Persistent Constraint Contextual Logic Programming

Salvador Abreu
Vitor Nogueira
Universidade de Évora

Overview

- Questions pertaining to applied Logic Programming and a proposed approach
  - Program and computation structure
  - Persistence
- Experience & Future work

UNIVERSIDADE DE ÉVORA

Need for Modularity

- Programming in-the-large
  - collaborative development
  - well-defined interfaces
  - independent components
- Prolog
  - compact
  - efficient unification over complex terms
  - doesn't scale well

Prolog Modules

- Several approaches
  - Quintus/SICStus modules
  - Logic & Objects
  - Lambda-Prolog
  - etc.
- ISO standard
  - slow in the making
  - delicate issues...

ISO standard "Part 2"

- a.k.a. "ISO Modules"
  - complex compromise
  - multiple-implementation origin
  - syntactic & semantic heavyweight
    - doesn't really take advantage of Prolog's strengths, e.g. unification / Logic variable & code compactness
- suffer from Prolog's worst weakness:
  - lack of consensus

Contexts

- Program structuring mechanism
- unit = a parametrized named set of predicates
  - similar to a module
- context = an ordered collection of qualified units
  - combination of units
  - similar to a theory (set of predicates)
  - attribute of computations

Conte… …ivial Example

- A “conte… …ello Worl…

```
:- unit(hello(WHAT)).

…h(L) :-
    length(WHAT, L).
```



```
?- hello(“world”) :> length(L).

L = 5
```

Contextual Logic Programming

- Computation attributes
  - Current Context
  - Calling Context
- Rules
  - Prolog & Context Goal Expansion (reduction)
  - Context Extension & Switch
  - Context Traversal
  - Context & Calling Context Enquiry

Context?

- List of qualified units
  - each defines a set of predicates
  - predicates share variables with respective unit arguments
- goal-in-context
  - reduced (proven) in the theory formed by the qualified union of predicates in the context

Contextual Logic Programming Summary

Goal	Name	Description
U :> G	Context extension	Evaluates G in the context [U|C]
R :< G	Context switch	Evaluates G in the context R
:< R	Context enquiry	Unifies R with C
:^ G	Super	Evaluates G in the context TAIL(C)
:# G	Lazy call	Evaluates G in the calling context
:> R	Context	Unifies R with the calling context

Context Search Process

```
let P=predicate(G)
let CC=C

repeat:
 let C=[U|T]
 if U defines P then
  return C
 else
  C := T
```

Simple example

```
:-unit(u).
  p(1).
  p(2).

:-unit(v).
  p(a).
```

```
?- u :> v :> p(X).
X = a

?- v :> u :> p(X).
X = 1
X = 2
```

```
?- [v, u] :< p(X).
X = a

?- [u, v] :< p(X).
X = 1
X = 2
```

Parametric example

```
:-unit(u(P)).
  p(X) :- q(X).
  p(s(X)) :- p(X).

  q(P).
```

```
:-unit(v(X)).
  q(f(X)).
```

```
:-unit(u2(P)).
  p(X) :- :# q(X).
  p(s(X)) :- p(X).

  q(P).
```

```
?- u(z) :> p(X).
X = z
X = s(z)
X = s(s(z))
...
```

```
?- u(z) :> v(y) :> p(X).
X = z
X = s(z)
...
```

```
?- u2(z) :> v(y) :> p(X).
X = f(y)
X = s(f(y))
...
```

```
?- u2(z) :> p(X).
X = z
X = s(z)
...
```

Contextual Logic Programming

- Attractive because
  - lightweight incremental change to Prolog
  - proper implementation design favors efficiency
- Language issues
  - unit composition wrt predicate definitions?
  - contexts opaque or 1st class entities?
  - “lazy” vs. “eager” call?

Contexts as Objects

- Parallels with OO terminology
  - Object / Context
  - Method / Predicate
  - Message / Goal
  - Inheritance (is-a) / Context structure
  - Inclusion (has-a) / Arguments

Contexts as Objects

- Patterned units & coding style
  - Access Predicates
  - Structured Contexts
  - Access to “subclass instance” context

UNIVERSIDADE DE EVORA

Programming with contexts

```
:-unit(point(X, Y, Z)).

point(X, Y, Z).
x(X).
y(Y).
z(Z).

item :- «instantiate»(X, Y, Z).
```

- access predicates
  - for the entire unit
  - for specific args
- generators:
  - argument-less predicate (item/0)
  - binds unit variables
  - “database” predicates

Programming with contexts

- Structured Contexts
  - Unit arguments may contain other contexts
  - Context switch operator :< disregards current context
  - Models containment ("has-a" in OOP)

```
:- unit(line(P1, P2)).

line(P1, P2).
point1(P1).
point2(P2).

item :- P1 :< item, P2 :< item.

length(L) :-
    P1 :< (x(X1), y(Y1), z(Z1)),
    P2 :< point(X2, Y2, Z2),
    L is sqrt((X2-X1)^2 +
              (Y2-Y1)^2 +
              (Z2-Z1)^2).
```

Glass Box Approach

- New Context Operators can be defined
- Example: alleviating the need to remember how many parameters a unit has?
  - define a new operator, e.g. <u>:/></u>
  - use it as <u>:></u>
  - non-contextual definition (i.e. this is a regular Prolog predicate, not defined in any unit)

```
:- current_op(X, xfy, ':>'),
   op(X, xfy, ':/>').

A :/> G :-
  current_unit(A, U),
  U :> G.
```

Programming with contexts

```
:- unit(where(CONDITION)).

item :- :^ item, CONDITION.
```

- A common situation: filtering
  - dynamically create a context which specifies complex conditions
  - build on top of previous contexts
  - activate it with item/0

```
person :/> (
  age(A),
  where(A > 17) :> :< CX),
...
CX :< item,
...
```

Context examples

- web session
  - a given page specifies a context for each possible subsequent pages in the session
  - topmost unit indicates possible actions (the defined predicates)
- logic symbol table
  - association list or other data structure

A Prototype Implementation GNU Prolog/CX

- simple WAM extension
- high performance
  - approximately 20% overhead
  - insignificant loss with very deep contexts

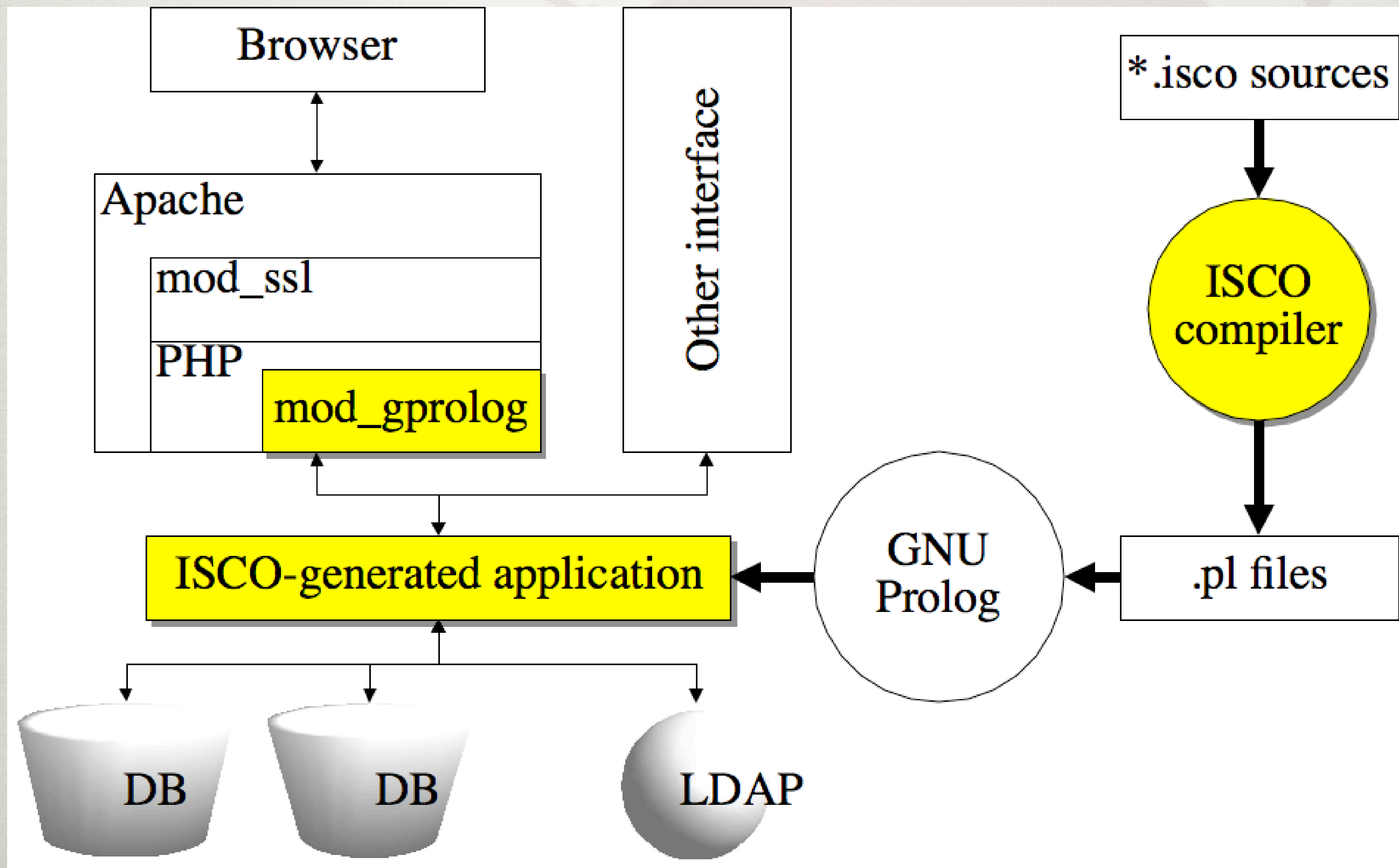
Persistence

- Logic Programming
  - expressiveness as a result of declarativity
  - built-in search
  - integrated with constraints (ala CLP)
- RDBMS
  - close to 1st order logic
  - built-in persistence
  - immediate sharing of changes

Prolog vs. RDBMS

- Updates
  - more diversity of operations in RDBMS
    - insert/delete/update vs. assert{a,z} & retract
  - sharing in a concurrent execution setting
  - dynamic code is generally worse than static code
- Indexing
  - more general (DBMS)
  - faster in particular cases (Prolog)

ISCO: THE BIG PICTURE



ISCO

- RDBMS access
  - tables as Prolog predicates
  - tuple-at-a-time
  - implements inheritance (ala object-relational)
  - convenience features
    - non-positional arguments
    - default values
    - ...
- Allows for updates (insert/delete/update)

UNIVERSIDADE DE ÉVORA

Constraints in ISCO queries

- ISCO was designed to have several back-ends
  - SQL is the most relevant one (currently)
    - simple dynamic query generation
    - based on current variable bindings & FD constraints
    - desirable to be able to "see" constraints other than min/max for specific variables

Contexts and ISCO

- Purpose
  - discipline access to ISCO predicates
  - improve program structure
    - modularity
    - abstraction
- Means
  - associate each ISCO class with one unit
  - the unit is parametrized with the class arguments

Combined Example

- typical access predicates
- generator predicates (item/0) are ISCO
- inheritance (both ISCO and cx)

```
:-unit(person(ID, NAME, BIRTHDATE)).

id(ID).
name(NAME).
birthdate(BIRTHDATE).

person(ID, NAME, BIRTHDATE).

item :- person_gen(ID, NAME, BIRTHDATE).
```

Conclusions

- GNU Prolog/CX good for ISCO
- Contexts are a useful way to design and code web-based IS
- Scaled well in an application lifecycle

Future work & WIP

- Integration into CMS (Plone, Mambo, etc.)
- XML (OWL, Xquery, ...) front and back-ends
- Smarter SQL generation
- Semantic integration of multiple sources: support tools (use contexts as theories)
- ISTO: integration of Temporal aspects

Obrigado 8-)

- Mais informação em:
  - http://www.di.uevora.pt/~spa/
  - http://www.di.uevora.pt/~vbn/
  - http://dev.si.uevora.pt/isco/
- Contactos
  - spa@di.uevora.pt
  - vbn@di.uevora.pt

UNIVERSIDADE DE ÉVORA