Ontology Driven Software Development with Mercury

Michel Vanden Bossche, Peter Ross, Ian MacLarty, Bert Van Nuffelen, Nikolay Pelov

Melbourne – August 14th, 2007

Based on SWESI '07 paper “Ontology Driven Software Engineering for Real Life Applications”
Outline

1. Motivation and History
2. Architecture Overview
3. OWL
4. Mercury
5. OWL → Mercury (Hedwig)
6. Use Case: eInsurance Application
The Company at a Glance

- **Mission Critical**
  - **What** Software Consultancy Firm
  - **Who** Software Engineers with a formal CS background (MSc, PhD)
  - **When** Founded in 1993
  - **Where** Brussels (Belgium) and Melbourne (Australia)

- **Origins** Logic Programming (BIM Prolog) and Open Systems
- **Vision** Much better CQFT\(^1\) requires a **Paradigm Shift** in SE
- **Products** Business-Critical Customer-Facing Applications
- **Customers** Information Intensive Companies

---

\(^1\) Cost, Quality, Flexibility, Time
Motivation

Software Development Hard

- Hard to write correct software
- Often a difference between what the client wants and what the programmer thinks the client wants
- Hard to maintain software as specs change
- Hard to deliver software predictably in terms of cost and time
GAP Between Users and Programmers
Using Ontologies to Help Bridge the Gap

- Business Users
  - Informal Knowledge (Expert's Brain)
- Ontologies
  - Formal Description (OWL)
- IT
  - Executable Program (Mercury)
Benefits of OWL as a Modelling language

- Business feels **more involved in project**
- Makes **requirements explicit**
  - Business people understand better the complexity of their domain
  - Better time and cost estimates
  - Early feedback, helps with project management
- **Simple Formal semantics**
  - Provide an unambiguous “contract” between Business and IT
- **Long Term Business Asset**
  - **Ontologies** not tied to a particular technology
  - Knowledge **not lost in code**
- **W3C Standard**
**OWL**

- **Web Ontology Language**
- **Formal Description of a Domain**
  - **Classes** (sets of individuals)
    - Class Toys
  - **Individuals** (elements of classes)
    - http://toys.com.au/toys.owl#buzzLightYear is an element of Toys
  - **Properties** (binary relations)
    - number_of_batteries(buzzLightYear, 2)
    - married_to(harry, sally)
  - **Datatypes** (XML Schema)
    - string, float, int, 1..10
OWL Classes

- SubClass Hierachy (subset relations)
- Union, Intersection, Complement
- Can assert individuals are members of Classes

Example
- Class **ElectronicToys**
  - **ElectronicToys** is a subclass of **Toys**
  - Individual **buzzLightyear** is a member of **ElectronicToys**
  - **AnnoyingElectronicToys** is the intersection of **AnnoyingToys** and **ElectronicToys**
OWL Properties

- **Domains must be a class**
- **Ranges can be a Class or a Datatype**
  
  **Examples**
  - Property `designer` has domain `Toy` and range `Person`
  - Property `number_of_batteries` has domain `ElectronicToy` and range `positive integer`

- **Cardinality constraints**

  **Examples**
  - Each `Toy` should have at least one `designer` (but maybe more)
  - Every `ElectronicToy` should have exactly one value for their `number_of_batteries` property
OWL Properties (cont.)

- **Range constraints**
  
  Examples
  
  - Any OldToy should have a manufactured_year of less than 1960
  - At least one designer of a Toy should be a member of the class ImaginativePerson

- **Transitive, Symmetric, Functional, Inverse Functional, InverseOf**

  Examples
  
  - older_than is a Transitive property
  - married_to is a Symmetric property
  - wife is the inverse of husband
Limitations of OWL

- Not widely spread and not well-known (although gaining traction)
- Open world assumption makes working with negation and aggregation difficult
- OWL does not assume unique names, which complicates reasoning (we have adopted UNA)
- Limited expressiveness, although can be extended with SWRL

So far, expressive enough in practice
Using Ontologies to Help Bridge the Gap

- Business Users
- Informal Knowledge (Expert's Brain)
- Formal Description (OWL)
- Executable Program (Mercury)
- Ontologies
- IT

2007-08-14
Requirements for a Mercury – OWL API

- Ontologies should be integrated into the build system for the application. **Should not just be passive documentation.**
- **Compile-time errors**, not runtime errors (like a lot of RDMS APIs that use SQL query strings).
- Spec changes → Code changes
- Mercury has a lot of compile-time checking features which we can exploit.
Mercury

- Developed at Melbourne University
- Logic Language with similar semantics and syntax to pure Prolog
- Added benefits of strong type, mode and determinism systems
- Module system
**Pro**
- Good engineering tool for developing large-scale robust applications
- Many compile time-checking features
- Efficient

**Cons**
- Not widely known, therefore difficult to sell
- Requires experts to maintain, perceived as risky

**Try to ease client's fears by**
- Coding business logic in OWL, a W3C standard
- Writing domain specific interpreters for the ontologies in Mercury
Generate binary predicates for properties (after inferring all entailed facts from ontology):

```prolog
:- pred number_of_batteries(uri, int).
number_of_batteries("buzzLightYear", 2).

:- pred designer(uri, uri).
designer("buzzLightYear", "janet").
designer("barbie", "sarah").
designer("lego", "harry").
```
For each class we generate an inst:

:- inst 'Toys'
    ---> "buzzLightYear"
    ; "barbie"
    ; "lego".

:- inst 'ElectronicToy'
    ---> "buzzLightYear".

:- inst 'EducationalToys'
    ---> "lego".
Mercury API for OWL (cont.)

- We use these insts in the mode declarations of the predicates.
- Mode declarations give information about how a predicate can be called.
- Determinism comes from cardinality restrictions.

```prolog
:- mode number_of_batteries(in('ElectronicToy'), out) is det.

:- mode designer(in('Toy'), out('Person')) is multi.

:- mode designer(in('EducationalToy'), out('Teacher')) is det.
```
Mercury API for OWL (cont.)

- For classes we also generate a unary predicate:

  ```prolog
  :- pred 'Toy'(uri).
  :- mode 'Toy'(ground >> 'Toy') is semidet.
  :- mode 'Toy'(out('Toy')) is multi.
  
  'Toy'("buzzLightYear").
  'Toy'("barbie").
  'Toy'("lego").
  ```
Example Code

- Some example code using the API:

```prolog
:- pred fulfill_order(uri::in('Item'), ...) is det.

fulfill_order(Item, ...) :-
  ( if 'Toy'(Item) then
    ( Item = "barbie",
      ... code for ordering barbie ...
    ; Item = "lego",
      ... code for ordering lego ...
    ; Item = "buzzLightYear",
      number_of_batteries(Item, Batteries),
      ... code for ordering buzz with batteries ...
    )
  else
    ... code for ordering other items ...
  ).
```
Actual API a bit more complex, because...

- No empty inst in Mercury, so this only works for non-empty classes. Most classes will be empty in initial development stage.
- Subtype insts not supported very well in Mercury standard library.
- Some classes and properties may change at runtime.
Real API

- Abstract type for each OWL class
- Typeclass for each OWL class
- Functions for converting between type and uri of the right instance
- Casting predicates
- "snapshot" argument for classes and properties that change at runtime.

```prolog
:- type 'Toy'.
:- typeclass 'Toy'(T).
:- instance 'Toy'('Toy').
:- instance 'Toy'('ElectronicToy').
:- pred designer(T::in, 'Person':out) is multi <= 'Toy'(T).
```
Non-Toy Application

- **What?**
  - eInsurance, “Non-Life”, Business Transaction at Point of Sales
  - 4000+ Brokers, Agents, Partners, Clients
  - Key selling point: fully dynamic “Shopper Screen”
  - Maximize “Straight Through Processing” ⇒ Many rules
  - Dynamic roles, powers, preferences
  - Reuse back-ends systems for some back-office functions

- **Key Development Constraint**
  - Only *35% of requirements* known at kick-off
Result

- All requirements accepted (Shopper Screen refused by others)
- **OWL, RDF, Mercury, DSL Interpreter** (Rules), **AJAX UI (XUL)**...
- **Semantic Service Broker** based on **OWL-S** for back-ends
- **Scalable** stateless application engine, < 3 sec response time
- **Portable**: Windows, Linux, Unix, MacOS
- Development team: 10 (MC) + 2 (Customer)
- Completed in 1/3 person-months (p.m) of the next closest quote
- Completed in 1/3 p.m for a similar application (1.5 MLOC of Java)
- **45 KLOC** (program), **212 classes** + **40 K instances** (ontology)
Running Application
Questions & Comments