

MANDARAX + ORYX

An Open-Source Rule Platform

Presentation by

Jens Dietrich, Massey University, New Zealand

Gerd Wagner, Eindhoven University of Technology, The Netherlands

Content

1. Introduction
2. Pros and Cons
3. Project History and Organisation
4. Design Objectives
5. Persistency
6. On Standards
7. Deployment and Rule Lifecycle
8. Mandarax related projects: oryx, chameleon, agents for the Semantic Web, commercial applications
9. Usage Scenarios
10. Next steps
11. EU Rent Example

Introduction: Mandarax

- **Mandarax** is an **open source Java rule platform** based on the logic programming concepts of
 - **derivation rules** with negation-as-failure
 - top-down rule evaluation (**backward reasoning**) and
 - generating answers by logical term unification (matching of complex terms)
- Mandarax is **extensible**: it comes with reference implementations (including a rule engine)...
- ... but you can plug in your own components

Introduction: Derivation Rules versus Production Rules

- Derivation rules:
 - IF Condition
 - THEN Conclusion
- Production Rules:
 - IF Condition
 - THEN Action
- Derivation rules have a logical semantics and can be processed
 - top-down (like Prolog rules), or
 - bottom-up (like in deductive databases)
- A derivation rule engine can process facts from **secondary storage** and **remote information sources**

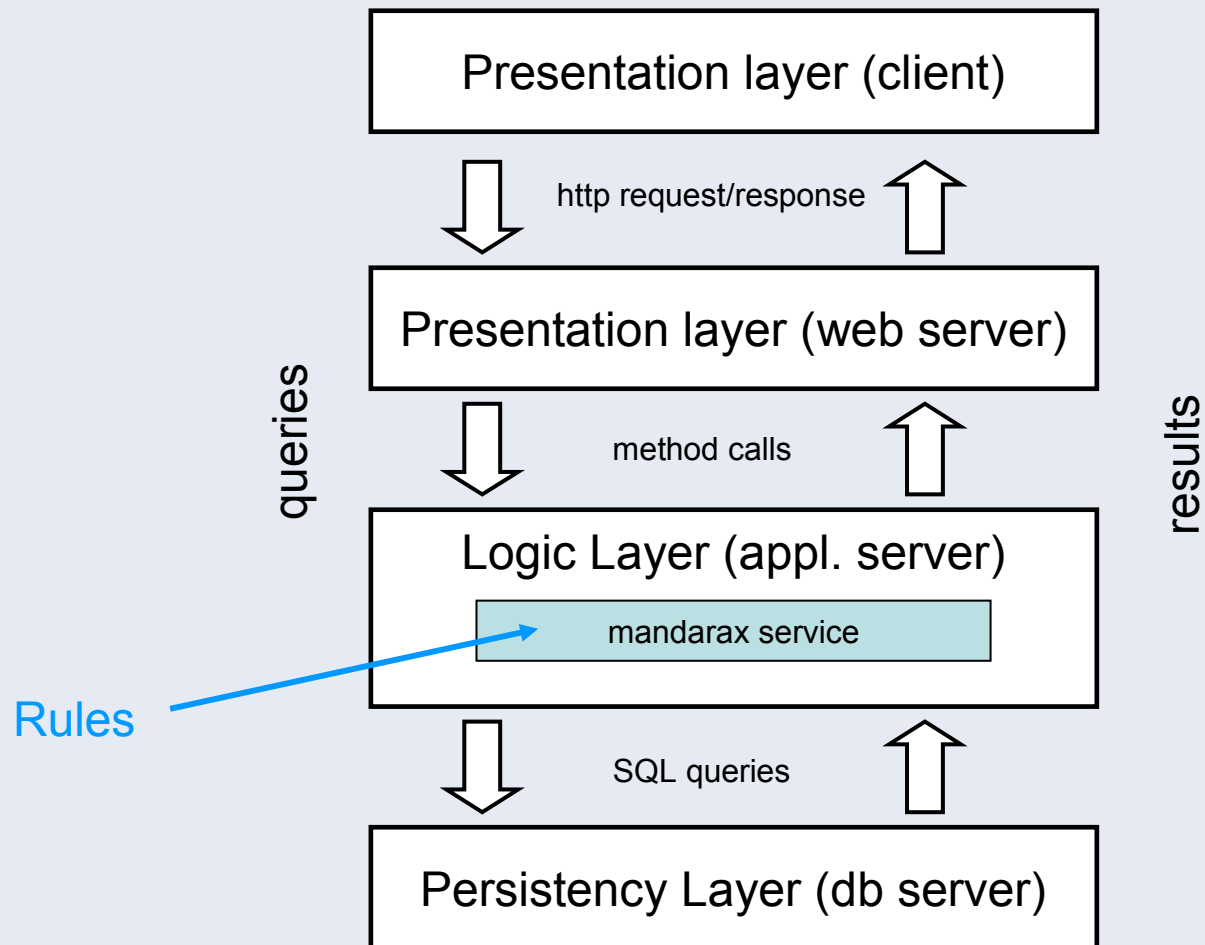
Introduction: Derivation Rules versus Production Rules

- Production rules have some kind of operational semantics, but **no clear logical semantics**
- Production rules can **implement derivation rules** by using the special action ASSERT
- Production rule engines **cannot** (easily) process facts from **secondary storage** and **remote information sources**

Introduction: Oryx

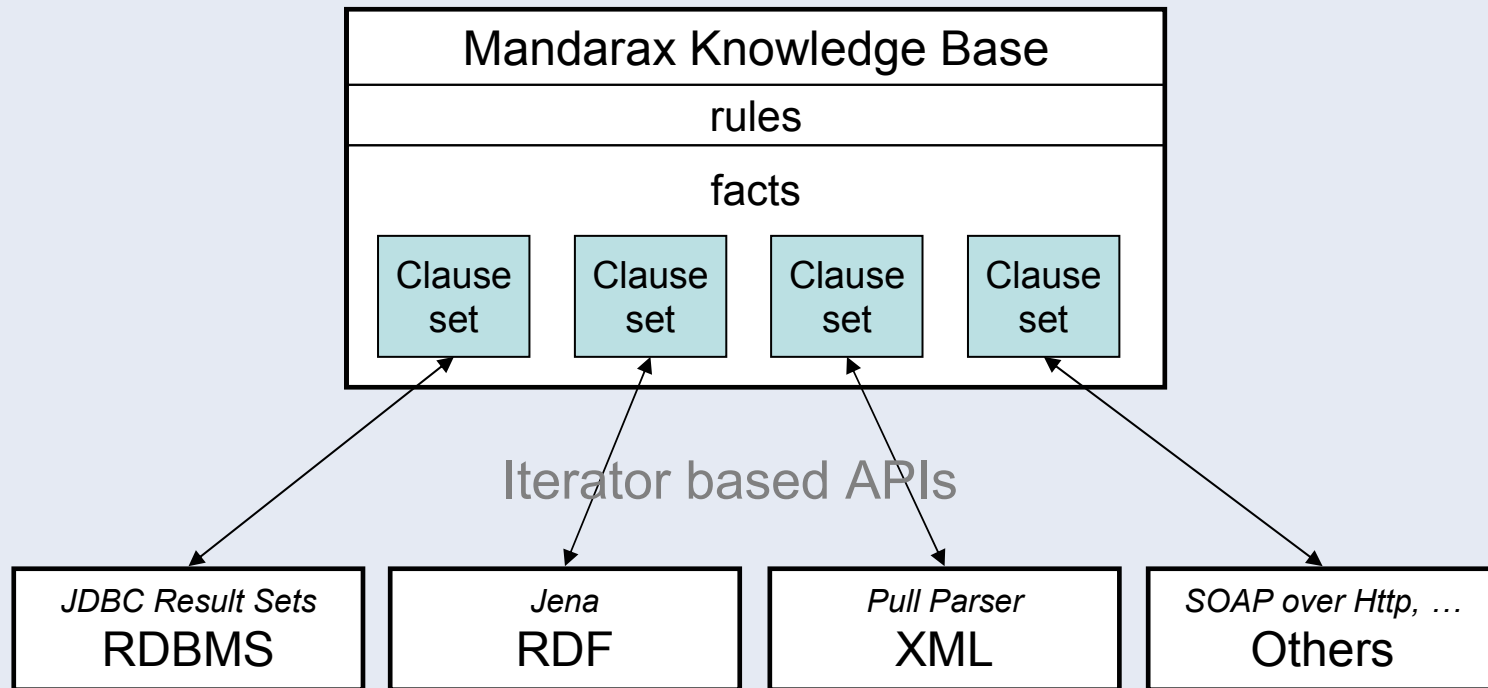
- **Oryx** is an **open source rule management tool** based on Mandarax
- Oryx includes
 - a **repository** for managing the vocabulary
 - a **formal-natural-language**-based rule editor
 - a user interface library based on Windows and Web interface components
- The repository of Oryx can be generated with the help of ORM/UML models

Rule-Based Query Answering in 3-Tier C/S Architectures



Design Objectives: Avoiding Data Redundancy

- Facts are integrated on demand using iterator-based interfaces (clause sets).
- By default, facts are not replicated (besides there are cache options to do so).
- Assumptions: large number of facts (Millions)



Example: Clause Set in a Portfolio Risk Example

The screenshot shows the Oryx Knowledge Base Editor interface. The main window displays a knowledge base structure with several clause sets:

- exists_special_risk**
 - if special risk for a share: a risk (reason: a description) then there is a special risk for a share
- has_risk**
 - if the the maximum of the country rating of a share and the sector rating of a share is between 0 and 4 and the price/earnings ratio...
 - if the the maximum of the country rating of a share and the sector rating of a share is between 0 and 4 and there is no special risk...
 - if the the maximum of the country rating of a share and the sector rating of a share is between 5 and 7 and there is no special risk...
 - if 8 is less than or equals the maximum of the country rating of a share and the sector rating of a share then a share is a **HIGH** risk...
 - if special risk for a share: a risk (reason: a description) then a share is a a risk risk investment
- has_special_risk**
 - ! Facts from database for: has_special_risk

A callout box labeled "show clauses" action points to the "Facts from database for: has_special_risk" entry. Below the main window, a smaller window titled "Clauses" displays the results of this action:

clause set: Facts from database for: has_special_risk

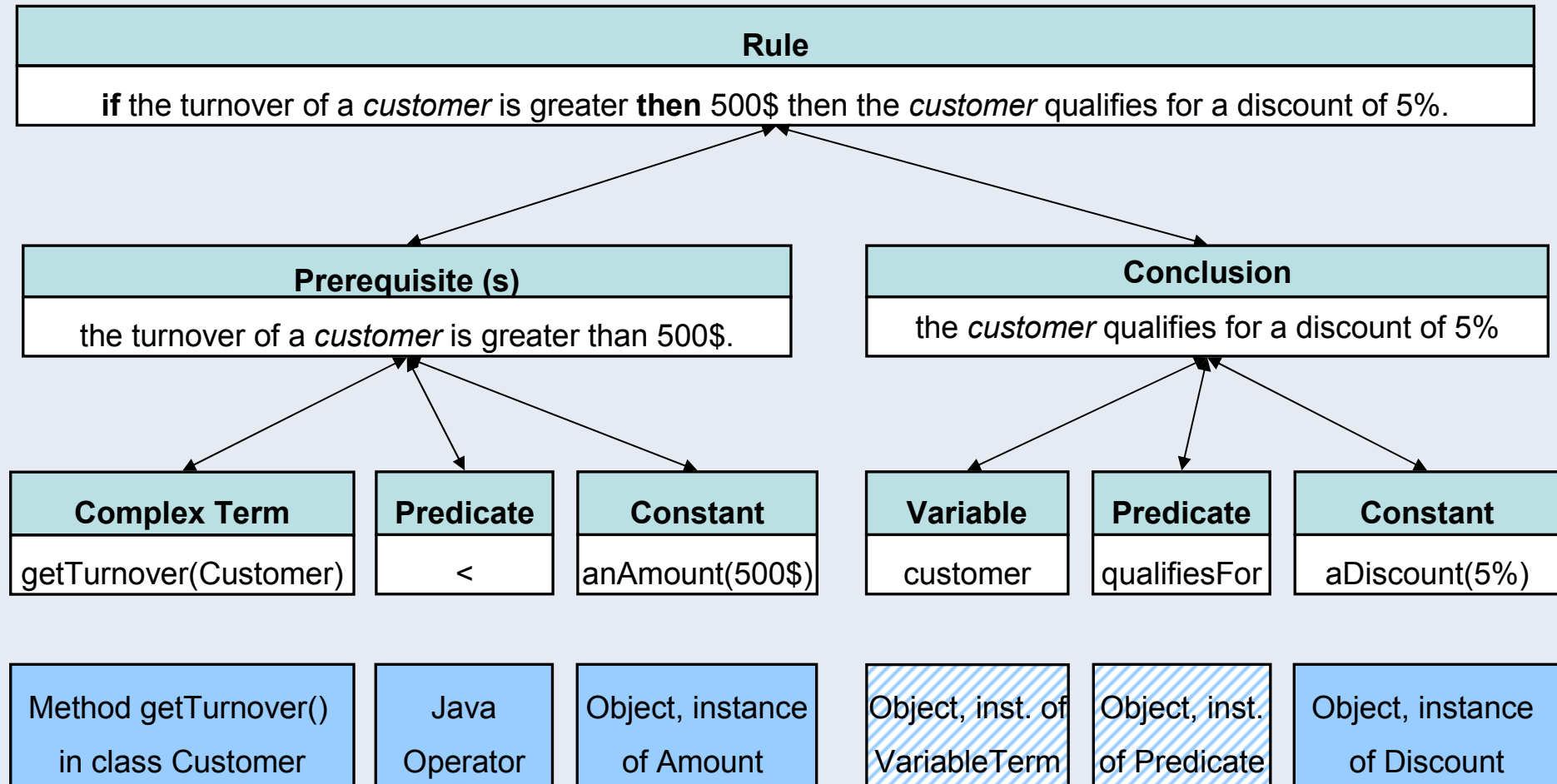
clauses:

- ! special risk for **DTE: HIGH** (reason: *UMTS debts.*)
- ! special risk for **IPHA: HIGH** (reason: *Change in mining legislation in Sou*)

Facts containing information about special risks associated with certain investments are maintained in a database and integrated in "real-time" (when the query is issued). This database can be remote (e.g., a service provided by Standard and Poor's or Moody's).

Design Objectives: Using the Java Object Model

Logic Concepts (Mandarax)

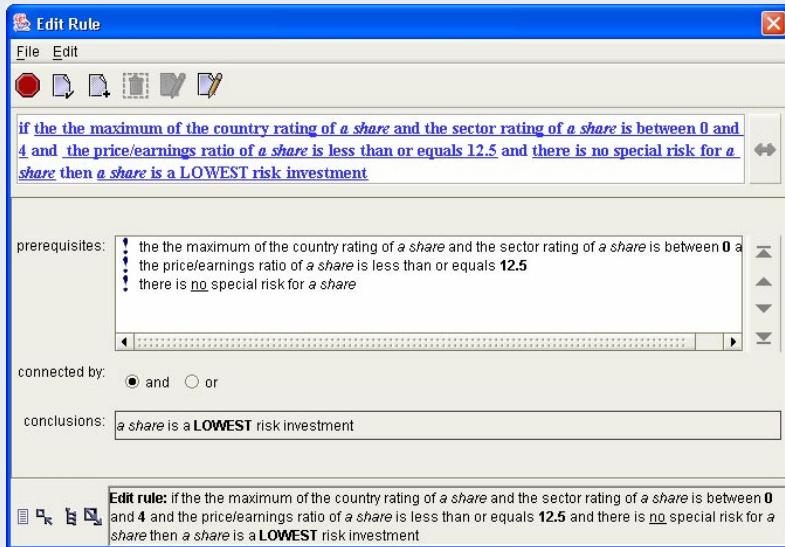


OO Concepts (Java)

Design Objectives: Avoiding Code Redundancy

- Objects are treated as constant terms in the sense of predicate logic.
- Java operators and methods can be used as predicates (like ==, <, equals(), ...).
- Java methods can be used as functions (like toString, getTurnover(), ..).
No code redundancy: existing functionality is reused, mandarax focuses on high level rules.
- Interface to other semantic models possible, including SQLPredicates and SQLFunctions.

Example: Using SQL Statements as Functions



if the maximum of the **country rating of a share** and the sector rating of a share is between 0 and 4 and the price/earnings ratio of a share is less than or equals 12.5 and there is no special risk for a share then a share is a LOWEST risk investment

The “country rating of a share” function is defined using the SQL query

```
SELECT COUNTRY_RATINGS.RATING FROM COUNTRY_RATINGS, SHARES  
WHERE COUNTRY_RATINGS.COUNTRY=SHARES.COUNTRY AND  
SHARES.SYMBOL=?
```

If the variable is bound (e.g., the country is the Netherlands), the query is performed and the result record is used to evaluate the prerequisite. Database services like Joins and Aggregation can be used.

Design Objective: Iterator Based Query Interface

- Mandarax queries are issued to a rule base using an inference engine.
- The inference engine returns an iterator (result set cursor). This facilitates a “computation on demand” design.
- The knowledge (rules, facts) supporting the result are revealed as part of the API.
- The API is very similar to the JDBC query API. Therefore, existing JDBC knowledge within the organisation can be used. A native JDBC interface is available as well: the mandarax JDBC driver.
- With an iterator based interface, query results can be integrated as clause sets: knowledge bases can be clustered.

MetaLogic - Facilities

- Plug-in support for algorithms used (loop checking, unification).
- Comparators: priorities (for conflict resolution) can be assigned automatically. E.g., policies like “prefer rules with more prerequisites” or “prefer facts over rules” can be set.
- Result set filters: results sets can be filtered, filters are themselves result sets (process is transparent for application, similar to database views).
- Examples for filters include: identify results with same substitutions but different derivations, prefer results based on a shorter derivation. Using result set filters is an alternative approach to handle conflicting rules / knowledge.

On Persistency

- RuleML is an XML-based language for rules.
- Mandarax was the first application supporting RuleML.
- ZKB is based on RuleML. Objects are represented by IDs. Object references are resolved by a naming service (interface resembles JNDI) called Object Persistency Service (OPS).

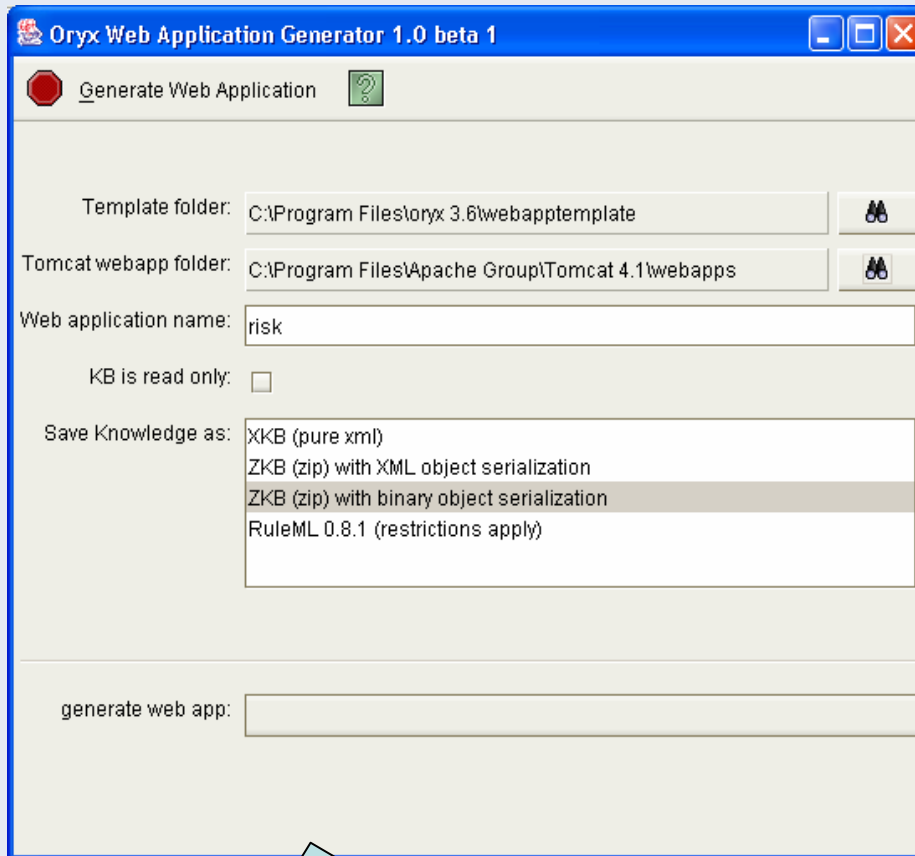
On Standards: JSR-94

- JSR-94 Java Rule Engine API is a Java Specification Request. Participating team members are BEA, Fujitsu, IBM, ILOG and Oracle.
- The API is very abstract, and does not contain a detailed specification of the data structures used.
- Therefore, migration between two JSR94-compatible rule engines is expected to be relatively expensive (compared with standards such as JDBC).
- Mandarax – JSR94 layer currently implemented by contributors from Top Logic.

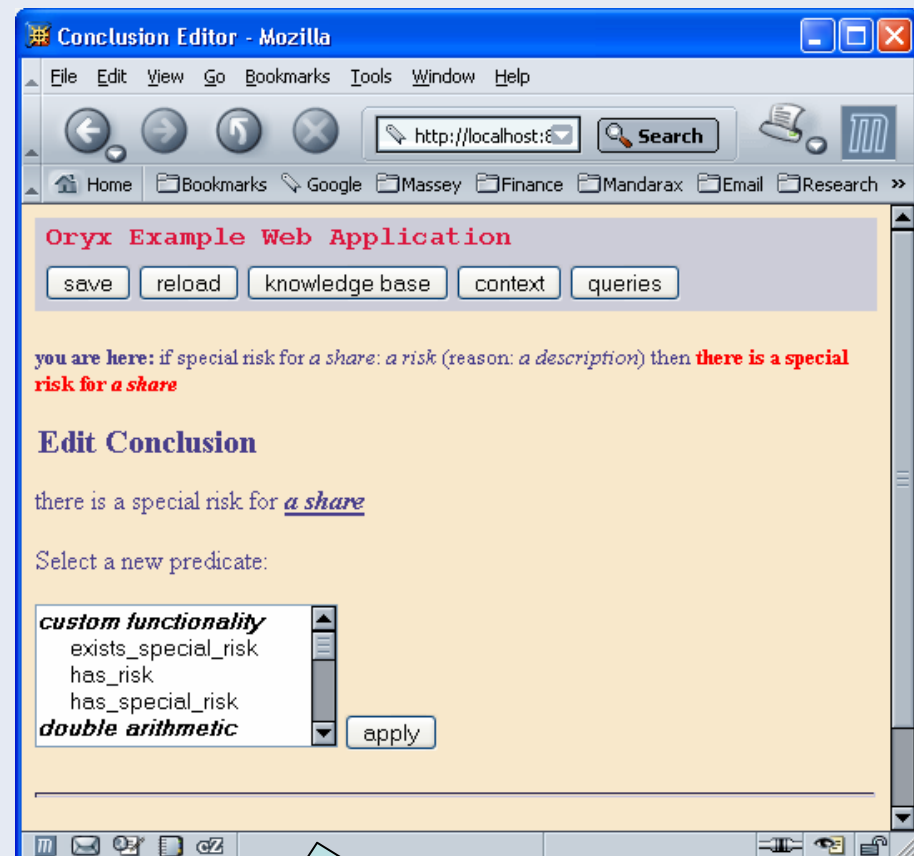
Deployment of Rules

- Mandarax comes with two JDBC driver implementations: a local and a type 4 network driver .
- The network driver has an open architecture to configure transport protocol and encoding. By default, the encoding is XML serialisation, and the transport used is http (based on the Apache Http client).
- Therefore, a knowledge base can be deployed as **knowledge server** on a standard web or application server, and applications can query the knowledge base in a proxy-friendly fashion.
- The Oryx web interface application can be used to maintain the knowledge server using a standard web browser.

Example: Web-Deployment of the Portfolio Risk Example



Deployment of knowledge base as web application (using a Tomcat-compatible Webserver)



Rules are maintained using a standard web browser.

Example: Client Connecting to the Knowledge Server

```
// 1. load mandarax jdbc driver
Class.forName("org.mandarax.jdbc.DriverImpl");

// 2. initialize connection
String url = "jdbc:mandarax:net:zkb:/risk.zkb@" +
            "http://localhost:8080/mandarax-server";
Connection con = DriverManager.getConnection(url);

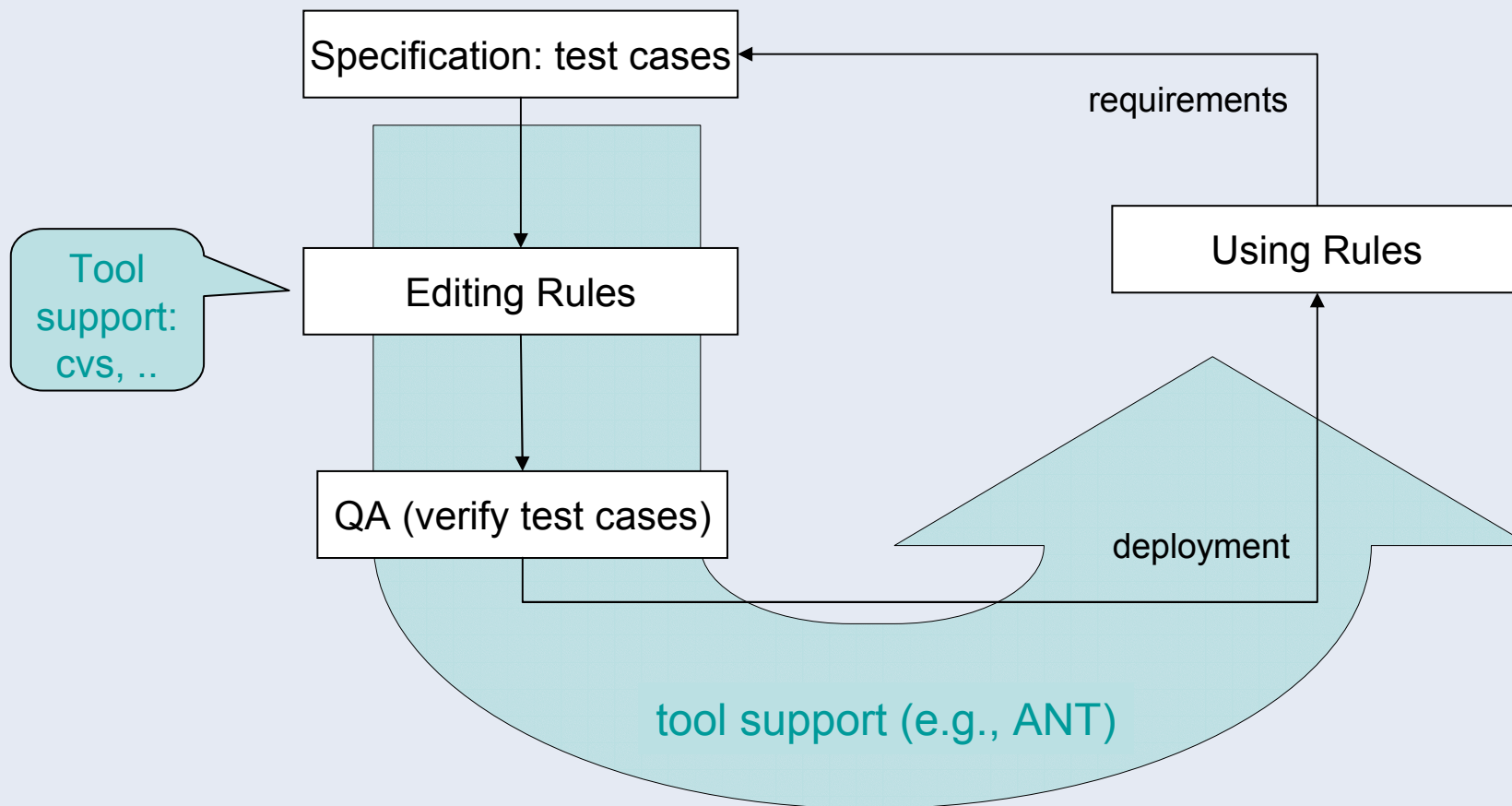
// 3. Issue query
String sql = select * from has_risk where share='IPHA';
ResultSet result = con.createStatement().executeQuery(sql);
Object risk = result.getObject("risk");

// 4. clean up
result.close();
```

Standard JDBC code (like querying Oracle), URL references web server.

Rule Lifecycle

- Editing rules resembles an iteration in the software development lifecycle.
- Expected outcomes can be captured as / translated into (JUnit) test cases by the rule author.
- Updated rules are verified against these test cases and deployed.



Mandarax Projects

- **Chameleon** – Classical Web-based Expert System to support farmers in combating bush encroachment. Can be used in other problem domains as well, open source (MySQL, Apache Tomcat, project itself hosted as sourceforge)
- **WAFER** (Long Term Care Regulatory Web Accessible Facility EnRollment (WAFER))– Commercial Application by Government of Texas. Nominated by Government of Texas for NASCIO* award.
https://www.nascio.org/awards/2003awards/digGovt_GtoB.cfm, G2B category.
- More commercial applications: Kleinwort Benson, Bauer & Partner, Digital Sesame, Erry, China Telecom Guangzhou R & D Center.

The Pros and Cons of Mandarax/Oryx

- **Pros:**
 1. Improved feedback – user sees not only the result of applying the rules, but the rules supporting the result can also be visualized (“the application computed this result based on the following rules: ...”).
 2. Standard software interfaces (e.g., JDBC) and tools can be used, uses existing skills (no new PL !) and existing implemented functionality (methods, stored procedures etc).
 3. Rule engine can compute different alternative solutions.
- **Cons:**
 1. No full vendor support is available.
 2. Performance may be insufficient for large-scale applications.

Usage Scenarios

- Project with other open source tools (Tomcat, JBoss).
- Projects with query based reasoning, with focus on semantic web technologies or with large fact bases in databases.
- Knowledge bases which must be queried with standard clients through proxies.
- Prototyping, with the option to migrate to commercial product later. Standards (RuleML, JSR-94) facilitate this approach.

Project History

- First Mandarax like prototype 1998 (Smalltalk, commercial application for Mercedes Benz)
- Java Open source project started in 2000.
- Project supported by Swiss/German Bauer&Partner group.
- Oryx (graphical user interface) extension started in 2001.
- Various commercial and academic applications. Users include Kleinwort Benson, Government of Texas, City University London, German Institute of Artificial Intelligence (DFKI).

Project Organisation

- Hosted on Source Forge: <http://mandarax.sourceforge.net>
- Uses agile software engineering approach: ANT as project backbone, and test driven development using JUnit (currently 800+ test cases).
- 10 people have contributed to Mandarax, 4 active “committers” (people who can commit code into the CVS) from Massey University, Bauer/Partner TopLogic, SUN Microsystems, TU Munich.

Code Organisation

- Original idea: provide an abstract API layer with adhoc implementation to be replaced later.
- The abstraction layer contains interfaces, abstract classes and exceptions.
- The reference layer contains implementation classes. Instantiation is done using factories.
- Services (e.g., persistency) reference abstraction layer and are therefore available for alternative implementations.

Future Plans

- Support for future versions RuleML.
- Integration with leading Ontology tool Protégé, improved support for RDF.
- Add support for unit tests to the mandarax core (to facilitate rule validation).
- Generating repositories and rules from UML and ORM models.
- RDBMS based persistency.
- Compile mandarax knowledge bases into Java bytecode.