

A definition can be fully formal, partly formal or informal. It is fully formal if all of it is styled as described above. A partially-formal definition starts with a styled designation for a more general concept but other details depend on external concepts.

Styles of definition are explained separately for different types of concepts.

3.8.2.1. Definition of a Type or Role

A common pattern of definition begins with a designation for a more general concept followed by the keyword ‘that’ and then an expression of necessary and sufficient characteristics that distinguish a thing of the defined concept from other things of the more general concept. Another less used pattern also leads with a designation for a more general concept but then uses the word ‘of’ with another expression as explained above.

Two kinds of information are formally expressed by a fully formal definition.

1. A fact that the concept being defined is a category of a particular more general concept
2. A closed set projection that defines the concept.

Only the first kind of information is formally expressed by a partially formal definition. A partially formal definition leads with a styled designation that is for a more general concept. That designation is generally followed by the keyword ‘that’ and then an informal expression of necessary and sufficient characteristics.

3.8.2.2. Definition of an Individual Concept

A definition of an individual concept is just like a definition of a type except that it must be a definite description of one single thing.

A definition of an individual concept can generally be read as a statement using the following pattern. The leading “The” is optionally used depending on the designation.

[The] <designation> is the <definition>.

3.8.2.3. Definition of a Fact Type

A definition given for a fact type is an expression that can be substituted for a simple statement expressed using a form of expression of the fact type.

The definition must refer to the placeholders in the form of expression. This is done in order to relate the definition to the things that play a role in instances of the fact type. Whether or not the definition is formal, each reference to a placeholder appears in the ‘term’ font and is preceded by the definite article, “the”.

A definition of a fact type can generally be read using the pattern below, which is shown for a binary fact type but works for fact types of any arity (“a” represents either “a” or “an”).

> A fact that a given <placeholder 1> <fact type designation> a given <placeholder 2> is a fact that <definition>.

For example: A fact that a given statement expresses a given proposition is a fact that the proposition is what is meant by the statement.

Similarly, the equivalence understood from a definition of a fact type can generally be read using the following pattern:

> A <placeholder 1> <fact type designation> a <placeholder 2> if and only if <definition>.

For example: A statement expresses a proposition if and only if the proposition is what is meant by the statement.

3.9. Specifying a Rule Set

A rule set is specified in a document section having several individual entries for rules and clarifications. The introduction to a rule set includes the rule set’s name, description, vocabulary, note, and source.

3.10. Rule and Clarification Entries

Each entry in a rule set is an element of guidance—expressed as one of the following:

- An operative business rule statement.
- An operative business rule clarification statement.
- A structural business rule statement.
- A structural business rule clarification statement.

Business rules include only those rules under business jurisdiction. Similarly, business rule clarifications include only those clarifications under business jurisdiction. Entries can also be made for structural rules and clarifications that are not under business jurisdiction. These entries are, respectively:

- A structural rule statement.
- A structural rule clarification statement.

Each entry includes the statement itself and optionally includes the following captions: name, guidance type, description, source, synonymous form, note, example, enforcement level.

3.10.1. Rule Statement or Clarification Statement

A rule statement or clarification statement can be expressed formally or informally. A statement that is formal uses only formally styled text — all necessary vocabulary is available (by definition or adoption) such that no external concepts are required. Such a statement can be represented as a logical formulation.

3.10.2. Enforcement Level

The ‘Enforcement Level’ caption labels the enforcement level that applies to an operative business rule.

4. Metalog

The main idea of the Semantic Web is to provide a flexible “basic semantic language”, RDF, which makes possible to codify the basic bricks of reasoning: then on top of this “universal semantic alphabet”, more and more sophisticated technologies can be employed, so to bring the expressive power to higher levels [Metalog]. What has been somehow missing has been technologies that follow not just the technological axis, but the people axis, i.e. technologies that empower the people and try to make the semantic web closes to the widest possible audience, possible sacrificing some of this power. For this purpose Metalog uses a so-called Pseudo Natural Language (PNL), which is much similar to natural language, and therefore allows an easy interfacing to the more complex underlying technologies of the Semantic Web.

In the examples part of this document we demonstrate verbalization of business rules with Metalog. The Metalog approach is composed by 3 major layers.

4.1. The Metalog Model Level

The first layer consists of an enrichment of the RDF model. RDF provides the basis for the data structuring on the web in a consistent and accurate way. However, RDF is only the first step towards the construction of what is called *semantic web*. RDF provides only the basic vocabulary in which data can be expressed and structured. But the problem of accessing and managing of the data arises. Metalog provides the way to express logical relationships like “and”, “or” and “implication”. These are denoted with ml:and, ml:or and ml:imply respectively. It provides a negation (not) operator, and the classic comparison, which is used to name variables. Finally, it also provides two annotation extensions, ml:annotation and ml:ns. The “semantic layer” is built on top of RDF using so-called *RDF schema*.

4.2. The Metalog Logic Level

The second layer consists of a “logical interpretation” of RDF data into logic. It is essentially a subset of equational First Order Logic. The basic intuition of the recursive interpretation mapping:

- The ml:and, ml:or and ml:imply and “not” operators are translated into logical conjunction, logical disjunction, logical implication and logical negation respectively. The interpretation of operators, in almost all useful logics, is the same, except logical interpretation of the *not* connector. There are different alternatives to consider, giving different logics. The Metalog choice follows from the intrinsic nature of the World Wide Web (where the Metalog system is mainly expected to be used), as a distributed knowledge basis, with possible partial information available: the interpretation of the *not* connector has been chosen to be *negation as failure (NAF)*, that is to say, the Closed World Assumption is adopted.

- Ground mathematics, and equality/inequality, can be naturally mapped into this logic (this also means that the corresponding infinitary axioms for the comparison and equality predicates has to be presented in MLL).
- Each literal/URI-reference is mapped into some constant via an injective mapping.
- ml:name, ml:annotation and ml:ns are mapped into distinguished (i.e. distinct and outside of the image of the literals/URI) constants.
- Each triple (S, P, O) that neither has Metalog extensions, nor is part of an RDF container description is mapped into the predicate P(S,O).
- RDF Container description of type Seq (Bag, Alt resp.) containing *k* objects are mapped into a distinguished *k*-ary operator SEQ_k (BAG_k, ALT_k resp) applied to (the mapping) of its *k* operands.

4.3. The PNL

The third level is a language interface (PNL) for writing structured data and reasoning rules. In principle, data and rules can be written directly in RDF, using RDF syntax and the Metalog schema, but this is not convenient from the practical viewpoint. The PNL stresses user-friendliness as much as possible: a program is a collection of natural language assertions. Here is the simple example of the Metalog session:

```
comment: the simplest session.
comment: we start defining what things are.
CAR represents the car “Mini_Cooper” from the branch
“http://www.example.com/car”.
IS represents the verb “is” from the collection
“http://www.relationships.example.com/verbs”.
comment: now we say something.
CAR IS “available”.
comment: now we ask something.
do you know whether CAR IS AVAILABLE?
```

When this program is loaded in Metalog, it is actually colored in different ways, so to better outline the various components of the discourse. The first, second, fifth and sixth sentences would output as red, indicating that these are comments.

The third and fourth sentences would appear as green lines: green indicates that these are so called “representation” parts. Representations are useful to denote shorthand: in these sentences, we associate some entity (like CAR) to its corresponding concept (the concept “Mini_Cooper” from the branch “http://www.example.com/car”.). This means in certain sense that, anywhere, writing the whole the car “Mini_Cooper” from the branch “http://www.example.com/car”. is pretty much equivalent to writing its representation (the shorter CAR).

The last two sentences (the sixth and the eight) would appear on the Metalog screen in blue, signaling that they can be either “assertions” or “queries”. An assertion is a sentence where we state something. For instance, the first of such a blue lines in the Metalog discourse (CAR IS “available”.) states precisely what it says. A query, instead, is a sentence where we are asking Metalog for answers. In fact, the second blue line is a query (do you know whether CAR IS AVAILABLE?). Queries can be easily distinguished because they end with a question mark (“?”). Representations, assertions, and queries are the three types of sentences in any Metalog discourse.

Metalog queries not just return truth, but more meaningful answers, which is achieved by the process of instantiations of the answers (where free variables are bound to results, much like Prolog), and, above all, via return process called *feedback*. Feedback allows re-computing back an answer, and trying to model it in the form of natural language reply. To this extent, feedback utilizes special annotations, which codify at the RDF level the information expressed by the representations, and use them back after a reply is given by the inference engine, so to build up a meaningful answer. For example, the above query would return that

```
CAR IS “available”.
```

together with the opportune (*minimal*) set of representations needed to understand the answer.

5. Attempto Controlled English

Attempto Controlled English (ACE) is a controlled natural language, i.e. a precisely defined subset of full English that can automatically and unambiguously be translated into discourse representation structures, a variant of first-order logic. Thus ACE is both human and machine understandable.

The following is a brief introduction into ACE. For a full account readers should consult the ACE documentation found at the Attempt website [www.ifi.unizh.ch/attempto].

5.1. Vocabulary

The vocabulary of ACE comprises

- predefined function words (e.g. determiners, conjunctions, prepositions)
- user-defined, domain-specific content words (nouns, verbs, adjectives, adverbs)

The Attempto system provides a large basic lexicon of content words. Users can define additional content words with the help of a lexical editor, or can import existing lexica.

5.2. Grammar

The grammar of ACE defines and constrains the form and the meaning of ACE texts. ACE's grammar is expressed as a small set of construction rules.

5.2.1. ACE Texts

An ACE text is a sequence of anaphorically interrelated sentences. There are

- simple sentences
- composite sentences

Furthermore, there are query sentences that allow users to interrogate the contents of an ACE text.

5.2.2. Simple Sentences

Simple ACE sentences have the following general structure:

subject + verb + complements + adjuncts

Each simple sentence has a subject and a verb. Complements (direct and indirect objects) are necessary for transitive verbs (*insert something*) and ditransitive verbs (*give something to somebody*), whereas adjuncts (adverbs, prepositional phrases) are optional.

Here are examples of simple sentences.

> A customer waits.
>
> A customer inserts 2 cards .
>
> A card is valid.

All elements of a simple sentence can be elaborated upon to describe the situation in more detail.

To further specify the nouns *customer* and *card*, we could add adjectives

> A new customer inserts 2 valid cards.

possessive nouns and of-prepositional phrases

> John's customer inserts a card of Mary.

or proper nouns and variables as appositions

> The customer Mr Miller inserts a card A.

Other modifications of nouns are possible through relative sentences

> A customer who is new inserts a card that he owns.

which are described below since they make a sentence composite.

We can also detail the *insert* event, e.g. by adding an adverb

> A customer inserts some cards manually.

or equivalently

> A customer manually inserts not more than 2 cards.

or by adding prepositional phrases, e.g.

> A customer inserts a card into a slot.

We can combine elaborations to arrive at

> John's customer who is new inserts a valid card of Mary manually into a slot A.

5.2.3. Composite Sentences

Composite sentences are recursively built from simpler sentences through coordination, subordination, quantification, and negation.

Coordination by *and* is possible between sentences and between phrases of the same syntactic type.

> A customer inserts 2 cards and the machine checks their codes.
>
> A customer inserts a card and enters a code.
>
> An old and trusted customer enters a card and a code.

(NB. The coordination of the noun phrases *a card and a code* represents a plural object.)

Coordination by *or* is possible between sentences and between verb phrases.

> A customer inserts a card or enters a code.

Coordination by *and* and *or* is governed by the standard binding order of logic, i.e. *and* binds stronger than *or*. Commas can be used to override the standard binding order. Thus the sentence

> A customer inserts a VisaCard or inserts a MasterCard, and inserts a code.

means that the customer inserts a VisaCard and a code or a MasterCard and a code.

There are two forms of subordination: relative sentences and if-then sentences.

Relative sentences starting with *who*, *which*, *that* allow to add detail to nouns, e.g.

> A customer who is new inserts a card that he owns.

With the help of if-then sentences we can specify conditional or hypothetical situations, e.g.

> If a card is valid then a customer inserts it.

Note the anaphoric reference via the pronoun *it* in the then-part to the noun phrase *a card* in the if-part.

Quantification allows us to speak about all objects of a certain class, or to denote explicitly the existence of at least one object of this class. To express that all involved customers insert cards we can write

> Every customer inserts a card.

This sentence means that each customer inserts a card that may, or may not, be the same as the one inserted by another customer. To specify that all customers insert the same card – however unrealistic that situation seems – we can write

> There is a card that every customer inserts.

ACE does not know the passive voice. To state that every card is inserted by a customer we write somewhat indirectly

> For every card there is a customer who inserts it.

The textual occurrence of a quantifier opens its scope that extends to the end of the sentence, or – in coordinations – to the end of the respective coordinated sentence.

Negation allows us to express that something is not the case, e.g.

> A customer does not insert a card.
>
> A card is not valid.

To negate something for all objects of a certain class one uses *no*

> No customer inserts more than 2 cards.

or, equivalently, *there is no*

> There is no customer who inserts a card.

To negate a complete statement one uses sentence negation

> It is not the case that a customer inserts a card.

5.2.4. Query Sentences

Query sentences permit us to interrogate the contents of an ACE text. There are yes/no-queries and wh-queries.

Yes/no-queries establish the existence or non-existence of a specified situation. If we specified

> A customer inserts a card.

then we can ask

> Does a customer insert a card?

to get a positive answer.

With the help of wh-queries, i.e. queries with query words, we can interrogate a texts for details of the specified situation. If we specified

> A new customer inserts a valid card manually.

we can ask for each element of the sentence, e.g.

Who inserts a card?

Which customer inserts a card?

What does the customer insert?

How does the customer insert a card?

Note, however, that we cannot ask for the verb itself.

5.3. Constraining Ambiguity

To constrain the ambiguity of full natural language ACE employs three simple means

- some ambiguous constructs are not part of the language; unambiguous alternatives are available in their place
- all remaining ambiguous constructs are interpreted deterministically on the basis of a small number of interpretation rules
- users can either accept the assigned interpretation, or they must rephrase the input to obtain another one

5.4. Avoidance of Ambiguity

Here is an example of how ACE replaces ambiguous constructs by unambiguous constructs. In full natural language relative sentences combined with coordinations can introduce ambiguity, e.g.

A customer inserts a card that is valid and opens an account.

In ACE the sentence has the unequivocal meaning that the customer opens an account. This is reflected as

A customer inserts {a card that is valid} and opens an account.

To express the alternative – though not very realistic – meaning that the card opens an account the relative pronoun *that* must be repeated, thus yielding a coordination of relative sentences.

A customer inserts a card that is valid and that opens an account.

with the interpretation

A customer inserts {a card that is valid and that opens an account}.

5.5. Interpretation rules

However, not all ambiguities can be safely removed from ACE without rendering it artificial. To deterministically interpret otherwise syntactically correct ACE sentences we use about 20 interpretation rules. Here are some examples. If we write

The customer inserts a card with a code.

we get the interpretation

The customer {inserts a card with a code}.

that reflects ACE's interpretation rule that a prepositional phrase always modifies the verb. However, this is probably not what we meant to say. To express that the code is associated with the card we can employ the interpretation rule that a relative sentence always modifies the immediately preceding noun phrase.

The customer inserts a card that carries a code.

yielding the interpretation

The customer inserts {a card that carries a code}.

or – to specify that the customer inserts a card and then a code – as

> The customer inserts a card and a code.

Adverbs can precede or follow the verb. To disambiguate the sentence

> The customer who inserts a card manually enters a code.

we employ the interpretation rule that the postverbal position has priority.

> The customer who {inserts a card manually} enters a code.

5.6. Anaphoric References

Usually specifications consist of more than one sentence.

> A customer enters a card and a code. If a code is valid then SimpleMat accepts a card. If a code is not valid then SimpleMat rejects a card.

To express in the sentences above that the occurrences of *card* and *code* should mean the same card and the same code, ACE provides anaphoric references via the definite article

> A customer enters a card and a code. If the code is valid then SimpleMat accepts the card. If the code is not valid then SimpleMat rejects the card.

During the processing of ACE texts the ACE system replaces any anaphoric reference by the most recent and most specific accessible noun phrase that agrees in gender and number, yielding

> A customer enters a card and a code. If [the code] is valid then SimpleMat accepts [the card]. If [the code] is not valid then SimpleMat rejects [the card].

What does "most recent and most specific" mean? Given the sentence

> A customer enters a red card and a blue card.

then

> The card is correct.

yields

> [The blue card] is correct.

while

> The red card is correct.

yields

> [The red card] is correct.

What does "accessible" mean? According to Discourse Representation Theory noun phrases introduced in if-then sentences, universally quantified sentences or negations are not accessible as antecedents of anaphora. Thus *the card* in

> A customer does not enter a card. The card is correct.

cannot refer to *card*.

Anaphoric references are also possible via personal pronouns

> A customer enters a card and a code. If it is valid then SimpleMat accepts the card. If it is not valid then SimpleMat rejects the card.

or via variables

> A customer enters a card CARD and a code CODE. If CODE is valid then SimpleMat accepts CARD. If CODE is not valid then SimpleMat rejects CARD.

Anaphoric references via definite articles and variables can be combined.

A customer enters a card CARD and a code CODE. If the code CODE is valid then SimpleMat accepts the card CARD. If the code CODE is not valid then SimpleMat rejects the card CARD.

Note that proper nouns like SimpleMat always refer to the same object.

5.7. Extending ACE to Handle Modality

Obviously, ACE is highly suited to express business rules and policy rules. However, these rules often use modal constructs like *can*, *cannot*, *must*, *must not*, *is/does always*, *is/does never* etc. that are not at all, or not satisfactorily, covered in ACE.

Currently ACE provides only variables as a restricted means to express modality. For instance, to express *is/does always* we can say

For every time T ... is/does ... at T.

and to express *is/does never*

There is no time T such that ... is/does ... at T.

There remains a definite need to extend ACE by better means to express modality. However, for obvious reasons we want to remain within first-order logic, we do not want to extend ACE in ways that increase the ever-present danger of ambiguity, and we do not want to increase the size – and decrease the performance – of the ACE parser. Taking these constraints into account, we have developed two proposals to add modality to ACE:

- fixed modal phrases, and
- sentence reification.

5.7.1. Fixed Modal Phrases

Fixed modal phrases are patterned after ACE's sentence negation

It is not the case that a customer inserts a card.

where the fixed phrase *it is not the case* negates the embedded sentence *a customer inserts a card*. Similarly we can build

It is possible that a customer inserts a card.

It is obligatory that a customer inserts a card.

It is necessary that a customer inserts a card.

...

by introducing the appropriate fixed modal phrases that modify the respective embedded sentence. The DRS derived from one of these sentences consists of the representation of the embedded sentence wrapped within the respective fixed modal phrase, and thus remains within first-order logic.

In the same way we can introduce negation as failure

It is not provable that a customer inserts a card.

to complement ACE's logical negation.

Deductions from sentences using fixed modal phrases must be carefully designed to prevent incorrect inferences since there is an interplay between the monotonicity of the embedded sentence and the fixed modal phrases. For example, though we want to deduce

A user accesses the data.

from

A privileged user accesses the data.

we probably would not want to deduce

It is allowed that a user accesses the data.

from

It is allowed that a privileged user accesses the data.

5.7.2. Sentence Reification

The idea of sentence reification is simple: sentences get labels that can be referred to as variables in other sentences. It is straightforward to express by sentence reification the examples used for the illustration of fixed modal rules. With

S1: A customer inserts a card.

we get

S1 is possible.

S1 is obligatory.

S1 is necessary.

S1 is not provable.

Note that the labelled sentence is the embedded sentence of the fixed modal phrases. Note furthermore that labelled sentences do not have a truth value.

Obviously, sentence reification is at least as expressive as fixed modal rules. However as we will see presently, sentence reification is in fact much more powerful than fixed modal rules. Let's look at two very similar looking sentences.

A customer promises a clerk to enter a card.

A customer expects a clerk to enter a card.

While in the first sentence the entering agent is the customer, in the second sentence the entering agent is the clerk. Deriving the correct logical representations of the two sentences is tricky. Sentence reification, however, clearly distinguishes the two cases.

S2: A customer enters a card. The customer promises a clerk S2.

S3: A clerk enters a card. A customer expects S3.

(Note that there are still unresolved problems concerning anaphoric references.)

Sentence reification has the advantage that ACE needs only to be extended by labels and references to labels; no other changes to the language or to the parser will be necessary. Sentence reification has one disadvantage, though. Splitting a modal sentence into a labelled one and another one that refers to the label may not be acceptable to all users of ACE.

5.8. Modality in ACE and in SBVR Structured English

While ACE's fixed modal phrases are syntactically – though perhaps not semantically – identical to the Prefixed Rule Keyword Style of SBVR Structured English (see above), we do not plan to offer in ACE the equivalent of the Embedded (Mixfix) Rule Keyword Style of SBVR Structured English, i.e. modal constructs like *can*, *cannot*, *must*, *must not*, or similar.

Though Hobbs [http://www.isi.edu/~hobbs/disinf-chap2a.pdf] has shown that these modal constructs can be represented in first-order logic by appropriate reification, the representations proposed by Hobbs would allow incorrect deductions. Furthermore, the necessary extensions of ACE could lead to new forms of ambiguity that we would rather avoid.

Using sentence reification to express modal constructs like *can*, *cannot*, *must*, *must not*, or similar is certainly possible, but leads to unnatural sentences. For instance, to express

    A customer can enter a card.

we could write

    S: A customer enters a card.
    A customer can S.

which is hardly acceptable English.

5.9. Alternatives to ACE: PENG

ACE allows users free-form input, but expects them to learn and to recall ACE's construction and interpretation rules. Though the number of construction and interpretation rules is small, occasional users could possibly formulate syntactically incorrect ACE sentences, or be unaware that a sentence has another meaning than the intended one.

To keep users from generating syntactically incorrect sentences and to restrict the possibility of wrong interpretations, Schwitter [http://www.ics.mq.edu.au/~rolfs/peng/] has developed the controlled language PENG that frees the users from knowing the grammatical restrictions of the controlled language. PENG's look-ahead text editor ECOLE guides the user during the composition of a PENG text indicating at each instance the restrictions that apply, respectively the alternatives that are allowed.

The idea of guiding users through the composition of a PENG texts seems very attractive. It remains to be seen, though, whether users do value the guidance they get by ECOLE, or feel unnecessarily constrained by it.

Concerning syntax and semantics PENG overlaps largely with ACE – which is no surprise since Schwitter was one of the original authors of ACE – does, however, not cover some of the newer extensions of ACE like plural. There does not seem to be a plan to add modality.

5.10. Alternatives to ACE: CLCE

Sowa [http://www.jfsowa.com/clce/specs.htm] has developed Common Logic Controlled English (CLCE), a formal language with an English-like syntax.

Concerning syntax ACE and CLCE are very similar, and their language constructs overlap to a large degree. There are, however, some significant differences. On the one side, CLCE is more expressive since it supports ontology for sets, sequences and integers, and also allows in-line declarations of words with references to databases. On the other side, unlike ACE CLCE does not cover plurals. CLCE also does not cover modality that ACE plans to introduce. While ACE successfully disambiguates a large number of constructs that in full English would be ambiguous, CLCE is much more restrictive and even introduces parentheses for disambiguation. In this sense CLCE is less expressive and less natural than ACE.

Concerning semantics, CLCE like ACE supports full first-order logic. However, it is not clear whether a CLCE parser exists that translates CLCE texts into first-order logic.

Like ACE, CLCE is intended for bi-directional translations, i.e.

controlled language ⇒ first-order logic ⇒ controlled language

According to the documentation found at [http://www.jfsowa.com/clce/specs.htm] the development of CLCE is not yet completed.

5.11. Verbalisation in ACE

The Attempto Parsing Engine (APE) translates the ACE text

John enters a card. Every card is correct.

unambiguously into the discourse representation structure drs/2, a variant of first-order logic.

```
drs([A,B,C,D,E],[named(B,‘John’),
object(B,named entity,person),structure(B,atomic),
quantity(B,cardinality,count unit,A,eq,1),structure(D,atomic),
quantity(D,cardinality,count unit,C,eq,1),object(D,card,object),
predicate(E,event,enter,B,D),drs([F,G],[structure(G,atomic),
quantity(G,cardinality,count unit,F,eq,1),
object(G,card,object)])=>drs([H,I],[property(I,correct),
predicate(H,state,be,G,I)])])
```

APE is implemented as a Definite Clause Grammar that is in principle reversible, i.e. can also be used to generate texts. However, APE's grammar rules use feature structures and procedural attachments that render APE non-reversible, i.e. APE cannot readily be used to translate the above discourse representation structure back into the original ACE text.

To achieve the verbalisation of the discourse representation structure in ACE, we will proceed in two steps.

First, since different ACE sentences can result in the same discourse representation structure, we will define a subset of ACE as the target language for verbalisation. This subset of ACE will, for instance, not allow for relative phrases, and use instead sentence conjunctions. Furthermore, there will be only a restricted form of anaphoric references.

Second, we will complement each grammar rule of APE with a complementing rule for the translation of elements of discourse representation structures into elements of ACE, i.e. for verbalisation.

Once we will be able to translate discourse representation structures into ACE, we can proceed to the next step, verbalising other formal notations in ACE. To do so we will use discourse representation structures as interlingua, i.e. translate the respective formal notation into a discourse representation structure, and then translate this into ACE. A prerequisite for verbalisation in ACE is that the respective formal notation is equivalent to (a subset of) first-order logic.

6. The First Version I1 Controlled English (I1CE-V1)

The I1 approach to verbalization under development is based on the template method of *SBVR Structured English* and aims at combining it with the DRT-based approach of *Attempto* for improving the quality of the English output. Like SBVR Structured English we are also using verbalization templates and font style markup. Verbalization templates allow a simple mapping from formal/visual representations to English expressions, while the markup allows a non-ambiguous reading of these expressions, and thus also supports capturing vocabularies and rules.

The expressions obtained from applying verbalization templates may, however, not be perfectly correct and natural. The correctness and naturalness of the obtained expressions can be improved by additionally applying certain linguistic improvement rules and by using more advanced linguistic techniques such as the DRT representations of Attempto.

Our approach deviates from SBVR by supporting

1. a systematic use of UML role names, which correspond to OWL property names.
2. a systematic treatment of classification both at the first-order and at the second-order level

The following keywords are used in the I1 Controlled English (I1CE) :

- *IF, THEN, OR, AND, NOT* – designate logical connectives
- the, a, an, that - *other keywords as explained in the section on SBVR*

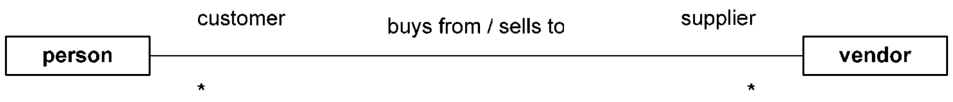
We use the following font types for marking up the different parts of an I1CE expression:

- **type term –** designates a type (that is part of a vocabulary being used or defined)
- *type term* **–** This markup is applied to a type term in the special case where the term is used to name the represented concept rather than to refer to things denoted by the term. This is a reference to the concept itself.
- connecting verb phrase – designates a (user-defined) domain predicate symbol
- *predefined connecting verb phrase* – designates a predefined predicate symbol
- name – designates an individual or data value

6.1. Association Facts

UML associations and OWL object property axioms are verbalized as ***association fact type expressions***.

For instance, consider the binary association



which has both an association name and a range class role name, as well as an inverse association name and a domain class role name. It can be verbalized by the following three equivalent fact type expressions:

person buys from **vendor**

person has **vendor** as *supplier*

person is *customer* of **vendor**

Its inverse can be verbalized correspondingly as

vendor sells to **person**

vendor has **person** as *customer*

vendor is *supplier* of **person**

This association fact type expression corresponds to the following OWL property axiom (expressed in the abstract syntax of OWL):

```
ObjectProperty(supplier domain(person) range(vendor)
      inverseOf(customer))
```

Since, in general, the class and property identifiers used in OWL do not allow a natural verbalization, we introduce three OWL annotation properties for the verbalization of class, association, and role names:

`vterm` – *verbalization terms* for verbalizing class names.

`vp` – *verb phrases* for verbalizing verb phrase predicate symbols.

`rgrolename` and `domrolename` – *role names* for verbalizing range and domain class role name predicate symbols.

We have to distinguish between the *range class role name* (rgrolename) and the *domain class role name* (domrolename). The corresponding OWL annotation property axioms to introduce these annotations are:

```
Namespace(i1ce=<http://rewerse.net/I1/I1CE#>)
AnnotationProperty(i1ce:vterm)
AnnotationProperty(i1ce:vp)
AnnotationProperty(i1ce:domrolename)
AnnotationProperty(i1ce:rgrolename)
```

With the help of these annotation properties we can express the above association by the following OWL axioms:

```
Namespace(pp=<http://rewerse.net/I1/ex1#>)
Class(pp:person annotation(i1ce:vterm("person")))
ObjectProperty(pp:supplier domain(pp:person) range(pp:vendor)
   annotation(i1ce:vp("buys from"))
   annotation(i1ce:rgrolename("supplier"))
   inverseOf(pp:customer))
ObjectProperty(pp:customer domain(pp:vendor) range(pp:person)
   annotation(i1ce:vp(“sells to”))
   annotation(i1ce:rgrolename(“customer”))
   inverseOf(pp:supplier))
```

We obtain association fact statements from association fact type expressions in the obvious way: by replacing the type terms (here **person** and **vendor**) with individual names (such as Tony Miller and DELL):

Tony Miller buys from DELL

Tony Miller has DELL as *supplier*
Tony Miller is *customer* of DELL

6.1.1. Expressing Functionality or Non-Functionality

It is straightforward to improve the above verbalization results by taking the functionality or non-functionality of an association into consideration. If an association is functional, we add the determiner "the" before the range role name; if it is inverse functional, we add the determiner "the" before the domain role name.

Otherwise, if the association is not functional (or not inverse functional) we use the determiner "a" before the range (or domain) role name. This gives, for instance,

Tony Miller has DELL as a supplier

Tony Miller is a customer of DELL

A functional role name, such as "*owner*" in

person is *owner* of **portfolio**

can be used as a function for referring to the particular instance (or, equivalently, to all instances) playing that role:

int is the age of the *owner* of **portfolio**

instead of

int is the age of **person** *AND* **person** is *owner* of **portfolio**

6.1.2. Deriving Predicates from Fact Types by Partial Instantiation

From a *n*-ary fact type we can derive predicates of arity smaller than *n* by partially instantiating the fact type. For instance, from the binary fact type

person buys from **vendor**,

we can derive the unary predicate

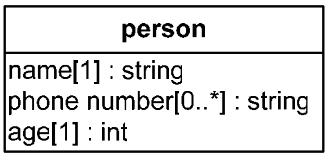
person buys from DELL

which defines the derived person subclass "DELL customer".

6.2. Attribution Facts

UML attributes and OWL datatype property axioms are verbalized as ***attribution fact type expressions***.

For instance, consider the following attributes of **person**:



These attributes can be verbalized as

person has **string** as *phone number*

or, inversely, as

string is *phone number* of **person**

This attribution fact type expression corresponds to the following OWL datatype property axiom:

```
DatatypeProperty(pp:phone_number domain(pp:person)
   range(xsd:string) annotation(i1ce:rcrolename("phone
number")))
```

In order to include a natural attribute name in such an OWL axiom we propose to use our `rgrolename` annotation property since an attribute name corresponds to a range class role name of the corresponding association with the attribute datatype as the range class.

Again, we obtain the corresponding fact statements by instantiating the type terms (here, e.g., **person** and **string**), i.e. by replacing them with instance names (such as Tony Miller and *"8027644"*):

> Tony Miller has *"8027644"* as *phone number*
>
> *"8027644"* is *phone number* of Tony Miller

As for association facts, we can express the functionality or non-functionality of an attribute by using the determiners "the" and "a":

> *"8027644r"* is a *phone number* of Tony Miller
>
> *34* is the *age* of Tony Miller

6.2.1. Constrained Attribution Facts

Attribution facts may be constrained by an attached data constraint. Consider, for instance, the fact type expression

> **int** is the *age* of **person** *AND* **int** *is greater than* *20*

Here, the attached data constraint expression is

> **int** *is greater than* *20*

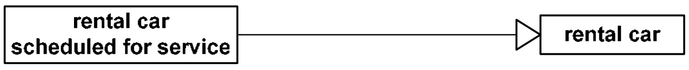
which is formed with the help of the predefined predicate "*is greater than*" and the data value "*20*".

If a data constraint refers to a functional attribute, such as "*age*", we can use it as a function and simplify the above fact type expression into

> the *age* of **person** *is greater than* *20*

6.3. Generalization Statements

A generalization statement, such as



or its equivalent in OWL,

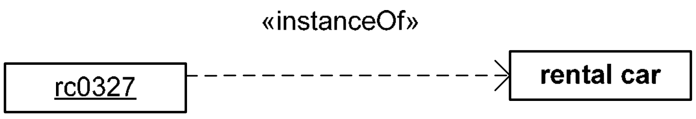
```
      Class(pp:rental_car_scheduled_for_service partial
pp:rental_car)
```

can be verbalized in the form of

> A *rental car scheduled for service* is a *rental car*

6.4. Classification Facts

A classification fact as expressed by the following UML *instanceOf*-dependency



can be verbalized as

> rc0327 is a *rental car*

where "rc0327" is an identifier for an individual. It corresponds to the following OWL fact:

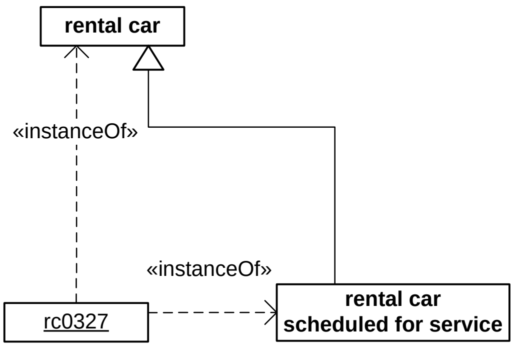
```
        Individual(pp:rc0237 type(rental_car))
```

Classification facts are not instances of a (user-defined) domain fact type. Rather they are instances of the meta-fact type

entity is a ***entity type***"

6.5. Categorization Facts

There are also 'subclassification' facts, better called *categorization facts*, such as expressed by



which can be verbalized as

The *rental car* rc0327 is a *rental car scheduled for service*

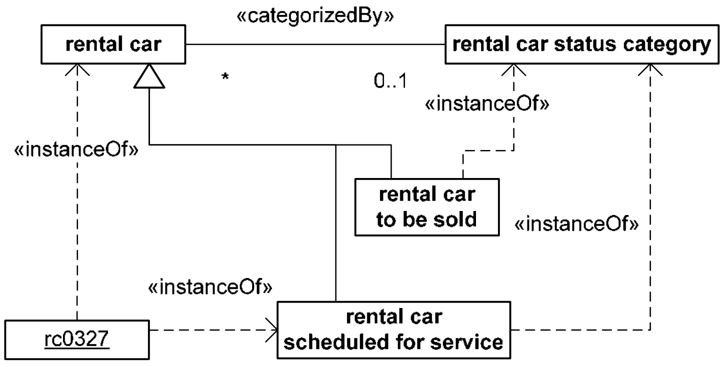
This categorization fact corresponds to the following combination of an OWL individual description and an OWL implication:

```
        Individual(pp:rc0237 type(pp:rental_car)
           type(pp:rental_car_scheduled_for_service))
        Class(pp:rental_car_scheduled_for_service partial
pp:rental_car)
```

For suchh categorization facts we can form the corresponding binary categorization fact type:

rental car is a ***rental car status category***

assuming that "*rental car scheduled for service*" is an instance of the second order entity type "**rental car status category**", which is an example of a *categorization type*, i.e. a type whose instances are categories (subtypes) of a certain type. This is depicted in the following diagram:



By instantiating the categorization type of a binary categorization fact type, we obtain a unary categorization fact type like, for instance,

rental car is a *rental car scheduled for service*

Categorization types and binary categorization fact types cannot be expressed in OWL DL, since they are second order concepts. In OWL Full, they could be expressed in the form of second order classes and user-defined higher order properties, but they are not supported by a built-in categorization property:

```
Class(pp:rental_car_status_category
annotation(i1ce:vterm("rental
   car status category")))
ObjectProperty(pp:categorization_type domain(pp:rental_car)
   range(pp:rental_car_status_category)
   annotation(i1ce:vp("is categorized by"))
   annotation(i1ce:rgrolename("categorization_type")))
```

6.6. Equality and Inequality Facts

Equality (and inequality) facts, stating that two or more names refer to the same individual (or to different individuals), as expressed in OWL by means of "sameAs" (and "differentFrom"), can be verbalized with the help of the key words "is the same as" (and "is different from"). For instance,

James Bond is the same as 007

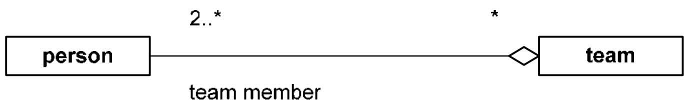
is the verbalization of the OWL equality fact

```
Individual(pp:James_Bond sameAs(pp:007))
```

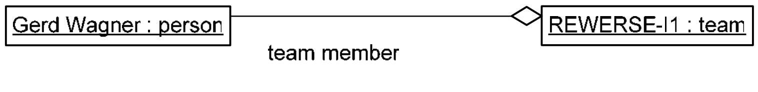
...

6.7. Aggregation Facts

Aggregation is a special binary relationship that is asymmetric and transitive and carries the intended meaning of a part-whole relation. In UML it is visualized with the help of a diamond shape at the range class association end which represents the role of aggregate:



OWL does not support this important ontological concept, so in OWL one has to express it with the help of a user-defined object property. An example of a corresponding aggregation fact (or link) is



Aggregation links and associations can be verbalized in the same style as links and associations, except that one can always employ the special (implicit) verb phrase predicate symbol *is part of*.

6.8. Multiplicity constraints

Multiplicity constraints are an important class of integrity rules. As has been proposed in [ORM], they can be verbalized with the help of various quantifier key words:

– universal quantification (*each*),

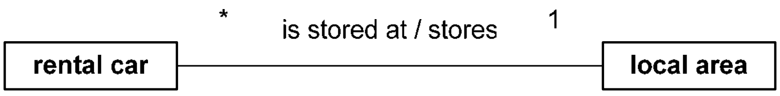
– existential quantification (*some*, *at least one*),
– at-most-one quantification (*at most one*)
– at-least-n quantification (*at least* n),
– at-most-n quantification (*at most* n),
– exactly-one quantification (*exactly one*)

For each association, we have to specify, if it is

1. *total* or not, using the symbols +T or –T
2. *functional* or not, using the symbols +F or –F
3. *inverse total* or not, using the symbols +IT or –IT
4. *inverse functional* or not, using the symbols +IF or –IF

6.8.1. Totality and functionality constraints

An example of an association with a totality constraint is



which is verbalized by the fact type

rental car is stored at **local area**

or inversely as

local area stores **rental car**

This association is both total and functional. The totality constraint can be verbalized as

(+T) *Each* **rental car** is stored at *at least one* **local area**

The functionality constraint is verbalized as

(+F) *Each* **rental car** is stored at *at most one* **local area**

Both constraints, totality and functionality, can be combined in one statement:

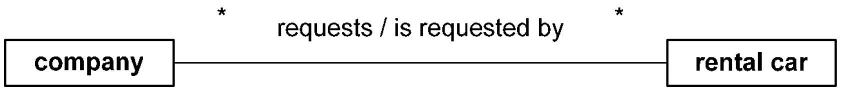
(+TF) *Each* **rental car** is stored at *exactly one* **local area**

According to SBVR, this rule is not a *business rule* but a *structural rule*, so it should be verbalized as a necessity statement:

(+TF) *It is necessary that each* **rental car** is stored at *exactly one* **local area**

6.8.2. Neither total nor functional

The association



which is verbalized by the fact type

company requests **rental car**

or inversely as

rental car is requested by **company**

is neither total nor functional, which is expressed by the following clarification statement:

(-T) *It is possible that some* **company** requests *no* **rental car**
(-F) *It is possible that the same* **company** requests *more than one* **rental car**

This association is also neither inverse total nor inverse functional, which may be verbalized as:

(-IT) *It is possible that some* **rental car** is requested by *no* **company**
(-IF) *It is possible that the same* **rental car** is requested by *more than one* **company**

6.9. Some Rule Examples

The following examples are based on the case study of a fictitious car rental company called EU-Rent, which is described in the appendix. We first give I1CE representation of a rule, then express it using OCL and finally show how it is expressed in ACE.

6.9.1. Defining a derived categorization fact type by means of attribution fact types

First version:

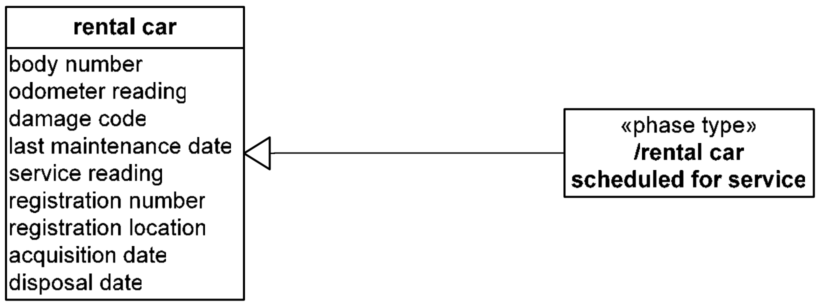
IF **rental car** *has* **service reading** *AND* **service reading** *is greater than* *5000 km* *OR* **rental car** *has* **last maintenance date** *AND* **last maintenance date** *is more than* *3 months* *ago* *THEN* **rental car** is a *rental car scheduled for service*.

Second version, using *that* as an anaphoric reference:

IF **rental car** *has a* **service reading** *that is greater than* 5000 km *OR* **rental car** *has a* **last maintenance date** *that is more than* 3 months *ago THEN* **rental car** is a *rental car scheduled for service*.

Third version, exploiting the functionality of the attributes *service reading* and *last maintenance date*:

IF the *service reading* of ***rental car*** is greater than 5000 km OR the *last maintenance date* of ***rental car*** is more than 3 months ago THEN ***rental car*** is a *rental car scheduled for service*.



OCL:

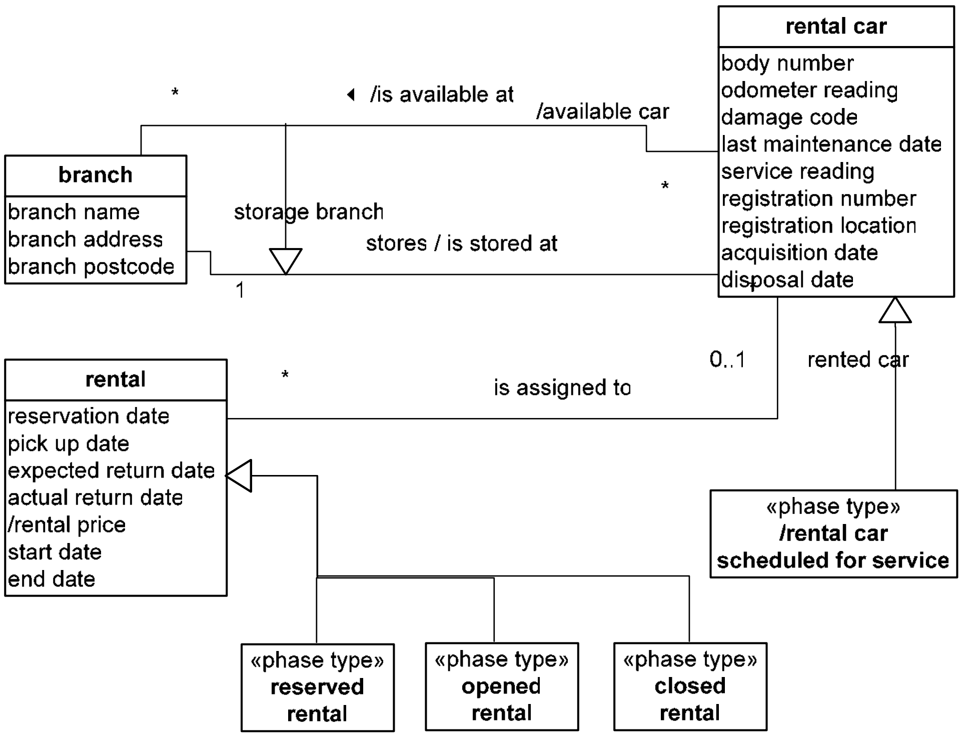
```
context RentalCar::isScheduledForService:Boolean derive:
if lastMaintainanceDate>=3 or serviceReading>=5000
then true
else false
endif
```

ACE:

```
If the last-maintenance-date of a rental-car is at least 3
months old or the service-reading of the rental-car is at
least 5000 km then the rental-car is scheduled-for a service.
```

6.9.2. Defining a derived association fact type by means of other association fact types and categorization fact types

IF ***rental car*** <u>*is stored at*</u> ***branch*** AND NOT ***rental car*** <u>is a *rental car scheduled for service*</u> AND ***rental car*** <u>*is*</u> NOT <u>*assigned to*</u> *a* ***rental*** THEN ***rental car*** <u>*is available at*</u> *the* ***branch****.*



OCL:

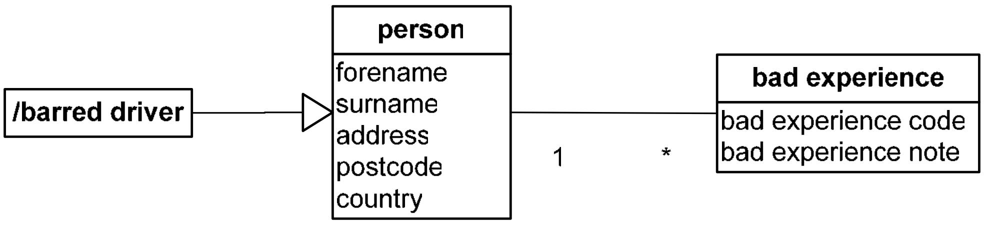
```
context Branch::availableCar: RentalCar derive:
self.storedCar->select( c |
   not oclIsKindOf(RentalCarScheduledForService)
   and c.Rental->isEmpty())
```

ACE:

```
If a rental-car is stored-at a branch and the rental-car is not
assigned-to a rental and the rental-car is not scheduled-for a
service then the rental-car is available.
```

6.9.3. Defining a derived categorization fact type by means of association fact types

IF a ***person*** has *more than 3* ***bad experience****s THEN the* ***person*** *is a* barred driver**.**



OCL:

```
context Person::BarredDriver:Boolean derive:
if self.badExperience->size()>3
then true
else false
endif
```

ACE:

```
If a driver has more than 3 bad-experience-records then the
driver is barred.
```

7. Conclusion

We have shown that an approach based on verbalization patterns for all important cases of fact types can lead to acceptable verbalization results in many practically relevant cases. However, it lacks a systematic method of applying anaphoric references for simplifying the generated verbalizations. Such a method is provided by the Attempto approach. It is therefore an issue for future research to develop a method for combining fact type verbalization patterns with the anaphoric reference mechanism of Attempto.

References

[GW05b] G. Guizzardi & G. Wagner. A Unified Foundational Ontology and some Applications of it in Business Modeling. In P. Green and M. Rosemann (Eds.), Business Systems Analysis with Ontologies, IDEA Publishing, 2005.

[GW05a] G. Guizzardi & G. Wagner. Towards Ontological Foundations for Agent Modelling Concepts Using the Unified Foundational Ontology. In P. Bresciani et al. (Eds.): AOIS 2004, LNAI 3508, pp. 110 – 124, Springer-Verlag, 2005.

[Metalog] The Metalog Project, http://www.w3.org/RDF/Metalog/. W3C, 1998-2004.

[NFRA] Reasoning in Attempto Controlled English. Norbert E. Fuchs, Uta Schwertel. Institute für Informatik, Universität Zürich, {fuchs, uschwert}@ifi.unizh.ch, http://www.ifi.unizh.ch/attempto.

[ORM] T.A. Halpin, Information Modeling and Relational Databases, Morgan Kaufmann Publishers, 2001

[OWL] OWL Web Ontology Language, http://www.w3.org/2004/OWL.

[RDFCAS] G. Klyne and J.J. Caroll (Eds.), Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C, 2004.

[SBVR] Semantic of Business Vocabulary and Business Rules (SBVR). Revised Submission to BEI RFP br/2003/06/03, http://www.omg.org/cgi-bin/doc?bei/2005-03-01.

[UML] UML, http://www.uml.org/.

8. Appendix

8.1. Metalog representation of the example in section 6.9.1

The vocabulary for this rule:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
xmlns:rdfs=”http://www.w3.org/2000/01/rdf-schema#”>
<rdfs:Class rdf:ID=”RentalCar”/>
<rdfs:Class rdf:ID=”RentalCarScheduledForService”>
  <rdfs:subClassOf rdf:resource=”#RentalCar”/>
</rdfs:Class>
<rdf:Property rdf:ID=”lastMaintenanceDate”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”http://www.w3.org/1999/02/22-rdf-syntax-
 ns#Literal ”/>
</rdf:Property>
<rdf:Property rdf:ID=”serviceReading”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”http://www.w3.org/1999/02/22-rdf-syntax-
ns#Literal”/>
</rdf:Property>
...
</rdf:RDF>
```

The XML/RDF syntax of the rule:

```
<implies>
  <rdf:Seq>
    <rdf:li>
      <and>
        <rdf:Alt>
          <rdf:li>
            <Predicate name=”less”>
              <rdf:Seq>
                <rdf:li>
                  <Variable>LAST_MAINTENANCE_DATE</Variable>
                </rdf:li>
                <rdf:li><Constant>3</Constant></rdf:li>
              </rdf:Seq>
            </Predicate>
          </rdf:li>
          <rdf:li>
            <Predicate name=”greater_or_equal”>
              <rdf:Seq>
                <rdf:li>
                  <Variable>SERVICE_READING</Variable>
                </rdf:li>
                <rdf:li><Constant>5000</Constant></rdf:li>
              </rdf:Seq>
            </Predicate>
          </rdf:li>
        </rdf:Alt>
      </and>
    </rdf:li>
    <rdf:li>
      <Predicate name=”RentalCarScheduledForService”>
        <rdf:Seq>
          <rdf:li><Variable>RENTAL_CAR</Variable></rdf:li>
        </rdf:Seq>
      </Predicate>
    </rdf:li>
  </rdf:Seq>
</implies>
```

PNL representation of the rule:
if LAST_MAINTENANCE_DATE is greater than “3” months and SERVICE_READING greater_or_equal than “5000” km then RENTAL_CAR is “RentalCarScheduledForService”.

8.2. Metalog representation of the example in section 6.9.2

Metalog

The vocabulary for this rule:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
xmlns:rdfs=”http://www.w3.org/2000/01/rdf-schema#”>
<rdfs:Class rdf:ID=”RentalCar”/>
<rdfs:Class rdf:ID=”RentalCarScheduledForService”>
  <rdfs:subClassOf rdf:resource=”#RentalCar”/>
</rdfs:Class>
<rdfs:Class rdf:ID=”Branch”/>
<rdfs:Class rdf:ID=”Rental”/>
<rdfs:Class rdf:ID=”ReservedRental”>
  <rdfs:subClassOf rdf:resource=”#Rental”/>
</rdfs:Class>
<rdfs:Class rdf:ID=”OpenedRental”>
  <rdfs:subClassOf rdf:resource=”#Rental”/>
</rdfs:Class>
<rdfs:Class rdf:ID=”ClosedRental”>
  <rdfs:subClassOf rdf:resource=”#Rental”/>
</rdfs:Class>
<rdf:Property rdf:ID=”rentedCar”>
  <rdfs:domain rdf:resource=”#Rental”/>
  <rdfs:range rdf:resource=”#RentalCar”/>
</rdf:Property>
<rdf:Property rdf:ID=”availableCar”>
  <rdfs:domain rdf:resource=”#Branch”/>
  <rdfs:range rdf:resource=”#RentalCar”/>
</rdf:Property>
<rdf:Property rdf:ID=”storageBranch”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”#Branch”/>
</rdf:Property>
<rdf:Property rdf:ID=”isAvailableAt”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”#Branch”/>
</rdf:Property>
<rdf:Property rdf:ID=”stores”>
  <rdfs:domain rdf:resource=”#Branch”/>
  <rdfs:range rdf:resource=”#RentalCar”/>
</rdf:Property>
<rdf:Property rdf:ID=”isStoredAt”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”#Branch”/>
</rdf:Property>
<rdf:Property rdf:ID=”isAssignedToRental”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”#Rental”/>
</rdf:Property>
<rdf:Property rdf:ID=”lastMaintenanceDate”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal”/>
</rdf:Property>
<rdf:Property rdf:ID=”serviceReading”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal”/>
</rdf:Property>
```

```
...
</rdf:RDF>
```

The XML/RDF syntax of the rule:

```
<implies>
  <rdf:Seq>
    <rdf:li>
      <and>
        <rdf:Seq>
          <rdf:li>
            <Predicate name=”isStoredAt”>
              <rdf:Seq>
                <rdf:li><Variable>CAR</Variable></rdf:li>
                <rdf:li><Variable>BRANCH</Variable></rdf:li>
              </rdf:Seq>
            </Predicate>
          </rdf:li>
          <rdf:li>
            <not>
              <rdf:Seq>
                <rdf:li>
                  <Predicate name=”isAssignedToRental”>
                    <rdf:Seq>
                      <rdf:li><Variable>CAR</Variable></rdf:li>
                    </rdf:Seq>
                  </Predicate>
                </rdf:li>
              </rdf:Seq>
            </not>
          </rdf:li>
          <rdf:li>
            <not>
              <rdf:Seq>
                <rdf:li>
                  <Predicate name=”RentalCarScheduledForService”>
                    <rdf:Seq>
                      <rdf:li><Variable>CAR</Variable></rdf:li>
                    </rdf:Seq>
                  </Predicate>
                </rdf:li>
              </rdf:Seq>
            </not>
          </rdf:li>
        </rdf:Seq>
      </and>
    </rdf:li>
    <rdf:li>
      <Predicate name=”isAvailableAt”>
        <rdf:Seq>
          <rdf:li><Variable>CAR</Variable></rdf:li>
          <rdf:li><Variable>BRANCH</Variable></rdf:li>
      </rdf:Seq>
      </Predicate>
    </rdf:li>
  </rdf:Seq>
</implies>
```

PNL representation of the rule:
if CAR “isStoredAt” the BRANCH and CAR not “isAssignedToRental” and CAR not “RentalCarScheduledForService” then CAR “isAvailableAt” the BRANCH.

8.3. Metalog representation of the example in section 6.9.3

Metalog

The vocabulary of this rule:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<rdf:RDF xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
xmlns:rdfs=”http://www.w3.org/2000/01/rdf-schema#”>
```

```
<rdfs:Class rdf:ID=”Person”/>
<rdfs:Class rdf:ID=”BadExperience”/>
<rdfs:Class rdf:ID=”BarredDriver”>
  <rdfs:subClassOf rdf:resource=”#Person”/>
</rdfs:Class>
<rdf:Property rdf:ID=”badExperienceRecords”>
  <rdfs:domain rdf:resource=”#RentalCar”/>
  <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#integer/”>
</rdf:Property>
...
</rdf:RDF>
```

The XML/RDF syntax of the rule:

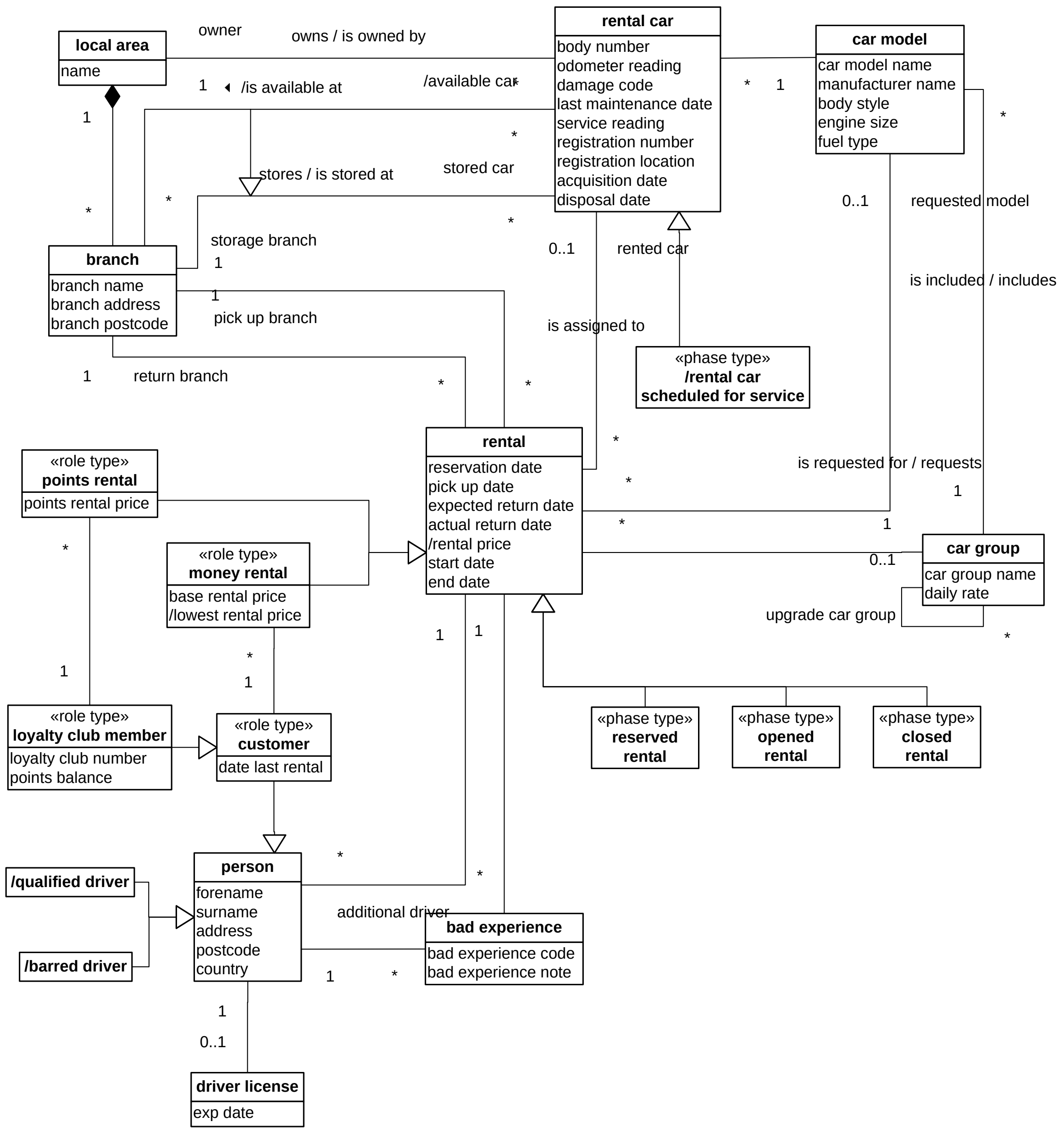
```
<implies>
  <rdf:Seq>
    <rdf:li>
      <Predicate name=”greater”>
        <rdf:Seq>

<rdf:li><Variable>BAD_EXPREIENCE_RECORDS</Variable></rdf:li>
          <rdf:li><Constant>3</Constant></rdf:li>
        </redf:Seq>
      </Predicate>
    </rdf:li>
  </rdf:Seq>
  <rdf:Seq>
    <rdf:li>
      <Predicate name=”BarredDriver”>
        <rdf:Seq>
          <rdf:li><Variable>PERSON</Variable></rdf:li>
        </rdf:Seq>
      </Predicate>
    </rdf:li>
  </rdf:Seq>
</implies>
```

PNL representation of the rule:

if BAD_EXPERIENCE_RECORDS greater than “3” then PERSON is “BarredDriver”.

8.4. EU-Rent domain model



8.4.1. EU-Rent vocabulary definition

Branch

branch

Definition: EU-Rent facility from which **rental car**s are picked up and to which **rental car**s are returned

return branch

Concept type: role

Definition: **branch** in the rental where **rented car** is returned

pick-up branch

Concept type: role

Definition: **branch** in **rental** where **rented car** is picked up

storage branch

Concept type: role

Definition: **branch** where **rental car** is stored

branch is included in **local area**

Constraint: *Each* **branch** is included in *exactly one* **local area**

branch stores **rental car**

Constraint: *Each* **branch** stores *several* **rental car**s

When a rental is booked the branch, from which the rented car is to be picked up must be specified. If a one-way rental is required, the return branch must also be specified - otherwise the rented car must be returned to the pick-up branch.

Rental Car

rental car

Definition: vehicle owned by EU-Rent and rented to its customers

rented car

Concept type: role

Definition: the **rented car** is assigned to *the* **rental**

available car

Concept type: role

Definition: a **rental car** *that* is available at *the* **branch**

rental car scheduled for service

Concept type: role

Definition: a **rental car** *that* has **maintenance date** *more than* *3 months* OR **service reading** *is more than* 5000 km

available car is available at *the* **branch**

Concept type: derived fact type

Definition: *a* **rental car** *that* is stored at *the* **branch** AND NOT assigned to *a* **rental** AND *is* NOT *a* ***rental car scheduled for service***

rental car has **car model**

Concept type: is-property-of fact type

rental car is stored at **storage ranch**

Constraint: *Each* **rental car** is stored at *most one* **branch**.

rented car is assigned to **rental**

Constraint: A rental car can be assigned to several rentals, but rental periods should not overlap

rental car is owned by *the* **local area**

Constraint: *Each* **rental car** is owned by *at most one* **local area**.

A rental car, owned by a EU-Rent branch and rented to customers.

Attribute	Description
body number	Assigned by manufacturer. Together with manufacturer name (in Car Model) provides a unique business identifier for car
odometer reading	Kilometers recorded since acquisition
damage code	Indicates type of damage sustained by car (if damaged)
Service reading	Kilometers recorded since last service maintenance
last maintenance date	Date of last maintenance
registration number	Registration or license number, as issued by vehicle licensing authority
registration location	Location within country (state, canton, city etc) in which car is registered. Not applicable for some countries
acquisition date	Date on which car first became available for rental
disposal date	Date on which car was sold or written off (if no longer owned by EU-Rent)

Person

additional driver

Concept type: role

Definition: *a* **person** that is allowed to drive the **rented car** of a **rental** and not a *customer*

customer

Concept type: role

Definition: a **person** who is contactually responsible for the **rented car** of a **rental**

barred driver

Concept type: derived role

Definition: *a* **person** who has *3 or more* **bad experience** records.

Note: **barred driver** is not allowed to drive a **rental car** of a **rental**

qualified driver

Concept type: derived role

Definition: a **person** with age over 21 and his **driving licence** is valid

customer has **date last rental**

date last rental

Definition: The date of last rental by the customer

loyalty club member

Concept type: role

Definition: a **customer** who has been identified as one who can drive the **rented car** of a **loyalty club rental**

Note: loyalty club member pays only by points and has no other discounts

loyalty club member *has* **loyalty club number**

loyalty club member ***<u>has</u>*** **points balance**

person <u>has</u> **bad experience**

Constraint:	person may have no more than 3 bad experience.

Rental

<u>return branch</u>

Concept type:	role
Definition:	a **branch** in a **rental** where **rented car** is returned

<u>pick-up branch</u>

Concept type:	role
Definition:	a **branch** in a **rental** where **rented car** is picked up

rental

Definition:	fact type that is 'customer agrees to rent from pick-up branch'
Description:	A contract between a **customer** and EU-Rent to rent a **rental car**.
Constraint:	Any change to the pick-up date/time of a rental must specify a date/time later than the (real world) current date/time.
Constraint:	For a rental, after rental pick-up, none of the following may be changed: requested car group, requested car model, pick-up date/time.

<u>reserved</u>

Concept type:	role
Definition:	**rental car** of the **rental** has not been picked up yet

<u>opened</u>

Concept type:	role
Definition:	**customer** of the **rental** has picked up **rental car**

<u>closed</u>

Concept type:	role
Definition:	**customer** of the **rental** has returned **rental car**

rental <u>has</u> **car group**

Concept type:	is-property-of fact type
Note:	The customer may request a change of car group at any time between rental booking and pick-up, but the rental must always have a car group.

rental <u>requests</u> **car model**

Concept type:	is-property-of fact type
Note:	car model is an option that the renter may request at any time between rental booking and pick-up. EU-Rent accepts the request as a preference when allocating cars to rentals, but does not guarantee to provide a car of the requested model.

rental <u>has</u> *<u>pick-up branch</u>*

Concept type:	is-property-of fact type
Note:	The pick-up branch cannot be changed. If the renter wishes to do so, EU-Rent regards it as a cancellation and a new rental.

rental <u>has</u> *<u>return branch</u>*

Concept type: is-property-of fact type

Note: The return branch is usually the same as the pick-up branch, but may be different. It may be changed at any time up to the end of the rental, but the rental must always have a return branch.

rental has **bad experience**

Constraint: the rental may have no more than 3 bad experience records

money rental

Concept type: role

Definition: rental that is paid with money, not loyalty club points

lowest rental price

Concept type: derived attribute

Definition: lowest rental price that is honored for a money rental. The lowest rental price is recalculated by applying discounts to the base rental price at every touch point (reservation, pick-up, return, etc.), provided that the car group is not changed.

base rental price

Definition: the estimated price of rental before any discounts have been applied

points rental

Concept type: role

Definition: the **rental**, which is paid by **loyalty club point**s and assigned to *loyalty club member*

points rental price

Definition: The price of a *points rental* in points

A Rental starts with a rental reservation, which must specify a pick-up Branch, a return Branch and a Car Group. A specific car model may be requested.
A specific car is assigned to the Rental in time for pick-up by the customer.

Attribute	Description
reservation date	
pick up date	
expected return date	
actual return date	
rental price	Derived attribute, stores the actual price of the rental when all discounts have been applied
start date	The start date of the rental, defined during reservation
end date	The end date of the rental, defined during reservation