Web Logic Programming

Informatics System Design LS

Dott. Ing. Giulio Piancastelli

giulio.piancastelli@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008

Outline

Part I

Background

What is the World Wide Web? (1/2)

A proper description, understanding, formalization and divulgement of the World Wide Web architectural principles and design criteria has been achieved only recently by

- the Representational State Transfer (REST) architectural style for distributed hypermedia systems [Fielding, 2000]
- the Resource-Oriented Architecture (ROA) as a REST-based set of guidelines and practices for creating services on the Web [Richardson and Ruby, 2007]

What is the World Wide Web? (2/2)

The World Wide Web is a network-based application with specific *architectural style* and *computation abstractions*

Architectural style

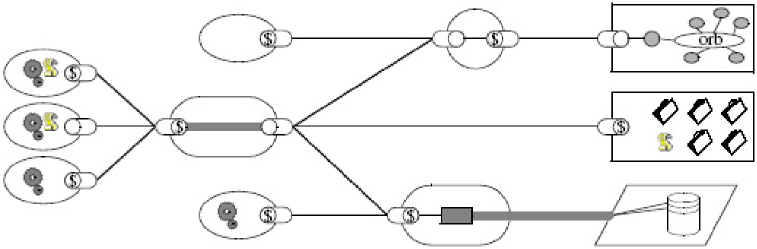
An architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style

Abstractions

The Web features 3 different kinds of abstractions

- data
- connectors
- components

The Architecture of the World Wide Web



The Representational State Transfer style is composed by the following styles: client-server, stateless, cache, uniform interface, layered system, code-on-demand

Computation Abstractions: Resources

REST focuses on the *resource* as the main data abstraction, defined as any conceptual target of a hypertext reference. Resources have

- a name
- data representing their state
- behavior to perform actions such as changing their state, building up their representations, managing interaction with other resources

REST prescribes communication amongst resources to happen through a *uniform interface* by transferring a *representation* of the current state of a resource

Computation Abstractions: Connectors and Components

Other REST abstractions include

connectors	providing a generic interface for accessing and manipulating a resource
components	encapsulating the activities of accessing resources and transferring resource representations

The Web as a RESTful Hypermedia System

REST resources become World Wide Web resources when

- resource names are defined by the URI standard [Berners-Lee et al., 1998]
- resource uniform interface and communication protocol are prescribed by the HTTP standard [Fielding et al., 1999]
- the data format of resource representations is defined by MIME types [Freed and Borenstein, 1996a, Freed and Borenstein, 1996b]

Resource-Oriented Architecture Addressability

- URIs should be descriptive
- URIs should have a definite structure varying in predictable ways
- Each name encompasses the names of other resources, and ultimately the name of the resource associated with the domain at the URI root

Bookshelf Sharing example

The name of a book on the `shelf` of the `jdoe` user

```
http://example.com/users/jdoe/shelf/5
```

encompasses the names

```
http://example.com/users/jdoe/shelf
http://example.com/users/jdoe
http://example.com/users
http://example.com
```

What is Logic Programming?

Logic programming is the use of logic as both a declarative and procedural representation language [Kowalski, 1974]

Horn clauses

Logic programming is based on a subset of first-order logic expressed by *definite Horn clauses*, that is disjunctions of literals having exactly one positive literal, taking the form

$$L_1, \ldots, L_n \Rightarrow L \equiv \neg L_1 \vee \ldots \vee \neg L_n \vee L$$

Resolution

Logic programming uses *SLD resolution* as the basic inference rule to

- demonstrate the satisfiability of a set of clauses, or equivalently
- compute the solution to a query on a given set of relations

Modular Logic Programming

Basic logic programming does not embody mechanisms helping to master the complexity of realistically sized applications

- A logic program consists of a flat set of clauses
- Logic programming lacks abstractions for structuring and modularizing programs

A number of different approaches to modularizing logic programming have been proposed [Bugliesi et al., 1994], for example

Modular Logic Programming defines an algebra of modules over a small number of operations which are formalized in terms of the basic logic programming semantics [Brogi et al., 1994]

ISO Prolog augment the most widely used logic language with constructs for declaring and using modules, much in the spirit of "conventional" programming paradigms [SC22, 2000]

Contextual Logic Programming

Contextual Logic Programming [Monteiro and Porto, 1993] is an extension of logic programming, relying on the possibility of defining systems consisting of separate logic programs to be combined together, where

- programs are structured as sets of predicates called *units*, which can be dynamically combined in an execution *context*
- matching predicates in goals are to be located in all the units which make up the current execution context

The latest implementation of a Contextual Logic Programming system is GNU Prolog/CX [Abreu and Diaz, 2003] publicly released in 2006

Object-Oriented Logic Programming

Object-Oriented Logic Programming [McCabe, 1992, Omicini, 1995] extends the model of Contextual Logic Programming trying to comprise typical object-oriented concepts such as

- encapsulation
  - information hiding
  - visibility
- inheritance
  - extension
  - overriding

Sometimes implementations (e.g. GNU Prolog/CX) directly manipulate contexts as if they were objects [Abreu and Diaz, 2003]

Logic Programming on the Web: State of the Art

Research on the significance of logic declarative languages in the specific application domain of the World Wide Web focussed on three main themes

- the provision of libraries, such as PiLLoW, to manipulate HTML and XML documents and to operate with the HTTP protocol [Cabeza and Hermenegildo, 2001]
- the "merge" between agent- and web-based technologies resulting in so-called Internet agents [Denti et al., 1997]
- the representation of information in the form of *logic pages*, as promoted, for instance, by the LogicWeb language and system [Loke, 1998]

However, a resource programming model is still missing

PiLLoW Programming Example

```
:- include(library(pillow)).
main(_) :-
    get_form_input(Input),
    get_form_value(Input, person_name, Name),
    response(Name, Response),
    output_html([
        'Content-type: text/html\n\n',
        html([title('Telephone database'),
              img$[src='phone.gif'],
              h2('Telephone database'),
              hr$[],
              Response])]).
response(Name, Response) :-
    form_empty_value(Name) ->
      Response = 'You have to provide a name.'
    ; phone(Name, Phone) ->
      Response = ['Telephone number of ', b(Name), ': ', Phone]
    ; Response = ['No telephone number available for ', b(Name)].
% A series of phone/2 facts follows...
```

LogicWeb Programming Example

```
<HTML>
<HEAD><TITLE>Seng Wai Loke's Home Page</TITLE></HEAD>
<BODY>
I'm from the Department of Computer Science</A> at the
<A HREF="http://www.unimelb.edu.au/">University of Melbourne</A>.
<!--
<LW_CODE>
interests(["Logic Programming", "AI", "Web", "Agents"]).
friend_home_page("http://www.cs.mu.oz.au/friend1/").
friend_home_page("http://www.cs.mu.oz.au/friend2/").
interested_in(X) :-
    interests(Is), member(X, Is).
interested_in(X) :-
    friend_home_page(URL),
    lw(get, URL)#>interested_in(X).
</LW_CODE>
-->
</BODY>
</HTML>
```

Part II

Abstraction

The Abstraction Problem

Most programming models for World Wide Web applications has been based on different abstractions than resources

page	dealing with the concept of resource representations
controller	dealing with a programming framework architectural abstraction

Most of the procedural and object-oriented abstractions that have been used to program the Web have shown different shortcomings

modules	are more of an organizational feature than a computation abstraction
objects	principles (e.g. inheritance) and patterns (e.g. MVC) can not be properly mapped onto resources

Web Resources

Resources have:

- a name, i.e. an URI
- data representing their state
- behavior to perform actions such as changing their state, building up their representations, managing interaction with other resources through HTTP

Web Logic Resources

Names

Mapped onto logic atoms, much as it happens for logic module names

Data

Represented by logic facts

Behavior

Encoded in logic rules

From Resources to Contexts

The resource naming structure, e.g.

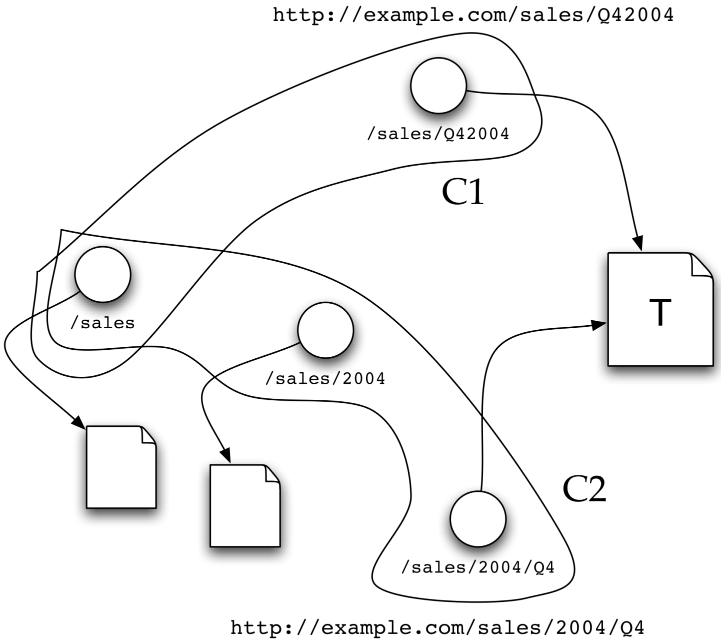
`http://example.com/sales/2004/Q4`

suggests that each resource does not exist in isolation, but lives in an information *context* composed by the resources associated to the names encompassed by the name of that resource

Since more than one name can identify the same resource, the context of a resource has to be associated with its name rather than directly with the resource itself

Sales example

```
http://example.com/sales/2004/Q4
http://example.com/sales/Q42004
```


http://example.com/sales/Q42004
/sales/Q42004
C1
/sales
/sales/2004
T
C2
/sales/2004/Q4
http://example.com/sales/2004/Q4

Web Logic Resources, revisited

The locus of computation is no more the single logic program representing a resource: the notion of context is introduced in the World Wide Web domain

Definition

Given a resource R with a name $N(R)$ such that

$$N(R) \subseteq N(R_1) \subseteq \ldots \subseteq N(R_n)$$

then, the associated context $C(R)$ is generated by the following composition

$$C(R) = T(R) \cdot T(R_1) \cdot \ldots \cdot T(R_n)$$

Implicit Web Logic Resources

In addition to web resources associated with a URI, we have identified four *implicit* resources corresponding to special entities that are part of the context of any application resource

- the environment resource R_E (identified by the special atom `environment`) representing the environment where the web application lives
- the application resource R_A (identified by the special atom `application`) representing the application itself
- the user resource R_U (identified by the special atom `user`) representing an application's user
- the session resource R_S (identified by the special atom `session`) representing an interaction session of a user with the web application

The generation of the context associated to a resource R becomes

$$C(R) = T(R) \cdot T(R_1) \cdot \ldots \cdot T(R_n) \cdot T(R_S) \cdot T(R_U) \cdot T(R_A) \cdot T(R_E)$$

Web (Logic) Resources vs (Logic) Modules

Modules are an organizational linguistic feature, while resources are a primary computation abstraction

- Resource Uniform Interface
- No arbitrary `import` or `export`
- Composition is already architecturally dictated

Web Logic Resources vs Logic Contexts

In Web Logic Programming, each resource has a context associated to each of its names, but these *web contexts* are somehow different from *logic contexts*

- Web contexts are isolated from each other
- Web context names are never parameterized
- No need for a dynamic web context augmentation is envisioned

Web (Logic) Resources vs (Logic) Objects

Foundational characteristics of object-orientation are missing from the resource abstraction in the World Wide Web system

- The resource abstraction does not bring any notion of inheritance with itself
- Paths are not hierarchies

Part III

Computation

The World Wide Web computation model

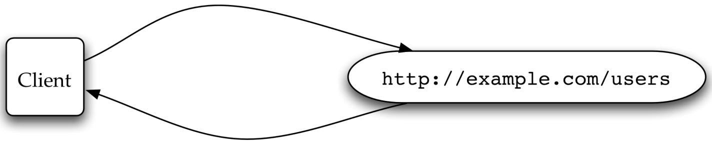
According to REST and ROA, the World Wide Web computation revolves around transactions in the HyperText Transfer Protocol

- HTTP [Fielding et al., 1999] is a document oriented protocol aimed at transferring *representations* of a resource current state
- HTTP requests contain two key computation elements

  method information indicates how the sender expects the receiver to process the request

  scope information indicates on which part of the data set the receiver should operate the method
- HTTP responses typically contain the representation of the target resource (new) state as the result of the computation

```
POST /users HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

name=jdoe&sex=f
```



```
HTTP/1.1 201 Created
Location: http://example.com/users/jdoe
```

Web Logic Programming Computation Model

A computation in logic programming is a deduction of consequences of a set of facts and rules defining relationships between entities

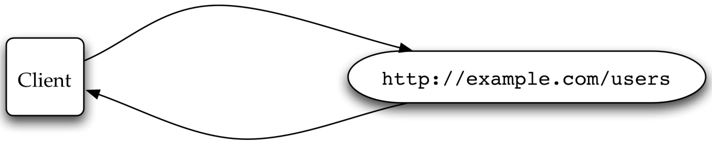
- Sets of facts and rules are called *logic theories*
- *Logic queries* are used to trigger the application of deduction rules on a theory

Each HTTP request gets translated to represent a deduction by

- retaining the request *scope information* to indicate the target *logic theory*, and
- mapping the request *method information* onto a *logic goal*

The computation takes place in the *context* associated to the resource target of the request. The information resulting from goal solution is translated to a suitable *representation* in order to be sent back as the payload of the HTTP response

```
POST /users HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

name=jdoe&sex=f
```



```
HTTP/1.1 201 Created
Location: http://example.com/users/jdoe
```

Goal

```
'http://example.com/users' : post(Request,
                                  Response,
                                  View).
```

Computation

```
post(Request, Response, _) :-
    create_user(Request),
    param(Request, (name, User)),
    user_url(User, Url),
    header(Response, (location, Url)),
    status(Response, (201, created)).
```

Context Traversal

Being the context $C(R)$ the composition of a number of theories, the computation is carried on so that the query G is asked in turn to each theory

- the goal fails if no solution is found in any theory
- the goal succeeds as soon as it is solved using the knowledge base contained in a theory $T(R_i)$

When the goal G gets substituted by the subgoals $S_j(G)$ of the matching rule in the theory, the computation proceeds from the context of the resource R_i rather than being restarted from the original context

Goal

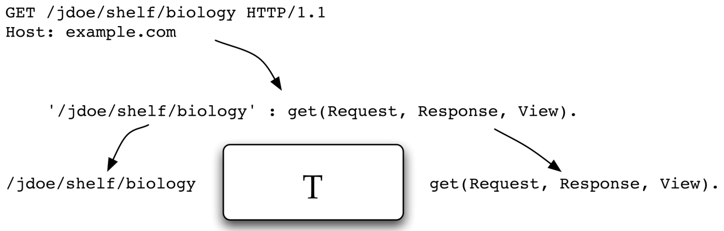
```
'/jdoe/shelf/biology' : get(Request, Response, View).
```

Computation in /jdoe/shelf/biology

```
pick_biology_books(Books) :-
    parent_id(Shelf),
    pick_books(Books, Shelf, category(biology)).
```

Computation in /

```
pick_books(Books, Shelf, category(C)) :-
    findall(B, Shelf : book(B), AllBooks),
    filter(AllBooks, C, Books).
```



```
GET /jdoe/shelf/biology HTTP/1.1
Host: example.com

    '/jdoe/shelf/biology' : get(Request, Response, View).

/jdoe/shelf/biology          get(Request, Response, View).
```

```
/jdoe/shelf/biology
```

get/3
pick_biology_books/1

```
% ...
pick_biology_books(Books),
% ...
```

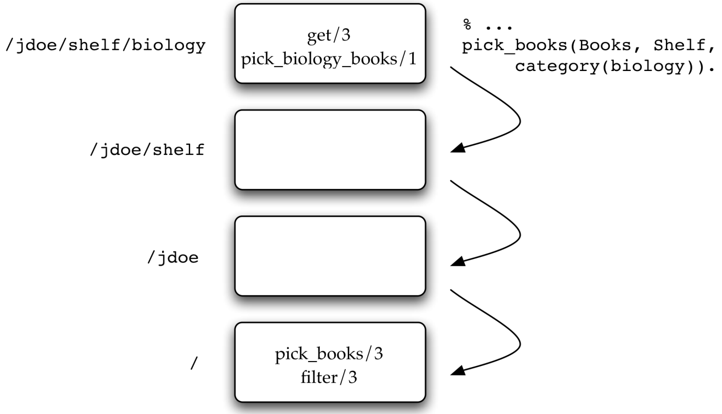
(1)

```
/jdoe/shelf/biology
```

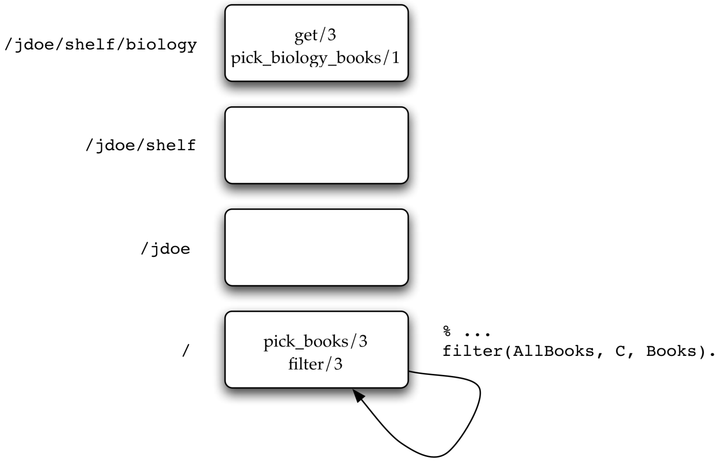
get/3
pick_biology_books/1

```
parent_id(Shelf),
pick_books(Books, Shelf,
     category(biology)).
```

(2)



```
% ...
pick_books(Books, Shelf,
           category(biology)).
```



```
% ...
filter(AllBooks, C, Books).
```

Operational Semantics

Operational semantics is a way to give meaning to computer programs in a mathematically rigorous way, by describing how a valid program is interpreted as sequences of computational steps [Plotkin, 1981]

Transition systems

A common way to rigorously define the operational semantics is to provide a state transition system for the language of interest

Inference rules

State transition systems are usually defined in the form of a set of inference rules which define the valid transitions in the system

Operational Semantics in Logic Programming

An operational semantics is used in logic programming to define derivations in the computation model in a declarative style, by considering a derivation relation and introducing a set of inference rules for it

Derivation tuples

A tuple in a derivation relation is written as

$$C \vdash G[\theta]$$

where C is a context, G is a goal, and θ is a set of equalities representing a substitution

Inference Rules

An inference rule is written in the following form

$$\frac{\mathit{Antecedents}}{\mathit{Consequent}}\ \{\mathit{Conditions}\}$$

where the *Consequent* is a derivation tuple, the *Antecedents* are zero, one, or two derivation tuples, and the *Conditions* are a (possibly empty) set of arbitrary prepositions

- Declaratively, the *Consequent* holds if the *Conditions* are true and the *Antecedents* hold
- Procedurally, to establish the *Consequent*, if the *Conditions* are true, the *Antecedents* need to be established

Derivation Tree

The notion of derivation can be formalized as a tree such that

- any node is labeled with a derivation tuple
- all leaves are labeled by an empty goal
- the relation between any node and its children is the one between the consequent and antecedents of an instance of an inference rule whose conditions are true

Derivations in Web Logic Programming

Given a context C and a goal G, the system will try to construct a derivation whose root is labeled by the $C \vdash G[\theta]$ tuple, giving θ as the computed answer substitution result if the derivation succeeds

Operational Semantics: Definitions (1/2)

A logic theory T is defined as a 2-tuple $< L_T, K_T >$ containing the identifier L_T for the theory, also called the *label*, and the knowledge base K_T, comprising the logic predicates specified in the theory. Formally, we define L_T as

$$L_T = \{a \mid a \in atom, a \in urispace\}$$

that is, a label is a particular atom belonging to the space of URI identifiers [Berners-Lee et al., 1998]. The knowledge base K_T is defined as the following set of clauses

$$K_T = \{h \leftarrow B \mid h \in atom, B \in goal\}$$

where the clauses have the form $h \leftarrow B$, the clause head h is an atom, and the clause body B is a goal. A goal can be a basic goal (e.g. `goal`) or a *labeled* goal (e.g. `label:goal`, with *label* $\in L_T$) or a set of basic and labeled goals.

Operational Semantics: Definitions (2/2)

Predicates defined in a theory T are explicitly specified by the set

$$\mathit{defined}(T) = \{\widehat{p} \mid \exists\ p \leftarrow Q \in K_T\}$$

where $\widehat{p}$ is used to represent the principal symbol of the predicate p.
For the purpose of the Web Logic Programming language, it holds that

$$\mathit{available}(T) \equiv \mathit{defined}(T)$$

that is, the set of available predicates in a theory T corresponds to the set of defined predicates in T.
The set of variants of the clauses in the knowledge base K_T is defined as

$$\mathit{variant}(K_T) = \{h\theta \leftarrow B\theta \mid h \leftarrow B \in K_T\}$$

where θ is a renaming substitution.

Goal Resolution: The Null Rule

The *Null Rule* states that the null goal is derivable in any context, with the empty substitution ϵ

$$\frac{}{C \vdash \emptyset[\epsilon]}$$

Goal Resolution: The Conjunction Rule

The *Conjunction Rule* dictates that, to derive a conjunction of goals in a context, you need to derive the first conjunct, and then the other conjunct in the very same context (with updated substitutions)

$$\frac{C \vdash G_1[\theta] \wedge C \vdash G_2\theta[\sigma]}{C \vdash G_1, G_2[\theta\sigma\lceil variables(G_1, G_2)]}$$

Even if the context may change during the derivation of the first goal, the derivation of the second goal must start from the *original context*, and not from the context where the derivation of the first goal may have possibly ended

Goal Resolution: The Reduction Rule

The third rule is called *Reduction Rule*

$$\frac{T \cdot C \vdash B\theta[\sigma]}{T \cdot C \vdash g[\theta\sigma\lceil variables(g)]} \left\{ \begin{array}{c} h \leftarrow B \in variant(K_T) \\ \theta = mgu(g, h) \end{array} \right\}$$

This rule describes how goals are reduced: if the predicate of an atomic goal is defined in the current theory, that is, a corresponding clause is found, the goal is reduced by using a variant of the clause

Context Navigation: The Implicit Up Rule

The *Implicit Up Rule* describes how a predicate not available in a theory can be derived by using the context, and therefore accounts for the dynamic binding of the Web Logic Programming language

$$\frac{T_b \cdot C \vdash g[\theta]}{T_a \cdot T_b \cdot C \vdash g[\theta]} \{\widehat{g} \notin available(T_a)\}$$

When a predicate g is not available in the current theory (here represented as T_a) the derivation process moves up to the next available theory in the context

Notes

The moving direction strictly follows the path in the URI identifying the resource context where the derivation has started

Context Navigation: The Context Switch Rule

The *Context Switch* rule describes the derivation of a goal in a specific context, different from the current one

$$\frac{C_R \vdash G[\theta]}{C \vdash n_R : G[\theta]} \; \{n_R \in L_T\}$$

To derive a goal G labeled with a resource identifier, the system switches from the current context to the context associated with that identifier, then starts the derivation of G in the new context

Notes

- This is the preferred method to invoke a computation on a resource external to the path associated with the current context
- Switching context instead of merging preserves the encapsulation of information that the representation of resources as separated logic theories encourages

Context Navigation: The Explicit Up Rule

The *Explicit Up* rule describes a convenient shortcut to invoke a derivation of a goal directly on the immediate ancestor of a resource

$$\frac{T_b \cdot C \vdash g[\theta]}{T_a \cdot T_b \cdot C \vdash parent : g[\theta]}$$

The Web Logic Programming language offers the special `parent` identifier to let programmers refer to the parent of the current theory in the composition representing a context

Notes

When compared with the *Implicit Up* inference rule, the only difference is that, in the *Explicit Up* case, the check for $\widehat{g}$ to belong to the set of available predicates in the current theory (represented as T_a in the rule) is entirely missing

From Semantics to Implementation

One of the advantages of operational semantics of a programming language over other kinds of semantics is its close relation to a possible implementation of an interpreter for that language

Meta-interpretation

When dealing with logic programming languages, building a meta-interpreter is

- quite often the quickest way to provide a core language implementation to prototype frameworks and applications
- one of the well-known techniques for implementing logic contexts [Denti et al., 1993]

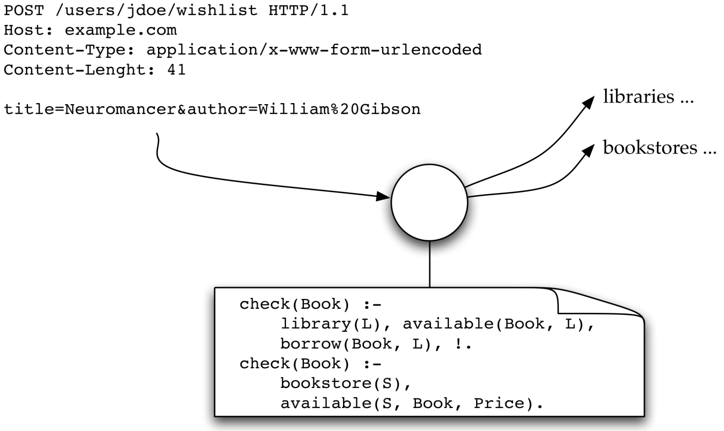
A Meta-Interpreter Sketch for Web Logic Programming

```
% Define a syntactic way to express the relation
% between a context and a subgoal
:- op(600, xfy, ':').

% Every goal needs to have an associated context
solve(':'(Context, Goal)) :- solve(Context, Goal).

% Null Rule
solve(_, true) :- !.

% Conjunction Rule
solve(Context, (Goal1, Goal2)) :- !,
    solve(Context, Goal1), solve(Context, Goal2).

% Resolution Rule with Context Navigation
solve(Context, Goal) :-
    '$member'(Context, Resource),
    rule(Resource, Goal, Body),
    solve(Resource, Body).
```

Dynamic Resource Behavior

Resource behavior can be regarded as dynamic under two independent aspects:

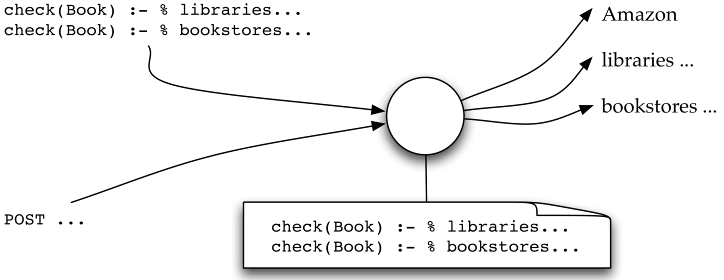
1. Two or more URIs can be associated to the same resource at any point in time: thus, a resource may live in two different contexts at the same time and feature different behavior according to the context where the computation takes place
2. Behavioral rules are expressed as first-class abstractions in logic programming languages, where programs can be treated as data and vice versa: the HTTP protocol allows changing resource data by means of the PUT method, so that it becomes possible to imagine behavioral changes of a contextualized resource at runtime



```
POST /users/jdoe/wishlist HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Lenght: 41

title=Neuromancer&author=William%20Gibson
```

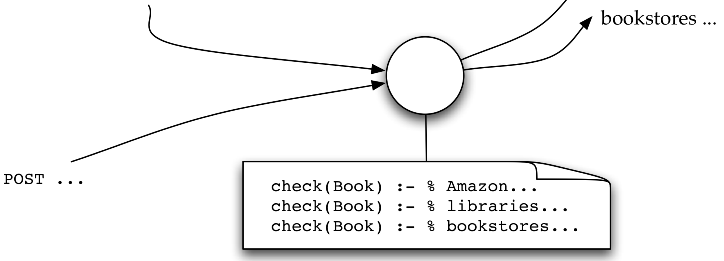
```
check(Book) :-
    library(L), available(Book, L),
    borrow(Book, L), !.
check(Book) :-
    bookstore(S),
    available(S, Book, Price).
```

```
PUT /users/jdoe/wishlist HTTP/1.1
Host: example.com
Content-Type: text/plain
Content-Lenght: 164

check(Book) :-
    available(amazon, Book, Price),
    Price < Threshold,
    place_order(amazon, Book), !.
check(Book) :- % libraries...
check(Book) :- % bookstores...
```



```
check(Book) :- % libraries...
check(Book) :- % bookstores...
```

```
PUT /users/jdoe/wishlist HTTP/1.1
Host: example.com
Content-Type: text/plain
Content-Lenght: 59

check(Book) :- % libraries...
check(Book) :- % bookstores...
```



```
check(Book) :- % Amazon...
check(Book) :- % libraries...
check(Book) :- % bookstores...
```

Part IV

Proposed Projects (and Theses)

Model

Find useful refinements of the Web Logic Programming computation model

1. Study typical resource combination and computation patterns in web applications
2. Define new operators and inference rules accounting for those patterns, then combine them with the existing model

Framework

Design and implement a prototype framework for Web Logic Programming around the following elements:

- the tuProlog (2P) Java-based light-weight Prolog engine for the basic computation core
- the Apache Tomcat container for World Wide Web exposure and JavaServer Pages representation technology
- the 2PTags technology

2PTags

A set of tags to perform logic queries from JavaServer Pages, with the aim of accessing the logic representation of resources to create their hypermedia representations

References I

Abreu, S. and Diaz, D. (2003).
Objective: In minimum context.
In Palamidessi, C., editor, *Logic Programming*, volume 2916 of *Lecture Notes in Computer Science*, pages 128–147. Springer Berlin.

Berners-Lee, T., Fielding, R. T., and Mainster, L. (1998).
Uniform Resource Identifiers (URI): Generic Syntax.
Internet RFC 2396.

Brogi, A., Mancarella, P., Pedreschi, D., and Turini, F. (1994).
Modular Logic Programming.
ACM Transactions on Programming Languages and Systems, 16(3):1361–1398.

Bugliesi, M., Lamma, E., and Mello, P. (1994).
Modularity in logic programming.
Journal of Logic Programming, 19-20:443–502.

References II

Cabeza, D. and Hermenegildo, M. (2001).
Distributed WWW Programming Using (Ciao–)Prolog and the PiLLoW Library.
Theory and Practice of Logic Programming, 1(3):251–282.

Denti, E., Lamma, E., Mello, P., Natali, A., and Omicini, A. (1993).
Techniques for implementing contexts in Logic Programming.
In Lamma, E. and Mello, P., editors, *Extensions of Logic Programming*, volume 660 of *LNAI*, pages 339–358. Springer.

Denti, E., Natali, A., and Omicini, A. (1997).
Merging Logic Programming into Web-based Technology: A Coordination-based Approach.
In De Bosschere, K., Hermenegildo, M., and Tarau, P., editors, *ICLP'97 Post-Conference 2nd International Workshop on Logic Programming Tools for Internet Applications*, pages 117–128, Leuven (B).

References III

Fielding, R. T. (2000).
Architectural Styles and the Design of Network-based Software Architectures.
PhD thesis, University of California, Irvine.

Fielding, R. T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999).
Hypertext Transfer Protocol – HTTP/1.1.
Internet RFC 2616.

Freed, N. and Borenstein, N. (1996a).
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.
Internet RFC 2045.

Freed, N. and Borenstein, N. (1996b).
Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.
Internet RFC 2046.

References IV

Kowalski, R. (1974).
Predicate logic as programming language.
In *Proceedings IFIP Congress*, pages 569–574. North Holland Publishing Co.

Loke, S. W. (1998).
Adding Logic Programming Behaviour to the World Wide Web.
PhD thesis, University of Melbourne, Australia.

McCabe, F. G. (1992).
Logic and Objects.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Monteiro, L. and Porto, A. (1993).
A Language for Contextual Logic Programming.
In *Logic Programming Languages: Constraints, Functions, and Objects*. The MIT Press.

References V

Omicini, A. (1995).
Programmazione Logica Orientata agli Oggetti.
PhD thesis, Università di Bologna.

Plotkin, G. D. (1981).
A structural approach to operational semantics.
Technical Report DAIMI FN-19, University of Aarhus.

Richardson, L. and Ruby, S. (2007).
RESTful Web Services.
O'Reilly.

SC22, J. T. C. I. J. (2000).
Information technology — Programming languages — Prolog — Part 2: Modules.
International Standard ISO/IEC 13211-2.