Moving from Zachman Row 2 to Zachman Row 3

Business Rules from an SBVR and an xUML Perspective

Markus Schacher, 21.05.06

1 Introduction

During the past few years, KnowGravity has been involved in research on the Business Rules Approach (BRA) as well as in the Unified Modeling Language (UML) and the Model Driven Architecture (MDA) of the Object Management Group (OMG). In this context we developed an environment, based on our CASSANDRA platform, for creating executable UML models (also called xUML models). The primary goal of CASSANDRA/xUML is to offer a platform for executable specifications that are as technology-independent as possible. In other words, CASSANDRA/xUML models are typically **Platform Independent Models (PIMs),** or **Zachman Row 3 models**.

In contrast to other approaches to xUML in the OMG, with CASSANDRA/xUML we do not try to define a small subset of UML that may be made executable; rather we try to make the full semantics of the UML executable. With the Business Rules Approach in mind we try to push the boundaries even further to and beyond the limits of UML. During the past two years we have added more rules-oriented features (or rules-oriented interpretations of existing features in UML) to CASSANDRA/xUML. Today, CASSANDRA/xUML supports **derivation rules** as well as **constraints** and **process rules**. Furthermore, we use a variant of UML's **OCL (Object Constraint Language)** that uses a more business-oriented syntax than the original UML2 OCL.

In this document, I will contrast business rules stated in a business-oriented language as described by the OMG specification **"Semantics of Business Vocabulary and Business Rules" (SBVR)** with business rules stated in a more IT-oriented form and expressed in our interpretation of xUML. Furthermore, I will show how another new OMG specification, the **"Business Process Modeling Notation" (BPMN)** can be mapped to xUML. SBVR as well as BPMN (to a certain degree) are languages intended to express **Zachman Row 2 models** and I will show how these models contrast with the **Zachman Row 3 models** of xUML. As an illustrative example, I will use "Micro EU-Rent", a simplified version of the well-known car rental case study "EU-Rent".

The remaining document is structured as follows: section 2 is a general introduction, where I will briefly introduce what business rules are, as seen from the business perspective (Zachman row 2, represented using SBVR), as well as seen from the IT perspective (Zachman row 3, represented using xUML). Based on these perspectives, I will discuss the following topics:

- In section 3, I will discuss how an SBVR business vocabulary relates to a business object model in xUML
- In section 4, I will discuss how derivation rules are represented in SBVR as well as in xUML
- In section 5, I will discuss how process rules are based on BPMN and represented in SBVR as well as in xUML
- In section 6, I will discuss how constraints are represented in SBVR as well as in xUML
- Finally, in section 7, I will discuss some advanced topics such as polymorphic and dynamic rule sets.

2 Business Rules

Business rules may be viewed from the business perspective as well as from an IT perspective.

2.1 Business Rules from the Business Perspective

The Business Rules Group defines a business rule from a business perspective as "guidance that there is an obligation concerning conduct, action, practice, or procedure within a particular activity or sphere". The primary subject of this guidance is the set of business processes performed by humans and potentially (partially)

supported or automated by IT systems. From a business perspective, business rules ought to have a motivation and may be subject to enforcement. SBVR basically distinguishes two different categories of business rules, as follows:

- **Structural Business Rules** specify necessities on business concepts, i.e. they are "true by definition" and thus represent form of definitions or conventions[1]. As an example, a rule that specifies how the total price of a rental is calculated is a structural business rule.
- **Operative Business Rules** specify business obligations, i.e. they exist in order to produce (or avoid) some effect in the business. Operative business rules may be violated and thus are usually enforced by some means. As an example, a rule stating that "blacklisted customers must not be given a rental" is an operative business rule.

Business rules used in this way describe the business, so they are part of row 2 ("Enterprise Model") of the Zachman Framework. According to SBVR, business rules from a business perspective are based on noun concepts (also called object types) and verb concepts (also called fact types) formalized in a vocabulary. Such a vocabulary defines the semantics about these concepts as well as providing community-specific terms for those concepts.

2.2 Business Rules from the IT Perspective

The Business Rules Group defines a business rule from an IT perspective as "a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the business". There exist multiple classification schemes for business rules from the IT perspective. I will use the following simple scheme of three categories:

- **Derivation rules** specify how some information is derived when some other information is given. As an example, derivation rules may define how much of a discount a particular rental gets.
- **Process rules** are rules that affect processes, i.e. they trigger, permit, or prohibit some activities. As an example, a process rule may prohibit a rental from being given to a blacklisted renter.
- **Constraints** are propositions that must always hold. As an example, a constraint might state that the sum of the total price of all open rentals of a renter may never exceed the renter's credit limit.

Business rules used in this way describe IT systems that will support a business, so they are part of row 3 ("Systems Model") of the Zachman Framework. In general, only some of the rules stated at row 2 will also be part of row 3 -- exactly those rules that are subject to automation by an IT system[2]. For those rules that are "transferred" to row 3, the following heuristics can be applied:

- Structural business rules will be turned into derivation rules.
- Operative business rules will either be turned into process rules or into constraints.

From an IT perspective, business rules are usually based on some form of business objects, represented in a Business Object Model (BOM). Such a BOM serves as an intermediary between technical artifacts, such as database tables, classes, Enterprise Java Beans, etc. and the "user compatible" names for them and their properties (mainly attributes and associations) that will be used to express rules.

---

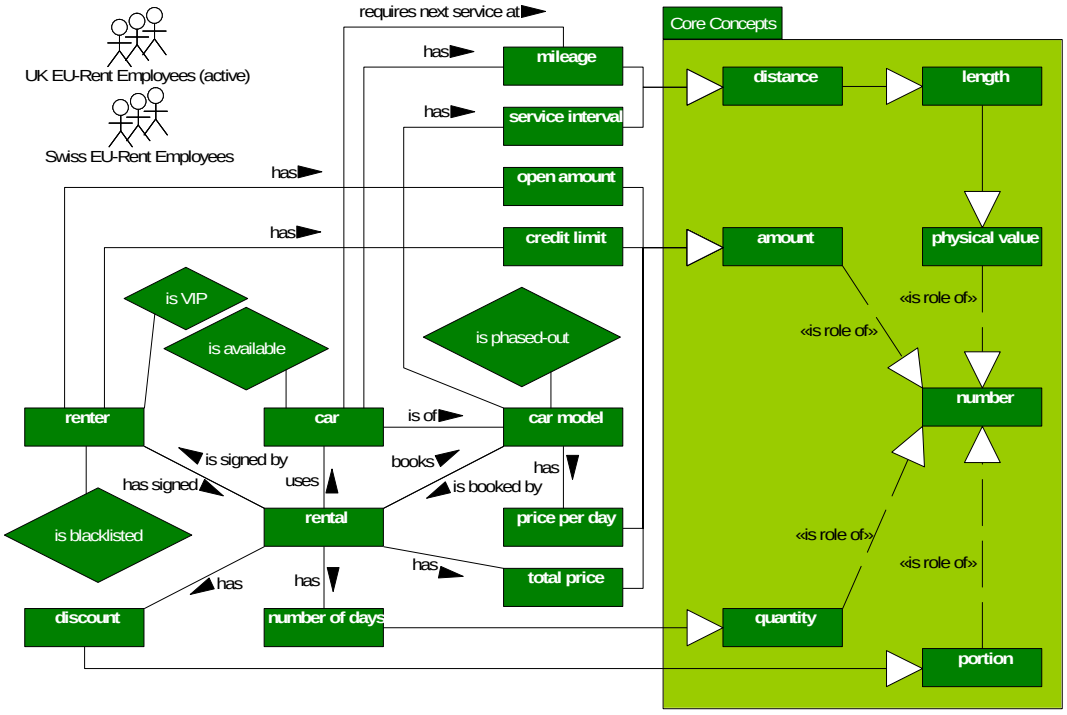
[1] At the time of writing, there is still some discussion among the SBVR authors on the relationship between structural business rules and definitions. This document reflects the personal view of the author.

[2] We restrict here row 3 (and below) of the Zachman Framework to IT systems. However, in general, elements from row 2 that are not implemented by IT systems often need to be mapped to solutions involving people.

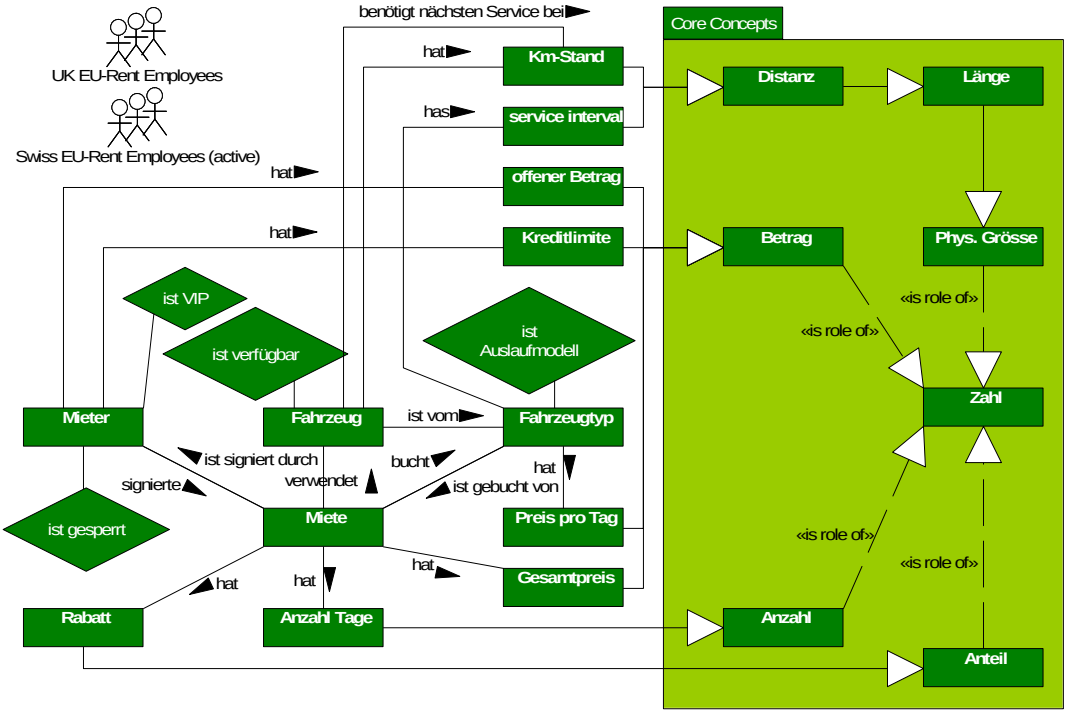
3 Defining a Vocabulary

3.1 Business Perspective in SBVR

In SBVR, a vocabulary is a representation of a so-called "body of shared concepts". The following fact diagram uses SBVR-UML to represent an initial vocabulary for Micro EU-Rent. This vocabulary introduces noun concepts (such as "renter") and verb concepts (such as "is signed by"). It builds on an existing vocabulary named "Core Concepts" that, besides other things, provides highly-reusable concepts that involve numbers. The terms used to name these concepts are those used by the community "UK EU-Rent Employees".



Other communities may use different terms to represent the same concepts. The following diagram shows the same "body of shared concepts" from the perspective of the "Swiss EU-Rent Employees" community:



The exact meaning of the concepts represented by an SBVR vocabulary needs to be defined. Often, additional properties -- such as synonyms, the source of a definition, the owner of a definition, etc. -- are provided as well. Furthermore, the definitions of these concepts are often supplemented by structural business rules, for example:

Rule R1:
It is necessary that each rental *uses* at most one car.
Rule R2:
It is necessary that each rental *is signed by* exactly one renter.
Rule R3:
It is necessary that each rental *books* exactly one car model.

The syntax used to express the rules above is SBVR Structured English (SBVR-SE). The color scheme in these expressions reflects the kinds of vocabulary words used: noun concept, *verb concept* and SBVR-SE keyword.

If the same rules are seen by the community "Swiss EU-Rent Employees", they would look like this (in SBVR Structured German):

Rule R1:
Es ist notwendig, dass jede Miete *verwendet* höchstens ein Fahrzeug.
Rule R2:
Es ist notwendig, dass jede Miete *signiert durch* genau einen Mieter.
Rule R3:
Es ist notwendig, dass jede Miete *bucht* genau einen Fahrzeugtyp.

These few examples of rules clearly illustrate how an SBVR vocabulary serves as a basis for expressing rules. And, if the vocabularies of various communities represent a common set of shared concepts, the very same rules can automatically be represented by expressions using the terms known to the corresponding communities.

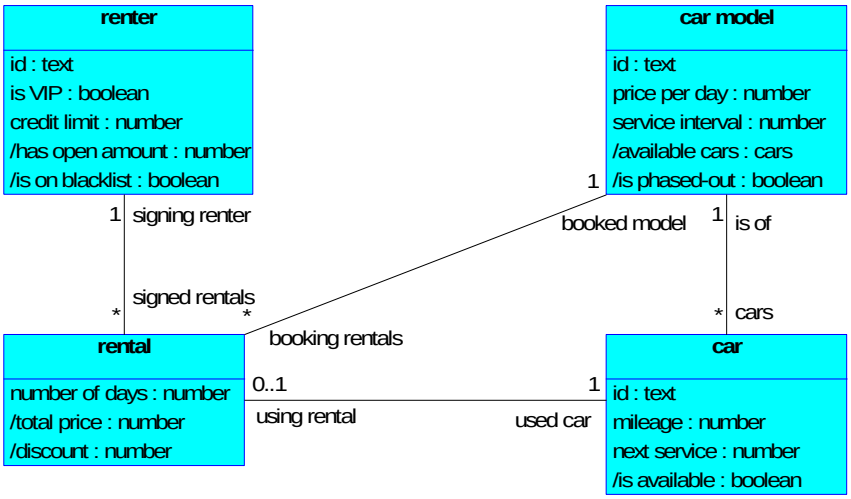
3.2 IT-Perspective in xUML

One of the first things to do when moving from Zachman row 2 to Zachman row 3 is to define the requirements for the required IT system(s). With respect to vocabulary, this primarily means that we need to define which of the concepts from the vocabulary shall be represented in the IT system. For Micro EU-Rent, this could appear as follows:

Id	Prio	Type	Description
Rq1	H	Functional	Information about renters and rentals shall be maintained in the IT system.
Rq2	H	Functional	Information about cars and car models shall be maintained in the IT system.
Rq3	H	Functional	Information about prices shall be maintained in the IT system.

The following xUML class diagram represents an initial Business Object Model (BOM) for Micro EU-Rent. In contrast to the vocabulary above, it shows named attributes[1] as well as associations with role names.

[1] The attributes called "id" are a special feature of CASSANDRA/xUML that provides automatic support for "meaningful" unique identifiers of instances; they have no special relevance here.



Compared to an SBVR vocabulary, an xUML BOM usually also reflects cardinalities on associations. These form a kind of rule: the "1" end represents a constraint on how many instances of one class may be associated with an instance of another class. The three structural business rules R1, R2, and R3 introduced in the business perspective above are now enforced by the cardinality constraints in the xUML BOM.

In contrast to the fact-based approach of SBVR, in xUML associations are always named by role names. Although role names are supported in SBVR as well, fact types are typically named by phrases involving a number of concepts. For example, the fact type "rental *books* car model" in SBVR is represented by the association role 'booked model' in xUML. In SBVR, fact types often have alternative readings, which are represented by other roles in xUML: the association role 'booking rentals' in xUML is a representation of the alternative fact type reading "car model *is booked by* rental" in SBVR. However, in xUML a role is stated in the plural if the corresponding cardinality is "many".

Also shown in the BOM above is something not often used in "classical UML": a number of derived attributes (prefixed by a slash "/"). These attributes will be the logical "hooks" for specifying derivation rules. (More on this later)

The process of transforming an SBVR vocabulary into an xUML BOM can be described by the following set of mapping rules:

1. Noun concepts are transformed into business objects represented as classes.
2. Value types (such as numbers) are transformed into basic types (boolean, number, text, etc.).
3. Unary fact types (also called "characteristics" in SBVR) are transformed into boolean attributes of the owning business objects.
4. Binary fact types are transformed into either:
   - attributes of a basic type, if one of the roles is an object type and the other is a value type, or
   - associations between two business objects, if both roles are object types.
5. Fact types of higher arity are currently not supported by xUML.
6. Object types that need to be referenced by a business user need an attribute "id" in their corresponding business object.

4 Expressing Derivation Rules

4.1 Business Perspective in SBVR

Let's assume that from the business perspective the following structural business rule has been formulated (in SBVR-SE):

Rule R4:

It is necessary that the total price *of* each rental *is* the number of days *of* that rental *x* the price per day *of* the car model that *is booked by* that rental *x* (100 - the discount *of* that rental) / 100.

This structural business rule supports the definition of "total price" and, as a structural business rule, is "true by definition". If we introduce the notion of a "context concept" for a rule (currently not part of SBVR), we could simplify the above structural business rule by specifying its context to be "rental", as follows:

Rule R4:

Context rental: It is necessary that the total price *is* the number of days *x* the price per day *of* the car model that *is booked by* that rental *x* (100 - the discount) / 100.

Furthermore, let's assume that the following decision table representing a set of structural business rules has been stated as well:

Ruleset RS1:

renter *has signed* rental and renter *is VIP*	false			true
rental *has* number of days	< 7 days	7...13 days	>= 14 days	---
rental *has* discount	**0%**	**5%**	**10%**	**10%**

Although these rules are not expressed in SBVR Structured English, they still comply with SBVR as they are based on the noun concepts and verb concepts defined in the vocabulary introduced earlier.

4.2 IT-Perspective in xUML

Again, first we have to define the requirements for the IT system. This time, we not only define functional requirements, but also a non-functional requirement that is typical for business rule systems, as follows:

Id	Prio	Type	Description
Rq4	H	Functional	Prices shall be automatically calculated by the IT system according rule R4.
Rq5	H	Functional	Discounts shall be automatically calculated by the IT system according ruleset RS1.
Rq6	M	Non-functional	The ruleset RS1 for discounts shall be maintainable by business people.

In the general case, the structural business rule about the "total price" stated above needs to be transformed into a set of derivation rules in xUML:[1]

1. 'total price' = 'booked model'.'price per day' * 'number of days' * (100 - discount) / 100;
2. discount = 100 * 'total price' / ('booked model'.'price per day' * 'number of days') - 100;
3. 'booked model'.'price per day' = 'total price' / ('number of days' * (100 - discount) / 100);
4. 'number of days' = 'total price' / ('booked model'.'price per day' * (100 - discount) / 100);

[1] A word in blue represents an element from the xUML model. The quotes are used in CASSANDRA/xUML to delimit symbol names that contain special characters (such as spaces); they have no further relevance here.

The reason for separating a single structural business rule into a number of derivation rules is because CASSANDRA/xUML (as most commercially-available business rule engines) currently does not include a constraint solver. The four derivation rules stated above form a kind of "definition cycle": if the value of more than one of 'total price', discount, 'booked model'.'price per day', and 'number of days' is unknown, we are unable to determine any of those values. Thus, we need to decide which elements are given and which are to be derived. This is the reason why, in xUML, some attributes are declared as derived (i.e. those with a preceding slash in front of the attribute name) and some not (i.e. the attributes that have declared, user-supplied values).

In the Business Object Model introduced earlier, we have already decided that 'total price' and discount will be derived attributes and the others will be declared attributes. The derivation of 'total price' will be specified by a mathematical formula and the derivation of discount will be based on the decision table specified at the business perspective. So, we end up with the following derivation rules (expressed in proper CASSANDRA/xUML syntax):

a) '/total price' := 'booked model'.'price per day' * 'number of days' * (100 - '/discount') / 100;
b) '/discount' := (
    10 if 'number of days' >= 14 or 'signing renter'.'is VIP';
    5 if 'number of days' >= 7 and 'number of days' < 14 and not 'signing renter'.'is VIP';
    0 otherwise);

Note that the equals symbol ("=") in the original derivation rules has been replaced by the assignment symbol (":="). This means that the whole expression on the right-hand-side of ":=" (not the result of its evaluation) is "assigned" to the derived attribute and thus becomes part of its specification. Please also note that the rule set assigned to '/discount' has been optimized to three rules[1] (from four columns in the original decision table) by introducing the "or" operator.

It is important to remember that the rules stated above are formulated in the context of the business object rental. They could also have been formulated in another context (e.g. renter) or in a global context, but they would then need to be expressed in a more complex form.

Now, whenever the '/total price' of a rental is required, derivation rule a) is applied. Since this derivation rule requires the given '/discount', derivation rules b) are applied each time as well. Which one of these actually determines the discount depends on their condition.[2] This process of chaining derivation rules in order to determine a particular outcome is also called "inferencing". Since we started with the outcome (the 'total price') and moved backward finally ending at the declared attributes ('booked model'.'price per day' and 'number of days'), this is a simple form of "**backward inference**" (sometimes also called "**backward chaining**").

[1] Or even two and a half rules, if one counts the "otherwise" clause as a "half rule".

[2] If the condition of more than one rule is true and these conclusions would yield different discounts, we have an inconsistency in our rule set, which would be reported by CASSANDRA/xUML.

5 Expressing Process Rules

5.1 Business Perspective in SBVR and BPMN

Let's assume that from the business perspective the following operative business rules have been formulated (in SBVR-SE):

Rule R5:
It is prohibited that a renter *has signed* a rental if that renter *is blacklisted*.
Rule R6:
It is prohibited that a rental *books* a car model that *is phased-out*.
Rule R7:
It is obligatory that a rental *uses* a car that *is available* and that *has* mileage that *is minimal*.[1]

These operative rules seem to have some effect on business processes; they control the business activities "create rental contract" (rules 1 and 2) and "assign car" (rule 3). More specifically, these rules prevent these business activities from being carried out as long as their necessary preconditions are not met.

In general, operative business rules affect the behavior of humans and/or IT systems, i.e. they affect business processes carried out by those humans and/or IT systems. We may distinguish the following kinds of effects operative business rules may have on business processes:

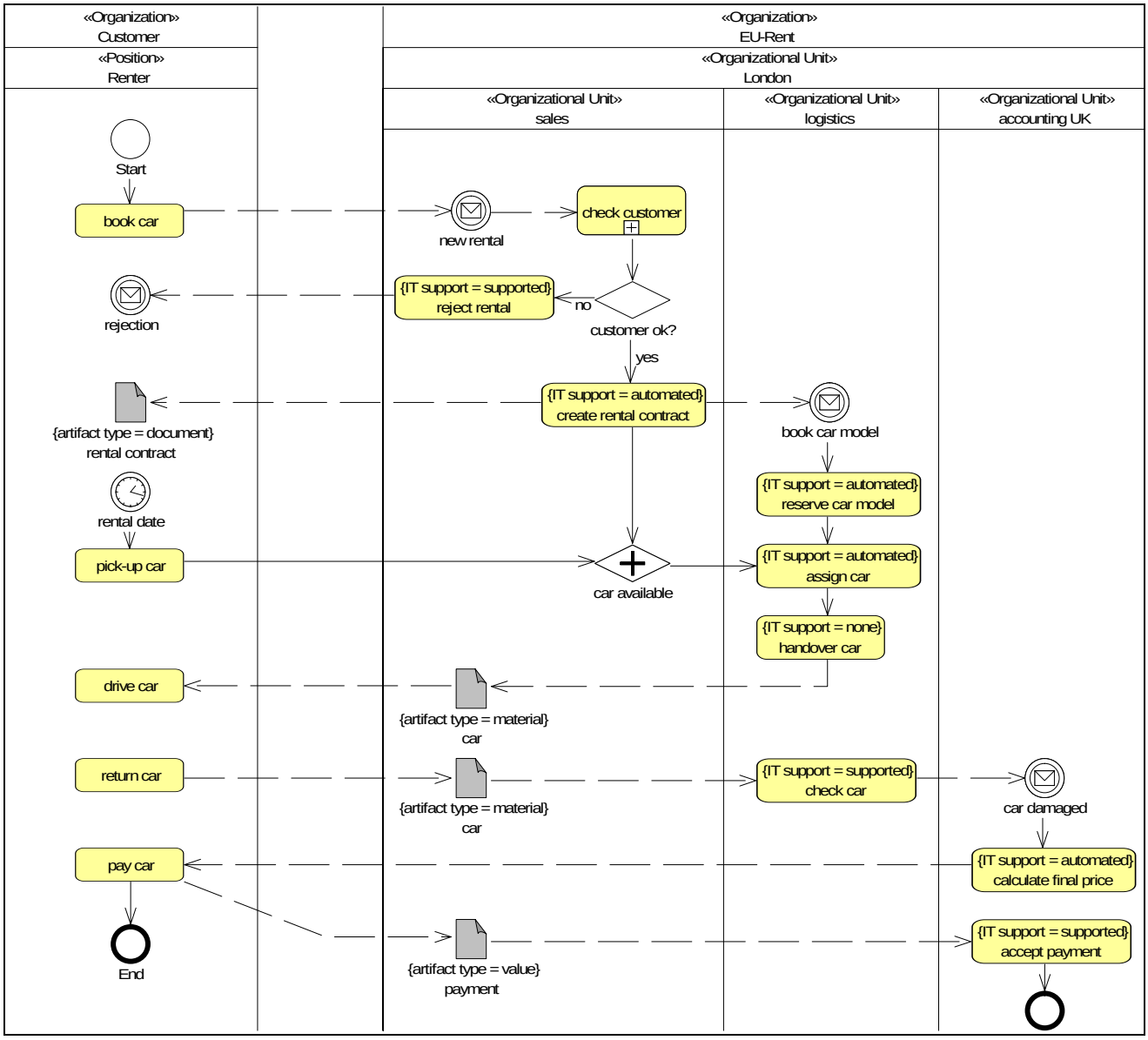
- They may **initiate** business processes (**obligations**)
- They may **allow** business processes (**permissions**)
- They may **disallow** business processes (**prohibitions**)

As soon as we introduce such rules that affect business processes (currently not fully supported by SBVR), we need to extend our vocabulary in order to have the necessary concepts available to state those rules. More specifically, we need to introduce the following process-related concepts into our vocabulary:

- **Business activities** are the primary components of business processes and (hopefully) produce some outcome. Usually, business activities can be further decomposed into other business processes (and thereby described in greater detail).
- **Business events** are occurrence types that initiate business processes. We may further distinguish the following kinds of business events: explicit business events (e.g. "customer appears at front desk"), implicit business events (e.g. "open amount is over threshold"), and time events (e.g. "end of day").
- **Business artifacts** are abstract or real things that are consumed, manipulated, or produced by business activities.
- **Business executives** are agents within a business that are declared to be responsible for the proper execution of some business activities. Such business executives may be organizations, organization units, job positions, or people. As a special but important case, IT systems may be regarded as business executives as well.

Business activities and business events are the "core competency" of business process models. Business artifacts are usually well covered by an SBVR vocabulary, and business executives are represented either in workflow models and/or in organization structure models. After the merger of BPMI.org and OMG in 2005, the Business Process Modeling Notation (BPMN) became an OMG specification. Since BPMN is a notation that supports all of the four key elements described above, I will use BPMN as a complementary vocabulary to SBVR to support the formulation of process rules. The following diagram shows a fragment of EU-Rent's business processes represented in BPMN (Business Process Modeling Notation):

[1] Actually, the real SBVR-SE representation of this rule would be more complex due to the special semantics of the unary verb concept *is minimal*, but we ignore that here.



Note that the business activity "check customer" shows a plus sign "+" which indicates in BPMN that this business activity is non-atomic and decomposed into another business process. Please also note that this diagram introduces some business events such as "new rental" and "book car model" that will later become important in the xUML model as well.

Now, we are able to state some rules based on a combination of an SBVR vocabulary and a business process model in BPMN (the color scheme has been extended to include **business activities**), as follows:

Rule R5a:

It is obligatory to **reject rental** for a renter if that renter *is blacklisted*.

Rule R6a:

It is prohibited to **create rental contract** for a car model that *is phased-out*.

Rule R7a:

It is permitted to **assign car** for a car only if that car *is available* and *has* mileage that *is minimal*.

In those rules, "for a" represents a keyword that introduces concepts from the context of a business activity and that are relevant to other parts of the rule (such as the "if" clause). In this form, obligations may be considered as triggers of business activities, whereas prohibitions, as well as permissions, may be considered as (necessary) preconditions for business activities.

5.2 IT-Perspective in xUML

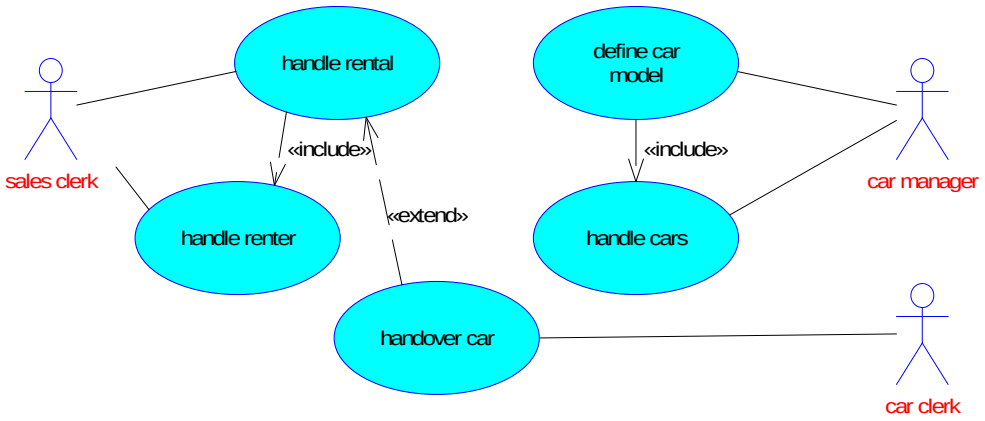
As you may have noticed in the business process diagram above, the business activities have already been classified according to their IT support, as follows:

- **IT support = automated:** This means that this business activity is completely automated by an IT system and needs no human intervention.
- **IT support = supported:** This means that this business activity is supported by an IT system, i.e. some interactive services are provided to the human user.
- **IT support = none:** This means that this business activity is not at all affected by the IT system and must be carried-out manually.

This decision can also be represented in the requirements catalog for the IT system, for example as follows:

Id	Prio	Type	Description
Rq7	H	Functional	Business activities "create rental contract", "assign car", and "calculate final price" shall be fully automated.
Rq8	H	Functional	Business activities "reject rental", "handover car", "check car", and "accept payment" shall be fully automated.

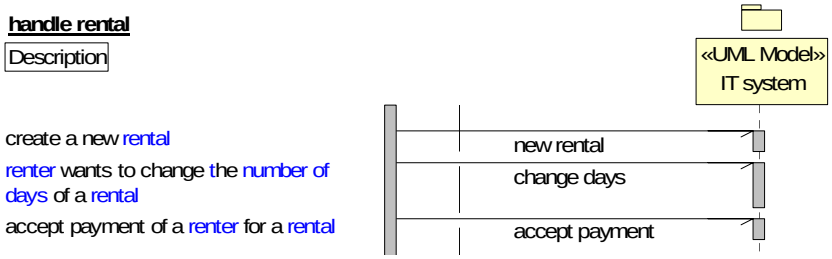
In xUML, events are an important element of the interface between an IT system and its environment. Business events that occur in the business (modeled in Zachman row 2) trigger some business activities that in turn request some functionality (also called "use cases") from an IT system (modeled in Zachman row 3). On the other hand, outcomes from these use cases are communicated by the IT system via events sent to its business environment. On the business side of the boundary between IT system and its business environment we often have people (also called "actors") who are responsible for the interaction with the IT system. The xUML Use Case Diagram below defines which actors will be given access to which use cases:



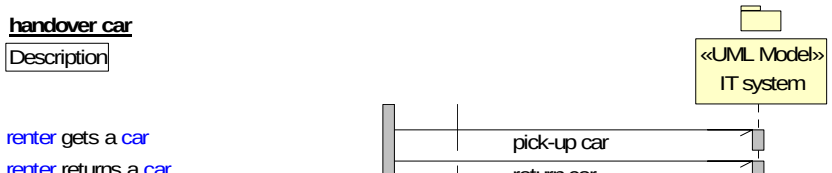
This use case diagram is intended to support the business activities as specified by the requirements. More specifically, we have the following traceability back to the BPMN business process diagram above:

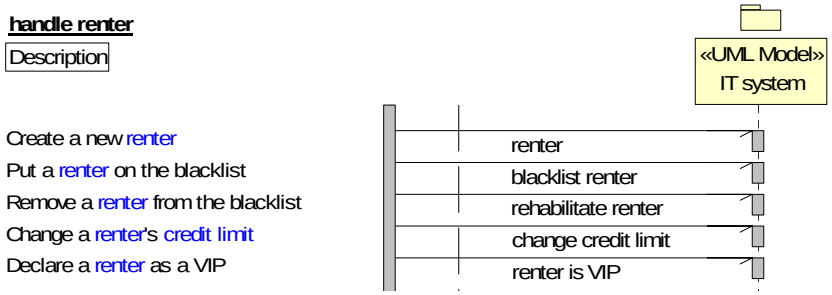
- Use case 'handle rental' supports the business activities "create rental contract", "reject rental", "calculate final price", and "accept payment".
- Use case 'handover car' supports the business activities "assign car" and "check car".
- Use case 'handle renter' supports the business activity "create rental contract".
- Use cases 'define car model' and 'handle cars' support business activities of the car management business process not shown in this document.

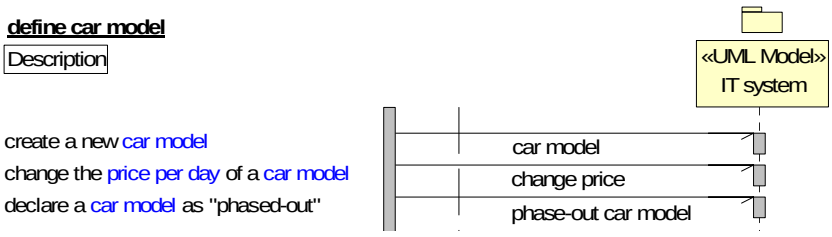
In xUML, the individual use cases are formally specified by so-called "black box sequence diagrams". These diagrams define, for each use case, which events may be raised by the actor. As an example, the diagram below shows the events available to the actor 'sales clerk' when carrying out the use case 'handle rental':

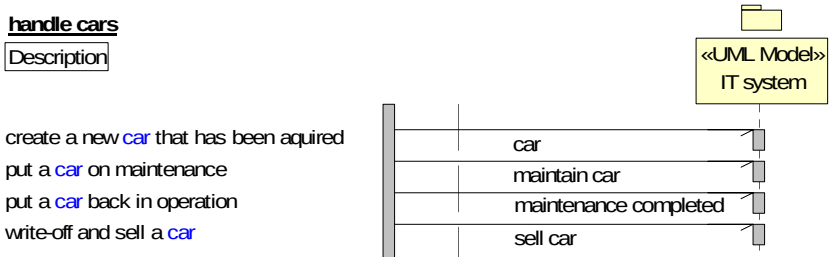


The four black-box diagrams below show the available events when carrying out the use cases 'handover car', 'handle renter', 'define car model', and 'handle cars':



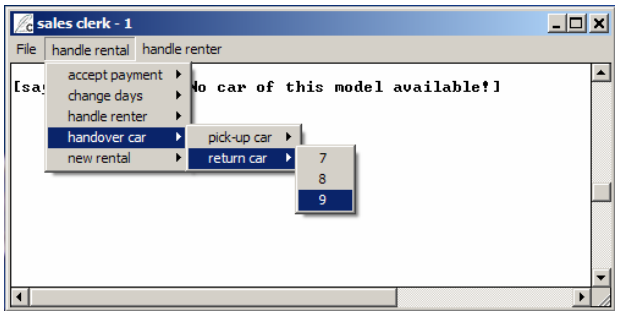






The events renter, 'car model', and car represent the "birth events" of the business objects renter, 'car model', and car, i.e. the events that create new instances of those business objects[1]. For reasons not relevant here, the xUML formalism strongly suggests naming the birth event of a business object with the same name as the business object itself.

From the use case model, CASSANDRA/xUML generates a simple, menu-oriented user interface for each actor. For example, the screen dump below shows the generated user interface for a 'sales clerk':

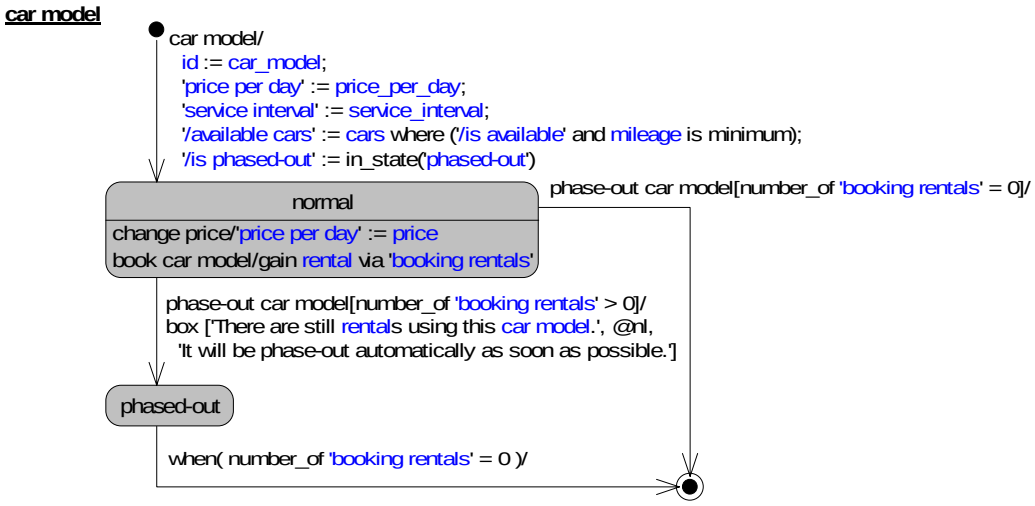


In xUML, the automated parts of business activities are encapsulated in the business objects and the dynamic behavior of business objects is specified by means of state transition diagrams. The most important elements of a state transition diagram are the following:

- A **state** is a named abstraction of a combination of properties of a business object showing different observable behavior from other states.
- A **transition** represents a reaction of a business object, usually triggered by the occurrence of an event and often accompanied by a change of the business object's state.
- An **event** is the named indication of some occurrence that may potentially cause some reaction by or some effect on a business object. Events often carry parameters that transport additional information about the event that has just occurred.
- An **action** is a formally-specified effect of a transition in a business object after the object has been hit by an event.
- A **guard** is a Boolean expression that may prohibit an action from being carried out, even if the corresponding event has occurred.

The example below shows the state transition diagram of the 'car model' business object:

[1] A new "rental" is created by the corresponding event "rental" as well, but this event is not shown in the sequence diagrams above. The reason for this is that a rental is created by a specific "renter" instance (e.g. Jim Smith"), which in turn receives the event "new rental" (shown in the "handle rental" sequence diagram) and propagates the event as "rental" internally.



A state transition diagram is a way to express event-related rules in a semi-graphical form. More specifically, a state transition diagram may represent two kinds of rules, as follows:

- **ECA (Event Condition Action) rules** specify that a particular action is to be carried out when a particular event occurs and a particular condition is true (the business object being in a particular state with, possibly, a guard). The example state transition diagram above includes the following ECA rules:
  - If event 'change price' occurs and the 'car model' is in state normal then set attribute 'price per day' to the value of the event parameter price.
  - If event 'phase-out car model' occurs and the 'car model' is in state normal and there are no 'booking rentals' then delete that 'car model'.
- **CA (Condition Action) rules** specify that a particular action is to be carried out when a particular condition becomes true (the so-called "change condition" indicated by the pseudo-event "when") and another particular condition is true (the business object is in a particular state with, possibly, a guard). The example state transition diagram above includes the following CA rule:
  - If no 'booking rentals' exist for a 'car model' anymore and the 'car model' is in state 'phased-out' then delete that 'car model'.

CA rules as described above are fairly common in business rules technology; they represent "**production rules**" that are processed by "**forward inference engines**" (sometimes also called "**forward chaining engines**").

An interesting question that arises is: "What happens if a business object is hit by an event but the business object is currently in a state in which it doesn't expect that event?" For example in the state transition diagram above, what happens if one of the events 'change price' or 'phase-out car model' occurs but the car model is in the state 'phased-out'? In CASSANDRA/xUML there are several options:

- **Ignore**: The event is simply ignored.
- **Warning**: A warning is shown to the user that this event was not expected in this state.
- **Abort**: An error message is shown to the user and any further processing of this event is aborted.
- **Rollback**: An error message is shown to the user, any changes already caused by this event are undone, and further processing of this event is aborted.

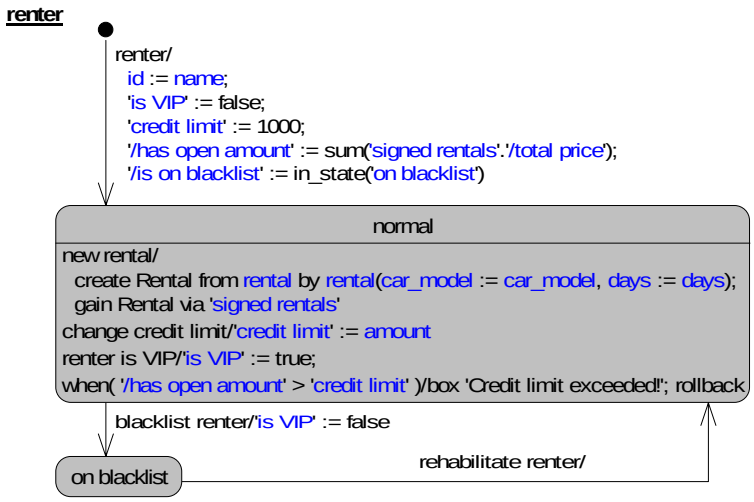
CASSANDRA/xUML provides a form of transaction support that is well suited to modeling business rules on a conceptual level. The runtime system of CASSANDRA/xUML automatically starts a transaction each time when a user raises an external event. During the processing of that event, every business object has a "veto power" to abort or rollback the transaction for business reasons. If no veto is made, the transaction is automatically committed by the runtime system after completing the processing of the user event.

With this in mind, let's see how the three original process rules stated earlier may be represented in the xUML model.

Rule R5a:

It is obligatory to **reject rental** for a renter if that renter *is blacklisted*.

This rule is represented in the state transition diagram of the business object renter. The event 'new rental' is only accepted (and processed) if the renter is in state normal, as stated below:

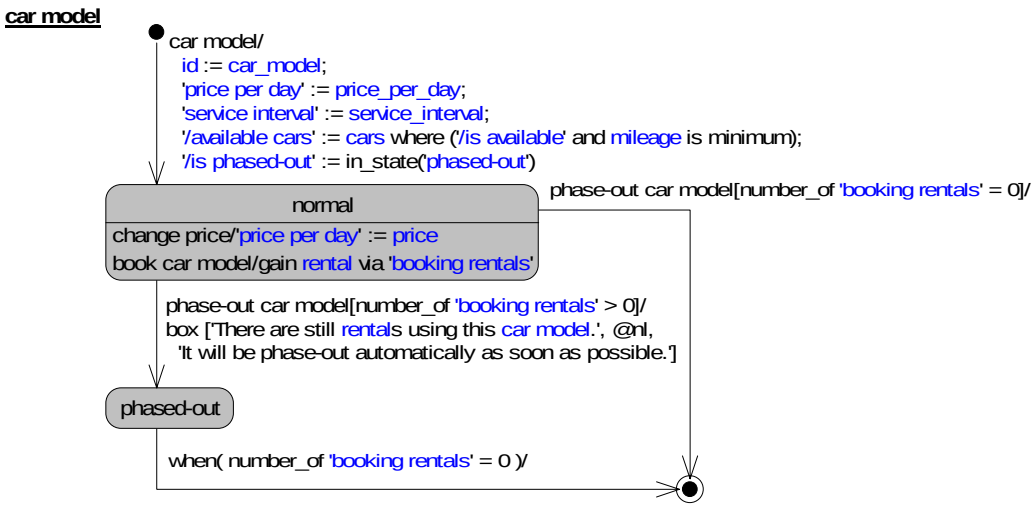


If 'new rental' occurs when the renter is in the state 'on blacklist', the transaction is automatically rolled back. This diagram also illustrates how some derivation rules are assigned to derived attributes (and thus become part of their specification) upon creation of a new renter.

Rule R6a:

It is prohibited to **create rental contract** for a car model that *is phased-out*.

When a new rental is created, this event ('new rental') is propagated via the business object rental to the requested 'car model' as the event 'book car model'.[1] Here, the event is only accepted if the 'car model' is in the state normal (without any further effect); otherwise it is rejected, i.e. the transaction is automatically rolled back:



[1] This could also be considered as the business event sent to the "logistics" organization unit (see BPMN diagram above).

The diagram above reflects another interesting detail. The event 'phase-out car model' is accepted when the 'car model' in the state normal. However, there are two outcomes for this event:

- If there are no rentals that book this car model, it will be deleted immediately.
- If there are still some rentals booking that car model, the car model simply changes its state to 'phased-out'. Then, from that state there is a "when" event that automatically deletes the car model as soon as there are no longer any rentals that book it.

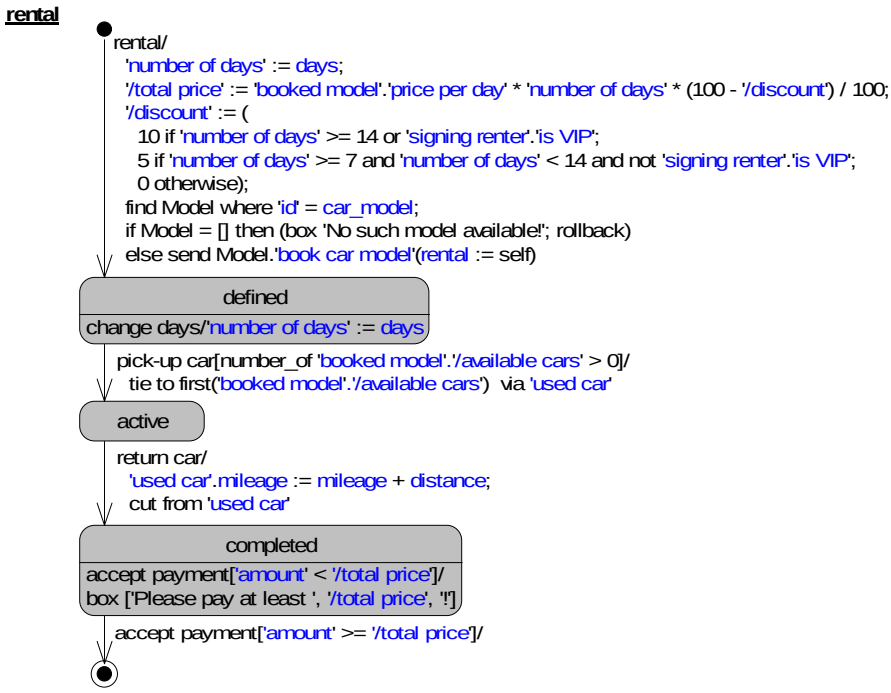
This is a typical behavior pattern that is quite common; it even has a name -- "intended death"[1].

Rule R7a:

It is permitted to **assign car** for a car only if that car *is available* and *has* mileage that *is minimal*.

In the xUML model, this rule has been divided into two parts:

- The condition of the rule is represented in the business object 'car model' (see state diagram above). The derived attribute '/available cars' represents all cars that are available and of these, those that have the least mileage.
- The permission part of the rule is represented by a guard on the transition leaving the state defined of the business object rental (see state diagram below) and leading to its state active: if there are some available cars of the booked 'car model', the first of these is tied to that rental. If no car is available, the event is simply ignored.



The state diagram above also shows some other interesting details. First, using an explicit "rollback" action, a conditional action in the initial transition prevents a rental from being created when a non-existing 'car model' is specified. Remember that the transaction is also rolled back by the business object 'car model' when the specified 'car model' is in the state 'phased-out'. Finally, this state diagram illustrates how some of the derivation rules are assigned to the corresponding derived attributes '/total price' and '/discount'.

[1] This is one of a family of so-called "death models" originally developed in the context of the SSADM methodology.

The mapping rules from the business perspective to the IT-perspective represented as an xUML model can be summarized as follows:

1. Business activities that are to be supported by an IT system will lead to use cases of the IT system.
2. Events become the interface between business objects and their environment (via use cases) and are specified via black box sequence diagrams.
3. Business activities that are to be automated will be specified by state diagrams of business objects.
4. Obligations to be automated will be represented as "when" events in state diagrams of business objects.
5. Prohibitions to be automated will be represented as events that are only accepted in certain states of business objects.
6. Permissions to be automated will be represented as guards on state transitions of business objects.

These are not the only possible mapping rules, but they are a useful starting point.

6 Expressing Constraints

6.1 Business Perspective in SBVR

Let's assume that from the business perspective the following operational business rule has been formulated (in SBVR-SE):

Rule R8:

It is prohibited that a renter *has* an open amount that *is larger than* the credit limit *of* that renter.

This rule is based on the noun concept "open amount", which is itself subject of the following structural business rule (in SBVR-SE):

Rule R9:

It is necessary that the open amount of each renter *is the sum of* the total price *of* each rental that *is signed by* the renter.

If we look a little closer on rule R8 above, we realize that it represents a kind of constraint on the business; it prohibits any activity in the business that may lead to a situation where a renter may have overdrawn his credit limit. This is non-trivial because this may happen in several situations:

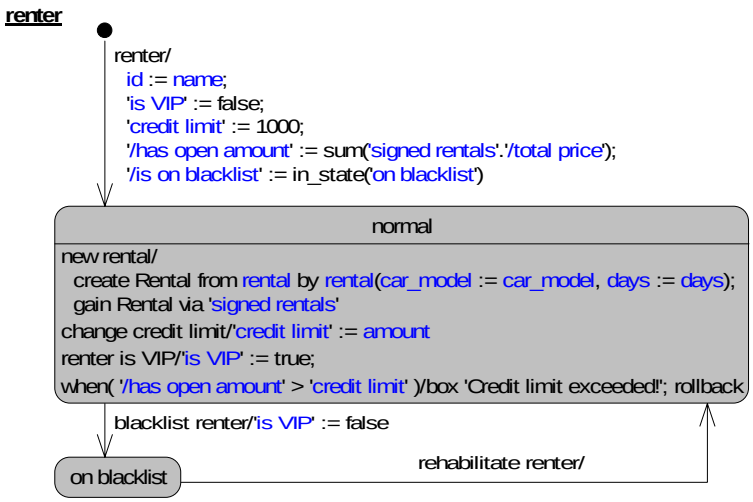
- If a renter wants to sign a new rental, but already has some active rentals close to his credit limit.
- If a renter wants to extend an existing rental, but his open amount is already is close to his credit limit.
- If the credit limit of a renter is decreased below the open amount of the renter.
- If the price per day of a car model rented by the renter is increased and the renter's open amount is already close to his credit limit.[1]
- If the discounting rules change.

However, in the business process, we do not want to care about all these different situations where this rule may be applicable -- we just want to state the rule. This is one of the fundamental reasons why business rules must be separated from business processes.

[1] This may not be the intention of the business, but that is a different issue (not pursued further here).

6.2 IT-Perspective in xUML

Constraints such as the example above may be stated very easily in xUML. In the context of a renter, it simply means that the condition 'open amount' =< 'credit limit' must always be guaranteed. This is achieved by inverting that condition and using it in a "when" event added to state normal of a renter to cause a rollback (or abort), as soon as the inverted condition becomes true, as follows:



Adding a "when" event that represents a constraint to on particular state means that this constraint is only maintained in that particular state. In other words, as specified above, a renter may exceed his credit limit if he is on the blacklist. If this is not intended, the "when" event needs to be moved to a superstate that encompasses the states normal as well as 'on blacklist'.

The state diagram above also shows the representation of rule R9 in xUML. It is a simple assignment of an aggregate expression to the derived attribute '/has open amount'.

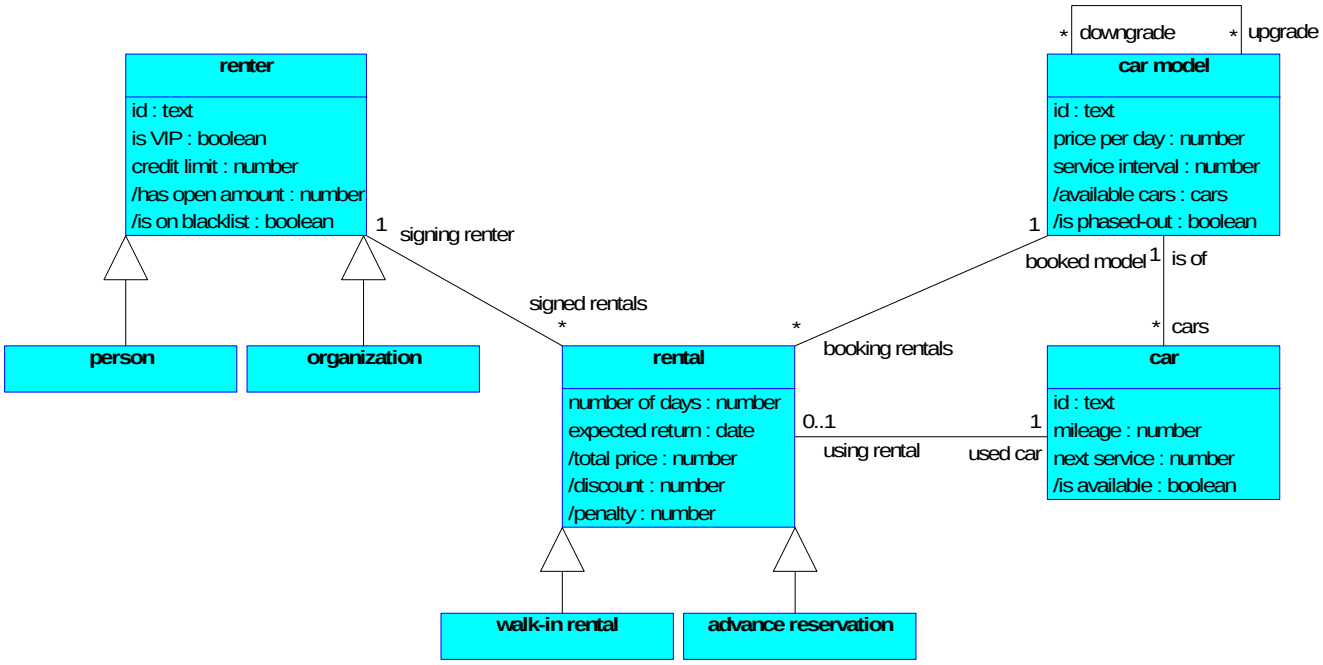
7 Advanced Topics

In this section, we will briefly look at some more advanced concepts of xUML that currently have no direct reflection in SBVR. Specifically, we will briefly talk about the following topics:

- Recursive rules, i.e. derivation rules that are based on derived attributes that are derived according the very same rules.
- Dynamic rule sets, i.e. rule sets that change dynamically depending on the particular state of a business object.
- Polymorphic rule sets, i.e. rule sets that exist in multiple instances of business objects and that may differ from instance to instance.

Finally, we will look at two potential extensions to the current xUML formalism that may lead to new dimensions in rule modeling. First, we look at a formalism to specify temporal rules, i.e. rules that represent constraints over time. Second, we try to make derivation rules even more declarative, thereby greatly increasing their reusability.

To illustrate these concepts, we need a small extension to our original "Micro EU-Rent" case study. The diagram below shows the xUML Business Object Model of the IT system supporting "Mini EU-Rent".



Compared to "Micro EU-Rent", the IT system of "Mini EU-Rent" needs to fulfill the following additional requirements:

- Cars will be rented to organizations as well as individuals. Depending on the type of renter, different rental conditions apply.
- Cars can be rented either in an "ad hoc" manner (called walk-in rental") or reserved in advance. Since advance reservations are preferred by EU-Rent, they get higher discounts.
- The actual rental duration is compared with the requested rental duration. If they are different, i.e. the actual duration is longer or shorter than the requested duration, a penalty fee is applied.
- If no car of the requested car model is available upon pick-up, the renter may get a free upgrade.

7.1 Recursive Rules

In the extended version of the case study we need to deal with upgrades of car models if no car of car model requested by a rental is available. For example, when a rental requested a "VW" and no "VW" is available at pick-up, the renter may get an "Audi" for the same price (or whatever has been declared as a valid upgrade). However, if no "Audi" is available as well, he may even get a "BMW" for the same price, etc.

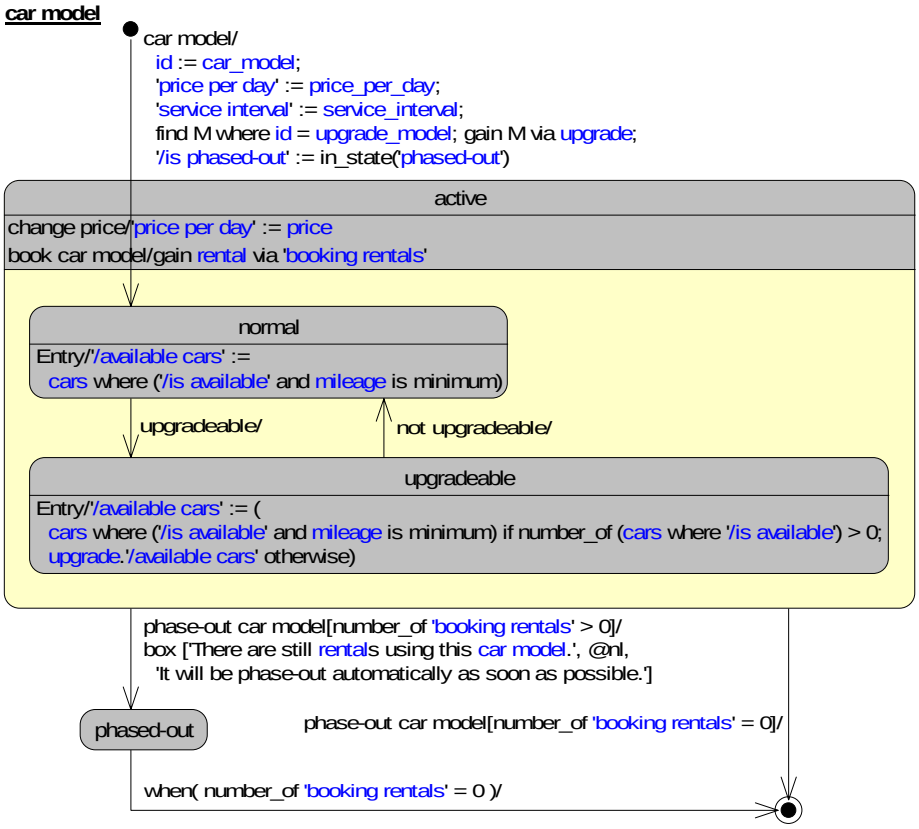
In section 5.2 we introduced the derived attribute '/available cars' that determined cars of a particular 'car model' that are available and that have the least mileage. Now, we've added a recursive association on 'car model' to be able to specify the upgrade options for that 'car model'. This additional association needs to be considered in the derivation rule of '/available cars' as well:

'/available cars' := (cars where ('/is available' and mileage is minimum) if number_of (cars where '/is available') > 0;
upgrade.'/available cars') otherwise);

This derivation rule says that '/available cars' are those cars of a 'car model' that are available and that have the least mileage if there are some available at all. However, if no cars are available the '/available cars' of the 'car model' declared as a possible upgrade are returned: the rules is recursively applied.

7.2 Dynamic Rule Sets

Sometimes it is necessary to apply different rules (or rule sets) to the very same instance of a concept, depending on some condition that involves that concept. Let's assume that a 'car model' is upgradeable only, if it is declared so. So, a 'car model' may either be in state normal or in state upgradeable. The recursive rule introduced in section 7.1 only applies, if a 'car model' is in state upgradeable. If this is not the case, the simpler rule as used in the "Micro EU-Rent" version of the case study applies. In xUML this requirement is simple to represent: upon state entry into the states normal or upgradeable, the different rules are assigned to the derived attribute '/available cars'. Finally, since bookings and price changes of the 'car model' must be allowed in either state, an enclosing superstate active needs to be introduced.

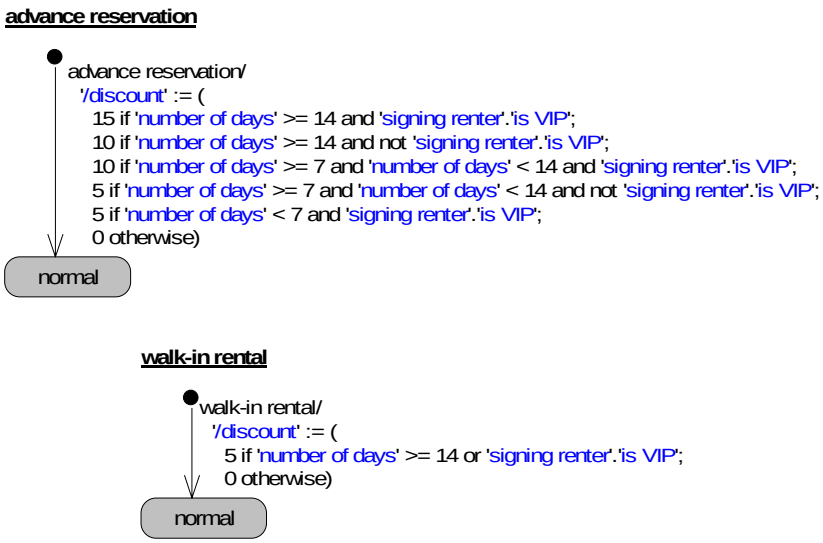


Remember that each object instances carries its own rules or rule sets, which means that we not only have multiple instances of the same class, but multiple instances of the same rules (or rule sets) as well.

7.3 Polymorphic Rule Sets

Sometimes it is required that different rules (or rule sets) are applied to a particular concept depending on some condition. For example, the discount of a rental might be different, if that rental is an advance reservation from the discount if it is a walk-in rental. However, in both cases the discount is based on the same fact type "rental *has* discount" referring to neither "advance rental" nor "walk-in rental". This is a situation that is quite common in object oriented modeling and that is called "polymorphism".

In xUML, this example of polymorphic rule sets is modeled very easy. The more general business object rental introduces the common derived attribute '/discount', whereas the actual rule set assignments are done by the more specific business objects 'advance reservation' and 'walk-in rental'. This is illustrated by the two simple state diagrams below.



Now, we may use the derived attribute '/discount' on business object rental (or the fact type "rental *has* discount") in any place without the need to care about what particular type of rental we have and which actual rules need to be applied.

7.4 Derivation Rules as Constraints

In section 4.2 introduced the following rule for the derived attribute '/total price':

'/total price' := 'booked model'.'price per day' * 'number of days' * (100 - discount) / 100

Whenever we know 'booked model'.'price per day', 'number of days', and discount this derivation rule tells us, how to determine (or calculate) the '/total price'. But on careful consideration, this is actually a constraint that could be stated as follows:

'total price' = 'booked model'.'price per day' * 'number of days' * (100 - discount) / 100

Do you see the two differences? The total price is not a derived attribute anymore and the assignment symbol (:=) has been replaced by an equal symbol (=). Such a constraint could be applied in many more situations than the corresponding derivation rule. It could be used to...

- ... calculate the 'total price' from the other three values (as the derivation rule)
- ... calculate any of the four values from the other three values
- ... verify the a given quadruple of values (e.g. 'total price' = 441, ' booked model'.'price per day' = 49, 'number of days' = 10, and discount = 10,) satisfies that constraint (which is the case in this example)
- ... generate a set of tuples of values that satisfy that constraint when any two of the other values are given
- ... generate a set of triples of values that satisfy that constraint when any one of the other values is given
- ... generate a set of quadruples of values that satisfy that constraint.

In other words: the constraint has a much higher reusability than the derivation! And the very same is true for decision tables as well: they could be simply regarded as complex constraints on values!

A rule engine that is based on constraint solver technology would be able to handle all these situations. Although the core implementation technology of CASSANDRA is a form of a constraint solver (i.e. the programming language Prolog), unfortunately the current implementation of CASSANDRA/xUML (more specifically, her UML Virtual Machine, UVM) is not a constraint solver. However, this is one of the development paths for CASSANDRA/xUML we are currently considering.

