Integrating RuleML and OO

Dagstuhl-Seminar 02061 - Rule Markup Techniques

Date: 03 FEB 2002 - 08 FEB 2002

By Jens Dietrich / Polytechnic of Namibia

Contact: jens.dietrich@unforgettable.com

Overview

- Rules and the Enterprise Computing Landscape
- SQL & Co: Integrating Facts
- Clauses versus Clause Sets
- Types
- MetaInfo

Rules in the Enterprise Computing Landscape 1/4

- The development of Enterprise Computing Tools focuses on a principle called ‘separation of concerns’, i.e. separate software modules are responsible for the various aspects of computing
- ‘Aspects’ include Persistency, Transaction management, Security etc
- This has to be seen in the context of the industrialization of software making
- An important issue to make this approach work is the existence of open interface standards like SQL, CORBA and XML based approaches such as SOAP (XML-RPC)

Rules in the Enterprise Computing Landscape 2/4

- Business Rule management could be yet another aspect, separating business logic from business objects (the domain model).
- The domain model is the result of a design process (using UML or related approaches) and is implemented by classes (their respective instances) using an OO language such as Java, Smalltalk, C++ or VB.
- In reality, the UML model is most often ambiguous (in particular w.r.t. semantics of the objects described) and inconsistent with the implementation.

Rules in the Enterprise Computing Landscape 3/4

- The domain model contains logic (behavior of its objects) that should not become responsibility of the rule module, e.g. a *get cache flow* method in a *loan* object. In particular, it is not the purpose of a rule module to do math!
- Therefore, a rule module should focus on high level rules governing core processes of the respective software
- Advantages of such modules are obvious: software can keep up with (changing) business processes, entering the slow and expensive software development lifecycle can be avoided, rules are maintained as ‘data’

Rules in the Enterprise Computing Landscape 4/4

- Rule management systems should not try to replace an expert and require two new experts!
- It tends to be difficult to manage large sets of rules, visualization / user interfaces are a real challenge, most useful (acceptable) rule systems will be rather small, but operating on large sets of data (SQL result sets, web data)
- Reasoning is mostly initiated by the client (e.g., a web store customer transaction), ‘pull’ favors a goal driven reasoning.

SQL & Co: Integrating Facts 1/4

- Rule modules must integrate with other modules in the Enterprise Computing Landscape
- This applies in particular w.r.t. data storage. Redundancy of data should be avoided (except replication for performance boosting – ‘caching’). In particular, facts built from db data should not be stored separately
- Data sources that should be integrated include relational, object-relational, object and XML databases accessed using SQL, ODMG OQL and XML / EJB queries.

SQL & Co: Integrating Facts 2/4

Example:

If favorite color of the customer is red offer product ABC to the customer

This rule operates on a set of facts like:

the favorite color of customer Jim is red;
the favorite color of customer Tom is blue;
the favorite color of customer John is green;
…

This set of facts is usually large (imagine companies like Amazon), and available in the companies relational database (table/view *customer_pref_colors* with columns *customer* and *color*).

SQL & Co: Integrating Facts 3/4

Example (ctd.):

- Due to the size of the database, we cannot hold all facts permanently in memory.
- What is more, the fact base needs to be up-to-date at query time since the database is part of a distributed (multi-client) environment.
- Therefore services like indexing, concurrency control, transaction handling must be added to knowledge bases to deal with this problem. On the other hand, those features are already implemented in the data storage module (the RDBMS in this case).

SQL & Co: Integrating Facts 4/4

Example (ctd.):

- Solution: keep facts (although not as facts but as records) in the database and integrate them ‘on-the-fly’. The knowledge base would then have a ‘fact set’ rather than a fact to represent knowledge:

  FACT_SET(SELECT CUSTOMER,PREF_COLOR) FROM CUSTOMER_PREF_COLORS)

- Besides the SQL query, additional information must be provided to issue the query and to build facts: The kind of database, kind of protocol, versioning and login information (URL, username), info how to build facts from records.

Clauses vs Clause Sets 1/8

- This is part of a more general problem: are facts integrated as explicit sets or as descriptive sets.
- Like in math: {1,2,3} vs {n|exists i: i*i=n}
- Def: a **clause set** of is an object that allows clients to loop over its clauses. Therefore the implementation of a clause set in a rule processing system must provide access to the next clause, and ways to find out whether there are more clauses available. This implies data structures similar to java iterators or enumerations and database cursors.

Clauses vs Clause Sets 2/8

Example 1 (ctd.): SQL Predicates

- The clause set is defined by a SQL query (the query string itself + connect info)
- The predicate is the structure of the result set (the table/view queried)
- Each row in the result set represents a fact from the result set
- The clause set iterator is the cursor fetching rows from the result set

Clauses vs Clause Sets 3/8

Example 1 (ctd.): SQL Predicates

Issues:

- Besides the connect info, a lot of additional (meta-) information is needed like:
- Transactional info (can facts be built from a dirty read)
- Caching info (can we cache facts for multiple sessions, timeouts)
- Type Mapping Info

Clauses vs Clause Sets 4/8

Example 2: OR Rules

- The clause set is a rule with prerequisites connected by OR, like **if A OR B then C**
- The clauses in the respective clause set are simply the rules **if A then C** and **if B then C**

Clauses vs Clause Sets 5/8

Example 3: Interactive facts

- The clause set is a “shadow” fact F, at query time the user is prompted whether F is a fact, if so, the clause set is a singleton containing F, otherwise empty
- Issues: When a clause set is asked for its clauses, contextual info should be provided (e.g. current variable bindings or references to knowledge containers)

Clauses vs Clause Sets 6/8

Example 4: Evaluating Objects

- In an OO environment (example: java), methods (example: *equals()* defined in *Object*) returning *booleans* are considered as predicates.
- A clause set is defined by the predicate and a set of objects for each parameter type (incl. the type that defines the method)
- The clauses are build by performing the method for each combination of parameters and selecting those combination where this yields true

Clauses vs Clause Sets 7/8

Example 5: Distributed Knowledge Bases

- The clause set is a query (with variables) + a reference to another knowledge base/ inference engine
- A clause set is defined by the combinations of variable bindings associated with the result of this query
- Concepts ala “collaborating agents” – experts ask other experts

Clauses vs Clause Sets 8/8

Representing Clause Sets in XML

- Clause Sets are downward compatible: facts and (and) rules can be considered as singleton clause sets
- Clause Sets are extremely polymorphic objects, and any XML standard that wants to support them must accept this
- At the end this could mean specs for some common cases (SQL !!) plus general purpose custom clause set tag with key-value parameters, perhaps plus a reference to a URI where the meaning of this is defined

Types 1/6

- Current Version of RuleML is not typed, but typing is desirable for some reasons:
- Datasources and processing systems interfacing with rule ml are generally typed: therefore using RuleML to communicate results in loosing information
- Typing is needed for rule processing systems in order to integrate the semantics of the resp. types: e.g. to compute 3+4 (which is 7 if both are integers and might by 34 if both are strings)

Types 2/6

- Once we have agreed that typing is good to have, which types should be chosen?
- Candidates: define our own type universe, take an established type universe like SQL-92 types, or allow types from any system and scope them (types like *SQL92:varchar* or *java:java.lang.String*
- If an open approach is chosen, what about type mapping if XML data is passed between various systems. Type mapping is usually defined by procedures and hard to be wrapped as XML. E.g., consider all the trouble related to type mapping between RDBMS and OO (“paradigm mismatch”)

Types 3/6

Example

- The SQL predicate introduced above returns facts where the SQL predicate associates two text typed terms. Using SQL typing, the type is something like *SQL92:VARCHAR*. If the type system used by the rule base is Java (and that is the approach we are in favor of), the first term repr. a customer. The customer could be simply mapped to J*AVA:java.lang.String*. But to take advantage of the OO approach, one does not want to represent customers as string, but rather as instances of a *Customer* class.

Types 4/6

Example (ctd.)

- This mapping is a rather complex **procedure** and might even include performing new queries to the database.

Types 5/6

Question: Can the prerequisites of the following rules be evaluated without having the respective facts in the knowledge base?

- if true then something
- if a=a then something
- if 2+2=4 then something
- if “John” contains “h” then something
- If product.getPrice()<100 then something

Using Intercultural Typing + Clause Sets, rule processing systems can integrate the semantics of the systems they integrate

Types 6/6

Yet another Example:

- c .. a customer
- f .. a function defined by the SQL query

  SELECT AMOUNT FROM CUSTOMERS WHERE NAME=?
- rule if 100<f(c) then grant discount of 10% to c
- No separate fact needed to verify this rule, the prerequisite can just be evaluated using the known meaning of f (=issue a query) and 100.

MetaInfo 1/3

Knowledge bases (or rule bases, rule containers ..) need certain meta info that must be represented somehow when making them persistent (e.g., XML RuleML).

- Versioning, history, user info
- Error handling
- Inference strategies
- Priority handling

MetaInfo 2/3

Exception Handling:

- If knowledge bases are integration platforms, many things can go wrong (e.g., broken network connections)
- It should be declared whether certain exceptions are critical or not. This would imply an *on_error* tag in the xml representation
- Possible exception handler (not complete!):
  - Ignore respective clause
  - Ignore respective clause set
  - Return default value (for exception in complex terms)
  - Interrupt inference
  - Log exception

MetaInfo 3/3

Inference Strategy:

- Inference strategies (e.g., non-monotonic inference) does not only depend on the inference engine, but also on the data structure of the knowledge base (e.g., priorities)
- Example container data structures:
  - Order clauses (clause sets) as they are
  - Order clauses according to a given priority label
  - Order clauses w.r.t. the number of premisses
  - Order clauses w.r.t. the number of variables in the prerequisites
  - Use more sophist. Order with major / minor sort keys