I1-D1

A First-Version Visual Rule Language

Project number:	IST-2004-506779
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-D1/D/PU/b1
Responsible editor(s):	Gerd Wagner
Reviewer(s):	Uwe Aßmann
Contributing participants:	Eindhoven, Skoevde, Heraklion, Librt
Contributing workpackages:	I1
Contractual date of delivery:	31 August 2004

Abstract

Rules are an important means for defining terms in a glossary or vocabulary and for formalizing (parts of) business policies. When using rules in Web applications, it will be essential to be able to visualize and verbalize them for making them accessible to non-technical experts. In this report of the REWERSE working group I1, visual rule languages are surveyed and elements of a visual rule language are proposed.

Keyword List

rules, visualization, Semantic Web

REWERSE Deliverable I1-D1
Report: A First-Version Visual Rule Language

Grigoris Antoniou
Institute of Computer Science
FORTH
P.O. Box 1385, 71110 Heraklion, Greece
antoniou@ics.forth.gr

Mikael Berndtsson
School of Humanities and Informatics
University of Skövde
Box 408, 541 28 Skövde
Sweden
spiff@ida.his.se

Silvie Spreeuwenberg
LibRT
Amsterdam, The Netherlands
silvie@librt.com

Kuldar Taveter
VTT Information Technology
(Technical Research Centre of Finland)
P.O.Box 1201, FIN-02044 VTT, Finland
kuldar.taveter@vtt.fi

Gerd Wagner
Faculty of Technology Management
Eindhoven University of Technology
PO Box 513, 5600 MB Eindhoven,
The Netherlands
G.Wagner@tm.tue.nl

Abstract
Rules are an important means for defining terms in a glossary or vocabulary and for formalizing (parts of) business policies. When using rules in Web applications, it will be essential to be able to visualize and verbalize them for making them accessible to non-technical experts. In this report of the REWERSE working group I1, visual rule languages are surveyed and elements of a visual rule language are proposed.

Acknowledgements
Draft versions of this report have been reviewed by Carlos Viegas Damasio (Univ. Nova de Lisboa, Portugal) and Uwe Assmann (TU Dresden, Germany). The reviewers' comments have greatly helped to improve this report. The remaining errors and weaknesses are the sole responsibility of the authors.

Contents

1 INTRODUCTION

Rules can be visualized in many ways. Different types of rules require different graphical expressions. In addition, there are several purposes of rule visualizations. For instance,

- expressing the syntactical structure of rules
- expressing dependencies between rules and their constituents
- including rules in model diagrams
- supporting testing and debugging of rules

For different purposes, different types of visualizations are appropriate.

Rules are built on *vocabularies*, and vocabularies are built on *names* and *terms*. Therefore, we also discuss the visualization of vocabularies in Section 2.

2 VISUALIZING VOCABULARIES

As an example of a (fragment of a) vocabulary consider the following list of terms and expressions about naturally occurring water resources: stream, brook, river, body of water, lake, ocean; streams have length, rivers empty into bodies of water; Danube, Black Sea.

In general, vocabularies include

1. ***proper names*** denoting individuals, such as *Danube*
2. ***terms*** denoting datatypes, such as *Integer*, or entity types, such as *River* and *Ocean*
3. ***fact type expressions*** denoting property or relationship types, such as *River has Length* or *River empties into Ocean*

There are various formalisms for representing vocabularies: e.g., predicate logic, UML class diagrams, RDF Schema and OWL. While UML and RDF have a graphical notation, neither predicate logic nor OWL come with any visual syntax.

RDF provides a graphical notation for visualizing fact statements (more precisely, conjunctive sentences including terminological sentences involving classes and properties) in the form of directed labeled graphs with two kinds of nodes.

UML class diagrams allow to visualize not only fact statements in the form of links relating two or more entities and/or data values, but also fact type expressions in the form of associations between types/classes.

In standard predicate logic, datatypes, entity types, property types and relationship types are not distinguished. They are all represented equally by means of predicates, and a vocabulary is called a *signature*. Because of this flattening of important meta-conceptual distinctions, standard predicate logic is not very suitable for representing knowledge and expressing business rules.

2.1 UML Class Diagrams

UML class diagrams are based on Entity-Relationship modeling and, thus, have a relational semantics (a general mapping to relational schemas). In Figure 1, an example of a UML class diagram representing a vocabulary including

1. the two fact types *Stream has length* and *River emptiesInto BodyOfWater*
2. the four facts *Danube is of type River*, *BlackSea is of type Ocean*, *Danube emptiesInto BlackSea* and *Danube has length 2858*

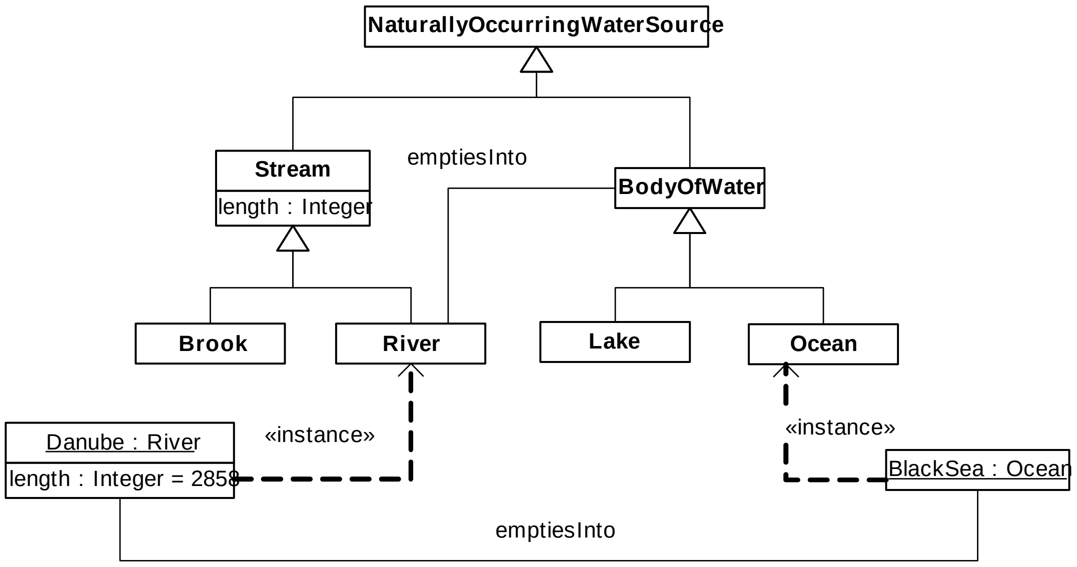


Figure 1: A vocabulary consisting of a datatype (Integer), 7 entity types (Stream, BodyOfWater, etc.), 2 properties (length, emptiesInto) and a proper name (Danube), expressed as a UML class diagram.

2.2 RDF Graphs

The vocabulary described in Figure 1, can also be represented in RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xml:base="http://www.geodesy.org/water/naturally-occurring"
   xmlns="http://www.geodesy.org/water/naturally-occurring"
>
<rdfs:Class rdf:ID="NaturallyOccurringWaterSource"/>
<rdfs:Class rdf:ID="Stream"
       <rdfs:subClassOf rdf:resource="#NaturallyOccurringWaterSource"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Brook"
       <rdfs:subClassOf rdf:resource="#Stream"/>
</rdfs:Class>
<rdfs:Class rdf:ID="River"
       <rdfs:subClassOf rdf:resource="#Stream"/>
</rdfs:Class>
<rdfs:Class rdf:ID="BodyOfWater"
       <rdfs:subClassOf rdf:resource="#NaturallyOccurringWaterSource"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Lake"
       <rdfs:subClassOf rdf:resource="#BodyOfWater"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Ocean"
       <rdfs:subClassOf rdf:resource="#BodyOfWater"/>
</rdfs:Class>
<rdf:Property rdf:ID="length">
       <rdfs:domain rdf:resource="#Stream"/>
       <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
</rdf:Property>
<rdf:Property rdf:ID="emptiesInto">
       <rdfs:domain rdf:resource="#River"/>
       <rdfs:range rdf:resource="BodyOfWater"/>
</rdf:Property>

<Ocean rdf:ID="BlackSea"/>
<River rdf:ID="Danube">
   <length>2858</length>
   <emptiesInto rdf:resource="#BlackSea"/>
</River>
</rdf:RDF>
```

RDF also defines a simple visualization in the form of RDF graphs, as shown in Figure 2.

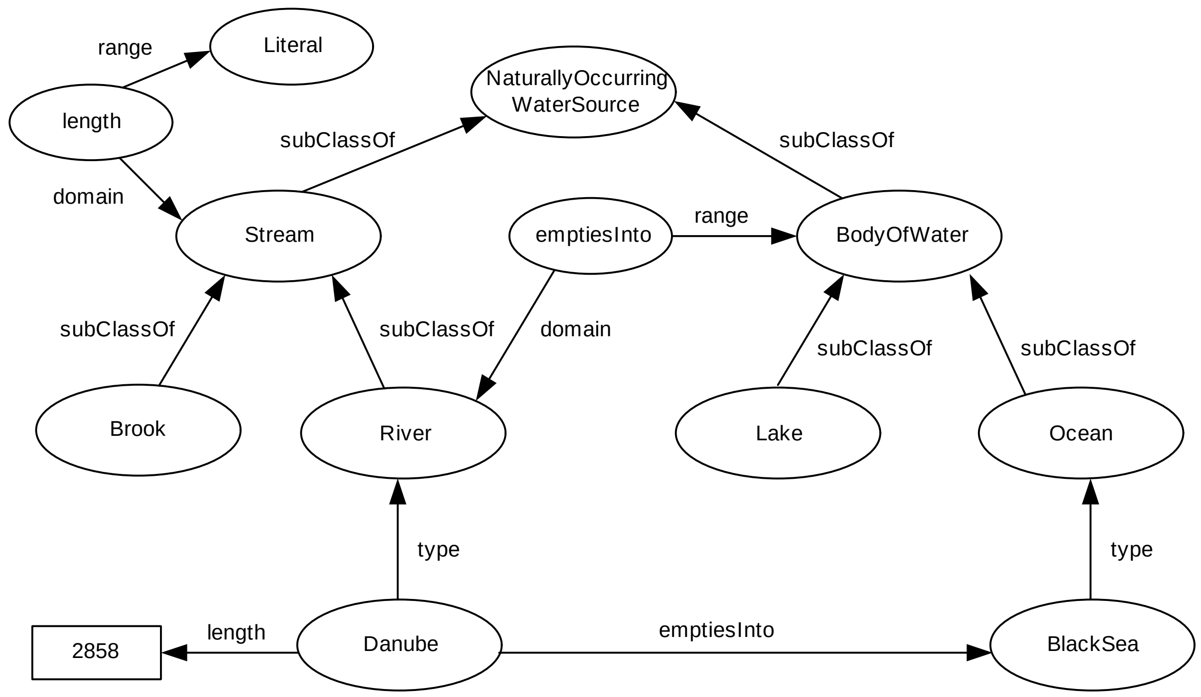


Figure 2: The example vocabulary visualized as an RDF graph.

2.3 Evaluation

The visual language of RDF graphs is very poor: it consists of just 3 language elements (tow kinds of labeled node shapes and one kind of labeled arc arrow). It is much poorer than that of UML class diagrams. In fact, it seems to be too poor for being used to visualize real-world vocabularies.

The language of UML class diagrams, however, seems to be a good basis for visualizing vocabularies. Therefore, it will be important for the REWERSE rule visualization effort to investigate and possibly adopt the ongoing work in the OMG for defining a standard representation of OWL ontologies as UML class diagrams.

3 VISUALIZING DERIVATION RULES

Derivation rules specify how certain logical sentences may be derived from others. They consist of one or more *conditions* and one or more *conclusions*. For specific types of derivation rules, such as definite Horn clauses or normal logic programs, the types of condition and conclusion are specifically restricted.

Previous work on the visualization of derivation rules is based on visualizing the dependency graph of a rule set or logic program in the form of an AND/OR tree. E.g., [DC91, BE91, NKD97] focus mainly on the visualization of proof trees and the control flow by displaying the success or failure of rules and the associated unification process. These works are motivated by the desire to support the debugging, and the execution analysis, of logic programs.

3.1 Cases

3.1.1 Case 1: Rules with Attribute-Value-Formulas

Consider the following rules that may be used by a car rental company to determine if a car must have service:

IF Condition = Damaged
THEN Must have service = no

IF Condition = StartingProblem
THEN Must have service = yes

IF Condition = Good AND Needs maintenance = yes
THEN Must have service = yes

IF Condition = Good AND Needs maintenance = no
THEN Must have service = no

Notice that in these rules the 0-ary predicates *Must have service* and *Needs maintenance* are expressed in the form of yes/no-valued attributes. Normally, in such an attribute-value language, attribute values must not be complex terms and the sharing of variables is not supported.

3.1.2 Case 2: Rules with Predicate-Logic-Formulas and Complex Terms

Consider the following rule that may be used by a car rental company to determine if a car is available for rental:

> *(D1) A car is available for rental if it is physically present, is not assigned to any rental order, is not scheduled for service, and does not require service.*

`D1` can be formalized as a derivation rule in a logic programming-style notation in the following form:

$$\begin{aligned}&\text{RentalCar.isAvailable}(x) \leftarrow\\&\quad \text{RentalCar.isPresent}(x)\\&\quad \wedge\neg\exists \text{y}(\text{isAssignedTo}(x, y))\\&\quad \wedge\neg\text{RentalCar.requiresService}(x)\\&\quad \wedge\neg\text{RentalCar.isScheduledForService}(x)\end{aligned}$$

where the variable x refers to instances of a RentalCar class and the variable y refers to instances of a RentalOrder class.

3.2 Survey

3.2.1 AND-OR Trees

Many Prolog development tools, such as SWI-Prolog or LPA-Prolog (see http://www.swi-prolog.org/ and http://www.lpa.co.uk/ind_pro.htm) support the visualization of the dependency graph [CGT90] of a rule set in the form of AND-OR trees. Also the 'extended rule/goal–graphs' proposed in [HSG03] visualizes rules in this manner. It uses three kinds of node symbols: circles for ordinary predicates, boxes for rules, and rhombuses for meta-predicates.

3.2.2 Class-Diagram-Based Visualization

Derivation rules may be expressed in UML class diagrams as implicational invariants in the Object Constraint Language (OCL). For instance, the above rule D1 could be expressed as

```
context RentalCar inv:
self.isPresent
and self.RentalOrder->isEmpty()
and not self.requiresService
and not self.isScheduledForService
implies self.isAvailable
```

However, OCL provides just a textual-symbolic, but no graphical representation.

In [TW01], it was proposed to include derivation rules in class diagrams using a graphical notation where a rule is visualized as a circle with incoming arrows from the conditions of the rule, and an outgoing arrow to the conclusion. The set of all incoming arrows represents a conjunction of conditions.

Negated conditions are visualized with an arrow that is crossed at its origin. It's an option to use a diamond for expressing a disjunction of conditions.

In order to express the atomic conditions and conclusions of a rule, a graphical notation for logical formulas in class diagrams is needed. The challenge is to identify the practically most important types of logical formulas and the simplest visualization for them. In [TW01], two important types of formulas have been identified:

1. The simplest case of a logical formula in a class diagram that can be used both for expressing conditions and conclusions is formed with the help of *status predicates, i.e.* Boolean attributes representing unary predicates, which apply to certain instances of the context class, thus defining subclasses. Such a formula consists of the name of a Boolean attribute and a variable referring to objects of the context class, such as `RentalCar.isPresent(`*x*`)`. These formulas are visualized by a rectangle named with the status predicate and included in the context class, as shown in Figure 3. A status predicate conclusion arrow implicitly defines a variable ranging over the instances of the context class. A status predicate condition arrow of a unary predicate rule implicitly refers to the variable defined by the unary predicate conclusion.
2. Another important type of simple formula for expressing conditions in a unary predicate rule refers to the existence (or non-existence) of *links,* i.e. the instances of an association. These formulas are called *link existence formulas*. The formula `¬∃y(isAssignedTo(`*x*`,`*y*`))` is an example of this. These formulas are visualized by attaching the origin of the condition arrow to the connection line representing the association concerned, as shown in Figure 3.

Thus, in the class-diagram-based visual language proposed in [TW01], the conclusion of a rule is a status predicate formula, while a condition may be a status predicate formula, a link existence formula, or a negated form of them. These restrictions guarantee that all free variables in conditions occur also in the conclusion.

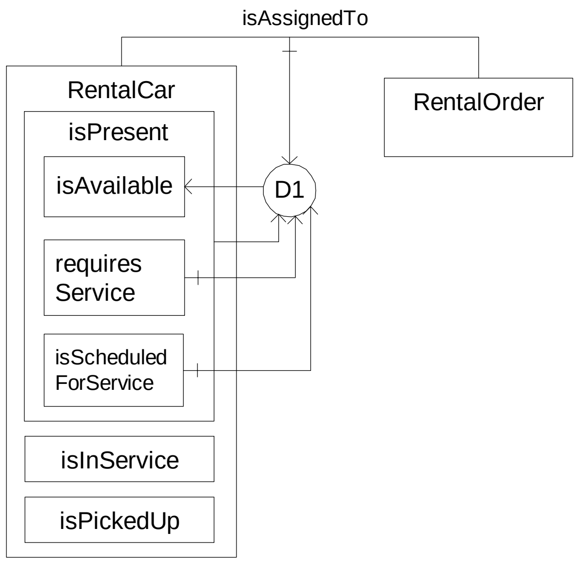


Figure 3: Visualizing the derivation rule D1.

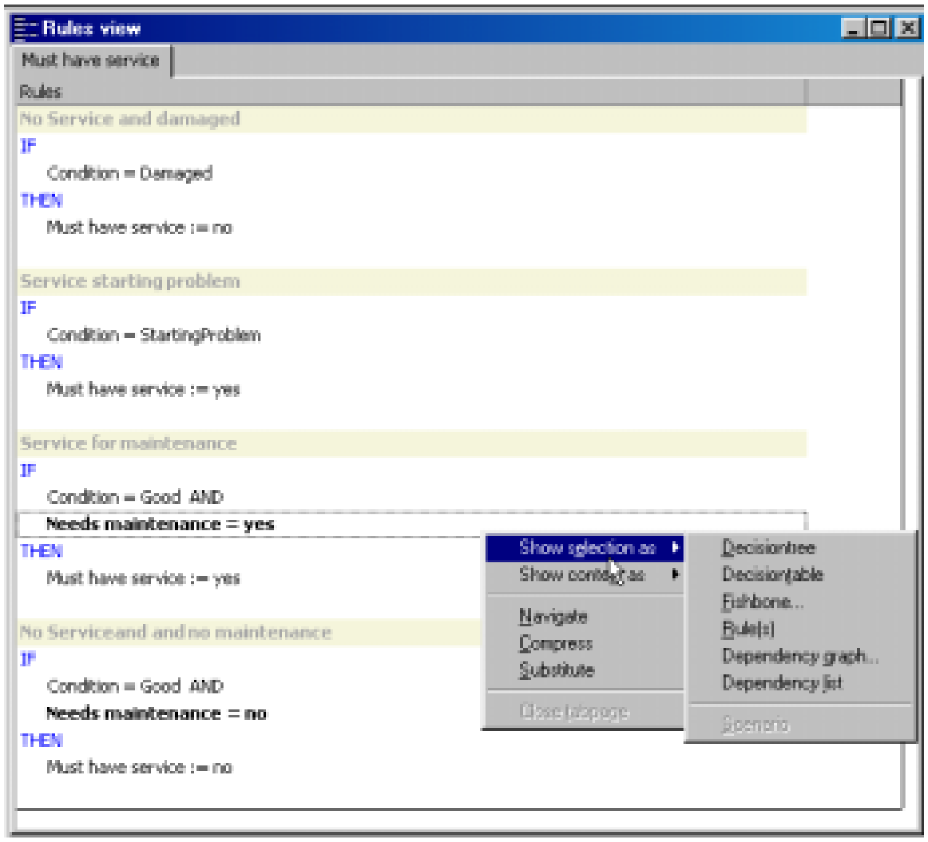
The proposed language allows an elegant visualization of certain types of simple rules. But it should be extended to allow the visualization of more expressive rules.

3.2.3 VALENS

VALENS is a tool that supports the validation process of rules by presenting them in multiple visual representations.

In its textual representation, each condition is represented on a new line between the IF and THEN statements. Conditions are always conjunct. Conditions using a term that is defined by other rules appear in bold.

After the THEN statement one or more attribute value assignments can be presented. There is no ELSE statement in the visualization nor disjunctions between conditions. Both would result in multiple rules.



```
No Service and damaged
IF
    Condition = Damaged
THEN
    Must have service := no

Service starting problem
IF
    Condition = StartingProblem
THEN
    Must have service := yes

Service for maintenance
IF
    Condition = Good AND
    Needs maintenance = yes
THEN
    Must have service := yes

No Serviceand and no maintenance
IF
    Condition = Good AND
    Needs maintenance = no
THEN
    Must have service := no
```

Figure 4: The example rules from case 1 in the textual notation of VALENS

3.2.3.1 Decision Trees in VALENS

Decision trees are used to graphically represent a set of rules containing complex business logic.

This does two things:

- It explains more readily (than by inspection) what the intent of the logic is.
- It eliminates the need to use constructs like "right-hand expression" in explaining how the decision table or decision tree is constructed – it simply lays out the terms and conditions necessary to achieve a given result.

Representation of a set of rules in the form of a decision tree is relevant if:

- Multiple terms and conditions are involved in establishing a given outcome (value of a term or fact).
- Each evaluation term has a small number of values (usually not more than 2-4).

The hierarchical format supports easy visualization of how the determination of appropriate outcome is to be made.

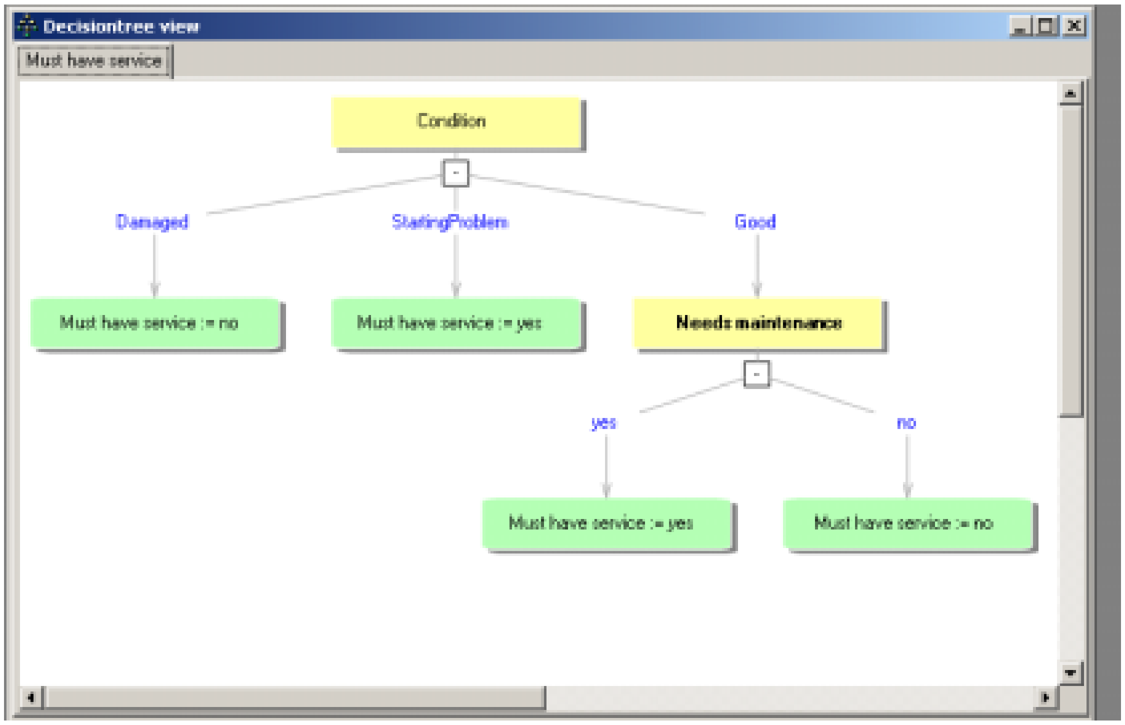


Figure 5: Decision tree in VALENS

Reading the Decision Tree

The graphical format of the decision tree is interpreted as follows:

- A rectangle represents an evaluation term used in the conditions. The ovals or rounded rectangles represent the outcomes.
- The text on the connecting lines between the rectangles represent other terms or values for which the term in the upper rectangle is tested.
- When a term is tested on multiple values they are separated by a comma, this represents an implicit OR (disjunction).
- When a negative test is performed (not equal to) the test value is preceded by an exclamation mark (!)
- The arrows represent conditions that are AND-ed (conjunction of terms).

Each full branch of the decision tree represents a single rule in the consolidated business logic that the decision tree represents. For example, the most-right branch in Figure X reads as follows:

Must have service must be set to no if all of the following are true:

- *the condition is good*
- *needs maintenance is no*

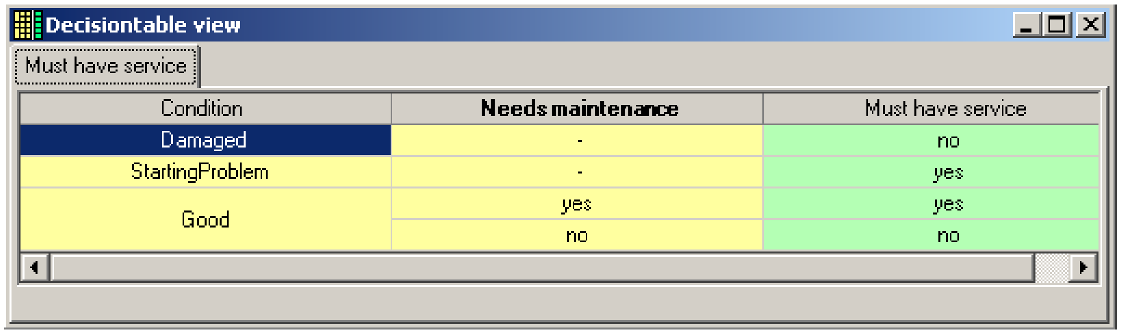
Notes:

- The consolidated business logic represented by the decision tree includes a total of five rules such as the above if all were written out in similar fashion. The decision tree makes the logic much easier to express and manage as a whole.

3.2.3.2 Decision Tables in VALENS

Decision Tables are typically used to represent sets of rules where all of the following conditions are true[1]:

- A significant number of rules are parallel – that is, they share the same subject, have exactly the same evaluation terms(s), and are equivalent (but not identical) in effect. In other words, the rules share a common pattern and purpose.
- Each evaluation term has a finite number of relevant values.
- Given the different values of the evaluation term(s), the outcomes cannot be predicted by a single formula.



Condition	**Needs maintenance**	Must have service
Damaged	-	no
StartingProblem	-	yes
Good	yes	yes
	no	no

Figure 6

Reading the Decision Table

The graphical format of the decision table is interpreted as follows:

- Columns represent the terms used in the rules. The name of each term is presented in the header of the column.
- The first columns (in yellow) represent terms that are used to determine the appropriate outcome, which is given in the last column(s) (green).
- The text in a cell represents some other term, value or condition that is tested for the term given in that column's header. .
- When a term is tested on multiple values, they are separated by a comma. This represents an implicit OR (disjunction).
- When a negative test is performed (not equal to) the test value is preceded by an exclamation mark (!)
- When the text in the next row for a column is the same as for the row above it, the text is not repeated. Rather, the cells are merged.
- Between columns, the conditions are implicitly AND-ed (conjunction of terms).
- Between rows, the rules are implicitly OR-ed (disjunction of terms).

Each full row of the decision tree represents a single rule in the consolidated business logic that the decision table represents. For example, the uppermost row in Figure 6 reads as follows:

- Must have service is no if condition is damaged

3.2.3.3 Fishbone diagram in VALENS

The fishbone diagram represents all rules that share the same right hand side of a rule showing all the conditions that relate to this right hand side.

[1] Excerpted from *Principles of the Business Rules Approach by Ronald G. Ross*, p. 161-162

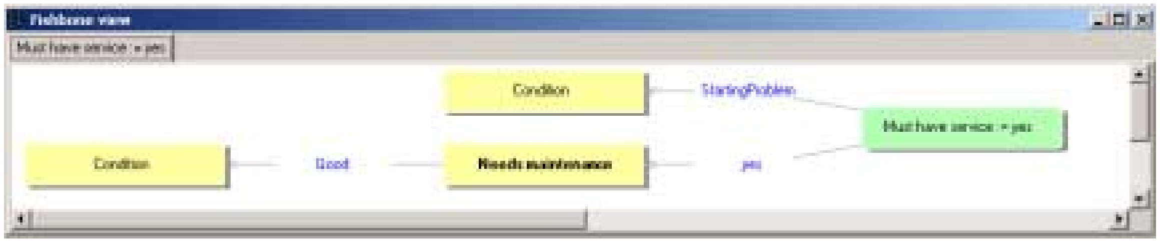


Figure 7

The interpretation of this representation is very similar to the decision tree view. The most lower branch in Figure 7 reads as follows:

If the condition of the car is good and the car needs maintenance than the car must have service.

3.2.3.4 Dependency Graph in VALENS

The dependency graph shows relationships between terms and rules in a graphical way.

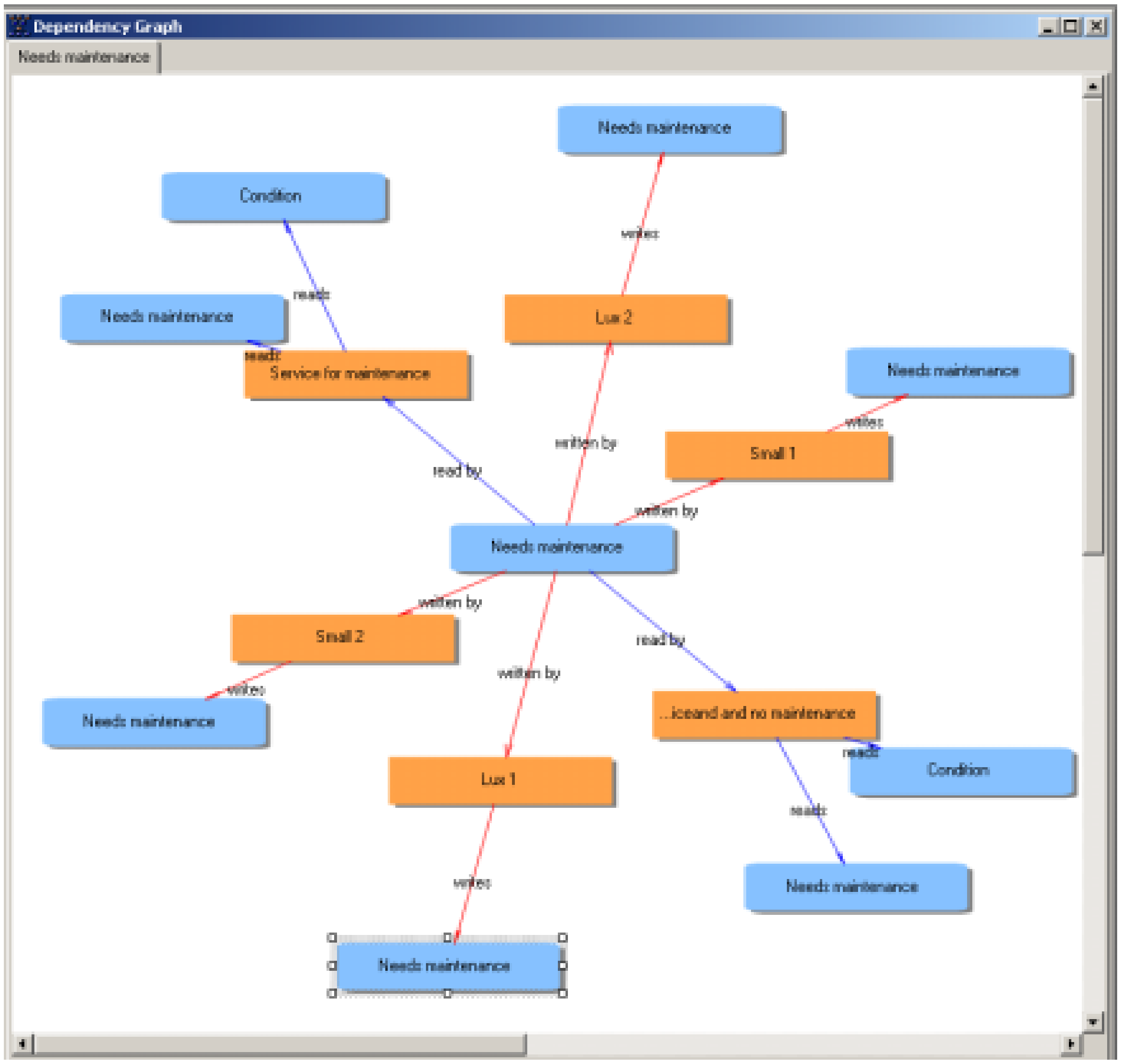


Figure 8

Reading the Dependency Graph

The graphical format of the dependency graph is interpreted as follows:

- The rectangle boxes represent rules.
- The rounded rectangle boxes represent terms.
- The lines between boxes represent a relationship between the term and the rule.
- A red line indicates a computes/computed by relationship between the term and the rule. This means that the term appears as the subject of the rule – that is, that the rule computes, derives or infers the value(s) or instance(s) of this term.
- A blue line indicates a uses/used by relationship between the term and the rule. This means that the term is used in the rule's logic used to compute, derive or infer the subject of the rule.

In many cases the graph can be represented more compactly by combining duplicate rules and terms. This compact table view can be activating by selecting 'compress' from the menu, toolbar or context menu.. The result for the table in Figure 18 appears as follows:

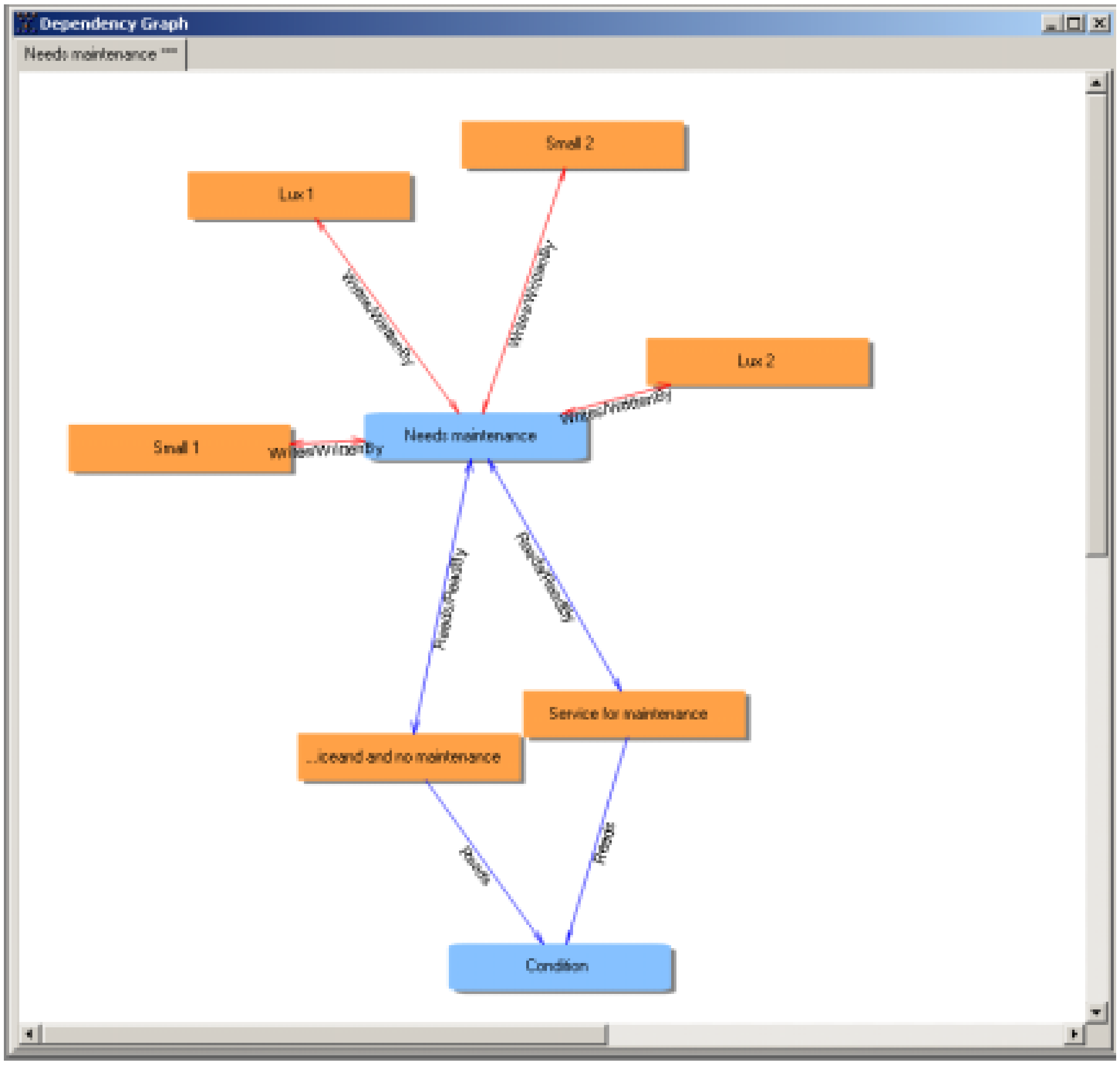


Figure 9

The following sample observations can be made based on the graph above:

- The attribute 'Needs Maintenance' is derived by four rules.
- The attribute 'Needs Maintenance' is used by two other rules.
- The attribute 'Needs Maintenance' is used in combination with the attribute 'Condition'.

This information can enhance understanding of the dependencies between terms and rules. Especially when a particular specification changes, the business rules and campaigns affected by this change can be determined easily.

3.2.3.5 Scenario diagram in VALENS

The scenario demonstrates the path of rule evaluations using the test case values to derive the final outcome (sometimes called reasoning path). One can see the values assigned to intermediate terms and how these are used in subsequent rules.

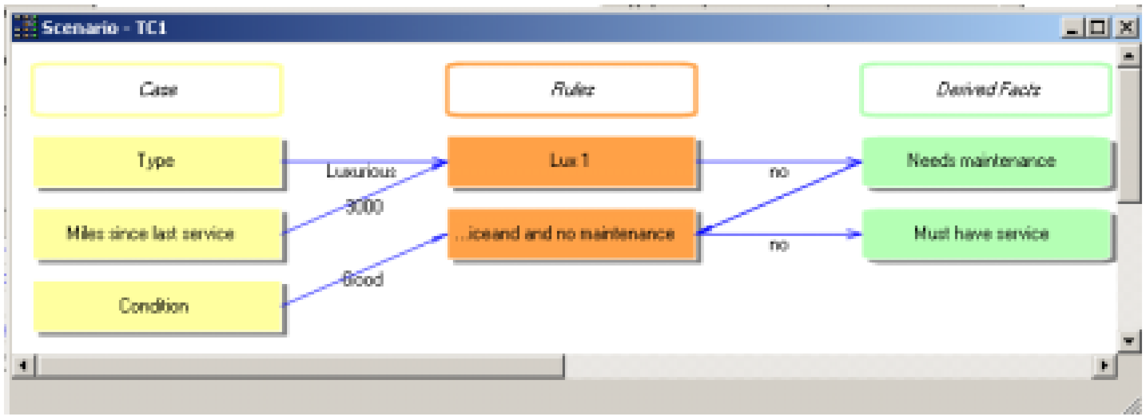


Figure 10

Reading the Scenario Diagram

The graphical format of the scenario graph is interpreted as follows:

- Rules are represented by the orange-colored boxes.
- Terms used in conditions are represented by the yellow-colored boxes.
- The text above the line from yellow or green boxes to orange boxes represent the value of the term that is used to make the condition of the rule (in the orange box) succeed.
- Terms used in actions are represented by the green-colored boxes.
- The text above the line from orange boxes to green boxes represent the value that is assigned to the term (in the green box) that by the rule.

From this diagram it can be concluded that a Car of type Luxurious, in good Condition with 3000 miles since the last service does not need service.

4 VISUALIZING TRANSFORMATION RULES

This section discusses the positional and transformation language Xcerpt [BS02] and the visual query language visXcerpt [BBS03], both for processing XML data.

4.1 Positional Querying

Most widespread query languages for XML, such as XQuery, rely on path expressions. For simple queries and transformations, this *navigational approach* is very natural and results in simple programs. But for more complex queries, especially for queries involving several variables, the navigational approach often leads to intricate programs. A further important aspect of navigational node selections is that they do not easily support the selection of several related nodes at once. Finally, the intertwining of construction and query parts often yields programs that are difficult to visualize properly.

With the *positional* query and transformation language Xcerpt the nodes to be selected are specified by variables in patterns called query terms. Query patterns are related to other patterns called construct terms through their common variables.

4.2 Main Constructs

An Xcerpt program may consist of at least one goal and of some (maybe zero) rules. Goals and rules are built up from database, query and construct terms that are first introduced.

Common to all terms is that they represent tree-like (or graph-like) structures. Different kinds of brackets cater for ordered and unordered term specifications, total and partial term specification. Graph structure is expressed using a reference mechanism.

- Database Terms are used to represent XML documents and the data items of a semistructured database. They are similar to ground functional programming expressions and logical atoms.
- Query Terms are similar to non-ground functional programming expressions and logical atoms. Query terms are unified with database or construct terms using a nonstandard unification called simulation unification.
- Construct Terms serve to reassemble variable (the bindings of which are specified in query terms) so as to construct new database terms.

Both rules and goals have the form *Construct Term* ← *Query Part*, where a construct term is constructed depending on the evaluation of a query part.

In addition to querying external resources, a query may also be evaluated against the program. In such a case, the heads of the program rules are queried and the associated rule is evaluated. Both forward and backward chaining are feasible.

4.3 visXcerpt: A Visual Rendering of Xcerpt

The syntax and semantics of Xcerpt as a whole is well suited as foundation for a visual language. As a consequence, textual Xcerpt's visual counterpart visXcerpt can be conceived as a mere rendering instead of a fully novel language. This rendering might be seen as an advanced layout. visXcerpt aims at easing the use of query technology by novice users, an issue that is particularly important in the Web context, since there are always queries not foreseen by developers.

For visual query languages it is considered important to have a *strong visual relationship between queries and queried data or query results*. A natural approach is to provide some sort of example of a valid result as query as first presented in QBE. Xcerpt query patterns with positional variables can be seen as samples of valid source data items, where some parts are left out and others represented by variables. Construction patterns can be seen as samples or templates of result data items.

4.3.1 Visual Representation of Terms

Xcerpt terms are visualized as boxes. A term label (or tag) is attached as a tab on the top of its associated box. Attributes are placed in a two-column table with names in the left column and values in the right column. The attribute table appears first in a box and is omitted if there are no attributes. Direct subterms (i.e. children) are visualized the same way as sub- or child boxes. Child boxes are arranged vertically in a parent box. For better distinction, they are colored differently. Different box borders are used as visual counterparts to the different Xcerpt parentheses (see Figure 11).

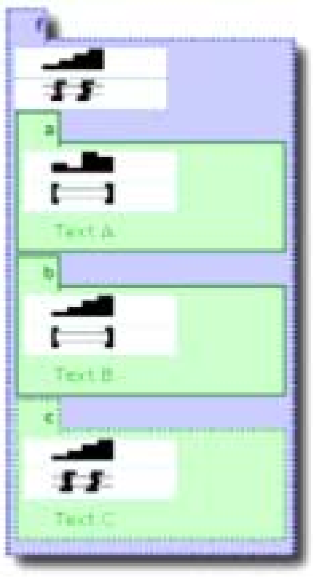


Figure 11: Term representation using different combinations of ordered/unordered and partial/total matching

Beyond the hierarchical structure that terms can express, Xcerpt provides a reference mechanism based on IDs associated to terms. The references are represented with an icon resembling a pointer, and referenced terms carry the anchor name in the title tab (see Figure 12).

Variables are represented as black boxes with the variable name written in white in the box. If a variable is restricted (bound) to a term, this term appears within the box of the variable.

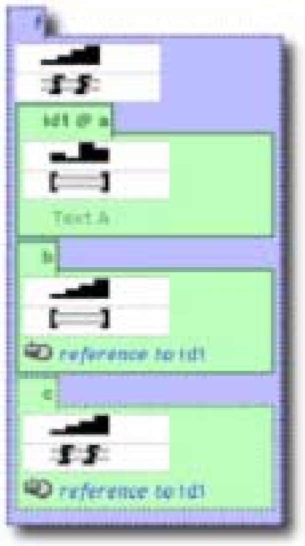


Figure 12: Identifiers appear in the tab, references are visualised by icons

4.3.2 Visual Representation of Rules and Goals

Rules and goals are visualized in visXcerpt by connecting a query part with a construct term by means of an arrow, so as to emphasize the fact that in an Xcerpt rule a result follows from a query. The construct term is positioned left of the arrow while the query part is positioned right of the arrow (see Figure 13).

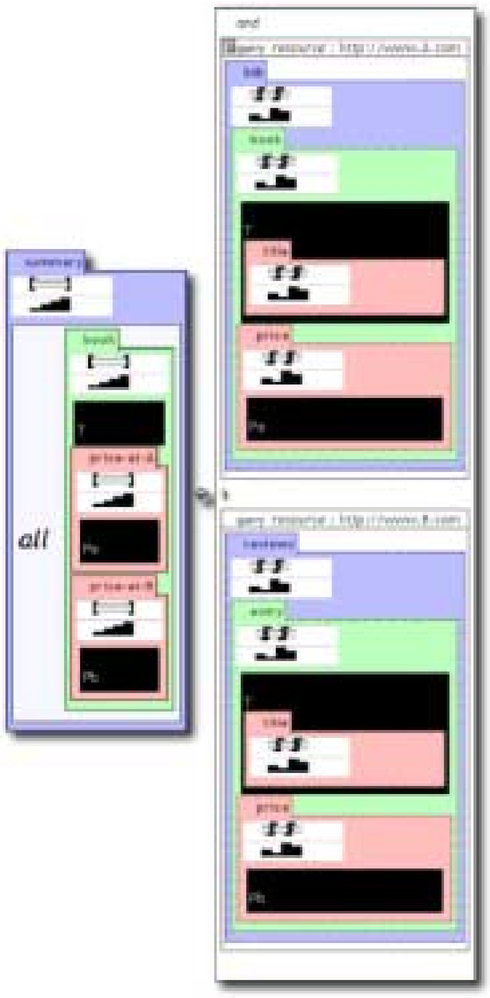


Figure 13: Visualisation of a rule

4.3.3 Dynamic Features

A static visualization is not sufficient for an interactive query system like Xcerpt. The visXcerpt prototype provides features that allow for easier navigation and improved comprehension:

- *Partial Viewing*. For large documents, only a part is displayed in the viewer. Vertical and horizontal scrollbars allow one to move the current view.
- *Information Hiding*. In large documents, it is desirable to be able to hide such information that is irrelevant for the current task. In visXcerpt, clicking on the title tab of an element "folds" the element together with all its contents such that only the title tab remains (in a shaded color).
- *References*. visXcerpt "moves them into hyperspace" by representing them as hyperlinks. That is, by clicking on a reference the visualisation scrolls or focuses on the occurrence of the referenced element. Hovering with the pointer above a term with ID highlights all occurrences of references to it. Backward navigation to references is supported through a popup menu of elements containing an ID.

Variable Highlighting. In (vis)Xcerpt rules, all variables that appear in the head of a rule are also required to appear in its body. Moreover, the same variable may occur in several parts of the body, even several times within the same query term. To support the user in designing visXcerpt rules, all occurrences of a variable are highlighted by inverting its color when the mouse hovers over one occurrence of the variable (see Figure 14). This eases the comprehension of term equality in positional queries and thus allows the user to recognize connections between different parts of a rule or within a term.

The visXcerpt prototype implementation also provides an editor to “try out” programs while developing them. Since the editor has to edit tree structured data, many of the editing capabilities commonly found in plain text editors have only limited applicability. Thus, in addition to common text editing primitives (like “cut”, “paste”), the visXcerpt editor provides primitives that are suitable for insertion of subtrees and nodes (e.g. “paste into at beginning/end” or “paste before/after”). The visual interface ensures that the syntax is always correct).

4.3.4 Future Work

The visXcerpt prototype will be extended by adding improved browsing facilities, like browsing from an element to such elements that refer to it. Furthermore, investigating suitable commands for editing tree- and graph-structured data is of major interest. As the current visXcerpt editor is implemented prototypically in HTML and JavaScript, a more efficient implementation is also sought for, possibly by extending already-existing XML editors.

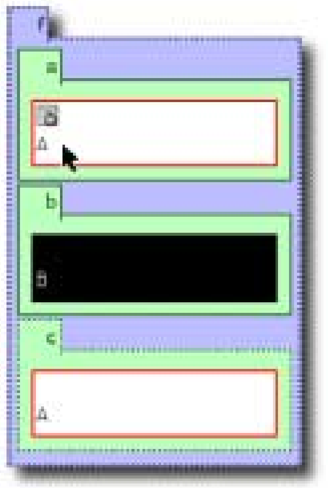


Figure 14: Variable highlighting

5 VISUALIZING REACTION RULES

Reaction rules are considered to be the most important type of business rule in [TW02]. Reaction rules consist of a mandatory triggering event term, an optional condition, and a triggered action term or a post-condition (or both). There are basically two types of reaction rules: those that do not have a post-condition, which are the well-known *Event-Condition-Action (ECA) rules*, and those that do have a post-condition, which can be called *ECAP rules*.

Reaction rules can be used for specifying the reactive behavior of a system and for expressing *interaction patterns*.

In symbolic form, a reaction rule is defined as a quadruple

$$\varepsilon, C \rightarrow \alpha, P$$

where ε denotes an event term (the triggering event), *C* denotes a logical formula (the state condition), α denotes an action term (the triggered action), and *P* denotes a logical formula specifying the effect or post-condition.

5.1 Case 1: The MIT Beer Game

The MIT Beer Game is a management simulation that was developed by the System Dynamics Group of the Sloan School of Management in the early 1960s.

The game is played by teams of four persons, each person playing one of the four roles of Retailer, Wholesaler, Distributor and Factory. The four roles are arranged in a simple beer supply chain (see Figure 15).

The factory has an unlimited supply of raw materials and each of the roles has an unlimited storage capacity. The supply lead time (the time between shipment by the supplier, and arrival at its destination) and order delay time (the time between the sending of an order, and the arrival of the order at its destination) are fixed.

Supplies take two full turns to arrive, the orders for new beer arrive at their destination in the next turn.

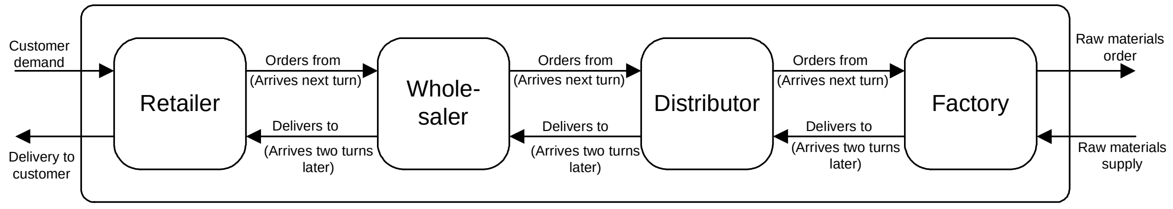


Figure 15: The supply chain structure of the MIT Beer Game

Each turn (usually a simulated week), the retailer receives a customer order and tries to ship the requested amount from its inventory. It then orders an amount of beer from its supplier, the wholesaler, which tries to ship the requested amount from its inventory, and so on. Orders that cannot be met are placed in backorder and must be met as soon as possible.

At the end of each week, each role has to calculate that week's costs. Remaining inventory is charged \$0.50 per item as holding costs and backorders are charged \$1.00 per item as shortage costs. The objective of the game is to be the team with the lowest overall costs or to be the player with the lowest cost within a team, after playing a fixed number of weeks.

The players have complete information about their own role, including a history of their own incoming orders, outgoing orders and backorders. However, they have no information about the other roles. The customer demand is only known to the retailer, and no player knows the inventory size of the other players, or how much the other players have in backorder.

Though the simulation is a very simplified version of the real life situation (each role has unlimited labour, capacity, machines and funds and there are no competitors), the average costs after running 36 weeks is \$2000 and the orders between the players show an extremely oscillating pattern. The calculated optimal strategy would have a total cost of only \$200 [Ster92].

When looking at the Beer Game from an agent-oriented viewpoint, we can identify four agents (the retailer, wholesaler, distributor and factory) that have similar behavior. We exclude the end customer and the raw materials supplier from the simulation and therefore assume that the environment sends beer orders to the retailer, receives beer deliveries from the retailer and supplies the factory with raw materials.

Each agent receives orders from its client. The client is either the end customer, in case of the retailer, or the downstream supply chain node, in the other cases. Each order specifies a quantity of beer, which is added to the order quantity for that week. The agent can also receive a beer delivery from its upstream node or, in case of the factory, a raw materials delivery from the environment. The delivered quantity is added to the quantity in stock.

At the end of the week, the orders for this week and the backorder quantity are added to form the total outstanding order. This total order quantity is compared to the current quantity in stock. If the latter is equal to or greater than the former, the quantity of beer shipped is equal to the total order quantity, and the backorder quantity is set to 0.

If the inventory is insufficient to fulfil the total order, the entire inventory is shipped and the remaining (unfulfilled) part of the total order is recorded as being in backorder. At the end of its turn, the agent decides on the amount of beer it wants to order from its supplier.

The first reaction rule, R1, is triggered periodically at the end of each business week. It compares the inventory with the total number of outstanding orders. Depending on the outcome of this test, either the entire order or the maximum available quantity of beer is shipped. The quantities of the inventory, backorder and this week's orders are then adjusted accordingly.

Reaction rule R1: shipping outstanding orders

```
ON      Event    End of Week
IF      Cond     Inventory >= (OrderedQuantity + BackorderQuantity)
THEN    Action   Ship( OrderedQuantity + BackorderQuantity)
        Effect   Inventory = Inventory@pre – (OrderedQuantity +
                 BackorderQuantity@pre)
                 AND BackorderQuantity = 0
ELSE    Action   Ship(Inventory@pre)
        Effect   BackorderQuantity = (BackorderQuantity@pre + OrderedQuantity) –
                 Inventory@pre
                 AND`Inventory = 0
                 AND OrderedQuantity = 0
```

The second reaction rule, R2, handles the ordering of new beer from the upstream agent. This action takes place at the end of the week. The trigger for this rule is a perception that the end of the week has been reached.

Reaction rule R2: Ordering stock

```
ON   Event    End of Week
DO   Action   SEND( Order( ComputeQty())
```

The third reaction rule, R3, is triggered by the reception of a new order from the customer. It adds the new amount of requested beer to the amount of beer that had already been ordered this week.

Reaction rule R3: receiving a new order

```
ON   Event    Order
DO   Effect   OrderedQuantity = OrderedQuantity@pre
              + Order.Quantity
```

The fourth reaction rule, R4 is triggered by the arrival of a new shipment of beer. The quantity of the beer in the event is added to the existing amount.

Reaction rule R4: receiving new beer

```
ON   Event    Delivery
DO   Effect   Inventory = Inventory@pre + Delivery.Quantity
```

The fifth and last reaction rule handles the calculating of the costs of the past week, and adds it to the total costs. Like the shipping of beer and the ordering of new stock, this action takes place at the end of a business week and is triggered by the perception of the end of the week.

Reaction rule R5: Calculating costs

```
ON   Event    End of Week
DO   Effect   Costs = Costs@pre + (0.5 * Inventory) + (1 * BackorderQuantity)
```

5.2 Survey

In this survey, we focus on rules that are explicitly visualized. Hence, rules that can only be derived from a model are not part of this survey. For example, the semantics of a simple ECA rule can be completely modelled in terms of existing statechart notations.

For the purpose of this survey we divide rule visualization into two categories:

- Visualisation of rules in modelling diagrams such as class diagrams, ER diagrams, or specific rule diagrams. Characteristics of these approaches are that they have a graphical notation and are used during software development.
- Visualization of how rules behave and interact with each other and the underlying data model at run-time. Characteristics of these approaches are that they are related to tools for analysing rule behaviour and debugging rule sets. Some of the suggested tools also have features for animating rule behaviour at run-time.

For the purpose of this survey we focus on the first category, i.e., visualisation of rules in modelling diagrams. The second category is considered as a related area and is briefly summarized at the end of this survey. In subsequent sections we present a survey of suggested approaches for rule visualization in modelling diagrams.

5.2.1 AORML

In the Agent-Object-Relationship modeling language (AORML) proposed in [Wag03], an entity is an agent, an event, a commitment/claim, or an ordinary object. Only agents can communicate, perceive, act, make commitments and satisfy claims. Objects are passive entities with no such capabilities.

Figure 16 shows the AOR Interaction Frame Diagram describing the possible interactions between two agents of the Beer Game supply chain.

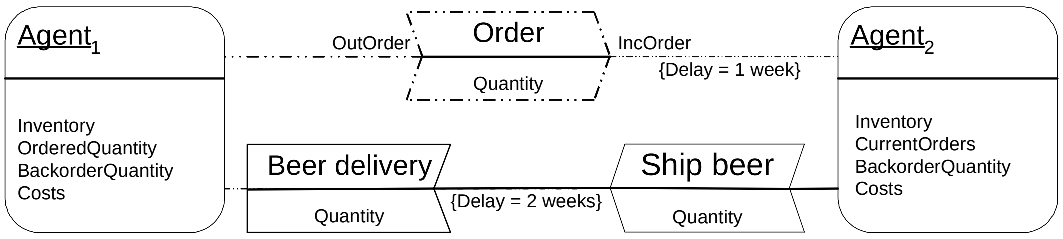


Figure 16: The interaction frame between two beer game agents

The incoming and outgoing orders are modeled as messages, the shipping of beer is modeled as a *non-communicative action event* and the reception of beer is modeled as a *non-action event*. The shipment of a quantity of beer by one agent and the reception of this shipment by the other agent are modeled as two different events (with a two week delay) abstracting away from the transportation process.

In AORML, a reaction rule is visualized as a circle with incoming and outgoing arrows drawn within the agent rectangle whose reaction pattern it represents. Each reaction rule has an incoming arrow with a solid arrowhead, which specifies the triggering event type. In addition, there may be ordinary incoming arrows representing state conditions (referring to corresponding instances of other entity types).

There are two kinds of outgoing arrows: one for specifying mental effects (changing beliefs and/or commitments) and one for specifying the performance of (physical and communicative) actions. An outgoing arrow with a double arrowhead denotes a mental effect. An outgoing connector to an action event type denotes the performance of actions of that type. Figure 4.2 shows an example of an *interaction pattern diagram* describing the interactions of a Beer Game agent. For instance, the reaction rule R3 has a trigger arrow and a mental effect arrow. It is triggered by `Order` messages and leads to a state change that is specified by the OCL postcondition:

```
OrderedQuantity = OrderedQuantity@pre + Order.Quantity
```

The Diagram depicting the reaction rules R1–R5 is shown in Figure 17.

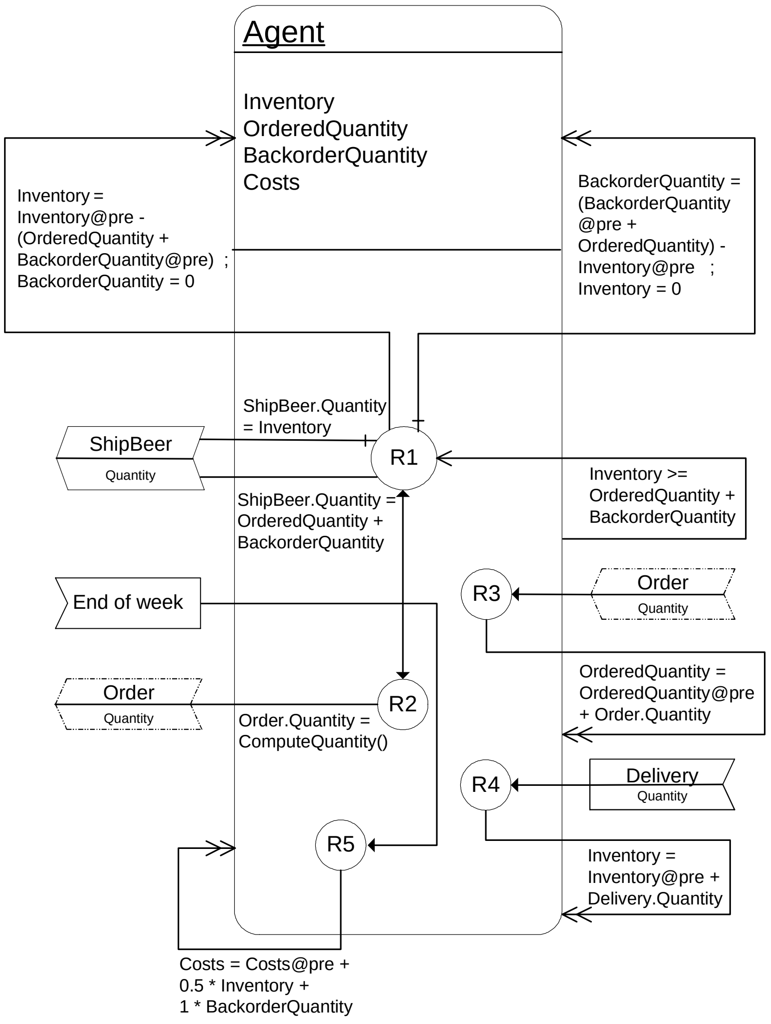

Agent
Inventory
OrderedQuantity
BackorderQuantity
Costs
Inventory = Inventory@pre - (OrderedQuantity + BackorderQuantity@pre) ; BackorderQuantity = 0
BackorderQuantity = (BackorderQuantity @pre + OrderedQuantity) - Inventory@pre ; Inventory = 0
ShipBeer.Quantity = Inventory
ShipBeer
Quantity
R1
Inventory >= OrderedQuantity + BackorderQuantity
ShipBeer.Quantity = OrderedQuantity + BackorderQuantity
End of week
R3
Order
Quantity
OrderedQuantity = OrderedQuantity@pre + Order.Quantity
Order
Quantity
R2
Order.Quantity = ComputeQuantity()
R4
Delivery
Quantity
R5
Inventory = Inventory@pre + Delivery.Quantity
Costs = Costs@pre + 0.5 * Inventory + 1 * BackorderQuantity

Figure 17: An AOR Interaction Pattern Diagram for Beer Game agents specifying 5 reaction rules.

5.2.2 A/OODBMT

A/OODBMT (Active Object-Oriented Database Modeling Technique) [Sil95], [SC96] uses OMT [RB+91] as a platform for introducing support for rule modelling. In particular, A/OODBMT introduces four models: i) the nested object model, ii) the behaviour model, iii) the nested rule model, and iv) the nested event model. Out of these models, i), iii), and iv) explicitly visualize rules. The suggested models in A/OODBMT have support for several different types of rules, e.g., ECA rules, exception rules, contingency rules, operation precondition rules, operation postcondition rules, and production rules. For the purpose of this survey we focus on ECA rules and exception rules.

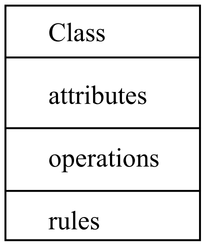


Figure 18: Rule support in A/OODBMT's nested object model.

The nested object model (NOM) extends the object model in OMT by adding rules to the class definition, see Figure 18 . Only the rule name is added to the class definition and the specification of the rule itself is done in a separate model, i.e. nested rule model.

The nested rule model (NRM) is composed of two diagrams: nested rule diagrams and (NRDs) and rule interaction diagrams (RIDs). One purpose of the nested rule diagram is to visually represent rules that constrain an object's structure, e.g., attribute constraint, attribute domain constraint, association cardinality constraint. Figure 19 presents the legend for a simple integrity constraint. The rule is represented as an ellipse. Inside the ellipse is the class that is related to the constraint, a short description what it is that is constrained (Constraint), and condition specified within a hexagon icon.

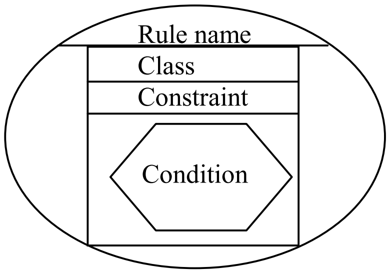


Figure 19: Legend for specification of a simple integrity constraint in A/OODBMT's nested rule diagram.

NRDs also support the representation of traditional ECA rules and exception rules, where exception rules are considered as a special case of ECA rules. Figure 20 presents the legend for ECA rules and exception rules. The triggering event is represented as a rhomb, which holds the name of the event. The condition is specified within a hexagon icon, and actions are specified within circles. The arrows dictate the flow of control. In the case of exception rules only one action will be executed. It should be noted that a NRD only show one rule at the time. Thus, it is not possible to see whether a rule action acts the triggering event for another rule. This type of interaction is shown in a separate diagram type, i.e., rule interaction diagrams.

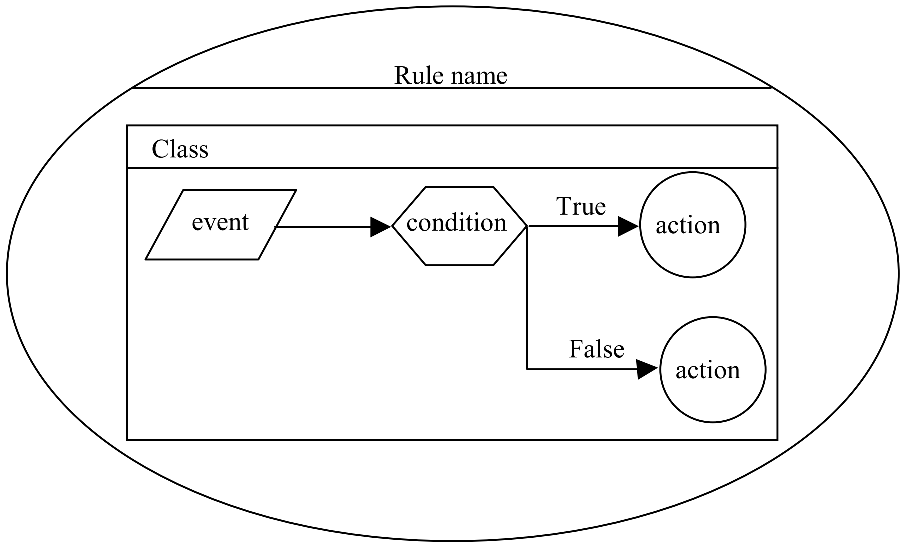


Figure 20: Legend for specification of ECA rules and exception rules in A/OODBMT's nested rule diagram.

The purpose of rule interaction diagrams (RIDs) is to show the interdependence between rules. RIDs are organized hierarchically to avoid problems with flat diagrams. A RID is represented by a box that shows a rule (ellipse) and which rule that triggers it (dotted ellipse). For example, in Figure 21 R3 triggers R2, which in turn triggers R1. R1 does not trigger any rules. A special rule object, rule connector, is introduced to depict the triggering rules of a rule. Navigation between the RIDs is done by expanding a rule connector object.

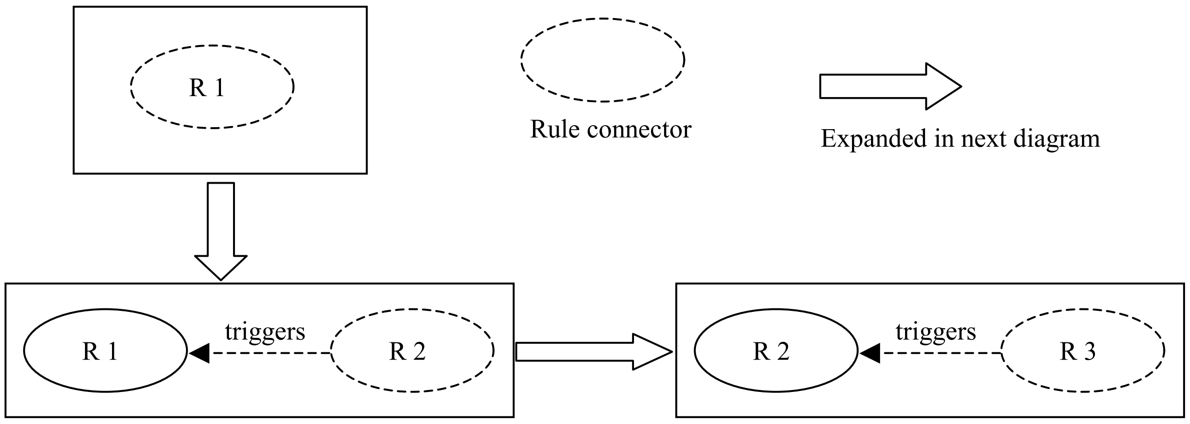


Figure 21: Legend for specification of rule interaction in A/OODBMT's rule interaction diagram.

The nested event model (NEM) describe in detail the events that are used in the other diagrams. Both primitive events (method events, transaction events, temporal events, periodic temporal events, and explicit events) and composite events (conjunction, disjunction, monitoring interval, relative temporal, closure, history, every-nth, negative, and sequence events) are supported. Figure 22 shows the legend for specifying a composite event. The constituent events are described with a rhomb notation, and they differ slightly depending upon event type, e.g., time events are specified within a box inside the rhomb, whereas method events are simply specified inside the rhomb. In addition a special bar notation is used to depict events that are detected over an interval.

The notation for the most common event operators are: i) sequence – no special notation, events are simply arranged in sequence, ii) conjunction – a special notation is used as shown in Figure 22, and the word "AND" is written underneath the event operator notation, and iii) disjunction – same as ii) but with the word "OR" instead.

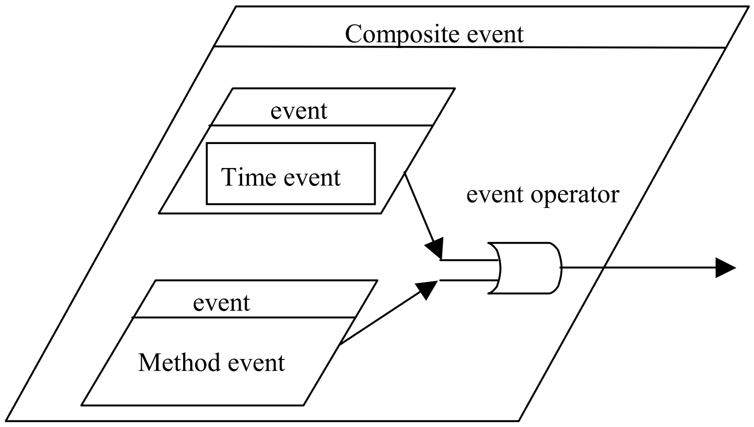


Figure 22: Legend for specification of composite events in A/OODBMT's nested event diagram.

The three rules R1-R3 in the MIT beer game are visualized in A/OODBMT both individually (each rule modelled in a nested rule diagram) and also together (the interaction between the rules are modelled in a rule interaction diagram). For example, R1, which includes the behaviour of both a traditional ECA rule and an exception rule, is modelled in Figure 23.

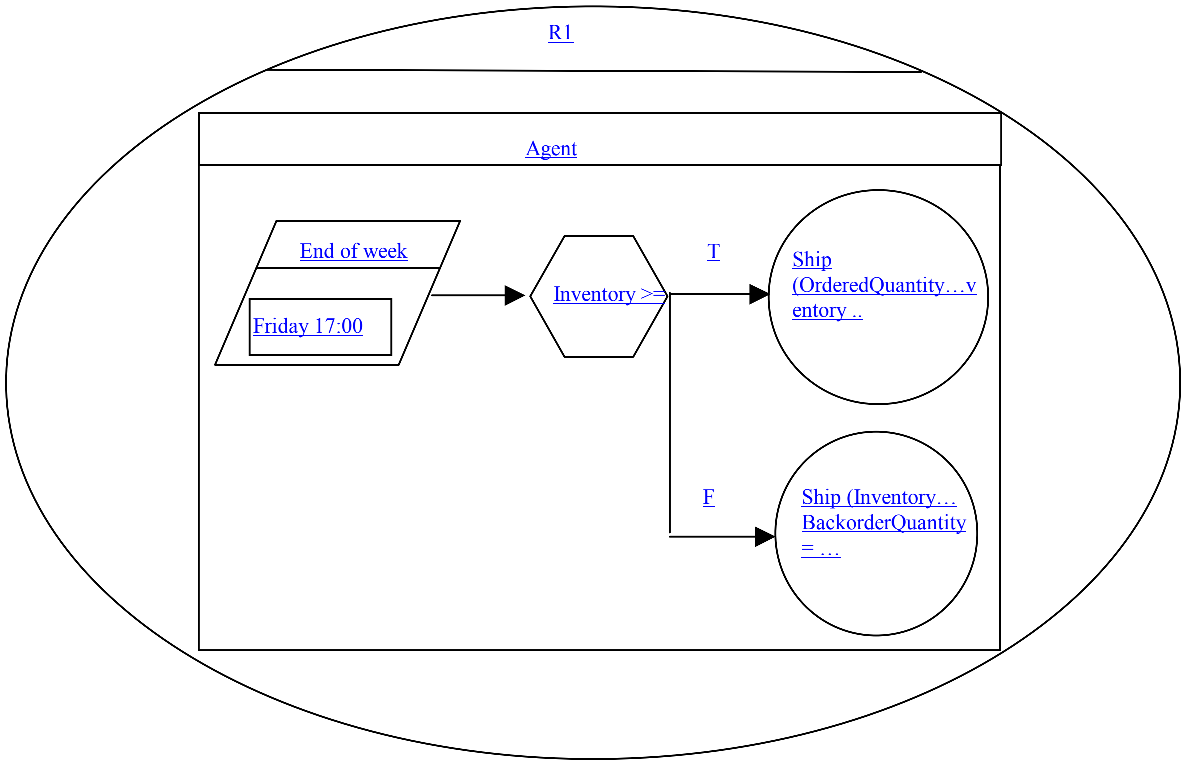

R1
Agent
End of week
Friday 17:00
Inventory >=
T
Ship (OrderedQuantity…v entory ..
F
Ship (Inventory… BackorderQuantity = …

Figure 23: Rule R1 modeled in a nested rule diagram

Although, R1 and R2 are triggered by the same "end of week event", it is only R2's action "send order quantity" that can trigger R3. Thus the interaction between the rules is straightforward to model in a rule interaction diagram as shown in Figure 24. Rules R1 and R3 are the top level since they do not trigger any other rules in this example.

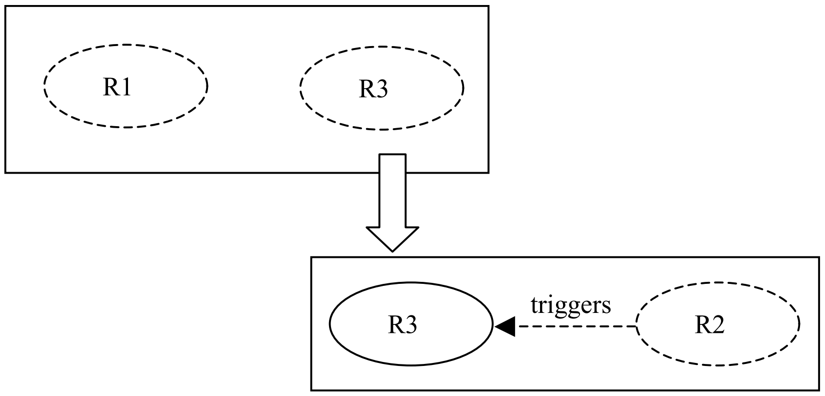


Figure 24: Rules R2 and R3 modeled in a rule interaction diagram

A/OODBMT introduces several new models and notations for supporting modeling of ECA rules. Although, A/OODBMT can model the example rules R1-R3 properly, the extensions made to the underlying method/notation, i.e., OMT, are rather extensive, so in that regard A/OODBMT is really a new method/notation rather than an extension to OMT.

5.2.3 *ECA^2 Nets and (ER)^2 Models*

The work in [Tan92, NTC92] extends the ER model with an ECA^2 net notation for modelling ECA rules. Briefly, in ECA^2 nets are high-level Petri nets and they have also been used in [Her97] for visualizing business rules that are structured as ECA rules.

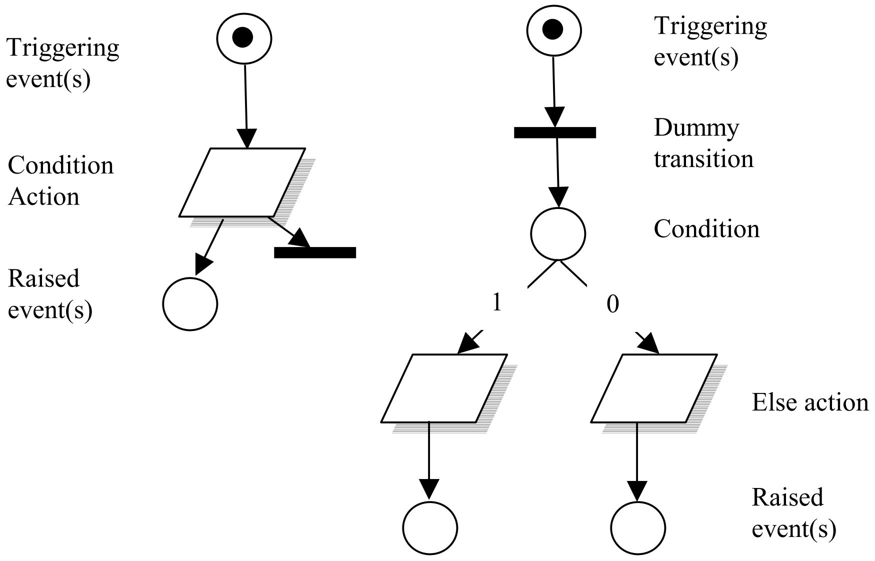


Figure 25: Legend for ECA2 nets

In Figure 25, places are either events or conditions, transitions are either dummy transitions or actions. Raised events can in turn act as triggering events for other rules. The parallelogram only holds the rule name, and details of the rule are specified by text in a separate document. ECA^2 nets are coupled to an ER-diagram by arc labels. First, an arc label can be drawn from an entity in the ER-diagram to a triggering event in an ECA^2 net. Secondly, a dashed arc label between conditions in an ECA^2 net and entities in the ER-diagram denotes that additional data is used by the rule for condition evaluation. A simple connection between an ECA^2 net and an ER-diagram is shown in Figure 26. A model that includes both an ECA^2 net and an ER-diagram is referred to as an (ER)^2 model.

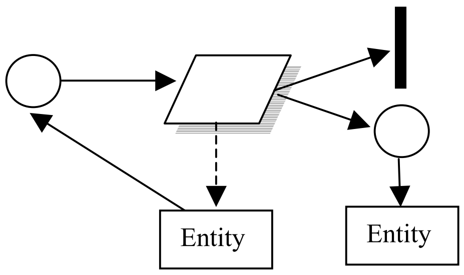


Figure 26: A simple (ER)2 model

The example rules R1-R3 can be visualized in an (ER)^2 model as shown in Figure 27. The event "Friday 17.00" triggers R1 (check inventory) and R2 (send order). Rule R2, in turn triggers R3 (update order quantity). The dashed arrows to "Agent" indicate that rules R1 and R3 use data from "Agent".

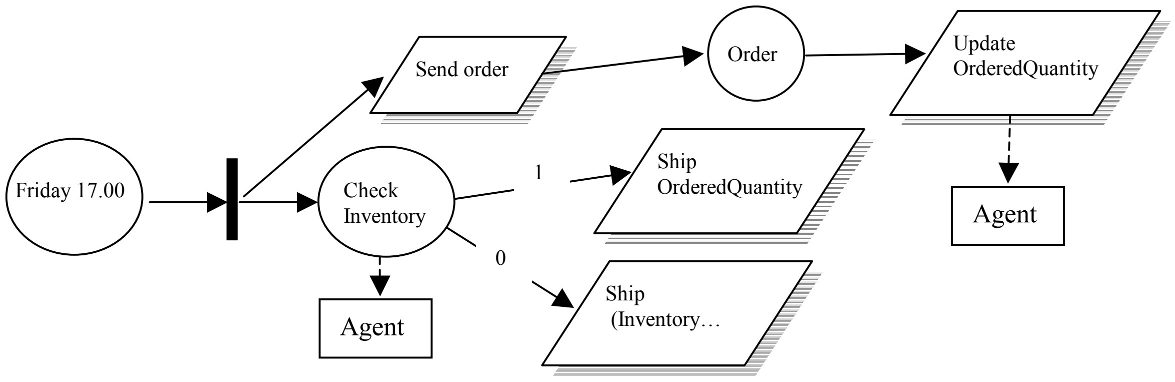


Figure 27: Rules R1-R3 in (ER)2.

Although the example rules R1-R3 can be modelled in an (ER)^2 model with the aid of a high-level Petri net, it is questionable whether this low-level modelling is useful in practice when designing an application together with a customer. However, it might be more useful to use an (ER)^2 model in the later design phases rather than the early phases of the software development.

The (ER)^2 notation is semantically overloaded in that the same notation is used for several concepts, for example, the same notation in used for dummy transitions and for actions.

5.2.4 IDEA

The IDEA methodology [CF97] introduces an informal notation for integrity constraints. Constraints in IDEA are divided into two categories: targeted constraints and untargeted constraints. Briefly, targeted constraints are defined in the context of a class, and untargeted constraints represent constraints that are not defined in the context of class.

Figure 28 shows the legend for visualising integrity constraints in IDEA. The dashed ellipse represent a constraint that can be specified in terms of execution mode, e.g., when is the condition evaluated (immediately after the event or at the end of the transaction), name of constraint, and a textual description of the integrity constraint. An arrow indicates that the constraint is defined in the context of a class.

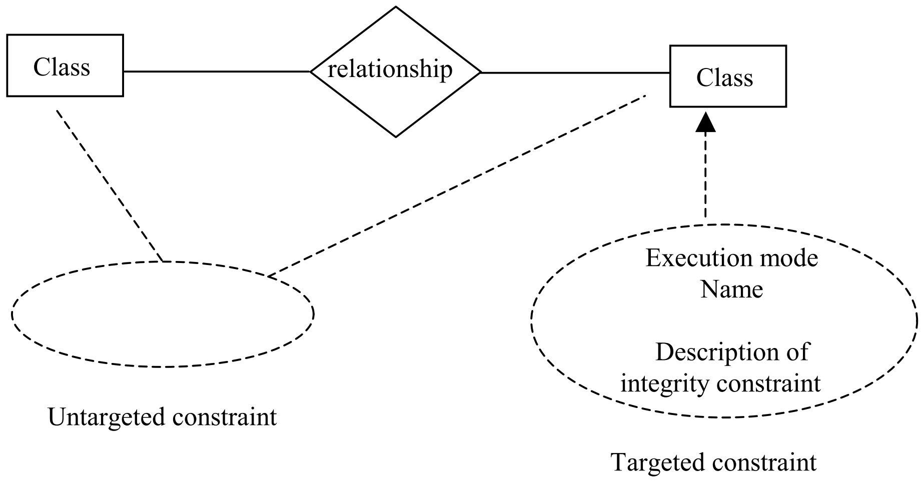


Figure 28: Legend for integrity constraints in IDEA.

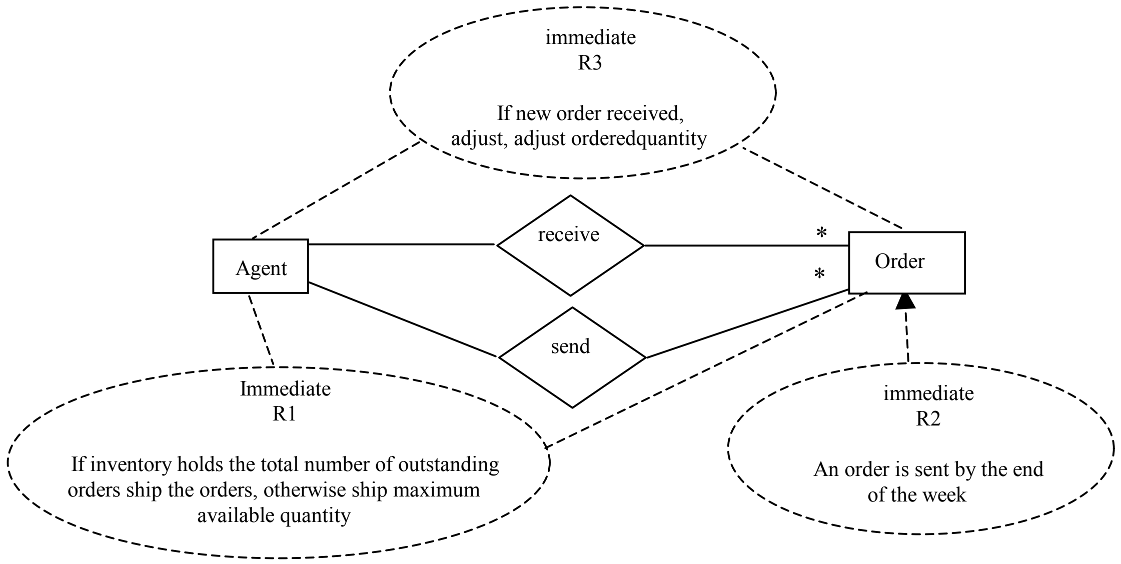

immediate
R3
If new order received, adjust, adjust orderedquantity
receive
Agent
*
*
Order
send
Immediate
R1
If inventory holds the total number of outstanding orders ship the orders, otherwise ship maximum available quantity
immediate
R2
An order is sent by the end of the week

Figure 29: Rules R1-R3 in IDEA notation.

The three rules R1-R3 in the MIT beer game are visualized in IDEA as shown in Figure 29. Briefly, R1 and R2 are both modelled as untargeted constraints to representing sending and receiving orders, whereas, R2 is modelled as a targeted constraint that describes when an order is sent.

As the notation in IDEA is informal it is not possible to distinguish between events, conditions, and actions. In addition it is not possible to see the interaction between the rules.

5.2.5 OMT-A and UML-A

OMT-A [Cal99] extends OMT [RB+91] with features for modelling ECA rules. In particular, graphical notations are added to class diagrams and state diagrams.

Modelling of rules in class diagrams is supported at three different levels of abstraction: i) top level, ii) rule level, and iii) ECA level.

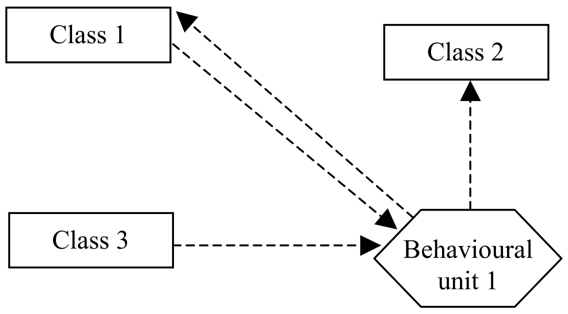


Figure 30: Rule modeling in OMT-A at the top level

Figure 30 presents a how rules are modelled at the top level in OMT-A. At this level, it is assumed that a designer is not able to exactly define the involved rules and how they interact with the classes. Instead, it is suggested that a "behavioural unit" is added to the class diagram for expressing reactive rule behaviour that might be implemented by one or more ECA rules. The semantics of the dashed arrows are "affects / is affected by". For example, "Rule 3" is triggered by an event from "Class 1" and the rule action affects "Class 1".

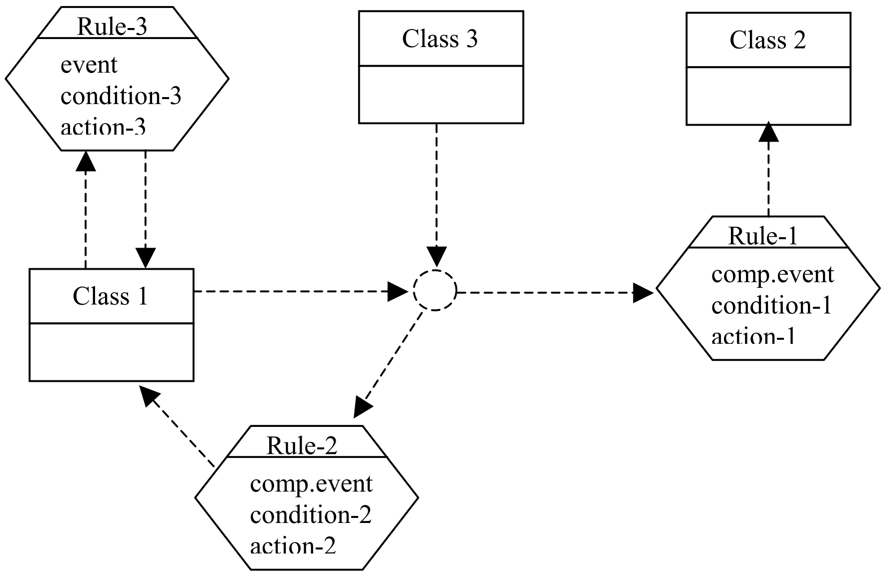


Figure 31: Rule modelling in OMT-A at the rule level

Figure 31 presents how rules are modelled at the "rule level" in OMT-A. At this level a designer is able to define individual ECA rules in terms of triggering event, condition to be evaluated, and action to be executed. The circle notation represents a composite event. At this stage of the modelling it is assumed that a designer can identify whether there are any composite events involved, but not necessarily specify what type of event operator that is needed. Hence, the empty circle and the rule attribute "comp. event". If it is possible to specify an event operator, then event operator is added to the circle and the rule attribute "comp. event" removed. In contrast to the model in Figure 30 rule-3 has been identified as a rule that only interacts with class-1.

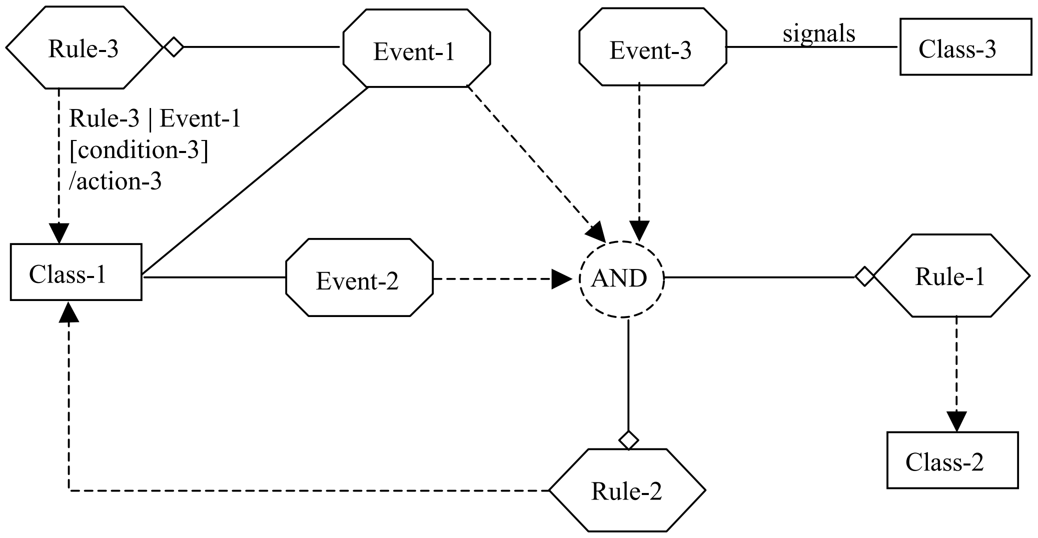


Figure 32: Rule modeling in OMT-A at the ECA level

Figure 32 presents how rules are modelled at the "ECA level" in OMT-A. At this level a designer is able to model composite events and explicitly specify the source of the event signal. The diamond symbol is the UML aggregate notation. A special event symbol is introduced.

UML-A [BC03] refined the graphical notations suggested in [Cal99] by introducing explicit rule components in UML statecharts. In addition, the work in [BC03] showed that many of the UML notations can be used "as is" in order to visualize rule behaviour.

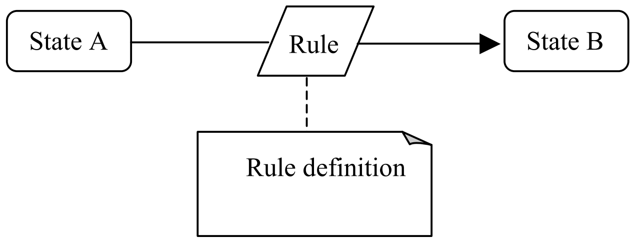


Figure 33: A simple ECA rule in UML-A

Figure 33 presents how a simple ECA rule is visualized in UML-A.

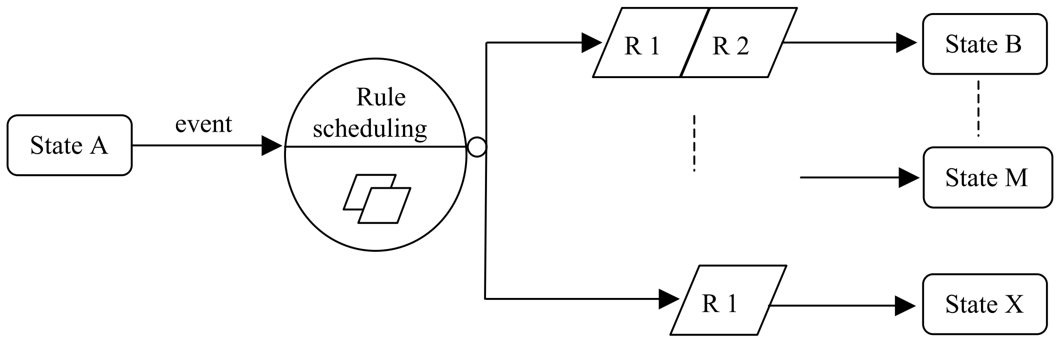


Figure 34: Multiple rule execution in UML-A

Figure 34 presents how multiple rule execution is visualized in UML-A. When an event triggers multiple rules, the rules must be scheduled, for example, according to priority or concurrent rule execution. In UML-A this is reflected by introducing a circle notation in which the scheduling is defined. In Figure 34, "Rule scheduling" would be replaced by, for example, "rule priority". As the choice of rule scheduling might lead to different states, an UML junction state is used to separate outgoing state transitions.

Rules R1-R3 are visualized in OMT-A and UML-A as shown in Figure 35 and Figure 36.

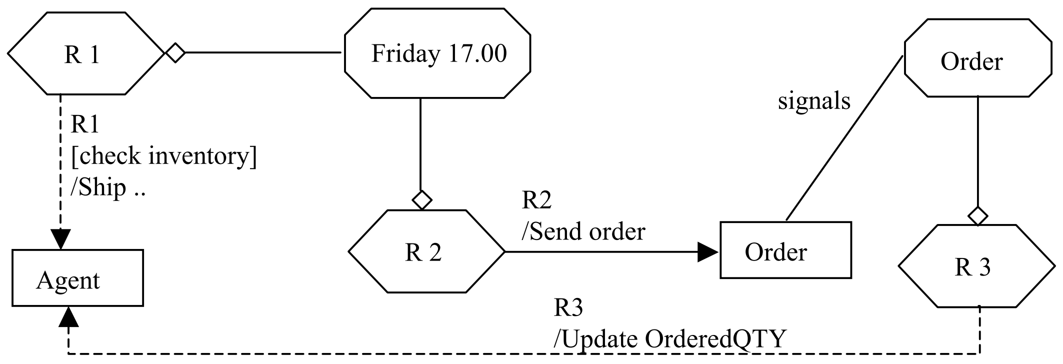


Figure 35: Rules R1-R3 in OMT-A

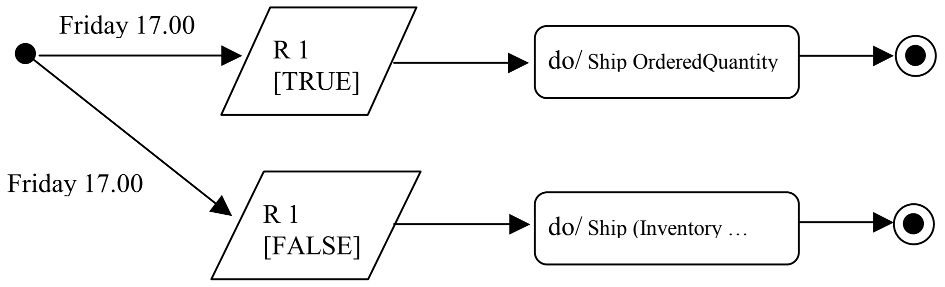


Figure 36: Rule R1 in UML-A

In the example, OMT-A cannot visualize that R1 is an exception rule and has an "else action". In addition it is not possible to see the interaction between the rules. Actions and conditions are not visualized in OMT-A. Instead they are part of the dashed lines that indicate a rule affect on a class.

As UML-A extends the UML state charts with notations for ECA rules, it has proper support for modelling R1-R3 as long as the rules do not span more than one class. For example, the exception rule R1 can easily be modelled in UML-A, but it is not possible to see that the event "Friday 17.00" also triggers another rule that is, in this example, related to another class.

5.2.6 OMT+

OMT+ [ZB+96] extends class diagrams in OMT [RB+91] by adding graphical notations for ECA rules, Figure 37. Hence, OMT+ supports modelling of both objects and rules in the same diagram. Rules are represented as ovals and a triggering relationship with classes by a dashed arrow line.

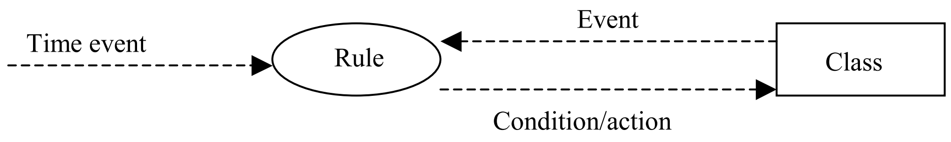


Figure 37: Legend for OMT+

The triggering events are either method invocations in the classes or events not related to a specific class, for example, time events. The "Condition/action" relationship between the rule oval and the class box represent which class is affected by the rule execution. For example, a rule might need to access a class during condition evaluation and invoke methods for action execution. In the examples presented in [ZB+96], “Rule” is replaced by a rule identifier (R 1), “Event” is replaced by the name of the triggering operation, and “Condition/action” is replaced by the operation in the class that is used during rule execution.

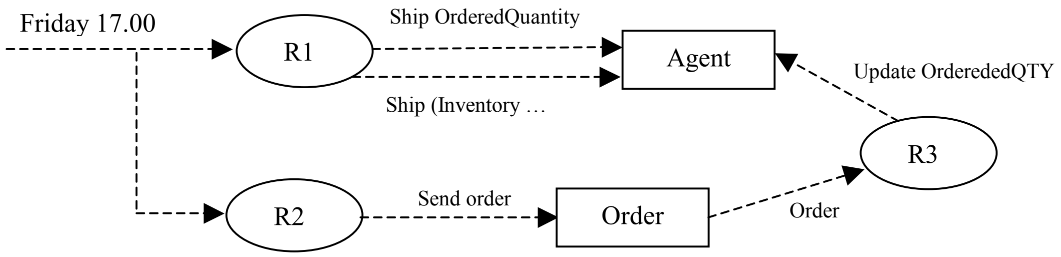


Figure 38: Rules R1-R3 modelled in OMT+

The major problem with the OMT+ notation is that the notation is semantically overloaded, for example, the same notation is used for events, conditions and actions. In addition it is not possible to see in Figure 38 that R2 triggers R3.

5.2.7 BROCOM

Business **R**ule-**O**riented **CO**nceptual **M**odelling (BROCOM) was proposed in [Her95] and [Her97]. The methodology is based on the metamodel that consists of the following submodels: ‘Business Rule’, ‘Data Model Components’, ‘Processor’, ‘Origin’, ‘Organizational Unit’, and ‘Process’.

The submodel ‘Business Rule’ consists of the four meta entity types: *Business rule*, *Event*, *Condition*, and *Action*. Every business rule has exactly one event, at most one condition, and one or two actions (*THEN* / *ELSE*). Events and conditions may be composite and therefore have recursive M:N relationship types. Actions of business rules may raise events. In the metamodel, this is represented by the relationship ***is_raised_by*** between the meta entity types *Action* and *Event*.

Followingly, reaction rules R1 – R5 of the MIT Beer Game, which were introduced in section 3.1, are presented according to the BROCOM methodology as the respective business rules *[BR1] – [BR5]*:

```
[BR1]  ON end of week
       IF (Inventory >= (OrderedQuantity + BackorderQuantity))
       THEN  Ship beer (OrderedQuantity + BackorderQuantity), ⇒ EVENT ‘beer shipped’
             SET Inventory := Inventory – (OrderedQuantity + BackorderQuantity)
             SET BackorderQuantity := 0
       ELSE  Ship beer (Inventory), ⇒ EVENT ‘beer shipped’
             SET BackorderQuantity := (BackorderQuantity + OrderedQuantity) – Inventory
             SET Inventory := 0
             SET OrderedQuantity := 0

[BR2]  ON end of week
       THEN  Send order (Order(ComputeQty())), ⇒ EVENT ‘order delivered’

[BR3]  ON order delivered
             SET OrderedQuantity := OrderedQuantity + Order.Quantity

[BR4]  ON beer delivered
             SET Inventory := Inventory + Delivery.Quantity

[BR5]  ON end of week
             SET Costs := Costs + (0.5 * Inventory) + BackorderQuantity
```

The sub model ‘Data Model Components’ of the BROCOM methodology encompasses the meta entity types for a conceptual data model. In accordance with the Entity Relationship Model, the meta entity types ***Entity type***, ***Relationship type***, and ***Attribute*** are incorporated into this sub model. The allowed semantics of ***references*** relationship between components of business rules and data model components is put together in Table 3-1, adopted from [Her97]. For example, the impact of the rule *[BR3]* on data model components is insertion of a new order and increasing the value of the attribute *OrderedQuantity*. And the other way round, the data model component (entity type) *Order* is referenced by the rule *[BR3]*.

Table 5-1. Relationship between business rules and modelling constructs.

Relationship from	Retrieval	Modification
Event ⇒ Data model component	No	No
Condition ⇒ Data model component	Yes	No
Action ⇒ Data model component	Yes	Yes

The sub model ‘Processor’ links rule components to specific processors who/which execute them, i.e., who/which detect the occurrence of events, evaluate conditions, and perform actions. A ‘Processor’ can be a ***human actor***, ***machine***, or ***software program***. A ‘Processor’ raises 0 to N events, evaluates 1 to N conditions, and performs 0 to N actions.

Business rules may *originate* outside or inside an organization which is addressed by the sub model ‘Origin’. Externally originating rules can be further divided into ***natural facts*** which are eternally fixed and (e.g., legal) ***norms*** which are specified by the society and may change. Internal origins can be either *primary* or *secondary*; an origin is primary if its content is originally described in a source document, whereas a secondary origin has previously been derived from another source.

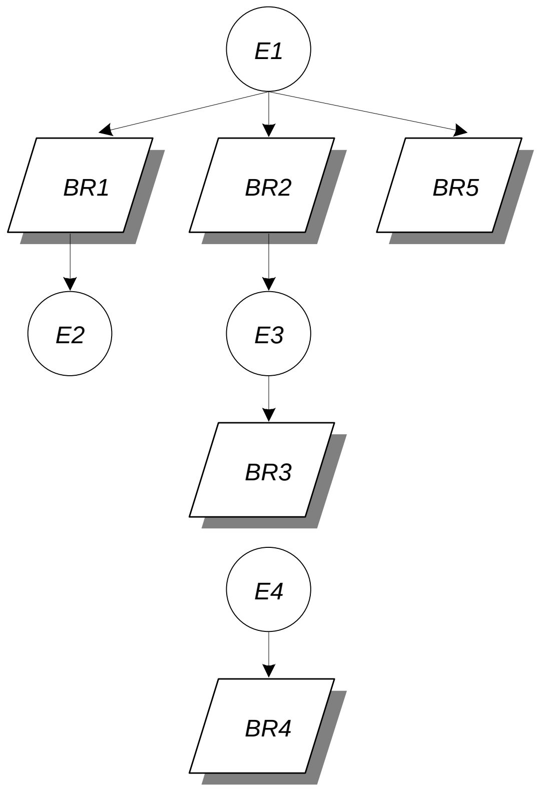


Figure 39: Order processing described by business rules

Within the sub model ‘Organizational Unit’, the assignment of business rule components to the ***organizational units***, which are responsible for processing the components, leads to *intra* and *inter* unit rules. This classification may help to support the administration of business rules in an organization. Organizational units ***own*** origins and ***encompass*** processors of business rules.

Within the sub model 'Process', actions of business rules can be related to events resulting in event rule nets describing the dynamic of ***processes*** like the example depicted in Figure 39. These rule nets were originally proposed [Tan92] and slightly simplified in the BROCOM approach. The example consists of the business rules *[BR1] – [BR5]*, which were presented above, and the respective events *E1* (*end of week*), *E2* (*beer shipped*), *E3* (*order delivered*), and *E4* (*beer delivered*). Processes can thus be specified by means of business rules. In the context of the metamodel only the behaviour of a business process is considered. Additional properties like process goals, values, and process owners are not further discussed.

In [Her97], the following five modeling steps are proposed:

1. Specification of the process structure.
2. Specification of the processes by using business rules.
3. Specification of the conceptual data model.
4. Specification of integrity constraints by using business rules.
5. Validation.

Steps one and two concern process specific business rules and steps three and four process independent business rules.

The BROCOM's sub model 'Processor' includes both human and automated actors. However, the BROCOM approach does not include the notion of communication/interaction, even though it acknowledges that actions performed by actors raise events. For example, raising the event 'order delivered' (the event E3 in Figure 39) by the rule [BR2], which occurs in an agent's "client" agent, and the reaction to this event by the rule [BR3] performed by the supplier agent could be more naturally modeled as sending a message from the requesting agent to the given agent, especially if they lie in different geographical locations. The same is not true for the rules [BR1] and [BR4] because, as has been explained in section 3.1, the shipment of a quantity of beer by one agent (the event E2) and the reception of this shipment by the other agent (the event E4) are modeled as two different events (with a two week delay) abstracting away from the transportation process.

5.2.8 Summary

Rule Modeling	**Diagrams**	**Semantically overloaded notation**	**Multiple rule firing**	**Rule interaction**
AOR	Rules are visualized in interaction pattern diagrams	No	Yes	Yes
A/OODMT	Rules visualized in nested rule diagram, rule interaction diagrams, nested event model	No	Yes	Yes
$(ER)^2$	Rules are visualized in ECA^2 nets	Yes	Yes	Yes
IDEA	Rules visualized in class diagram	Yes	No	No
OMT-A and UML-A	Rules visualized in class diagrams and statechart diagrams	OMT_A: Yes diagram UML-A: No	Yes	Yes
OMT+	Rules visualized in class diagram	Yes	Yes	No

5.3 Active Database Tools

Several active database tools have been proposed in the literature, for example, Dear [DJP93], ADELA [Fors95], SIEVE [CTZ95], and Vizar [CR+97]. These tools visualize rules and events to some extent. However, they are mostly limited to displaying whether a rule or event is "active" or not at run time. None of the tools seem to use a notation for rule visualization that can also be used for modelling rules.

6 ELEMENTS OF A FIRST-VERSION VISUAL RULE LANGUAGE

Our survey in the previous sections shows that there are many possibilities to visualize rules in different ways with different purposes. We have argued that a graphical notation for rules in UML class diagrams would be of particular importance, since class diagrams visualize vocabularies (resp. ontologies) and rules are based on vocabularies. Such a notation would therefore allow to show

- how rules are based on vocabularies
- how rules extend ontologies
- how rules can be used in a model-driven software engineering approach such as OMG's Model-Driven Architecture (MDA)

Based on our survey we can identify several requirements for such a visual rule language. In particular, we need

1. a graphical symbol for rules
2. a (semi-)graphical notation for certain types of logical formulas expressed in terms of other class modeling elements.
3. a notation for event types

6.1 A graphical symbol for rules

Rule is a new metaclass to be added to the UML metamodel. A graphical symbol for rules should have a shape that is different from all other model element shapes currently used in UML class diagrams. Candidate shapes are:

- circles, as in Figure 3 and Figure 17
- hexagons, as in Figure 35

It seems to be most natural to visualize the connections between rules and their components (conditions, conclusions, events, actions, postconditions) with the help of incoming and outgoing arrows.

6.2 A (semi-)graphical notation for logical formulas

Logical formulas may play the role of conditions, conclusions and postconditions in a rule. In each role, their syntax may be specifically restricted.

It seems to be natural to use the UML symbol for states, a rectangle with rounded corners, as in Figure 40, for expressing

1. status predicate conditions and conclusions by using the name of the Boolean attribute as the name of the state rectangle
2. general state (post-)conditions by putting a Boolean OCL expression in the state rectangle (instead of a name)

Checking and concluding the existence of links can be visualized as in Figure 3 by outgoing and incoming arrows to the connection line representing an association.

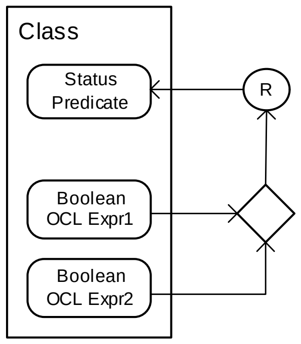


Figure 40: A derivation rule with a disjunctive condition.

How to express other types of atomic formulas is an issue for further research.

Negated conditions can be expressed as in Figure 3. Conjunctive conditions can be expressed by using multiple arrows as in Figure 3. Disjunctive conditions can be expressed by bundling them with the help of a diamond symbol (like the decision node of UML activity diagrams), as in Figure 40.

6.3 A notation for event types

For visualizing reaction rules, we need to distinguish two kinds of events:

- action events, which
  - can be the result of firing a reaction rule
  - can trigger a reaction rule
- non-action events, such as time events, which cannot be the result of firing a reaction rule but which can trigger a reaction rule

We can render these two kinds of event types graphically as in AORML (see the example shown in Figure 16 where *Order* and *ShipBeer* represent action event types and *BeerDelivery* respresents a non-action event type). This rendering of action events is the same as the rendering of an object node of type message in UML 2 activity diagrams, while this rendering of non-action events is the same as the rendering of an *AcceptEventAction* node in UML 2 activity diagrams. In UML 2 activity diagrams, there is a special graphical symbol for expressing time events that may also be used in a visual rule language: the.hour glass.

7 ISSUES FOR FUTURE WORK

This preliminary report is necessarily incomplete and leaves many issues unsolved. In particular, it would be important

- to relate rule modeling and visualization to the three abstraction levels defined by the Model-Driven Architecture fo the Object Management Group
- to integrate our rule modeling concepts with the UML metamodel
- to investigate what are the specific issues of modeling and visualizing Semantic Web rules based on RDF and OWL
- to investigate for which purposes which types of rule visualization tool would be useful
  - Graphical description tool for analysis and design
  - Validation and verification
  - Implementation
  - Maintenance

8 REFERENCES

[BBS03] S. Berger, F. Bry, and S. Schaffert: A Visual Language for Web Querying and Reasoning. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, Mumbai, India. 2003.

[BS02] F. Bry and S. Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *Proc. 2nd International Workshop on Web and Databases*, LNCS 2593, Springer 2002.

[BC03] Berndtsson, M. and Calestam, B. Graphical Notations for Active Rules in UML and UML-A. *ACM SIGSOFT Software Engineering Notes*, 28:2, ACM Press, 2003.

[BE91] Brayshaw, M., Eisenstadt, M. A practical graphical tracer for Prolog. International Journal of Man-Machine Studies, 35(5):597–631, 1991.

[Cal99] Calestam, B. *OMT-A: An Extension of OMT to Model Active Rules*, MSc dissertation HS-IDA-MD-99-001, Department of Computer Science, University of Skövde, Sweden, 1999.

[CF97] Ceri, S. and Fraternali, P. *Designing applications with objects and rules: The IDEA methodology*. International Series on Database Systems and Applications. Reading, MA: Addison-Wesley, 1997.

[CGT90] S. Ceri, G. Gottlob, L. Tanca: Logic Programming and Databases, Springer, 1990.

[CR+97] Coupaye, T., . Roncancio, C. L., Bruley, C., and Larramona, J. 3D Visualization of Rule Processing in Active Databases. In *Proceedings of the workshop on New paradigms in information visualization and manipulation, NPIV'97*, pp. 39-42. ACM Press, November 1997.

[CTZ95] Chakravarthy,S., Tamizuddin,Z., and Zhou, J. A Visualization and Explanation Tool for Debugging ECA Rules in Active Databases. In *Proceedings of the 2nd International Workshop on Rules in Database Systems*, volume 985 of Lecture Notes in Computer Science, pp. 197-212. Springer, 1995.

[DC91] Dewar, A. D., Cleary, J. G. Graphical display of complex information within a Prolog debugger. International Journal of Man-Machine Studies, 25(5):503–521, 1991.

[DJP94] Diaz, O., Jaime, A., and Paton, N. W. Dear: A debugger for active rules in an object-oriented context. In *Proceedings of the 1st International Workshop on Rules in Database Systems*, Workshops in Computing, pp. 180-193. Springer, 1994.

[For95] Fors, T. Visualization of Rule Behaviour in Active Databases. In *Proceedings of the IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3)*, 1995, pp. 215-231.

[Her95] Herbst, H. A Meta-Model for Specifying Business Rules in Systems Analysis. In: Iivari, J., Lyytinen, K., Rossi, M. (eds.), Proceedings of the Seventh Conference on Advanced Information Systems Engineering (CAiSE'95). Springer, 1995, pp. 186 – 199.

[Her97] Herbst, H. Business Rule-Oriented Conceptual Modeling. Physica-Verlag, 1997.

[HSG03] M. Hopfner, D. A. Seipel, Jürgen Wolff von Gudenberg: Comprehending and Visualising Software based on XML-Representations and Call Graphs. Proceedings of the IEEE International Workshop on Program Comprehension (IWPC'2003).

[MIT02] Website of the Web-based Beer Game. http://beergame.mit.edu/, 2002.

[NKD97] Neufeld, E., Kusalik, A., Dobrohoczki, M. Visual metaphors for understanding logic program execution. In: Davis, W., Mantel, M., Klassen, V. (eds.), Graphics Interfaces, pages 114–120, 1997.

[NTC92] Navathe, S. B., Tanaka, A.K., and Chakravarthy, S.Active Database Modelling and Design Tools: Issues, Approach and Architecture. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, 15(1-4):6-9, 1992.

[TN+91] Tanaka, A. K., Navathe, S. B., Chakravarthy, S., and Karlapalem, K. ER-R: An Enhanced ER Model with Situation-Action Rules to Capture Application Semantics. In *Proceedings of the 10th International Conference on Entity-Relationship Approach (ER'91)*, pages 59-76. ER Institute,Pittsburgh (CA), 1991.

[RB+91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. , *Object Oriented Modeling and Design*, Prentice Hall, 1991.

[SC96] Silva, M. J. V. and Carlson, C. R.. Conceptual Design of Active Object-Oriented Database Applications Using Multi-level Diagrams. In *Proceedings of the 10th European Conference on Object-Oriented Programming (ECCOP'96)*, volume 1098 of Lecture Notes in Computer Science, Springer, 1996, pp. 366-397.

[Sil95] Silva, M. J. V. *A/OODBMT, an Active Object-Oriented Database Modelling Technique*. PhD thesis, Illinois Institute of Technology, USA, 1995.

[Ster03] Sterman, J.D. Website of the Beer Game, http://mitsloan.mit.edu/faculty/r-beergame.php, MIT, 2003.

[Tan92] Tanaka, A. K. On *Conceptual Design of Active Databases*. PhD thesis, Georgia Institute of Technology, USA, 1992.

[TW01] K. Taveter.and G. Wagner. Agent-Oriented Enterprise Modeling Based on Business Rules, In Kunii, H.S.; Jajodia, S.; Sølvberg, A. (eds.): Proceedings of 20th International Conference on Conceptual Modeling (ER 2001), 27–30 November 2001, Yokohama, Japan. Springer-Verlag, Berlin, Lecture Notes in Computer Science (LNCS 2224), 2001, pp. 527–540.

[Wag03] Wagner, G. “The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behavior”, *Information Systems* 28:5 (2003), 475-504. See http://aor.rezearch.info/.

[WT03] Wagner, G. & Tulba, F. Agent-Oriented Modeling and Agent-Based Simulation. In Giorgini, P. & Henderson-Sellers, B. (Eds.), *Conceptual Modeling for Novel Application Domains*, volume 2814 of Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 205-216.

[ZB+96] Zimmermann, J., Brandig, H., Buchmann, A.P., Deutsch, A., and Geppert, A. Design, Implementation and Management of Rules in an Active Database System. In *Proceedings of the 7th International Conference on Database and Expert Systems Applications*, volume 1134 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 422-435.