



OCOPOMO

Open Collaboration in Policy Modelling

D4.2 SYSTEM AND USER DOCUMENTATION

C: USER MANUAL ON POLICY MODELLING AND SIMULATION TOOLS

Document Full Name	OCOPOMO_D4.2-C_DRAMS-UserManual.docx
Date	03/04/2013
Work Package	WP4, WP6
Lead Partner	Intersoft, SMA
Authors	Ulf Lotzmann, Ruth Meyer
Document status	v1.00 FINAL
Dissemination level	PUBLIC (PU)



Document Log

Version	Date	Comment	Author
0.0	08/08/2011	Initial version	Ulf Lotzmann
0.1	01/11/2011	First preview version	Ulf Lotzmann
0.2	02/02/2012	Syntax description completed, further additions in section 3	Ulf Lotzmann, Ruth Meyer
0.3	02/04/2013	Major revision, split up in user and system documentation	Ulf Lotzmann
0.4	03/04/2013	Correcting text and formatting.	Ulf Lotzmann
1.0	04/04/2013	Consolidation and finalisation	Ulf Lotzmann



TABLE OF CONTENTS

1. INTRODUCTION.....	5
2. DRAMS OVERVIEW.....	6
2.1. FUNCTIONALITY OUTLINE	6
2.2. RULE SCHEDULING MECHANISM	9
2.3. TRACEABILITY AND LINKS.....	12
3. USING DRAMS	13
3.1. USER ROLES	13
3.2. USER INTERFACE	14
3.2.1. Main Window.....	15
3.2.1.1. Data Dependency Graph	16
3.2.1.2. Rule Dependency Graph	17
3.2.1.3. Trace of Rule Schedule	17
3.2.2. Console Window	18
3.2.3. Output Writer Windows.....	19
3.2.3.1. Log output	19
3.2.3.2. Model Explorer	19
3.3. INTEGRATION IN REPAST MODELS	20
4. DRAMS LANGUAGE SYNTAX.....	21
4.1. STRUCTURE OF DEFINITION FILE	21
4.1.1. Type definitions.....	21
4.1.2. Fact templates.....	22
4.1.3. Facts.....	22
4.1.4. Rules	23
4.1.5. Clauses	23
4.1.6. Accessing slots of fact variables.....	24
4.1.7. Lag modes	24
4.1.8. Slot compare operators	26
4.1.9. Mathematical expressions	27
4.1.10. Fact names.....	29
4.1.11. Output writing facility	31
4.1.11.1. Log writer	32
4.1.11.2. Output writer.....	33
4.2. LHS CLAUSES.....	34
4.2.1. Fact base retrieval.....	34
4.2.2. Fact base queries	36
4.2.3. Exists	37
4.2.4. Composite of inner LHS clauses	38
4.2.5. Not operator	39
4.2.6. Foreach operator.....	39
4.2.7. Bind operator.....	40
4.2.8. Comparative operators.....	40
4.2.9. List operators.....	41
4.2.10. Set operators.....	42
4.2.11. Accumulator	43



4.2.12. Deftype components	44
4.2.13. Symbol generator.....	45
4.2.14. Print	46
4.2.15. Call	46
4.2.16. Agent birth and death.....	47
4.2.17. Breakpoint	47
4.3. RHS CLAUSES.....	48
4.3.1. Fact assertion.....	48
4.3.2. Fact retraction.....	48
4.3.3. Output writing	49
4.3.4. Print	50
4.3.5. Call	50
4.3.6. Agent birth and death.....	51
4.3.7. Breakpoint.....	51
5. REFERENCES	52
6. ANNEXES.....	53
6.1. DRAMS SYNTAX KEYWORDS	53
6.2. FREQUENTLY ASKED QUESTIONS	55
6.2.1. How can output writers be used?.....	55
6.2.2. What should be noted when using the DRAMS Java API?	56



1. INTRODUCTION

This user manual describes the usage of tools included into the *Simulation Environment* module of the integrated OCOPOMO toolkit. These tools provide a means for construction of declarative agent-based policy models, their encoding, debugging, and deployment to the executable simulation environment.

DRAMS, the *Declarative Rule-based Agent Modelling System*, provides the necessary rule engine functionality to enable modellers in the OCOPOMO project to develop declarative agent-based simulation models as discussed in [1].

While this document covers aspects of using DRAMS and the DRAMS language, information about employing DRAMS with agent-based simulation tools, API details or adding new features to the DRAMS core is given in a separate system documentation provided in D4.2-SD-3 *System Documentation of DRAMS*.

Structure of the here presented user manual is as follows:

- Section 2 gives an overview on features and peculiarities of DRAMS. In particular the subsection 2.2 provides useful information about the implemented scheduling algorithm.
- Section 3 is aimed at giving model developers a reference on all important topics. This includes instructions for installing DRAMS, for integrating DRAMS in java-based models and, finally, for using DRAMS.
- Section 4 provides a syntax description for the declarative language.

Other information related to the installation, maintenance, connection to the whole OCOPOMO platform, and technical details of the here-presented tools of the *Simulation Environment* module can be found in the main text of the D4.2 deliverable, as well as in D4.2-SD-3 *System Documentation of DRAMS*.

2. DRAMS OVERVIEW

2.1. FUNCTIONALITY OUTLINE

A rule engine is a software system that basically consists of:

- A fact base, which stores information about the state of the world in the form of facts. A fact contains a number of definable data slots and some administrative information (time of creation, entity that created the fact, durability).
- A rule base, which stores rules describing how to process certain facts stored in fact bases. A rule consists of a condition part (called left-hand side, LHS) and an action part (called right-hand side, RHS).
- An inference engine, which controls the inference process by selecting and processing the rules which can fire on the basis of certain conditions. This can be done in a forward-chaining manner (i.e. trying to draw conclusions from a given fact constellation) or backward-chaining manner (i.e. trying to find the facts causing a given result).

DRAMS is designed as a distributed, forward-chaining rule engine. It equips an arbitrary number of agent types with type-specific rule bases and initial fact base configurations. For each agent type, an arbitrary number of agent instances (objects) with individual fact bases can be created. All individual fact bases are initialized according to the agent type configuration, but may be adapted individually.

There is also a shared global fact base, containing “world facts”, e.g.

- a (permanently updated) fact reflecting the current simulation time,
- one fact for each agent instance present in the “world”, providing some information (e.g. reference ID) about the agent,
- model-specific environmental data, and
- (public) inter-agent communication messages.

Heart of the inference engine is the rule schedule, an algorithm deciding which rules to evaluate and fire at each point of time. The pseudo code in Figure 1 shows the basic structure of a possible rule schedule algorithm. In order to decide (for each fact base configuration without recompiling the rule base) which rules to evaluate for which agent instances, the scheduler relies on a data-rule dependency graph. This is constructed once from all specified rules and initially available data; the graph does not change unless rule bases are modified. As to detecting fact base modifications, the schedule keeps track of all (writing) fact base operations. A more detailed description of this method is given in the following section.

At each point of time, the rule processing within an agent (intra-agent process) is performed for all possible rules. At first, the conditions of a rule are checked, i.e. the LHS is evaluated. Each LHS consists of one or many (LHS) clauses, pertaining to the following basic categories:

- Clauses for retrieving data from fact bases. These operations are similar to data base queries, where as a result a set of facts (with 0, 1 or many elements) is retrieved. Each retrieved fact is called an instance of this clause, and all subsequent clauses of the LHS have to be evaluated for all instances. Thus, this clause type is spanning an evaluation tree. If one or more facts are retrieved, the evaluation result for this clause is true, otherwise false. In the latter case, the evaluation of this tree branch is terminated.

- Clauses which test whether data from the retrieved facts hold for specified conditions. If such a test fails, the evaluation of this tree branch is terminated.

The set of leaves of the evaluation tree is considered a set of possible input data configurations for firing the RHS of the rule. The RHS consists of one or many (RHS) clauses with the purpose of executing fact base operations (adding or removing a fact) or other actions (e.g. printing a statement to a log).

```
processSchedule(time t){  
  
    while new facts are available at time t  
    loop  
  
        find all agent instances for which new facts are available;  
  
        foreach agent instance  
        loop  
            find all rules for which new facts are available at time t;  
  
            foreach rule  
            loop  
                evaluate LHS;  
  
                if evaluation result==true then  
                    execute RHS;  
                    // e.g. generate new facts  
                end if;  
  
            end loop;  
  
        end loop;  
  
    end loop;  
  
end loop;  
}
```

Figure 1: The schedule algorithm

Accordingly, the expressiveness of the system is determined mainly by the number and capabilities of available clause types. In DRAMS, the following functionality is available for the LHS of a rule:

- Data from fact bases can be obtained either by retrieve or by query clauses. In both cases, a query on (in principle any existing) fact base is performed, and a number of facts, for which the slot values specified in the query match and optional time-related conditions hold, are returned. In the case of retrieve clause, these facts are used to create a corresponding number of instantiations, and for each instance a number of requested variables are bound with specified slots. In the case of query clause, only one instance is created, and a list of retrieved facts is bound to a result variable.
These two clause types are representative for the first category defined above, whereas the following LHS clause types belong to the second category.
- A unary BIND operator, which binds a specified variable with the evaluation result of an expression. The expression can be another single (already bound) variable, or a more complex arithmetic expression (with a standard repertoire of math operations, including generation of random numbers).



- A full set of binary logical operators, where the operators can be any expressions allowed for the BIND operator. The result is either bound to another variable or, alternatively, used as clause evaluation result.
- A set of LIST operators, including generation and modification of lists, counting and extracting of list elements, and several accumulator operations (sum, avg, min, max etc.).
- A set of SET operators, including creation and modification of sets, number or existence of elements, as well as union and intersection of two sets.
- A NOT clause, inverting the evaluation result of the specified inner clause. The inner clause can be any other LHS clause.
- A COMPOSITE clause, which can be seen as an encapsulated LHS with its own variable name space. For processing the specified inner clauses, the evaluation mode can be chosen between AND, OR and XOR.

RHS clauses dedicated to perform actions comprise:

- Asserting new facts to (in principle any existing) fact bases.
- With some restrictions, retracting existing facts from (in principle any existing) fact bases.

There is one special clause defined which can be part of both the LHS and RHS, providing two kinds of actions:

- printing formatted text (including values of variables) to a log (either to a console window or to a file);
- calling a method on the underlying model part; the peculiarity of this functionality is explained in more detail in the following section.

An important aspect of any multi-agent simulation system concerns the means by which agents can communicate with each other (inter-agent process). Technically, DRAMS provides three options:

- communication via the global fact base in a blackboard-like manner;
- writing facts to fact bases of other (remote) agents; this can be interpreted as the way humans typically communicate with each other, via speech or written messages;
- reading facts from fact bases of remote agents; this is conceptually similar to mind-reading, and should thus be avoided in most cases, but can be useful to find out (public) properties of another agent.

To facilitate creating a model using any of the described features of DRAMS, an appropriate user interface ought to be available. This user interface must provide adequate access to the rule engines, and furthermore it should support the user with creating models. In order to allow flexibility in usage, both as stand-alone tool or as part in an integrated toolbox, several levels of access are regarded. E.g. fact and rule bases can be accessed by:

- instantiation of Fact or Rule classes within the tool source code,
- reading/writing/editing of configuration files using a specific language (in OPS5 style),
- reading/writing of XML-based configuration files (in future versions),
- Interactively defining facts and rules within the GUI (in future versions).

The user interface is discussed in more detail in section 3.2.

2.2. RULE SCHEDULING MECHANISM

Basically, at each point in a simulation run, all rules for which facts queried in at least one of the retrieve or query clauses have been asserted or retracted are scheduled for evaluation. Successful evaluation of a rule's LHS results in the rule being entered into a conflict set of possible rules. All possible rules fire, in an order resolved by the rule scheduler. Firing a rule executes its RHS, which may include assertion of new facts or retraction of existing facts, thus triggering the rule scheduler again. Simulation time advances only after no more rules can be scheduled for evaluation [1].

For the implementation of the rule scheduler a data-driven approach was chosen. Basic assumption for this approach is that dependencies between rules can be expressed by shared data elements (facts), where a rule reading a fact is depending on all rules writing this fact. More precisely, rule y is depending on rule x , if rule y reads (retrieves, queries, tests for existence) facts written (asserted, retracted) by rule x . These dependencies are calculated and written in a directed fact-rule-dependency graph (an example is shown in Figure 2) prior to executing simulation runs. Each time t at which new facts are available, the possible rule execution paths within the fact-rule-dependency graph starting from these available facts can be depicted as a directed evaluation tree (Figure 3 shows the tree for the graph of Figure 2). This tree shows the dependencies between rules at time t . A graph with equal information for the initial fact configuration ($t=0$) can be displayed as rule dependency graph. This evaluation tree is the foundation of the evaluation algorithm of the rule scheduler. The branches of the evaluation tree for time t terminate/end at rules which

- retrieve/query facts asserted/retracted in a time step other than t (lagged fact base retrieval/query, see section 4.1.7 and Table 2),
- assert/retract facts in a future time step (deferred fact base operation).

All rules within each "stage" of the tree are independent from each other and are processed within a so called task. This, together with the fact that an evaluation tree for time t will be completely executed within the single time step t , implies that within each time step several tasks might be processed, evoking something like a temporal substructure within a time step. The flow of time within a simulation run is represented by t , i.e. after processing the evaluation tree for time t , a new evaluation tree might exist for time $t+x$ (where x typically equals 1). If no new tree is available, then the simulation run is terminated.

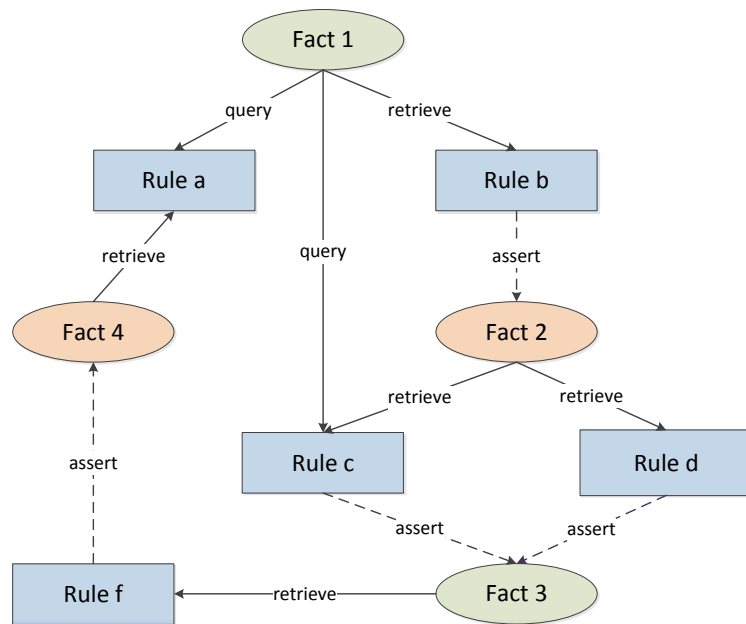


Figure 2: Data-rule-dependency graph (configuration at time t ; legend: green oval - existing fact, red oval - fact asserted by a rule, blue rectangle - rule, solid arc - reading fact base operations, dashed arc - writing fact base operation)

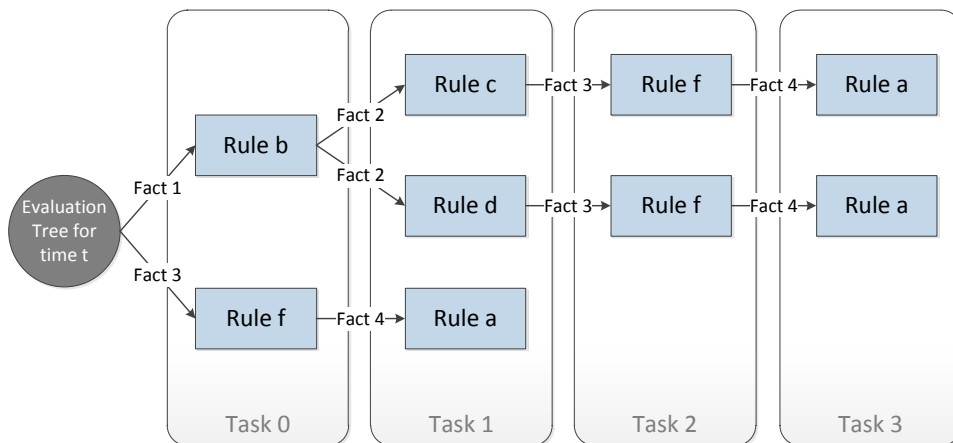


Figure 3: Evaluation tree for time step t

The rule scheduler supports two modes for time advance [1]:

- Time-driven mode (“active time”). In this mode, time advances in regular intervals. The current tick of a simulation run has to be provided by the simulations tool's scheduler (e.g. by the Repast Schedule class), and DRAMS has to be informed about the current tick using the RuleSchedule method processNextTick(Double tick); see Figure 4.

- Event-driven mode (“passive time”, discrete event mode). In this mode the simulation time is set to the next closest event time. Processing of the next event is triggered by the RuleSchedule method processNextTick(), the current simulation time is determined by DRAMS; see Figure 5.

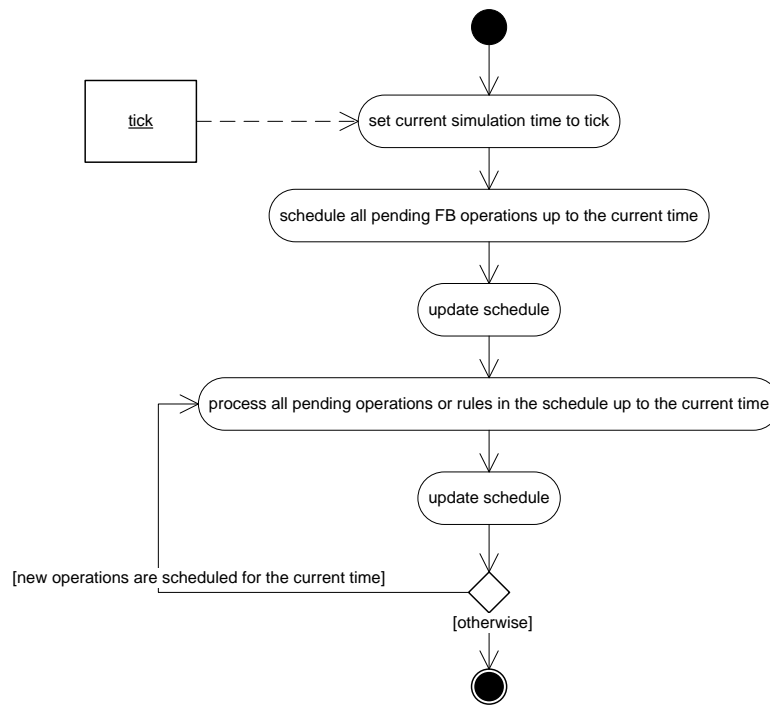


Figure 4: Active time (round based) activity diagram for processNextTick(tick)

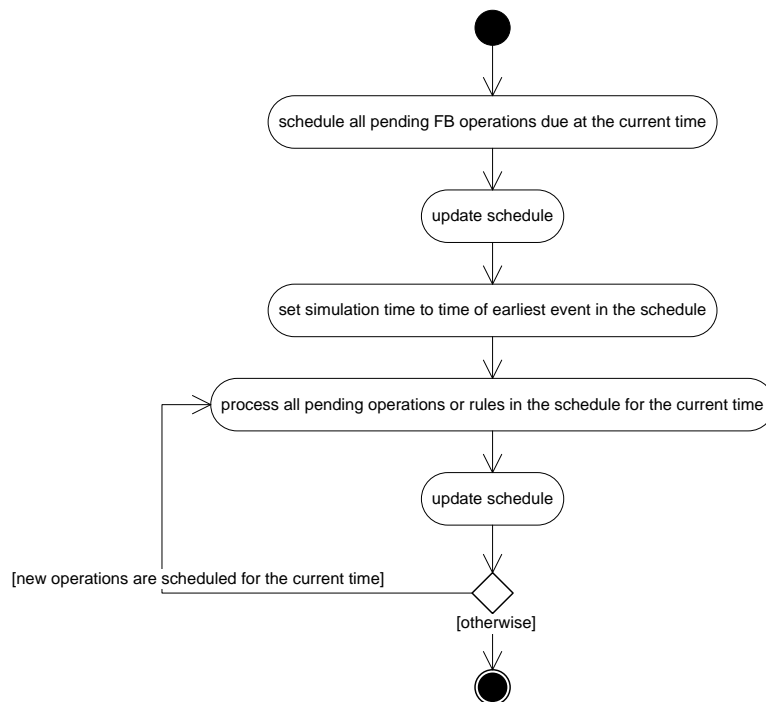


Figure 5: Passive time (discrete event) activity diagram for processNextTick()

2.3. TRACEABILITY AND LINKS

The feature of the OCOPOMO toolbox to preserve links (UUIDs) to CCD elements attached to rules, fact templates and facts throughout simulation runs makes it necessary to let DRAMS be aware of these links. The DRAMS parser is capable to recognise the links for the respective parts of the declarative code. During simulation runs, traces of UUIDs of employed code elements are generated and attached to simulation logs and numerical outcomes.

For each declarative code element, DRAMS tries to find a tag @link in a comment placed directly before that element. If the comment contains more than one @link tags, than the one closest to the element is chosen. The link information (UUID in hexadecimal or Base64 notation) must be written behind the tag (with or without space character(s) in between).

Code example:

```








/* This is a comment...
 * another comment...
 * @link _7G1DAIh4EeGe0_TgxxkLAA
 * and another comment...*/
(defrule Agent::"nice rule" ...)
  
```

3. USING DRAMS

3.1. USER ROLES

User roles for the underlying OCOPOMO process¹, which is supported by the here-presented DRAMS toolkit, were proposed in the D2.1 deliverable [2] as it is presented in Table 1.

Table 1: User roles applied in the OCOPOMO system.

Icon	User Role	Icon	User Role
	Politician		Analyst
	Civil servant		Modeller
	Stakeholder		<i>Administrator</i>
	Facilitator		

DRAMS software, provided within the *Simulation Environment* module of the OCOPOMO toolkit, supports the fourth phase of the OCOPOMO process, i.e., a development of executable policy models based on conceptual descriptions - CCD models provided as outcomes of the previous process phase.

Who uses DRAMS to construct executable models and to run simulations?



The design and iterative development of executable models is an expert task that requires an expertise, and even an experience, in the area of socio-economic modelling. It implies that the *Modeller* is the only user role involved in the highly focused and sophisticated activities related with the development of agent-based models². It also means that this user manual of DRAMS is especially dedicated for *Modellers*.

Who should set up and administer DRAMS?



Since DRAMS software is provided as a set of Eclipse plug-ins, it is installed into a local Eclipse environment (cf. installation instructions in the main D4.2 document), which is typically maintained by the end user - in this case, the *Modeller*. However, an assistance of *Administrator* could be helpful for the system set up, configuration, and technical support during the operation.

¹ <http://www.ocopomo.eu/results/glossary/ocopomo-policy-development-process>; see also Figure 1 in the main D4.2 document.

² The distribution of user roles into the respective phases of the OCOPOMO process belongs to methodological issues and is detailed separately in the D8.1 deliverable [7].

3.2. USER INTERFACE

The current user interface is designed to support the development of model prototypes and debugging of the DRAMS software [3] [4]. A number of SWING based windows can provide several different views:

- Main Window
 - an overall data dependency graph
 - an overall rule dependency graph
 - separate (data) dependency graphs for agent types
 - a trace of the rule schedule while running a simulation
 - an error log
- Console Window
 - an on-the-fly rule processing console with output view
 - fact base dumps with filter and search functionality
- optional a number of output windows
 - text-based log files
 - a model result explorer with traceability visualisation

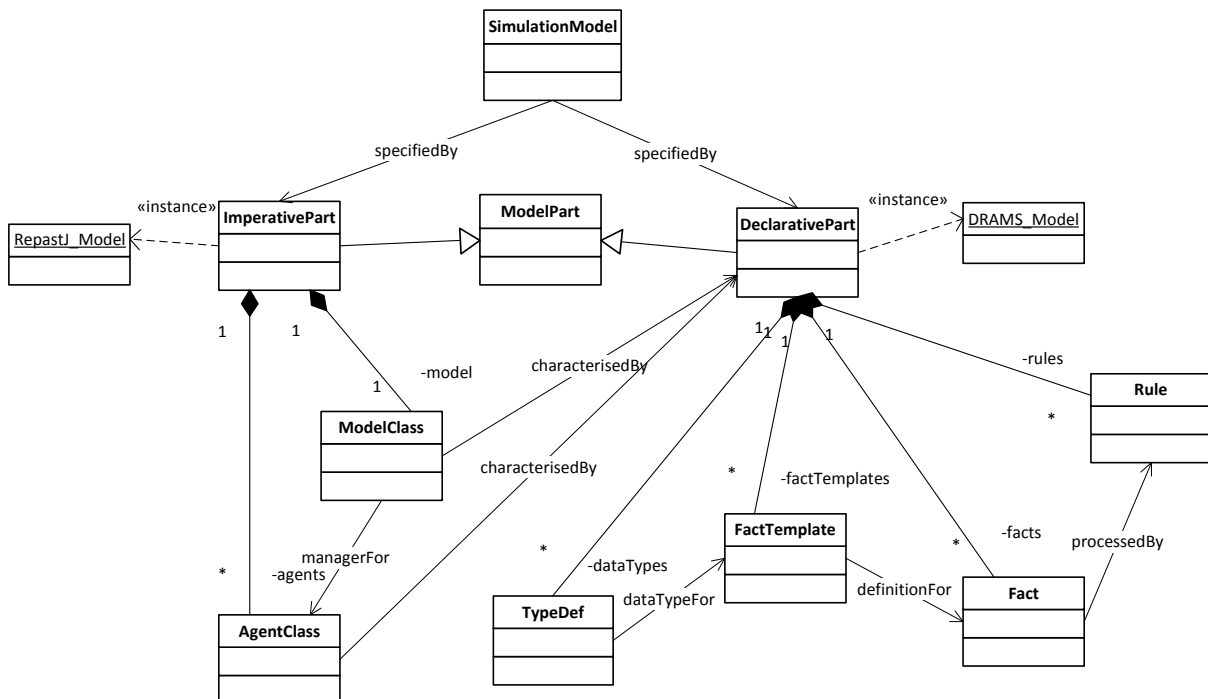


Figure 6: Meta-model for rule-based policy simulation models (preliminary version)

Together with Eclipse-based editing functionality, a fully featured Integrated Development Environment is available as component of the toolbox that supports all facets of handling simulation models, in this case agent-based policy models, in which basically the model structure is represented

as imperative code and the agent dynamics is described by declarative rules. This two-tier model design is reflected in the overview meta-model (Figure 6).

As for simulation models considered in OCOPOMO, the imperative parts are implemented by Java-based Repast models (RepastJ 3.1), while for the declarative parts the DRAMS rule engine (with its own OPS5-like language) is used. The model design determines the properties of the IDE in many ways and at any stage of use. For the editing part, it provides

- Eclipse built-in environment for editing Java code of Repast models, and
- Eclipse feature for editing declarative rules of the DRAMS model.

For the debugging part, it provides

- debugging and code inspection functionalities for Java code, and
- features for integrated debugging of Repast/DRAMS models, e.g. running the simulation model step by step, to set breakpoints (halt clause) at any point within any rule and to display the internal state of rule evaluation (visualisation of the evaluation tree) after reaching a breakpoint.

For the simulation execution part, it provides

- means to execute models within the Repast/Eclipse environment,
- a data collector (output writer) for outcome data of simulation runs.

3.2.1. Main Window

Figure 7 shows the structure of the DRAMS main window. The following menu entries are available:

- File:
 - save PNG image of the currently shown diagram
 - exit closes the simulation model execution
- View:
 - visualise cycles in the (directed) DDGs
 - enable or disable the printing of error messages in the error console
 - enable or disable the printing of warnings in the error console
 - discard all stored GUI setting (position and size of windows, preferences and code in console window, etc.)

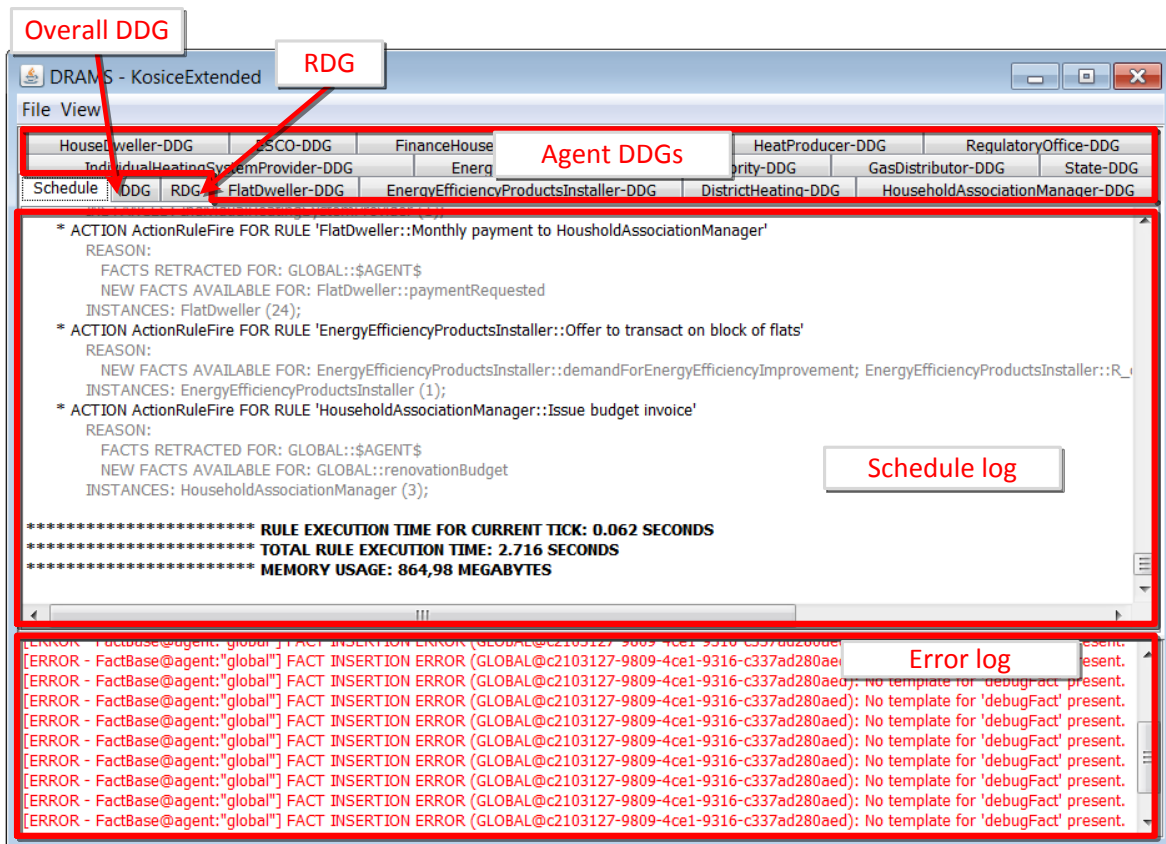


Figure 7: DRAMS main window

3.2.1.1. Data Dependency Graph

The overall *data dependency graph* shows the dependencies between facts and rules (see Figure 8). Facts are displayed as ellipses, containing the fact name prefaced with the fact base owner (agent type or GLOBAL). “Green” facts are available at model initialisation time, whereas “red” facts have to be generated during simulation runs. Rules are displayed as blue boxes, containing the rule name prefaced with the associated agent type (rule base owner).

Blue arcs connect a rule with all facts that are required on the LHS of the rule. A solid arc indicates that a fact is used in a retrieve clause, while a dashed arc indicates the involvement of a query clause. Facts that are asserted by the RHS of a rule are linked by solid green arcs. The number in square brackets represents the deferment time for this assertion (default: 0.0). Solid red arcs denote that the rule retracts the linked fact from its fact base. Possible cycles in the graph can be highlighted on demand [1].

Data dependency graphs (tabs “DDG”) have a filter function. Clicking on any node hides all information but the selected node with all directly related nodes. E.g., for any rule, all incoming and outgoing edges to facts can be more clearly visualised. A click on the background disables the filter.



Figure 8: Example of data dependency graph

3.2.1.2. Rule Dependency Graph

From the data dependency graph DRAMS automatically derives the overall *rule dependency graph*. A rule A depends in its execution on another rule B if it requires a fact F as input, i.e. on its LHS, that rule B produces as its output, i.e. on its RHS. In the example in Figure 9 the rule “compute-total-sales” of agent type Company depends on the rule “sell-to-customer” because it needs facts of type “sold”, which are asserted by “sell-to-customer” (see the DDG in Figure 8) [1].

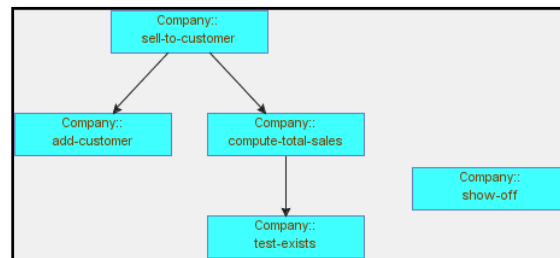


Figure 9: Example of rule dependency graph

3.2.1.3. Trace of Rule Schedule

The schedule log (tab "Schedule", see Figure 7) is shown only for the current tick, not for the entire simulation run. This is mainly due to optimising the execution speed and minimising the memory usage.

Some performance figures are added to the schedule log:

- Rule execution time for current tick. In contradiction to the figures presented for "time to run" in the simulation log (which reflect the time taken by all actions that have been carried out within the tick, including e.g. updating the user interface and writing data to files), the "pure" rule execution time is shown here.
- Total rule execution time

- Memory usage

If processing of a rule takes more than one second, a warning message "### PROFILER INFORMATION ###..." is printed to the error console).

3.2.2. Console Window

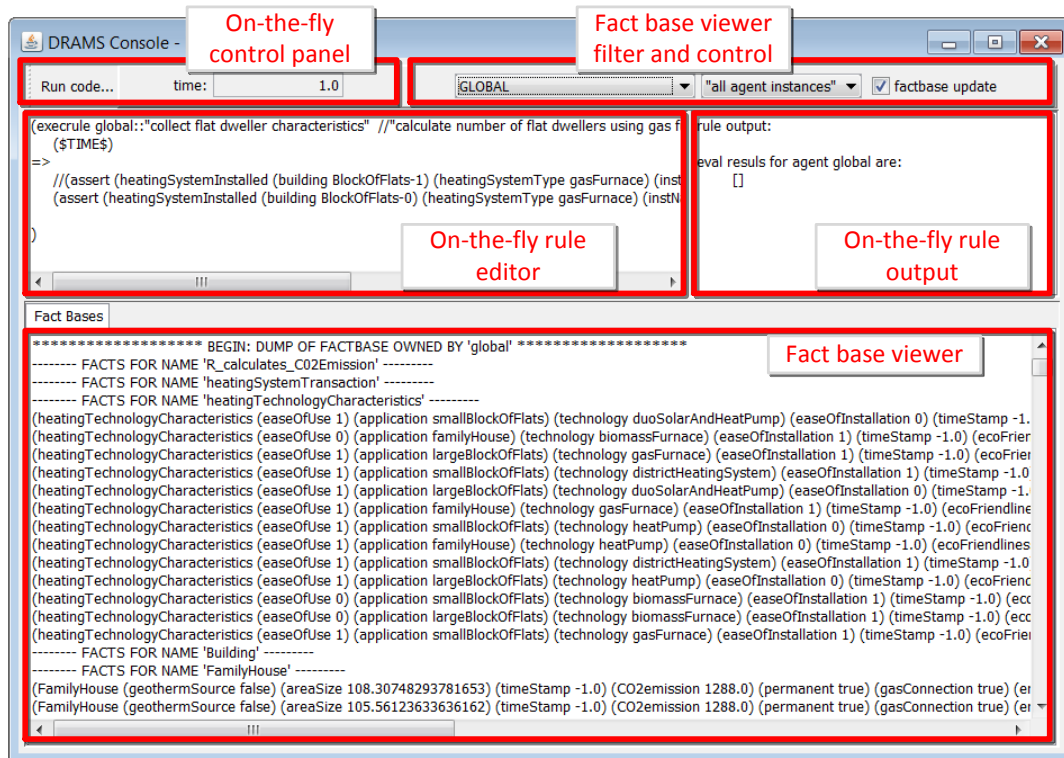


Figure 10: DRAMS console for on-the-fly rule processing and debugging

Figure 11 shows the on-the-fly rule processing console with, mainly dedicated for rule debugging. The following topics should be considered:

- The rule written in the on-the-fly rule editor has similar syntax to a regular rule defined in a .drams file, except the keyword which must be `execrule` (instead of `defrule`).
- The current simulation time is valid also for rules executed in the console. This is expressed by a text field indicating the current simulation time.
- Fact assertions from rules executed within the console become effective immediately. The fact base status is shown in the fact base inspector (tab "Fact Bases" at the bottom of the window).
- A filter for the fact base inspector is available which allows to select an agent type and/or a concrete agent instance for inspection.
- If the checkbox "factbase update" is activated, the fact base inspector is updated every time the simulation time changes.

3.2.3. Output Writer Windows

Different output window types are available for displaying an information generated during simulation runs. It is possible to define an arbitrary number of output windows for any simulation model.

3.2.3.1. Log output

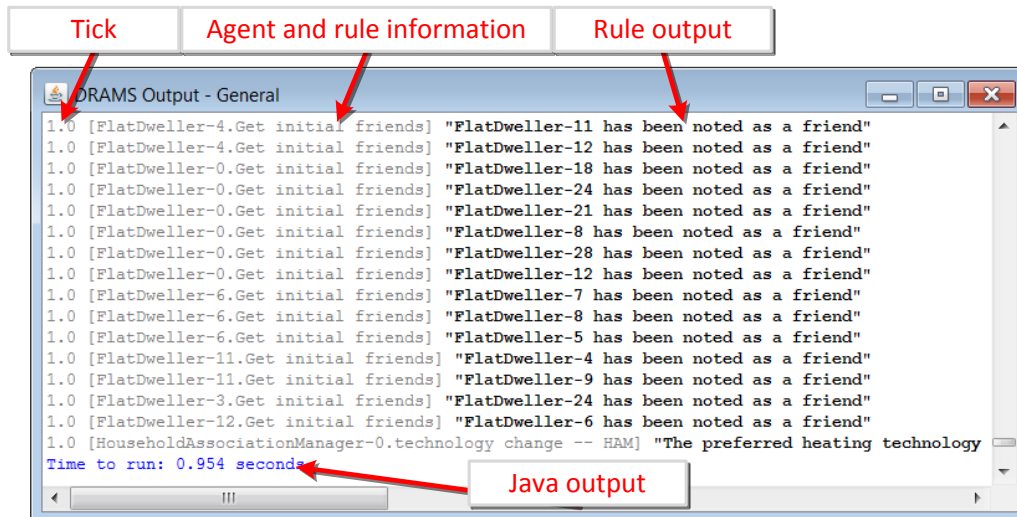


Figure 11: DRAMS output window for simulation logs

3.2.3.2. Model Explorer

The Model Explorer plugin can be used for visualisation of traceability information and model debugging (see Figure 12). For detailed information see [5] and [6].

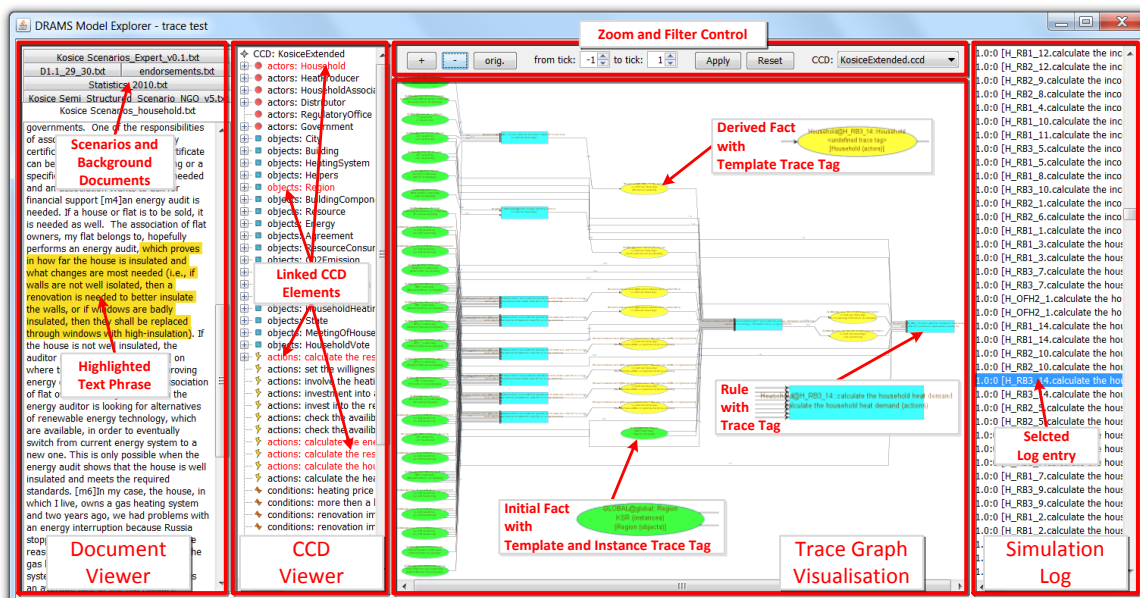


Figure 12: DRAMS Model Explorer window

3.3. INTEGRATION IN REPAST MODELS

As for typical simulation models (considered in OCOPOMO), the imperative parts are implemented by Java-based Repast models (RepastJ 3.1), while for the declarative parts the DRAMS rule engine (with its own OPS5-like language) is used. A class diagram for such a model is shown in Figure 13; it also shows the dependencies between the different components/packages involved (RepastJ 3.1, DRAMS, DRAMS Platform and the concrete model, which is shown exemplarily in the diagram).

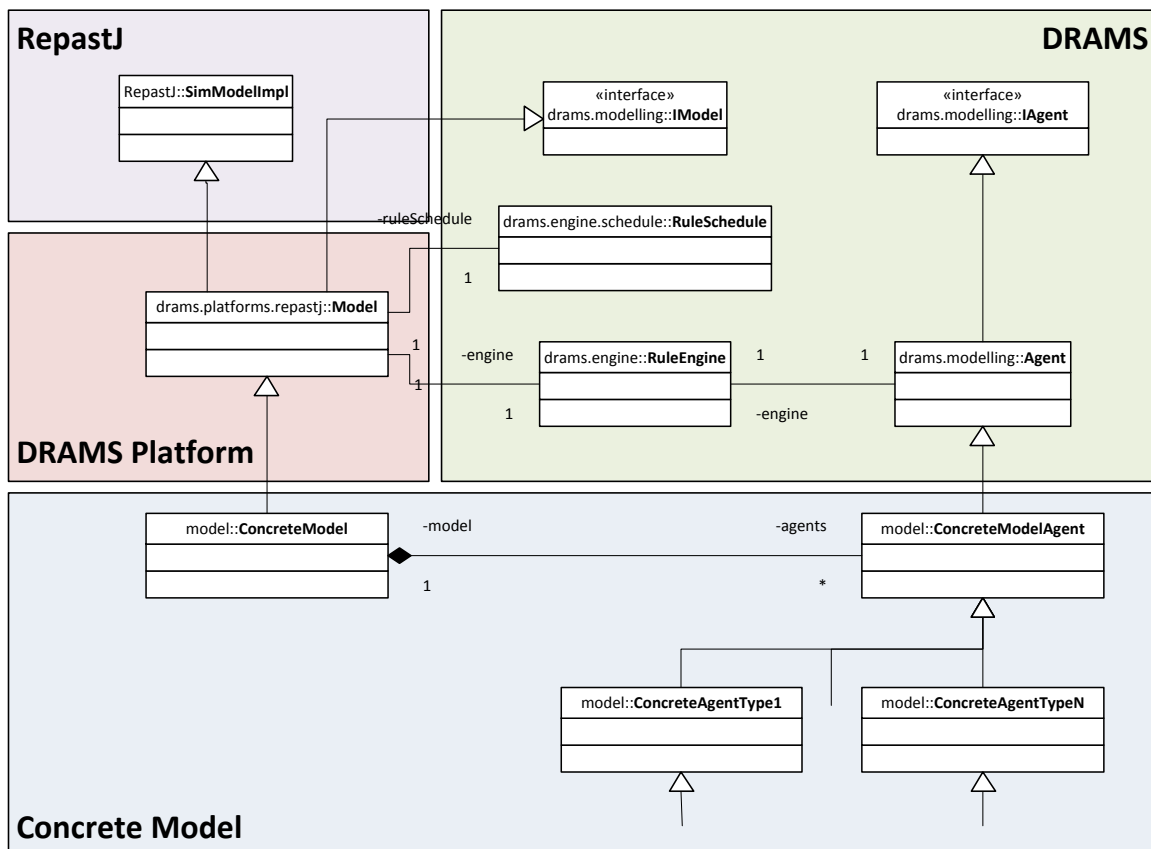


Figure 13: Example class diagram of a RepastJ-based DRAMS simulation model



4. DRAMS LANGUAGE SYNTAX

This section gives an overview on the DRAMS language implemented for specifying declarative model parts. This language is based on OPS5 style, and to some extent similar to languages used for other rule engines, e.g. JESS.

The syntax definitions presented in the following subsections are written in a EBNF like notation with the following constructs:

- `<symbol>`: symbols are put in arrow brackets
- `<charSeq>`: symbols written in italics define an arbitrary sequence of characters, e.g. names (strings) or numbers (usually Java types `int` or `double`). Strings can be written without quotes, if containing only characters allowed for Java identifier or one of the following special characters : `# + - ~ | $ & * / ^ % $`. Otherwise they have to be put in double quotes (").
- **terminal**: terminals are written in bold (without quotes)
- `(alternative1|alternative2)`: alternative symbols or terminals are put in parentheses and separated by a divider line
- `[option]`: optional symbols or terminals are written in square brackets
- `repeated*`: constructs repeated zero or more times are marked with an asterisk
- `repeated+`: constructs repeated zero or more times are marked with a plus sign

4.1. STRUCTURE OF DEFINITION FILE

The declarative code is specified within one or more text files (usually with the file extension ".drams") which have to be delivered to the RuleEngineManager (method `addDeclarativeCode()`).

Such a file contains arbitrary numbers of definitions for enumerated data types, fact templates, facts and rules, which can be written in any arbitrary order. All defined items (except data type definitions) have to be assigned to rule engines of distinct agent classes or to the global rule engine; the class hierarchy of the Java model is considered, i.e. if an item is assigned to an agent superclass, then this item will be present in rule engines of all subclasses. The RuleEngineManager takes care for the correct code distribution.

```
(<typeDefinition>|<factTemplate>|<fact>|<rule>)*
```

The following subsections describe the four possible item types.

4.1.1. Type definitions

With type definitions, custom enumeration data types can be specified, i.e. data types for which a set of discrete user-specified values are defined. Values can be strings or numbers.

```
(deftype <typeName> [<value> [, <value>]*])
```

Example:

```
(deftype HousingType [market, affordable])
```



4.1.2. Fact templates

Fact templates define signatures of facts, i.e. the name and structure of facts allowed to be stored in fact bases. A fact template definition consists of the name of the target agent type (agent class name or **global** for the global rule engine), the fact name and a number of data slots with the related data types. Apart from custom enumeration types (**deftype**), the following data types are allowed:

- all data types defined in package java.lang, e.g. **Integer**, **Double**, **String**
- all data types defined in package java.util, e.g. **UUID**
- general objects (**Object**)
- **Fact** and **Fact[]** can be used when regular facts or shadow facts have to be stored
- **IFact** and **IFact[]** ditto, but also for **SimpleShadowFacts** (e.g. **global::\$TIME\$**)

```
(deftemplate <targetAgentType>::<factName> [ (<slotName>:<dataType>)]*)
```

Example:

```
(deftemplate global::planningProposalUnitCost  
  (type:HousingType) (value:Double) (uniq:UUID))
```

4.1.3. Facts

Using fact definitions, fact bases can be endowed with concrete data. There are two syntax constructs available, one for asserting a single fact, and another for asserting multiple instances for a given fact name. In both cases, the target fact base(s) has/have to be specified (fact name, agent type and optionally agent instance; agent instances have to be specified only by name, the rule engine UUID can't be used because it won't be available at the time the facts are inserted), followed by a concrete configuration of the data slots. Optionally, an insertion time and a flag that defines the fact as permanent can be added.

```
(insertfact [permanent] [at <timeStamp>]  
<targetAgentType>[@<instance>]::<factName>  
 [ (<slotName> <value>)]*)  
  
(insertfacts [permanent] [at <timeStamp>]  
<targetAgentType>[@<instance>]::<factName>  
 [( [ (<slotName> <value>)]* ) ]*)
```

Examples:

```
(insertfact global::planningProposalUnitCost (type market)(value 10000.0))  
  
(insertfacts global::planningProposalUnitCost  
  ((type market)(value 10000.0)  
  (type affordable)(value 7000.0) )  
  
(insertfact Agent@"agent-1"::someFact (value 672))
```




4.1.4. Rules

Rule definitions basically consist of a rule identifier, a condition (or left-hand side, LHS) part and an action (or right-hand side, RHS) part. The identifier contains (similar to fact template definitions) a rule name and a target agent type (an agent instance definition is not allowed). Both the LHS and the RHS are composed of clauses; the available clause definitions are described in the following subsections.

```
(defrule <targetAgentType>::<ruleName>
  <LHSClause>+
=>
  <RHSClause>*
)
```

An experimental feature allows to associate rule to rule groups. There are two groups available:

XOR group (syntax token: +): at most one of the rules in the group can fire within a time step (the rule which meets the LHS condition first), no matter whether the LHS of some of the other rules become true in a later task of the time step.

```
(+ <defRule>+)
```

AND group (syntax token *): the RHSs of all rules within the group fire jointly only if the LHSs of all rules become true.

```
(* <defRule>+)
```

4.1.5. Clauses

The clauses available for the LHS and RHS sides are described in the sections 4.2 and 4.3, respectively. The following symbols are used within these clause definitions:

<variable> ::= ?<variableName> defines a variable.

<slot> ::= (<slotName> <expression>) defines a data slot for a fact base operation.

<slot_cond> ::=

```
([<compare_operator>] <slotName> (<expression>|<wildcard>)) |
|[<compare_operator>] <slotName1> <slotName2>|
```

defines a fact base condition, i.e. a compare operation between a data slot value and either an expression (or wildcard for certain cases) or another data slot value. For the optional compare operator, see section 4.1.8.

<lag> specifies the fact base query mode for some of the LHS clauses; see section 4.1.7.

<expression> ::= (<value>|<variable>|<mathExpression>) defines either a constant value, a variable or a mathematical expression. Also, lists of value are allowed, see section 4.1.8.

<mathExpression> ::= {<mathematicalExpression>} defines a mathematical expression; see section 4.1.8.

<wildcard> ::= * defines a wildcard slot value for "equal" and "not equal" slot operators.

<LHSClause> is a placeholder for any of the LHS clauses.



`<deferment>` ::= **deferredBy** `<timeValue>` specifies a time value by which a writing fact base operation (assertion or retraction) should be deferred.

`<list>` ::= (`<variable>` | [`<listValue>` [, `<listValue>`]*]) is a placeholder for a variable containing a list or the (constant) definition of a list .

`<set>` ::= (`<variable>` | [`<setValue>` [, `<setValue>`]*]) is a placeholder for a variable containing a set (a list with no repeated values) or the (constant) definition of a set .

`<factDescr>` ::= [`<targetAgentType>`[`@<instance>`]]:`<factName>` specifies the name of a fact, optionally combined with a fact base description (agent type and optionally instance) where the fact is/will be stored. For accessing the agent's local fact base, only `<factName>` must be specified; for remote fact base access reasonable combinations of both `<targetAgentType>` (as constant string) and `<instance>` must be provided. The global fact base is accessible with the keyword **global** as `<targetAgentType>`; in this case, an instance must not be specified.

`<factName>` ::= (`<customFactName>` | `$TIME$` | `$AGENT$` | `$SELF$`) is a placeholder for any custom fact name (a string with characters allowed for Java identifiers) or the name of one of the predefined (pseudo) facts; see section 4.1.10.

`<factList>` is a special type of list containing facts. Such lists are retrieved by query clauses, and processed by accumulator clauses.

`<printString>` is a string which may contain variable identifiers; the content of the variables is resolved and printed during runtime.

4.1.6. Accessing slots of fact variables

For variables bound to facts the slot values can be accessed with the following syntax: `?factVariable.slotValue`. This construct can be used in any kind of clauses where variables are allowed, also in expressions.

Code example:

```
?a <- (gLoBaL::$AGENT$ (type ["Employer", "Employee"]))  
=>  
(print "name of company member: ?a.name")
```

4.1.7. Lag modes

Lag modes are used to specify the temporal conditions of facts in LHS clauses which involve fact base queries (retrieve, query and exists clause). While by default only facts are retrieved which have been asserted at the current time step, lag modes allow to retrieve fact asserted at any other time. Table 2 gives an overview on the available lag modes.

```
<lag> ::= (  
  all |  
  last |  
  latest |  
  at <timeValue> |  
  before <timeValue> |  
  after <timeValue> |
```



```
(lag|relAt) <timeValue> |
(lagBefore|relBefore) <timeValue> |
(lagAfter|relAfter) <timeValue> |
)
```

Table 2: Lag mode specifications

Lag mode	Lag value	Behaviour at time t	Time for which rule evaluation is scheduled, if fact f asserted or retracted ³ at time t_a	Facts retrieved at time t_a for non-permanent data	Facts retrieved at time t_a for permanent data	Remark
	Insignificant	Retrieve only facts which have been asserted at current tick t	Next task within currently active time event	Facts asserted in the previous task ⁴ of t_a	All available facts	Default mode, used when no lag mode is specified
all	Insignificant	Retrieve all available facts	Next task within currently active time event	All available facts	All available facts	
at	Absolute time t_{abs}	Retrieve only facts which have been asserted at tick t_{abs}	Next task within currently active time event, if $t_{abs} == t_a$	Facts asserted at tick t_{abs}	All facts valid at t_{abs} (i.e. all facts asserted before or at t_{abs})	Currently works only if lag value is specified by constant
before	Absolute time t_{abs}	Retrieve only facts which have been asserted before tick t_{abs}	Next task within currently active time event, if $t_{abs} > t_a$	Facts asserted prior to t_{abs}	All facts valid prior to t_{abs} (i.e. all facts asserted before t_{abs})	Currently works only if lag value is specified by constant
after	Absolute time t_{abs}	Retrieve only facts which have been asserted after tick t_{abs}	Next task within currently active time event, if $t_{abs} < t_a$	Facts asserted later than t_{abs}	All available facts	Currently works only if lag value is specified by constant
relAt	Relative time t_{rel}	Retrieve only facts which have been asserted at tick $(t - t_{rel})$	First task of time event at $(t_a + t_{rel})$	Facts asserted at $(t_a - t_{rel})$	All facts valid at $(t_a - t_{rel})$ (i.e. all facts asserted before or at $(t_a - t_{rel})$)	Currently works only if lag value is specified by constant
lagBefore	Relative time t_{rel}	Retrieve only facts which have been asserted before tick $(t - t_{rel})$	First task of time event at $(t_a + t_{rel} + 1)$	Facts asserted before $(t_a - t_{rel})$	All facts valid prior to $(t_a - t_{rel})$ (i.e. all facts asserted before $(t_a - t_{rel})$)	Currently works only if lag value is specified by constant
lagAfter	Relative time t_{rel}	Retrieve only facts which have been asserted after tick $(t - t_{rel})$	Next task within currently active time event	Facts asserted after $(t_a - t_{rel})$	All available facts	Currently works only if lag value is specified by constant

³ only for not-clauses; in this case, these rules are candidates for being scheduled at the first tick, too

⁴ or initially available facts, if $t_a = 0$

last	Insignificant	Retrieve only facts which have been asserted at previous tick ($t - 1$)	First task of time event at ($t_0 + 1$)	Facts asserted at ($t_0 - 1$)	All facts valid at ($t_0 - 1$) (i.e. all facts asserted before or at ($t_0 - 1$))	
latest	Insignificant	Retrieve only facts with the highest available timestamp	Next task within currently active time event	Facts with most recent timestamp	All available facts	

4.1.8. Slot compare operators

A set of compare operators has been introduced to increase the expressiveness of pattern for unifying facts to clauses and to reduce the number of clauses involving fact base operations.

Usually, a clause specifies a number of slots for which the given value must be equal in the fact. In the extended syntax, an additional operator can be specified for e.g. unifying facts with "not equal" values, a specified order or intervals. Table 3 gives an overview on all available operators and the different effect when the clause specifies a single value or a list of values. The operators can be applied for any kind of comparable content, e.g. numbers and strings. Note that the equal ($==$) and not equal ($!=$) operators usually will be calculated more efficiently than the other operators.

Table 3: Slot compare operators

Syntax	Result for single value	Result for value list
(slotName ?value)	slot value equal to ?value	slot value equal to one of the values in the list defined by ?value
(== slotName ?value)	slot value equal to ?value	slot value equal to one of the values in the list defined by ?value
(!= slotName ?value)	slot value not equal to ?value	slot value equal to none of the values in the list defined by ?value
(< slotName ?value)	slot value less than ?value	slot value less than the lowest value in the list defined by ?value
(> slotName ?value)	slot value greater than ?value	slot value greater than the highest value in the list defined by ?value
(<= slotName ?value)	slot value less than or equal to ?value	slot value less than or equal to the lowest value in the list defined by ?value
(>= slotName ?value)	slot value greater than or equal to ?value	slot value greater than or equal to the highest value in the list defined by ?value
(<> slotName ?value)	Not defined - always false	within the open interval defined by the lowest and the highest values in the list defined by ?value
(<=> slotName ?value)	slot value equal to ?value	within the closed interval defined by the lowest and the highest values in the list defined by ?value
(!<> slotName ?value)	Not defined - always true	outside the open interval defined by the lowest and the highest values in the list defined by ?value
(!<=> slotName ?value)	slot value not equal to ?value	outside the closed interval defined by the lowest and the highest values in the list defined by ?value



4.1.9. Mathematical expressions

Mathematical expressions are written in curly brackets. Operands can be constants or variables (see <variable>). Beyond the basic arithmetic operations, the following constants and functions are (currently) available⁵:

Predefined Constants:

- **E** - The double value that is closer than any other to e, the base of the natural logarithms (2.718281828459045).
- **Euler** - Euler's Constant (0.577215664901533).
- **LN2** - Log of 2 base e (0.693147180559945).
- **LN10** - Log of 10 base e (2.302585092994046).
- **LOG2E** - Log of e base 2 (1.442695040888963).
- **LOG10E** - Log of e base 10 (0.434294481903252).
- **PHI** - The golden ratio (1.618033988749895).
- **PI** - The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter (3.141592653589793).

Supported Functions:

- **abs** (<value>) - The absolute value of a double value.
- **acos** (<value>) - The arc cosine of a value; the returned angle is in the range 0.0 through pi.
- **asin** (<value>) - The arc sine of a value; the returned angle is in the range -pi/2 through pi/2.
- **atan** (<value>) - The arc tangent of a value; the returned angle is in the range -pi/2 through pi/2.
- **cbrt** (<value>) - The cube root of val.
- **ceil** (<value>) - The smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
- **cos** (<value>) - The trigonometric cosine of an angle.
- **cosh** (<value>) - The hyperbolic cosine of val.
- **exp** (<value>) - Euler's number e raised to the power of a double value.
- **expm1** (<value>) - $e^x - 1$.

⁵ all static methods from `java.lang.Math` can be used



- **floor**(`<value>`) - The largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.
- **getExponent**(`<value>`) - The unbiased exponent used in the representation of val.
- **log**(`<value>`) - The natural logarithm (base e) of a double value.
- **log10**(`<value>`) - The base 10 logarithm of val.
- **log1p**(`<value>`) - The natural logarithm of (val+1).
- **max**(`<value1>`, `<value2>`) - maximum operator, also for lists of Integers and Doubles.
- **min**(`<value1>`, `<value2>`) - minimum operator, also for lists of Integers and Doubles.
- **nextUp**(`<value>`) - The floating-point value adjacent to val in the direction of positive infinity.
- **random**() - A double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
- **round**(`<value>`) - The closest 64 bit integer to the argument.
- **roundHE**(`<value>`) - The double value that is closest in value to the argument and is equal to a mathematical integer, using the half-even rounding method.
- **round**(`<value>`, `<number_of_decimal_places>`) - Round function, where the number of decimal places can be specified.
- **signum**(`<value>`) - The signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
- **sin**(`<value>`) - The trigonometric sine of an angle.
- **sinh**(`<value>`) - The hyperbolic sine of a double value.
- **sqrt**(`<value>`) - The correctly rounded positive square root of a double value.
- **tan**(`<value>`) - The trigonometric tangent of an angle.
- **tanh**(`<value>`) - The hyperbolic tangent of a double value.
- **toDegrees**(`<value>`) - Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
- **toRadians**(`<value>`) - Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
- **ulp**(`<value>`) - The size of an ulp of the argument.

For optimal execution speed, expressions are compiled into Java class files and dynamically loaded and bound into the rule code. However, there are a number of issues that have to be taken into account by modellers:

- A Java compiler (usually the JDK) must be available for all model projects that use expressions. See section **Chyba! Nenalezen zdroj odkazů.** for details.

- A general problem for the expression evaluation algorithm is that DRAMS can determine the type of any variable only at runtime, which makes it almost impossible to generate fairly efficient code for evaluating expressions when it should be allowed to flexible use variables containing data of any type.
The compromise chosen for DRAMS expressions is that all variables in expressions are treated as Doubles **unless** another data type is specified.
For example, the function `min(list_of_values)` for finding the minimum value in a list of values which is stored in variable `?list` must be written as
 - `{min(?list:Double[])}`, if `?list` contains a list of Doubles, or
 - `{min(?list:Integer[])}`, if `?list` contains a list of Integers.
- As expressions are translated into Java code and afterwards are compiled into Java class files, the usage of arbitrary Java code in expressions is possible in principle; e.g., all static methods of `java.lang.Math` are available by default. The usage of other libraries or local code might increase the power of these expressions, but also brings considerable risks. Here is a limited selection of examples what would be valid expressions:
 - `?value <- (is {min(?list:Double[])})` - the complete example from above
 - `?value <- (is {round(floor(4.6))})` - nested functions
 - `?value <- (is {(int)floor(4.6)})` - typecast
 - `?value <- (is {uchicago.src.sim.util.Random.uniform.nextDoubleFromTo(1.0, 4.0)})` - access to external libraries; of course this example only works if RepastJ is available in the model classpath
 - `?value <- (is {?v >= 0 ? 1 : 0})` - if variable `?v` is positive or zero then the expression result is 1, otherwise 0 (result type will be Integer)
 - `?value <- (is {(?self.name:String).length()})` - determining the length of the string stored in the slot "name" of a fact bound to variable "?self"
 - `?value <- (is {(?self.name:String).replace("-", "_")})` - some string operation (type of `?value` will be String)

4.1.10. Fact names

Fact names are used as key strings for accessing data in fact bases. Only characters allowed for Java identifiers should be used for custom fact names. Fact names starting and ending with a "\$" character are reserved for predefined facts. The following predefined facts are defined:

\$TIME\$

Description	This pseudo fact is situated in the global fact base and reflects the current simulation time.
Equivalent fact template	<pre>(deftemplate <i>global::</i>\$TIME\$ (value:Double) (owner:String) (timeStamp:Double))</pre>



Slots	<p>)</p> <ul style="list-style-type: none"> • value: the current simulation time • owner: default slot reflecting the fact owner ("GLOBAL") • timeStamp: default slot reflecting the time of creation ("0.0")
--------------	--

\$AGENT\$

Description	<p>For each created rule engine instance (respective each agent instance) a \$AGENT\$ fact is asserted to the global fact base. It contains information about the agent that (among others) can be used for accessing foreign agents' fact bases. The retraction of this fact for any agent triggers the death process for this agent, i.e. the related rule engine instance is no longer regarded by the rule scheduler.</p>
Equivalent fact template	<pre>(deftemplate <i>global</i>::\$AGENT\$ (name:String) (agent_type:String) (agent:String) (ruleengine_instance:UUID) (owner:String) (timeStamp:Double) (permanent:Boolean))</pre>
Slots	<ul style="list-style-type: none"> • name: the name of the agent instance • agent_type: the agent type (class name of Java agent representation) • ruleengine_instance: the unique identifier (UUID) of the agent instance; this value can be used as <i><instance></i> part of a <i><factDescr></i> (see 4.1.5) in order to access the fact base of this agent instance. Remark: in the current version, the related <i><targetAgentType></i> must be explicitly specified as a constant in order to allow the rule scheduler to work properly. • agent: this string combines the agent_type and ruleengine_instance in a way that can directly be used to identify a foreign agent's fact base. Remark: this functionality is not yet fully implemented and tested. • owner: default slot reflecting the owner of this fact (should have the same content as agent) • timeStamp: default slot reflecting the time of creation (value usually "-1.0") • permanent: default slot specifying the fact as permanent (value "true")

\$SELF\$

Description	<p>For each created rule engine instance (respective each agent instance) a \$SELF\$ fact is asserted to the agent's own fact base. Similar to the \$AGENT\$ fact it contains information about the agent that (among others) can be used for communication purposes.</p>
--------------------	---



Equivalent fact template	<pre>(deftemplate targetAgentType::\$SELF\$ (name:String) (agent_type:String) (agent:String) (ruleengine_instance:UUID) (owner:String) (timeStamp:Double) (permanent:Boolean))</pre>
Slots	<ul style="list-style-type: none"> • name: the name of the agent instance • agent_type: the agent type (class name of Java agent representation) • ruleengine_instance: the unique identifier (UUID) of the agent instance. • agent: this string combines the agent_type and ruleengine_instance in a way that can directly be used to identify a foreign agent's fact base. • owner: default slot reflecting the owner of this fact (should have the same content as agent) • timeStamp: default slot reflecting the time of creation (value usually "-1.0") • permanent: default slot specifying the fact as permanent (value "true")

4.1.11. Output writing facility

A flexible and powerful plugin-based facility for generating various kinds of simulations logs and output files has been implemented in DRAMS. It is planned to use this functionality for all kinds of outputs, including XML or CSV files with numerical data, text or CSV files for simulation logs.

The main concept behind this output writer facility is the specification of an arbitrary set of output writers in terms of deflog or defoutput definitions in .drams files. In consequence, if no such writers are defined, no output will be generated. A reasonable scenario might be to add one or more console windows during model development, and to add output writers as soon as the model should generate (persistent) simulation outcomes. For productive runs, the console windows might be disabled in order to save computation time.

A number of output writers is implemented and integrated as plugins into DRAMS, and can be directly used for writing simulation outcomes (see Table 4). In principle all these writers can be used for both logging and writing of simulation data.

Table 4: Available output writers

Writer ID	Description
graph-xml	XML files for numerical data according to specification of UNISOB (Francesco Poggi)
csv	CSV files for both numerical and textual simulation outcomes
text	Plain text files mainly for text logs
default	Default output stream
console	Console window for writing plain text (simulation logs) with text style



	<p>attributes for different types of outcomes. The meanings of the different colours are:</p> <ul style="list-style-type: none"> • blue: statements written by Java code (via System.out.print*) • red: statements written by Java code (via System.err.print*) • gray: text from DRAMS print clauses - meta information (e.g. tick, agent name, rule name) • black, bold: text from DRAMS print clauses - messages <p>See section 3.2.3.1</p>
tab-console	Single console window, where all specified output consoles are displayed a tabs within the window
explorer	DRAMS Model Explorer; see section 3.2.3.2

Depending on the writer ID, a file containing the log information or output data might be generated. The default location is a folder named "<modelFolder>/data/experiments/" (this can be changed from Java model code with `ResultGenerator.getInstance().setDefaultOutputPath(String defaultOutputPath)`). For each simulation run, a new subfolder with the name "run-<year><month><day><hour><minute><second><millisecond>" is created. The output folder for each writer can easily be changed by adding path information to the writer name. Don't add a file extension to the writer name, this will be done automatically according to the chosen format.

4.1.11.1. Log writer

A log writer processes all messages created by print clauses in DRAMS rules and, optionally, messages written by the Java output streams `System.out` and `System.err` (and probably some other channels in future versions).

Any number of log writers can be added to a simulation model by specifying appropriate definitions in .drams files (on the level of `deftemplate`, `defrule` etc.). The syntax is as follows:

```
(deflog <writer_ID>
  [prio [comp_op] <int_value> [<int_value>]]
  [<options_list>]
  <writer_name>
)
```

with

- <writer_ID> is one of the writer IDs specified in Table 4. Usually the `console`, `text`, `styled-console` and `csv` should be used here. The reason for writing logs into CSV files is that this would enable the "simulation analyst" to comfortably sort and filter information of logs as well as doing searches with standard spreadsheet software.
- <writer_name> is the name of the writer, i.e. the file name under which the log file is stored, or the caption of the output window;
- `[prio [comp_op] <int_value> [<int_value>]]` is an optional field specifying the priority of messages that will be processed by the writer. An optional comparative operator (one of `==`, `!=`, `<`, `<=`, `>`, `>=` together with one `<int_value>`, or `<=>`, `!<=>`, `<>`, `!<>` together with

two `<int_value>` specifying the range borders; no comparator will be treated as `>=`) can be used to define a writer specific for messages with priorities greater than, less than or equal to the specified level. If the priority field is omitted, then the writer will process all messages (i.e. the priority level for the writer is `Integer.MIN_VALUE`).

The priority of messages can be specified at `print` clauses, the default value is 0. In order to change the priority, the syntax definition for `print` has been extended by an optional attribute `prio`:

```
(print [prio <int_value>] "<printString>")
```

- `<options_list>` is an optional list of parameters that defines what (meta) information will be added to the output. By default, all meta information and Java system message are processed, but these can be deactivated for one or many of the following items:
 - `cumulated` - the output is written to one and the same facet definition (column in the data table), independent from the agent instance that produces the output; otherwise a new facet is generated for each agent instance generating data for this facet definition.
 - `no_tick` - simulation time will be omitted;
 - `no_task` - task number within simulation time will be omitted;
 - `no_type` - type of agent producing the message will be omitted;
 - `no_inst` - name of agent producing the message will be omitted;
 - `no_rule` - name of rule producing the message will be omitted;
 - `no_trace` - all trace information will be omitted;
 - `no_value` - only meta information will be printed;
 - `no_sys` - Java system messages will be omitted;
 - `no_err` - Java error messages will be omitted.

Example code for log writer definitions:

- `(deflog console "journal")` - the smallest possible log writer definition for a console similar to the Repast Output window
- `(deflog styled-console prio < 10 [no_task] "journal")`
- `(deflog styled-console prio >= 10 [no_task] "journal-high priority")`
- `(deflog csv [no_err, no_sys] "journal")`
- `(deflog graph-xml prio <=> 10 19 [no_task, no_sys, no_err, cumulated] "generalGraph")` - typical log file definition for (actually numerical) XML output

4.1.11.2. Output writer

Similar to the log writers, an output writer processes data samples provided by two newly introduced clauses: `sample` and `write`.

Any number of output writers can be added to a simulation model by specifying appropriate definitions in `.drams` files (on the level of `deftemplate`, `defrule` etc.). The syntax is as follows:



```
(defoutput <writer_ID>
  [<options_list>]
  <writer_name>
  [(<facet>:<dataType>)]*
)
<facet> := <facetName>[|<facetDescription>] |
  "<facetName>[|<facetDescription>]"
```

with

- <writer_ID> similar to log writer, see 4.1.11.1;
- <writer_name> similar to log writer, see 4.1.11.1;
- <options_list> similar to log writer, see 4.1.11.1;
- <facet> specifies names, optional description and types of data fields that can be part of the output. This information can be regarded as headers of table columns, whereas each data set constitutes a line in the table.
 - <facetName> the short name of the data field;
 - <facetDescription> an optional more narrative description for the data field;
 - <dataType> the data type for the field, e.g. Double or String.

Example code for output writer definitions:

```
(defoutput graph-xml "newTestOutput" [no_task, no_trace]
  ("value1|This is value 1.":Double) (value2:String) (value3:Double))
```

4.2. LHS CLAUSES

The following subsections give an overview on all currently available LHS clauses, grouped by functional relationship.

4.2.1. Fact base retrieval

```
[<variable> <-] (<factDescr> (<slot>|<slot_cond>)*)
```

```
[<variable> <-] (<lag> (<factDescr> (<slot>|<slot_cond>)*))
```

Short description

Retrieve sets of variable assignments from a fact base query matching a specified pattern of values.

Arguments

- <factDescr>: name of the fact to be retrieved, possibly combined with fact base owner description.
- <slot_cond>: pairs of <slotName>/<constant>, <slotName>/<boundVariable> or <slotName>/<mathExpression> specifying the fact base query pattern (i.e. condition)
- <slot>: pairs of <slotName>/<freeVariable> specifying the variable names which are to be assigned with values from the



Returns	specified slot names as result of the fact base query <ul style="list-style-type: none">• Optional <lag>: a lag mode and possibly the related lag value.
Evaluation Result	The optional result variable is bound with the retrieved fact(s). <ul style="list-style-type: none">• True, if one or more facts match the specified pattern• Otherwise false
Description	<p>This operation performs a fact base query, searching for facts that match the specified pattern. The pattern consists of constants and bound variables which are assigned to slot names. All facts with equal values at these slots are retrieved.</p> <p>For each returned fact, a set of variable assignments for the specified free variables is generated. In this set, values from the requested slots are assigned to the corresponding variable names. For each set of variable assignments a new branch of the evaluation tree is generated, i.e. the subsequent clause of the rule is evaluated for each set of variable assignments.</p> <p>If only a fact name is given, than the target of the query is the local fact base (associated with the rule engine that evaluates the actual retrieve clause). Remote fact bases can be accessed by specifying the name (and instance, if required) of any agent or global rule engine.</p> <p>The optional result variable is bound to a single fact, and the evaluation tree for the rule is split into several branches, one for each unified fact. Thus, this construct replaces the following lines of code:</p> <pre>?list <- (query (global::<i>\$AGENT</i>)) ?res <- (each ?list)</pre>
Predefined slot names	<ul style="list-style-type: none">• <code>owner</code>: string, specifying the agent that generated the fact.• <code>permanent</code>: Boolean value, indicating whether the fact is permanent or not. A permanent fact is retrieved in every time step (beginning with the time of assertion).• <code>timeStamp</code>: double value, specifying the time of assertion.
Example	<ul style="list-style-type: none">• the clause <code>(factName (slotName *))</code> will unify all facts that have a value (not null) defined for slot <code>slotName</code>;• the clause <code>(factName (!= slotName *))</code> will unify all facts that have no value defined for slot <code>slotName</code>.• <code>(global::boroughCharacteristics (!= landArea *) (housePrices ?p)(instName ?borough) (innerOrOuterLondon ?location) (residentialEarnings ?re))</code> - unifies all facts with borough characteristics that have no land area assigned yet• The fact base allows also slots for which a list of values is specified. For example it is possible to write the following code in order to let the <code><i>\$SELF</i></code> retrieve clause become true only for agent names "agent-1" or "agent-2":<ul style="list-style-type: none">○ <code>(<i>\$SELF</i> (name ["agent-1", "agent-2"]))</code>



- ```
(ruleengine_instance ?inst))
```
- `?agentList <- (is ["agent-1", "agent-2"]) // list could also be retrieved from another fact`  
`($SELF$ (name ?agentList) (ruleengine_instance ?inst))`
  - The slot compare algorithm is save in regard to lists probably stored in facts by analysing the fact templates.

#### 4.2.2. Fact base queries

```
[<variable> <-] (query [<lag>] (<factDescr> <slot_cond>*))
```

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Provide a list of facts as result from a fact base query matching a specified pattern of values.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Arguments</b>             | <ul style="list-style-type: none"> <li>• <code>&lt;factDescr&gt;</code>: Name of the fact, possibly combined with fact base owner description.</li> <li>• <code>&lt;slot_cond&gt;</code>: Pairs of <code>&lt;slotName&gt;/&lt;constant&gt;</code>, <code>&lt;slotName&gt;/&lt;boundVariable&gt;</code> or <code>&lt;slotName&gt;/&lt;mathExpression&gt;</code> specifying the fact base query pattern, optional with a compare operator</li> <li>• Optional <code>&lt;lag&gt;</code>: a lag mode and possibly the related lag value.</li> </ul>                                                                                                                     |
| <b>Returns</b>               | <code>&lt;factList&gt;</code> : the list of facts fulfilling the query                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Evaluation Result</b>     | <ul style="list-style-type: none"> <li>• True, if one or more facts match the specified pattern</li> <li>• Otherwise false</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>           | <p>This operation performs a fact base query, searching for facts that match the specified pattern. The pattern consists of constants and bound variables which are assigned to slot names. All facts with equal values at these slots are retrieved.</p> <p>The query result is stored in the (optionally) specified result variable (which must not be bound).</p> <p>If only a fact name is given, than the target of the query is the local fact base (associated with the rule engine that evaluates the actual retrieve clause). Remote fact bases can be accessed by specifying the name (and instance, if required) of any agent or global rule engine.</p> |
| <b>Predefined slot names</b> | <ul style="list-style-type: none"> <li>• <code>owner</code>: string, specifying the agent that generated the fact.</li> <li>• <code>permanent</code>: Boolean value, indicating whether the fact is permanent or not. A permanent fact is retrieved in every time step (beginning with the time of assertion).</li> <li>• <code>timeStamp</code>: double value, specifying the time of assertion.</li> </ul>                                                                                                                                                                                                                                                        |
| <b>Example</b>               | <ul style="list-style-type: none"> <li>• <code>?tflist &lt;- (query (global::\$AGENT\$ (!= type</code></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |



- Household))) - lists all agent facts that are not of type "Household"
- `?tflist <- (query (global:: $AGENT$ (name ["agent-1", "agent-2", "agent-7"])))` - lists the agent facts with the three names specified in the list

### 4.2.3. Exists

[<variable> <-] (**exists** (<factRetrieval>|<compositeClause>))

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | <ol style="list-style-type: none"> <li>1. Perform a fact base query in order to test whether facts are available that match a specified pattern of values without binding specified unbound variables.</li> <li>2. Performs composite clause evaluation without binding variables used in the inner LHS.</li> <li>3. <b>Possibly subject of change/extension.</b></li> </ol>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Arguments</b>             | One of the constructs described under 4.2.1 Fact base retrieval.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Returns</b>               | <p>Boolean value:</p> <ul style="list-style-type: none"> <li>• True, if one or more facts match the specified pattern</li> <li>• Otherwise false</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Evaluation Result</b>     | True.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>           | <p>This operation performs a fact base query, searching for facts that match the specified pattern. The pattern consists of constants and bound variables which are assigned to slot names. All facts with equal values at these slots are retrieved.</p> <p>If one or more facts match the query, then true is written to the specified result variable (which must not be bound), otherwise false.</p> <p>If only a fact name is given, than the target of the query is the local fact base (associated with the rule engine that evaluates the actual retrieve clause). Remote fact bases can be accessed by specifying the name (and instance, if required) of any agent or global rule engine.</p> <p>When used with a composite clause, it performs a composite clause evaluation without binding variables used in the inner LHS (see section 4.2.4).</p> |
| <b>Predefined slot names</b> | See 4.2.1 Fact base retrieval.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

#### 4.2.4. Composite of inner LHS clauses

**(and** <LHSClause>\*)

**(or** <LHSClause>\*)

**(xor** <LHSClause>\*)

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Logical conjunction ( <b>and</b> ), disjunction ( <b>or</b> ) or exclusive disjunction ( <b>xor</b> ) of inner LHS clauses.                                                                                                                                                                                                                                                                                                             |
| <b>Arguments</b>             | Set of inner LHS clauses.                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Returns</b>               | Boolean value: <ul style="list-style-type: none"> <li>• True, if <ul style="list-style-type: none"> <li>○ for conjunction (and), the evaluation result for all inner clauses is true</li> <li>○ for disjunction (or), the evaluation result for at least on inner clauses is true</li> <li>○ exclusive disjunction (xor), the evaluation result for exactly one inner clauses is true</li> </ul> </li> <li>• Otherwise false</li> </ul> |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"> <li>• Same as return value, if no result variable is specified</li> <li>• True, if result is assigned to a (free) variable</li> <li>• Otherwise false</li> </ul>                                                                                                                                                                                                                      |
| <b>Description</b>           | This clause represents an enclosed set of LHS clauses (sub-LHS). All bound variables from the outer LHS are also available within the sub-LHS. The scope of variables that are bound within a composite clause (and/or/xor statements) is the entire rule, unless an <code>exists</code> is specified around the composite clause (then the scope of variables is restricted to the inner clause).                                      |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>               | The "classical" if-then-else construct in declarative code: if ?num is less than 4, then ?z will be bound with 5, otherwise with 0 <pre>(or   (and     (&lt; ?num 4)     ?z &lt;- (is 5)   )   ?z &lt;- (is 0) ) (print "the result is ?z)</pre>                                                                                                                                                                                        |



#### 4.2.5. Not operator

`(not <LHSClause>)`

|                              |                                                                                                                                                        |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Invert the evaluation result of the inner LHS clause.                                                                                                  |
| <b>Arguments</b>             | Inner LHS clause with its associated arguments.                                                                                                        |
| <b>Returns</b>               | Same as inner clause.                                                                                                                                  |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"><li>• True, if the evaluation result of the inner clause is false</li><li>• Otherwise false</li></ul> |
| <b>Description</b>           | The evaluation result of the inner clause is inverted.                                                                                                 |
| <b>Predefined slot names</b> | Same as inner clause.                                                                                                                                  |
| <b>Example</b>               | n/a                                                                                                                                                    |

#### 4.2.6. Foreach operator

`[<variable> <-] (each <list>)`

|                              |                                                                                                                                                                                 |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Takes a list of values and assigns the result variable with each single value of the list by branching the LHS evaluation tree.                                                 |
| <b>Arguments</b>             | <ul style="list-style-type: none"><li>• <code>&lt;list&gt;</code>: list of arbitrary values or facts</li></ul>                                                                  |
| <b>Returns</b>               | Each single element of the list.                                                                                                                                                |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"><li>• True, if list is not empty.</li><li>• Otherwise false.</li></ul>                                                         |
| <b>Description</b>           | The clause <code>each</code> takes a list of (any kind of) values and assigns the result variable with each single value of the list by branching the LHS evaluation tree.      |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                             |
| <b>Example</b>               | <pre>?list &lt;- (query (global::\$AGENT\$)) ?res &lt;- (each ?list) (print "?res.name says: This message will be printed as many times as agent facts are in \$AGENT\$")</pre> |



#### 4.2.7. Bind operator

[<variable> <-] (**is** <expression>)

|                              |                                                                                                                                                                                                                                                              |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Assign an expression to variable or use expression as clause evaluation result.                                                                                                                                                                              |
| <b>Arguments</b>             | <ul style="list-style-type: none"><li>• &lt;expression&gt;: constant, variable, mathematical term</li></ul>                                                                                                                                                  |
| <b>Returns</b>               | Value of equal type as expression.                                                                                                                                                                                                                           |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"><li>• Same as return value, if no result variable is specified and the expression is of type Boolean.</li><li>• True, if result is assigned to a (free) variable.</li><li>• Otherwise false.</li></ul>      |
| <b>Description</b>           | Assigns the value of expression. The expression can be a constant, a variable or a mathematical term.<br>The result of the operation can be assigned to a variable, or alternatively used as clause evaluation criteria (by omitting the result assignment). |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                          |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                          |

#### 4.2.8. Comparative operators

[<variable> <-] (**==** <expression> <expression>)

[<variable> <-] (**!=** <expression> <expression>)

[<variable> <-] (**<** <expression> <expression>)

[<variable> <-] (**<=** <expression> <expression>)

[<variable> <-] (**>** <expression> <expression>)

[<variable> <-] (**>=** <expression> <expression>)

|                          |                                                                                                                                                                       |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b> | Compare two expressions.                                                                                                                                              |
| <b>Arguments</b>         | <ul style="list-style-type: none"><li>• &lt;expression&gt;:<ul style="list-style-type: none"><li>○ "left expression"</li><li>○ "right expression"</li></ul></li></ul> |
| <b>Returns</b>           | Boolean value: <ul style="list-style-type: none"><li>• True, if</li></ul>                                                                                             |



|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | <ul style="list-style-type: none"> <li>○ ==: left expression equals right expression</li> <li>○ !=: left expression not equal to right expression</li> <li>○ &lt;: left expression is less than right expression</li> <li>○ &lt;=: left expression is less than or equal to right expression</li> <li>○ &gt;: left expression is greater than right expression</li> <li>○ &gt;=: left expression is greater than or equal to right expression</li> <li>● Otherwise false</li> </ul> |
| <b>Evaluation Result</b>     | <p>Boolean value:</p> <ul style="list-style-type: none"> <li>● Same as return value, if no result variable is specified</li> <li>● True, if result is assigned to a (free) variable</li> <li>● Otherwise false</li> </ul>                                                                                                                                                                                                                                                           |
| <b>Description</b>           | <p>Performs a compare operation (specified by an operator) on two operands specified by expressions. Each expression can be a constant, a variable or a mathematical term.</p> <p>The result of the operation can be assigned to a variable, or alternatively used as clause evaluation criteria (by omitting the result assignment).</p>                                                                                                                                           |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

#### 4.2.9. List operators

[<variable> <-] (**first** <list>)

[<variable> <-] (**length** <list>)

[<variable> <-] (**listCreate** [<list>] (<list>|<expression>))

[<variable> <-] (**listRemove** <list> (<list>|<expression>))

[<variable> <-] (**member** <list> <expression>)

[<variable> <-] (**nth** <list> <expression>)

|                          |                                                                                                                                                                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b> | Perform operations on a list.                                                                                                                                                                                                                                                     |
| <b>Arguments</b>         | <ul style="list-style-type: none"> <li>● &lt;list&gt;: the list to operate on as a bound variable</li> <li>● &lt;expression&gt;: an element of a list (<b>listCreate</b>, <b>listRemove</b>, <b>member</b>) or a number indicating an element in the list (<b>nth</b>)</li> </ul> |
| <b>Returns</b>           | <p>&lt;variable&gt;: depending on the performed operation, this is</p> <ul style="list-style-type: none"> <li>● The number of elements in the list (<b>length</b>)</li> <li>● The specified element of the list (<b>first</b>, <b>nth</b>)</li> </ul>                             |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | <ul style="list-style-type: none"> <li>• True, if the specified value is an element of the list; otherwise false (member)</li> <li>• The new / altered list (<code>listCreate</code>, <code>listRemove</code>)</li> </ul>                                                                                                                                                                                                                                                                                                                                                   |
| <b>Evaluation Result</b>     | <p>Boolean value:</p> <ul style="list-style-type: none"> <li>• True, if result is assigned to a (free) variable</li> <li>• Otherwise false</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>           | <p>The different list operators perform operations on a given list. The unary operators <code>first</code> and <code>length</code> take only the list as argument and return its first element and its length (number of elements), respectively. The binary operators <code>listCreate</code>, <code>listRemove</code>, <code>member</code> and <code>nth</code> take an expression as a second argument, which specifies either (an element of) a list or – in the case of <code>nth</code> – a number.</p> <p>The result of the operation is assigned to a variable.</p> |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

#### 4.2.10. Set operators

```
[<variable> <-] (contains <set> <expression>)
```

```
[<variable> <-] (intersect <set> <set>)
```

```
[<variable> <-] (setCreate [<set>] (<set>|<list>|<expression>))
```

```
[<variable> <-] (setRemove <set> (<set>|<list>|<expression>))
```

```
[<variable> <-] (size <set>)
```

```
[<variable> <-] (union <set> <set>)
```

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b> | Perform operations on a set.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Arguments</b>         | <ul style="list-style-type: none"> <li>• <code>&lt;set&gt;</code>: the set to operate on as a bound variable</li> <li>• <code>&lt;expression&gt;</code>: an element of a set</li> </ul>                                                                                                                                                                                                                                                                                     |
| <b>Returns</b>           | <p><code>&lt;variable&gt;</code>: depending on the performed operation, this is</p> <ul style="list-style-type: none"> <li>• The number of elements in the set (<code>size</code>)</li> <li>• True, if the specified value is an element of the set; otherwise false (<code>contains</code>)</li> <li>• The new / altered list (<code>listCreate</code>, <code>listRemove</code>)</li> <li>• The intersection of the two specified sets (<code>intersect</code>)</li> </ul> |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | <ul style="list-style-type: none"> <li>• The union of the two specified sets (union)</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Evaluation Result</b>     | <p>Boolean value:</p> <ul style="list-style-type: none"> <li>• True, if result is assigned to a (free) variable</li> <li>• Otherwise false</li> </ul>                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b>           | <p>The different set operators perform operations on a given set. The unary operator <code>size</code> takes only the set as argument and returns its size (number of elements). The binary operators <code>listCreate</code>, <code>listRemove</code>, <code>contains</code>, <code>intersect</code> and <code>union</code> take either another set as a second argument or an expression, which specifies an element of the set. The result of the operation is assigned to a variable.</p> |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

#### 4.2.11. Accumulator

```
[<variable> <-] (avg (<factList> <slotName>) | (<list>))
```

```
[<variable> <-] (count (<factList> <slotName>) | (<list>))
```

```
[<variable> <-] (list <factList> <slotName>)
```

```
[<variable> <-] (max (<factList> <slotName>) | (<list>))
```

```
[<variable> <-] (min (<factList> <slotName>) | (<list>))
```

```
[<variable> <-] (set <factList> <slotName>)
```

```
[<variable> <-] (sum (<factList> <slotName>) | (<list>))
```

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b> | Perform accumulating operations on a fact list.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>         | <ul style="list-style-type: none"> <li>• Processing of fact lists: <ul style="list-style-type: none"> <li>○ <code>&lt;factList&gt;</code>: the list of facts (usually the result of a query) to operate on as a bound variable</li> <li>○ <code>&lt;slotName&gt;</code>: the name of a slot in the shared fact template of the facts in the list; the corresponding slot values have to be numerical for the mathematical operators <code>avg</code>, <code>max</code>, <code>min</code> and <code>sum</code> to work.</li> </ul> </li> <li>• Processing of other lists: <ul style="list-style-type: none"> <li>○ <code>&lt;list&gt;</code>: the list of (any kind of) values</li> </ul> </li> </ul> |
| <b>Returns</b>           | <code>&lt;variable&gt;</code> : depending on the performed operation, this is                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |



|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | <ul style="list-style-type: none"> <li>• A number: the average, maximum, minimum or sum of the specified slot/list values (<i>avg</i>, <i>max</i>, <i>min</i>, <i>sum</i>) or the number of slot values/list elements (<i>count</i>)</li> <li>• A new list containing all values of the specified slot as elements (<i>list</i>)</li> <li>• A new set containing all unique values of the specified slot as elements (<i>set</i>)</li> </ul>                 |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"> <li>• True, if result is assigned to a (free) variable</li> <li>• Otherwise false</li> </ul>                                                                                                                                                                                                                                                                                                               |
| <b>Description</b>           | The different accumulator operators perform operations on a given list of facts, which is usually the result of a fact base query, or on a list of any kind of values. The mathematical operators <i>avg</i> , <i>max</i> , <i>min</i> and <i>sum</i> expect numerical values as slot values, whereas the other operators ( <i>count</i> , <i>list</i> , <i>set</i> ) work on any type of slot value. The result of the operation is assigned to a variable. |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

#### 4.2.12. Deftype components

```
[<variable> <-] (components <type_definition>)
```

|                              |                                                                                                                                                                                                                                                                           |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | List components of the specified <b>deftype</b> .                                                                                                                                                                                                                         |
| <b>Arguments</b>             | <type_definition>: any type definition defined in the model.                                                                                                                                                                                                              |
| <b>Returns</b>               | <variable>: a list of components for the specified deftype                                                                                                                                                                                                                |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"> <li>• True, if type definition exists</li> <li>• Otherwise false</li> </ul>                                                                                                                                             |
| <b>Description</b>           | The clause returns a list of components for any defined type definition in the model.                                                                                                                                                                                     |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                       |
| <b>Example</b>               | With this clause, there are (at least) two solutions for the problem to select on of the components of a deftype by chance (if all components of the deftype have equal probabilities): <ol style="list-style-type: none"> <li>1. using clauses for each step:</li> </ol> |



```
?comps <- (components "SomeDeftype")
?numComps <- (length ?comps)
?rndSel <- (is {uchicago.src.sim.util.Random.
 uniform.nextIntFromTo(0, ?numComps:Integer - 1)})
?result <- (nth ?comps ?rndSel)
```

2. using the Java capabilities of expressions (obviously, this code fragment is shorter, but needs proper documentation...):

```
?comps <- (components "SomeDeftype")
?result <- (is
 {(?comps:Object[])[uchicago.src.sim.util.
 Random.uniform.nextIntFromTo(0,
 (?comps:Object[]).length - 1)]})
```

#### 4.2.13. Symbol generator

```
[<variable> <-] (gensym <symbol>)
```

```
[<variable> <-] (genuuid)
```

|                              |                                                                                                                                                                                                                                                                                            |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Binding variables to different kinds of values.                                                                                                                                                                                                                                            |
| <b>Arguments</b>             | <symbol>: The string that will be root word for the generated symbol.                                                                                                                                                                                                                      |
| <b>Returns</b>               | <variable>: depending on the performed operation, this is <ul style="list-style-type: none"><li>• unique symbol, consisting of the specified symbol string, concatenated with an unique number (gensym &lt;symbol&gt;)</li><li>• new UUID (genuuid)</li></ul>                              |
| <b>Evaluation Result</b>     | Boolean value: True                                                                                                                                                                                                                                                                        |
| <b>Description</b>           | Two clauses for binding variables to different content: <ul style="list-style-type: none"><li>• (gensym &lt;symbol&gt;) - generates an unique symbol, consisting of the specified symbol string, concatenated with an unique number.</li><li>• (genuuid) - generates a new UUID.</li></ul> |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                        |
| <b>Example</b>               | <ul style="list-style-type: none"><li>• ?sym &lt;- (gensym household) - each time this clause is unified, a string of the kind "household-000", "household-001" etc. is bound to variable ?sym</li><li>• ?id &lt;- (genuuid)</li></ul>                                                     |



#### 4.2.14. Print

```
(print [prio <int_value>] "<printString>")
```

|                              |                                                                                                                                                                                                                                                                                                                                            |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Prints a string to the console.                                                                                                                                                                                                                                                                                                            |
| <b>Arguments</b>             | <printString>: The string that will be printed to the console.<br>Optional <b>prio</b> <int_value>: Priority of the print statement.                                                                                                                                                                                                       |
| <b>Returns</b>               | Nothing                                                                                                                                                                                                                                                                                                                                    |
| <b>Evaluation Result</b>     | Boolean value: True                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b>           | The specified string is printed to the console. The string may contain an arbitrary number of words or tokens. Any token beginning with a "?" character is assumed to be a variable name; if a bound variable with this name exists, then the content of this variable is inserted in the output, otherwise the original token is printed. |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                        |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                        |

#### 4.2.15. Call

```
[<variable> <-] (call <methodName> ([<expression> [, <expression>]*]))
```

|                              |                                                                                                                                                                                                                                                                                                     |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Calls a Java method.                                                                                                                                                                                                                                                                                |
| <b>Arguments</b>             | <ul style="list-style-type: none"><li>• &lt;methodName&gt;: the set to operate on as a bound variable</li><li>• &lt;expression&gt;: method parameter</li></ul>                                                                                                                                      |
| <b>Returns</b>               | The return value of the method.                                                                                                                                                                                                                                                                     |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"><li>• True, if the method was executed successfully</li><li>• Otherwise false</li></ul>                                                                                                                                                            |
| <b>Description</b>           | A Java method with the specified name and zero, one or more parameters is invoked. The return value of the method can be used to bind a free variable.<br>The method to be called must be member of the agent class the rule with this clause belongs. <b>Possibly subject of change/extension.</b> |
| <b>Predefined slot names</b> | Same as inner clause.                                                                                                                                                                                                                                                                               |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                 |



#### 4.2.16. Agent birth and death

`(create <agent_type> <agent_name>)`

`(kill <agent_type> <agent_name>)`

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Triggers birth/death of an agent instance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Arguments</b>             | <ul style="list-style-type: none"><li>• <code>&lt;agent_type&gt;</code>: class identifier (string) of the agent to create</li><li>• <code>&lt;agent_name&gt;</code>: the name (string) of the instance</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Returns</b>               | Nothing.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Evaluation Result</b>     | Boolean value: <ul style="list-style-type: none"><li>• True, if the action was executed successfully</li><li>• Otherwise false</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>           | <p>Two clauses for agent birth and death processes:</p> <ul style="list-style-type: none"><li>• <code>create</code>: creates a new agent instance for the specified agent class with the specified name;</li><li>• <code>kill</code>: removes an agent instance of the specified agent class and name.</li></ul> <p>Two call-back methods for DRAMS (<code>createAgent()</code> and <code>killAgent()</code>) related to <code>create</code> and <code>kill</code> clauses are defined in the model interface. These should be implemented/adapted in the model super class according to the simulation tool specific agent management.</p> |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Example</b>               | <pre>(create ?type ?name) (kill ?type ?name)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

#### 4.2.17. Breakpoint

`(halt)`

|                          |                                                                                                                                              |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b> | Breakpoint definition.                                                                                                                       |
| <b>Arguments</b>         | n/a                                                                                                                                          |
| <b>Returns</b>           | Nothing.                                                                                                                                     |
| <b>Evaluation Result</b> | Boolean value: <ul style="list-style-type: none"><li>• True, if the action was executed successfully</li><li>• Otherwise false</li></ul>     |
| <b>Description</b>       | This clause causes the simulation timer to pause or halt the simulation run, according to the simulation tool specific implementation of the |





|                              |                                                        |
|------------------------------|--------------------------------------------------------|
|                              | related halt() method, defined in the model interface. |
| <b>Predefined slot names</b> | n/a                                                    |
| <b>Example</b>               | n/a                                                    |

### 4.3. RHS CLAUSES

#### 4.3.1. Fact assertion

```
(assert [permanent] [<deferment>] (<factDescr> <slot>*))
```

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Asserts a new fact to a fact base.                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Arguments</b>             | <ul style="list-style-type: none"> <li>• &lt;factDescr&gt;: name of the fact to be asserted, possibly combined with fact base owner description.</li> <li>• &lt;slot&gt;: pairs of (&lt;slotName&gt; &lt;constant&gt;) or (&lt;slotName&gt; &lt;boundVariable&gt;) specifying the content to be asserted</li> <li>• Optional &lt;deferment&gt;: the number of ticks after which the assertion will be valid.</li> </ul> |
| <b>Description</b>           | A new fact with name, target fact base and content as specified is asserted. The new fact can be made permanent, if the keywords <b>assert permanent</b> instead of <b>assert</b> are used. If no deferment value is specified, the timestamp of the new fact will be the current simulation time, otherwise it will be increased by the deferment value.                                                               |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                     |

#### 4.3.2. Fact retraction

```
(retract [<deferment>] (<factDescr> <slot>*))
```

```
(retract [<deferment>] <factVariable>)
```

|                          |                                                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b> | Retracts a fact from a fact base.                                                                                          |
| <b>Arguments</b>         | <ul style="list-style-type: none"> <li>• &lt;factDescr&gt;: name of the fact to be retracted, possibly combined</li> </ul> |



|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | <p>with fact base owner description.</p> <ul style="list-style-type: none"> <li>• <code>&lt;slot&gt;</code>: pairs of (<code>&lt;slotName&gt;</code> <code>&lt;constant&gt;</code>) or (<code>&lt;slotName&gt;</code> <code>&lt;boundVariable&gt;</code>) specifying the content of the fact to be retracted.</li> <li>• <code>&lt;factVariable&gt;</code>: variable bound to a fact to be retracted.</li> <li>• Optional <code>&lt;deferment&gt;</code>: the number of ticks after which the retraction will be valid.</li> </ul> |
| <b>Description</b>           | A fact with name, target fact base and (optional) content as specified is retracted. Alternatively a fact bound to a variable is retracted. If no deferment value is specified, the time at which the fact will become invalid will be the current simulation time, otherwise it will be increased by the deferment value.                                                                                                                                                                                                         |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

### 4.3.3. Output writing

```
(sample (<writer_name> <slot>*))
```

```
(write (<writer_name> <slot>*))
```

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Pass data to output writers.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>             | <ul style="list-style-type: none"> <li>• <code>&lt;writer_name&gt;</code>: name of the fact to be asserted, possibly combined with fact base owner description.</li> <li>• <code>&lt;slot&gt;</code>: pairs of (<code>&lt;slotName&gt;</code> <code>&lt;constant&gt;</code>) or (<code>&lt;slotName&gt;</code> <code>&lt;boundVariable&gt;</code>) specifying the content to be written</li> </ul>                                    |
| <b>Description</b>           | <p>To pass data to output writers, the following RHS-clauses (which are used in the same way as <code>assert</code> clauses) are available:</p> <ol style="list-style-type: none"> <li>1. Sample data for later writing: <code>(sample (&lt;writer_name&gt; &lt;slot&gt;*))</code></li> <li>2. Write a previously sampled or new data set: <code>(write (&lt;writer_name&gt; &lt;slot&gt;*))</code>.</li> </ol>                       |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Example</b>               | <ul style="list-style-type: none"> <li>• <code>(sample (myWriter (value1 24) (value2 ?sym)))</code> - samples the specified data with actual meta information (rule name, simulation time etc.)</li> <li>• <code>(write (myWriter))</code> - writes all previously sampled data to the destination (file) of the output writer. The meta information of the data fields are kept untouched, the actual meta information is</li> </ul> |



attached to the entire data set.

- `(write (myWriter (value1 25) (value2 ?sym)))` - writes the specified data to the destination (file) of the output writer. Actual meta information is attached to the data fields and to the entire data set.

#### 4.3.4. Print

```
(print "<printString>")
```

|                              |                                                                                                                                                                                                                                                                                                                                            |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Prints a string to the console.                                                                                                                                                                                                                                                                                                            |
| <b>Arguments</b>             | <printString>: The string that will be printed to the console.<br>Optional <b>prio</b> <int_value>: Priority of the print statement.                                                                                                                                                                                                       |
| <b>Description</b>           | The specified string is printed to the console. The string may contain an arbitrary number of words or tokens. Any token beginning with a "?" character is assumed to be a variable name; if a bound variable with this name exists, then the content of this variable is inserted in the output, otherwise the original token is printed. |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                                                                                                                                                        |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                                                        |

#### 4.3.5. Call

```
(call <methodName>([<expression> (, <expression>)*]))
```

|                              |                                                                                                                                                                                                                                                                                                               |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Calls a Java method.                                                                                                                                                                                                                                                                                          |
| <b>Arguments</b>             | <ul style="list-style-type: none"> <li>• &lt;methodName&gt;: the set to operate on as a bound variable</li> <li>• &lt;expression&gt;: method parameter</li> </ul>                                                                                                                                             |
| <b>Description</b>           | A Java method with the specified name and zero, one or more parameters is invoked. The return value of the method can be used to bind a free variable.<br>The method to be called must be member of the agent class the rule with this clause belongs. <b>Possibly subject of change/extension in future.</b> |
| <b>Predefined slot names</b> | Same as inner clause.                                                                                                                                                                                                                                                                                         |
| <b>Example</b>               | n/a                                                                                                                                                                                                                                                                                                           |



#### 4.3.6. Agent birth and death

`(create <agent_type> <agent_name>)`

`(kill <agent_type> <agent_name>)`

|                              |                                                                                             |
|------------------------------|---------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Triggers birth/death of an agent instance.                                                  |
| <b>Arguments</b>             | Two clauses for agent birth and death processes:                                            |
| <b>Description</b>           | create: creates a new agent instance for the specified agent class with the specified name; |
| <b>Predefined slot names</b> | n/a                                                                                         |
| <b>Example</b>               | <code>(create ?type ?name)</code><br><code>(kill ?type ?name)</code>                        |

#### 4.3.7. Breakpoint

`(halt)`

|                              |                                                                                                                                                                                                     |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Short description</b>     | Breakpoint definition.                                                                                                                                                                              |
| <b>Arguments</b>             | n/a                                                                                                                                                                                                 |
| <b>Description</b>           | This clause causes the simulation timer to pause or halt the simulation run, according to the simulation tool specific implementation of the related halt() method, defined in the model interface. |
| <b>Predefined slot names</b> | n/a                                                                                                                                                                                                 |
| <b>Example</b>               | n/a                                                                                                                                                                                                 |

## **5. REFERENCES**

- [1] S. Moss, R. Meyer, U. Lotzmann, M. Kacprzyk, M. Roszczynska and C. Pizzo, “D5.1 Scenario, policy model and rule-based agent design. Deliverable 5.1,” OCOPOMO, 2010.
- [2] M. Mach et al, D2.1 Platform Architecture and Functional Description of Components. Deliverable 2.1, OCOPOMO consortium, 2010.
- [3] U. Lotzmann and R. Meyer, “DRAMS - A Declarative Rule-Based Agent Modelling System,” in *25th European Conference on Modelling and Simulation, ECMS 2011*, T. Burczynski, J. Kolodziej, A. Byrski and M. Carvalho, Eds., Krakow, SCS Europe, 2011, pp. 77-83.
- [4] U. Lotzmann and R. Meyer, “A Declarative Rule-Based Environment for Agent Modelling Systems,” in *The Seventh Conference of the European Social Simulation Association, ESSA 2011*, Montpellier, 2011.
- [5] U. Lotzmann und M. A. Wimmer, „Provenance and Traceability in Agent Based Policy Simulation,“ in s *The 26th European Simulation and Modelling Conference, ESM 2012*, Essen, 2012.
- [6] U. Lotzmann und M. A. Wimmer, „Traceability in Evidence-based Policy Simulation,“ in s *27th European Conference on Modelling and Simulation, ECMS 2013 (accepted)*, Alesund, 2013.
- [7] S. Scherer et al, D8.1 Manual of the Methodology for Process Development and Guide to Policy Modelling Toolbox. Deliverable 8.1, OCOPOMO consortium, 2013.



## 6. ANNEXES

### 6.1. DRAMS SYNTAX KEYWORDS

|                         | <b>Keyword</b>     | <b>Category</b>                |
|-------------------------|--------------------|--------------------------------|
| <b>Top level</b>        | <b>deflog</b>      | Log writer                     |
|                         | <b>defoutput</b>   | Output writer                  |
|                         | <b>defrule</b>     | Rules                          |
|                         | <b>deftemplate</b> | Fact templates                 |
|                         | <b>deftype</b>     | Type definitions               |
|                         | <b>execrule</b>    | Console Window                 |
|                         | <b>insertfact</b>  | Facts                          |
|                         | <b>insertfacts</b> | Facts                          |
| <b>Tags<sup>6</sup></b> | <b>@link</b>       | Traceability and links         |
| <b>LHS</b>              | <b>&lt;</b>        | Comparative operators          |
|                         | <b>&lt;=</b>       | Comparative operators          |
|                         | <b>==</b>          | Comparative operators          |
|                         | <b>!=</b>          | Comparative operators          |
|                         | <b>&gt;</b>        | Comparative operators          |
|                         | <b>&gt;=</b>       | Comparative operators          |
|                         | <b>and</b>         | Composite of inner LHS clauses |
|                         | <b>avg</b>         | Accumulator                    |
|                         | <b>call</b>        | Call                           |
|                         | <b>contains</b>    | Set operators                  |
|                         | <b>count</b>       | Accumulator                    |
|                         | <b>create</b>      | Agent birth and death          |
|                         | <b>each</b>        | Foreach operator               |
|                         | <b>exists</b>      | Exists                         |
|                         | <b>first</b>       | List operators                 |
|                         | <b>gensym</b>      | Symbol generator               |
|                         | <b>genuuid</b>     | Symbol generator               |
|                         | <b>halt</b>        | Breakpoint                     |
|                         | <b>intersect</b>   | Set operators                  |
|                         | <b>is</b>          | Bind operator                  |
|                         | <b>kill</b>        | Agent birth and death          |
| <b>length</b>           | List operators     |                                |

---

<sup>6</sup> in comments



|                |                   |                                    |
|----------------|-------------------|------------------------------------|
|                | <b>list</b>       | Accumulator                        |
|                | <b>listCreate</b> | List operators                     |
|                | <b>listRemove</b> | List operators                     |
|                | <b>max</b>        | Accumulator                        |
|                | <b>member</b>     | List operators                     |
|                | <b>min</b>        | Accumulator                        |
|                | <b>not</b>        | Not operator                       |
|                | <b>nth</b>        | List operators                     |
|                | <b>or</b>         | Composite of inner LHS clauses     |
|                | <b>print</b>      | Print                              |
|                | <b>query</b>      | Fact base queries                  |
|                | <b>set</b>        | Accumulator                        |
|                | <b>setCreate</b>  | Set operators                      |
|                | <b>setRemove</b>  | Set operators                      |
|                | <b>size</b>       | Set operators                      |
|                | <b>sum</b>        | Accumulator                        |
|                | <b>union</b>      | Set operators                      |
|                | <b>xor</b>        | Composite of inner LHS clauses     |
| <b>RHS</b>     | <b>assert</b>     | Fact assertion                     |
|                | <b>call</b>       | Call                               |
|                | <b>create</b>     | Agent birth and death              |
|                | <b>halt</b>       | Breakpoint                         |
|                | <b>kill</b>       | Agent birth and death              |
|                | <b>print</b>      | Print                              |
|                | <b>retract</b>    | Fact retraction                    |
|                | <b>sample</b>     | Output writing                     |
|                | <b>write</b>      | Output writing                     |
| <b>General</b> | <b>&lt;</b>       | Slot compare operators, Log output |
|                | <b>&lt;=</b>      | Slot compare operators, Log output |
|                | <b>==</b>         | Slot compare operators, Log output |
|                | <b>!=</b>         | Slot compare operators, Log output |
|                | <b>&gt;</b>       | Slot compare operators, Log output |
|                | <b>&gt;=</b>      | Slot compare operators, Log output |
|                | <b>&lt;&gt;</b>   | Slot compare operators, Log output |
|                | <b>!&lt;&gt;</b>  | Slot compare operators, Log output |
|                | <b>&lt;=&gt;</b>  | Slot compare operators, Log output |
|                | <b>!&lt;=&gt;</b> | Slot compare operators, Log output |
|                | <b>*</b>          | Clauses                            |
|                | <b>after</b>      | Lag modes                          |
|                | <b>all</b>        | Lag modes                          |
|                | <b>at</b>         | Lag modes                          |





|                   |                                          |
|-------------------|------------------------------------------|
| <b>before</b>     | Lag modes                                |
| <b>deferredBy</b> | Clauses, Fact assertion, Fact retraction |
| <b>lag</b>        | Lag modes                                |
| <b>lagAfter</b>   | Lag modes                                |
| <b>lagBefore</b>  | Lag modes                                |
| <b>last</b>       | Lag modes                                |
| <b>latest</b>     | Lag modes                                |
| <b>recent</b>     | Lag modes (reserved)                     |
| <b>relAfter</b>   | Lag modes                                |
| <b>relAt</b>      | Lag modes                                |
| <b>relBefore</b>  | Lag modes                                |
| <b>permanent</b>  | Facts, Fact assertion                    |
| <b>prio</b>       | Log writer, Print (LHS), Print (RHS)     |

## 6.2. FREQUENTLY ASKED QUESTIONS

### 6.2.1. How can output writers be used?

Here is a more concrete code example how to write numerical data into csv files:

1. Put the writer definition in one of the .drams files, e.g. in code.drams. Hereby the head of the table that will be created is defined, i.e. "value1" to "value3" are the column headers:  

```
(defoutput csv "nameOfResultFile" (value1:Double) (value2:Integer) (value3:Double))
```
2. If all the three values that have to be written into the csv file are available in one and the same rule, just put a write statement in the RHS of that rule:  

```
(write (nameOfResultFile (value1 ?resultValue1) (value2 ?resultValue2) (value3 ?resultValue3)))
```

Alternatively, if e.g. the three different result values are produced by two different rules, the procedure can be as follows:

- 3a. Sample the result values:

```
(defrule rule1
 ...
=>
 (sample (nameOfResultFile (value1 ?resultValue1) (value3 ?resultValue3)))
 ...
)

(defrule rule2
 ...
=>
 (sample (nameOfResultFile (value2 ?resultValue)))
```



```
 ...
)
```

- 3b. Write the current sample to the file e.g. by another rule. The only issue here is that it has to be taken care that the rule writing the result is executed subsequent to rule1 and rule2:

```
(defrule rule3
 ...
=>
 (write (nameOfResultFile))
 ...
)
```

If, for some reason, it is intended to write the data into an xml file instead of the csv file, "csv" has to be replaced with "graph-xml" in the writer definition. No other changes are necessary.

### 6.2.2. What should be noted when using the DRAMS Java API?

When accessing fact bases from Java model code, the following issues should be regarded:

- There are several query methods to be used from Java code, but for the moment only one method should be used:  
`Collection<IFact> query(String name, String owner, Double timeStamp, String[] slots, Object[] values)`
- The result of all query methods is not of type `List<IFact>` (as it was in old DRAMS versions) but rather of type `Collection<IFact>`. Furthermore, there won't be a `List` object behind the `Collection`, in fact, at the moment it is a `HashSet` (but this also cannot be guaranteed for future version). As a consequence, if a special type of collection is needed for further processing (e.g. an `ArrayList`), then a new Instance of this particular collection type has to be instantiated and filled with the collection returned by the query:  
`List<IFact> factList = new ArrayList<IFact>(globalFB.query("boroughCharacteristics", globalFB.getOwner(), 0.0, null, null));`