A Two-level World Wide Web Model with Logic Programming Links

Seng Wai Loke

Dept. of Computer Science
University of Melbourne
Parkville, Victoria 3052, Australia
E-mail: swloke@cs.mu.oz.au

Andrew Davison

Dept. of Computer Engineering
Prince of Songkla University
Hat Yai, Songkhla 90112, Thailand
E-mail: ad@ratree.psu.ac.th

Abstract

We propose a two-level model for the World Wide Web: Web pages are stored at the first level, and the links between them pass through a link abstraction layer. This extra layer permits processing to be carried out as part of link traversal, thereby increasing the expressiveness of links beyond direct connection between pages. This mechanism is implemented using LogicWeb, which allows Web pages to be treated as logic programs. One advantage of LogicWeb is that it enables conceptual Web structures to be coded in the abstraction layer which are amenable to sophisticated querying and automated reasoning. Another benefit is that it allows the easy specification of rules which model the nondeterministic and unpredictable nature of link activation in the Web.

1 Introduction

The Web's basic link mechanism is uni-directional, with fixed source and destination pages. When a page is retrieved, it is simply displayed. We extend the expressiveness of links by proposing a two-level model for the Web: Web pages are at the first level, but the links between them pass through a link abstraction layer. This layer permits processing to occur during link traversal, thereby increasing the expressiveness of the links.

The layer can take many forms, such as a semantic network, concept hierarchy, or simply procedures to compute destination pages.

The two-level model is implemented using the *LogicWeb* system which views Web pages as logic programs which we call *LogicWeb programs* [8, 6].

We explore the use of logic programming for coding the link abstraction layer, exploiting its ability to represent declarative structures in a concise and readable manner, and its nondeterminism.

In §2, we describe LogicWeb, and in §3, the two-level model. §4 illustrates the expressiveness of links which utilise structured information such as semantic networks. In §5, we show how nondeterminism in logic programs easily models

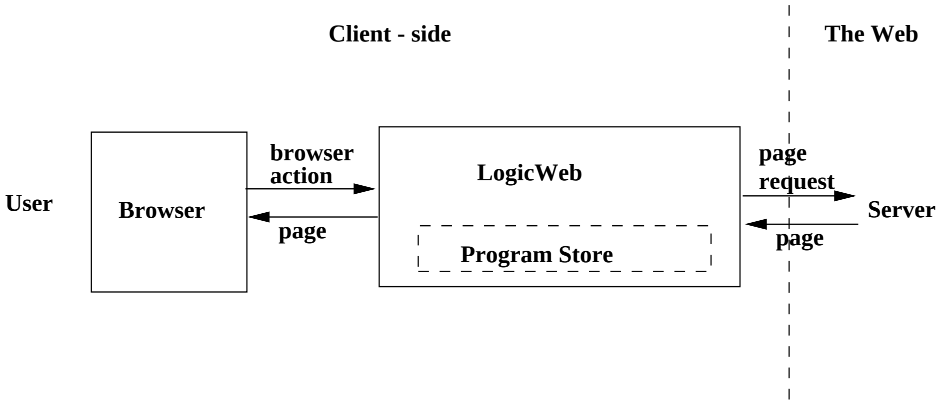

Figure 1: The LogicWeb system between the user and the Web.

the dynamic and unpredictable nature of the Web. We discuss related work in §6, and conclude in §7.

2 LogicWeb

LogicWeb allows each Web page to be viewed as a logic program. An ordinary Web page is represented by a set of clauses storing the page's contents and some meta-level information, such as its last modification date. Pages can also contain additional LogicWeb code.

The LogicWeb system stands between the user and the Web as shown in Figure 1. A prototype LogicWeb system has been implemented using the NCSA XMosaic Browser, C, and Prolog [8, 6], and was used as a testbed for the examples in this paper.

When a Web page is retrieved, it is displayed by the browser as normally. However, LogicWeb also translates all retrieved pages into programs. The basic transformation generates two facts for each Web page:

```
my_id("URL").
h_text("Page text").
```

`my_id/1` holds the program's own ID which is its URL, while `h_text/1` contains the complete text of the Web page (including its HTML tags) as a string.

The system parses each page looking for link information so that a `link/2` predicate can be generated. For instance, the anchor in the line:

```
I like <A HREF="http://www.prost.org/beermake.html">beer making</A>.
```

becomes:

```
link("beer making", "http://www.prost.org/beermake.html").
```

The system stores meta-level page information supplied by the Web server in `about/2` facts. Typically, this information includes the MIME (Multipurpose Internet Mail Extension) version, the server version, the page request date, the content type, the content length, and the time the page was last modified. For instance, the `about/2` fact for the last modified time of a page would be:

```
about(last_modified, "Sunday, 08-Dec-96 20:48:29 GMT").
```

Facts about the page structure can also be generated when required, including a `title/1` fact and similar information about section, sub-section, and other headings.

LogicWeb code is included in a Web page inside a "<LW_CODE>...</LW_CODE>" container. The language utilised by LogicWeb is a fairly elementary Edinburgh-style Prolog with some additional operators. The main new operator applies a goal to a program specified by a Web request method and its program ID (i.e. its URL):

```
lw(RequestMethod, URL)#>Goal.
```

`RequestMethod` refers to the underlying Web communication protocol, HTTP [1]. The HTTP methods currently supported by LogicWeb are HEAD, GET, and POST. The HEAD method retrieves meta-level information about a page, whereas the GET and POST methods retrieve both meta-level information and page contents. A GET request only sends the URL of the required page, while a POST request can send arbitrary data in addition to the URL. At the LogicWeb level, these three methods are identified by the `RequestMethod` terms `head`, `get`, and `post(Data)`.

Each of the methods retrieves a Web page, which is stored in the LogicWeb system as a logic program. Once the logic program has been created, the `#>/2` goal is applied to it. Consequently, the `lw/2` term can be viewed as a *specification* of the logic program to which the goal is applied.

The behaviour of the `#>` operator is actually more complex, in that the logic program defined by the `lw/2` term is only retrieved if it is not already present in the LogicWeb store (see Figure 1). If the program is present then the goal is applied to it, without the need for code to be brought over the Web.

The following example shows how LogicWeb predicates can be used to describe research interests in a home page:

```
<HTML>
<HEAD>
<TITLE>Seng Wai Loke</TITLE>
</HEAD>

<BODY>
<H1>Seng Wai Loke's Home Page</H1>
...
<!--
<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>
```

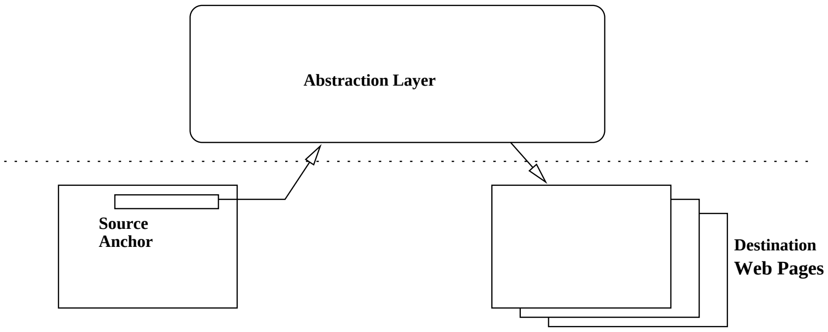

Figure 2: The two-level Web model. A link abstraction layer separates the source and destination of links.

```
-->
</BODY>
</HTML>
```

The code appears inside a "`<!--...-->`" container so that it is uninterpreted by the browser. `interested_in/1` states that the author is interested in several topics and in what his friends are interested in. The "`#>`" goal in the second clause of `interested_in/1` evaluates the goal `interested_in/1` against each friend's home page. Hidden from the user, each page is retrieved using a GET method request, and transformed to a logic program before the `interested_in/1` goal is evaluated.

`link_action/1` is an important built-in predicate, which is automatically invoked when a link is selected on a page. The URL of the link is passed as a string argument to the `link_action/1` call.

`link_action/1`'s normal meaning is to retrieve the page, place its program in the store, and display the page. However, it can be redefined by including a `link_action/1` predicate in the LogicWeb code part of the page.

This mechanism means that arbitrary computation can be carried out in response to link selection, and is the implementation basis of our two-level Web model.

3 The Two-level Web Model

Figure 2 depicts our model: the first level contains Web information (e.g., Web pages, multimedia data), and the second level determines the meaning of links (the link abstraction layer).

Links in the existing Web are uni-directional, have a fixed configuration, and go from one source page to one destination page. Links at the abstraction layer can be bi-directional, dynamically configured, and go from many sources to many destinations.

The link abstraction layer can implement many link formalisms, and we contend that logic programming is particularly suitable for two key aspects:

- *Structured information representation*: Logic programming can represent structured information in the form of deductive databases, or knowledge-based structures.

A fundamental reason for such structures is the ability to create abstract views of the Web. This will help improve a user's understanding of the Web, permit new information to be better integrated with existing Web knowledge, and facilitate search based on semantic criteria.

- *Nondeterminism*: The nondeterminism in logic programs allows links to deal with the dynamic and unpredictable nature of the Web.

4 Utilising Structured Information for Linking

This section shows how various kinds of structured information (e.g. semantic networks, deductive databases) can be implemented in the link abstraction layer. These structures can be predefined by the programmer, or be created dynamically by extracting details from Web pages.

4.1 IS-A Hierarchy Links

The IS-A hierarchy about AI, logic programming, and agents described here is loosely based on the Canto hypertext data model [11]. Its overall structure is shown in Figure 3.

`isa/2` represents an IS-A relation between concepts in the form of `isa(Subconcept, Concept)` links:

```
isa(logic_programming, ai).
isa(agents, ai).
isa(agent_theories, agents).
isa(agent_architectures, agents).
isa(agent_languages, agents).
isa(mobile_code, agent_languages).
isa(java, mobile_code).
isa(agent_tcl, mobile_code).
isa(agent0, agent_languages).
```

To build an IS-A *hierarchy*, we define the transitive closure on `isa/2`:

```
trans_isa(Concept1, Concept2) :-
  isa(Concept1, Concept2).
trans_isa(Concept1, Concept2) :-
  isa(Concept1, Mid),
  trans_isa(Mid, Concept2).
```

We associate URLs with concepts using `page_about/2`:

```
page_about(agents, "http://machtig.kub.nl:2080/infolab/egon/Home/agents.html").
page_about(agents, "http://www.cs.mu.oz.au/~swloke/res1.html").
page_about(java, "http://java.sun.com/").
page_about(agent_tcl, "http://www.cs.dartmouth.edu/~agent/agenttcl.html").
page_about(agent0, "http://www.scs.ryerson.ca/~dgrimsha/courses/cps720/agent0.html").
   :  % more page_about/2 facts
```

`trans_about/2` defines a transitive version of `page_about/2` which associates a concept with a URL if one of its sub-concepts is associated with that URL:

```
trans_about(Concept, URL) :-
  page_about(Concept, URL).
trans_about(Concept, URL) :-
  trans_isa(SubConcept, Concept),
  page_about(SubConcept, URL).
```

We assume that the complete IS-A structure is stored in the page:

```
http://www.cs.mu.oz.au/~swloke/link_kb.html
```

An IS-A query will be phrased as a syntactic extension of that page's URL:

```
http://www.cs.mu.oz.au/~swloke/link_kb.html-+-<concept-name>
```

A URL of this type will be understood to mean: retrieve a page associated with that `concept-name` in the IS-A hierarchy. For example, a page about agents can be obtained by dereferencing:

```
http://www.cs.mu.oz.au/~swloke/link_kb.html-+-agents
```

This behaviour is implemented by redefining the meaning of `link_action/1`. It must extract the concept from the string passed to it, search the IS-A hierarchy for a suitable URL, and then display that page:

```
link_action(URLstring) :-
  link_parts(URLstring, ISA-URL, Concept),
  lw(get, ISA-URL)#>page_about(Concept, ConceptURL),
  my_id(MyURL),
  ConceptURL \= MyURL,  % ensure that a different page is fetched
  show_page(ConceptURL).

link_parts(URLstring, ISA-URL, Concept) :-
  append(ISA-URL, [0'-,0'+,0'-|ConceptStr], URLstring),
  atom_chars(Concept, ConceptStr).   % convert string to atom

show_page(URL) :-
  lw(get, URL)#>h_text(Src),
  display_page(URL, [data(Src)]).
```

`link_action/1` uses `link_parts/3` to split the URL string into the URL for the IS-A hierarchy page and the concept. A suitable URL related to the concept is obtained by calling `page_about/2`, and the corresponding page is displayed with `show_page/1` (but only if it is different from the current page).

By changing the meaning of `link_action/1`, the same link can have quite different semantics. For instance, all pages related to the given concept can be returned. `link_action/1` uses `setof/3` applied to `trans_about/2` to obtain all suitable URLs. If only a single URL is found then `show_page/1` is called, otherwise the list of URLs is assembled into a HTML page by `report_urls/3` and printed.

```
link_action(URLstring) :-
  link_parts(URLstring, ISA-URL, Concept),
  setof(DestURL, lw(get, ISA-URL)#>trans_about(Concept, DestURL), DestURLs),
  ([DestURL1] = DestURLs ->
```

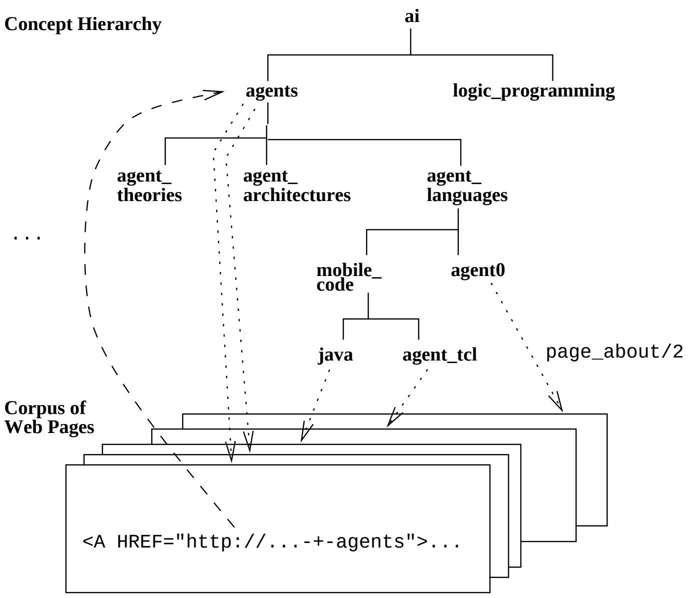

Figure 3: An IS-A hierarchy. The dashed arrow shows a link referring to a concept in the hierarchy. The dotted arrows show mappings from concepts to Web pages as specified by `page_about/2`.

```
    show_page(DestURL1)
 ;
    report_urls(DestURLs, Concept, Links),
    display_page("Multi-destination link", [data(Links)])
 ).
```

Figure 4 shows the page constructed when the link to "agents" is followed.

4.2 Page Name Links

We now consider an alternative link abstraction, which maps names to URLs. This technique is similar to Uniform Resource Names (URNs)[1].

Currently, when a Web page is moved, the links to it from other pages will cease to work. One solution is to define links in terms of page names rather than addresses, and use a central database (or databases) to map these names to their actual URLs. When an author moves a page he must update its URL in the relevant database, but the page name remains the same. This means that users of a page (who use the page name as a link reference) are unaffected by the movement of the page.

We imagine a database of page names and URLs of the form:

```
page_details(Name, URL)
```

e.g.

[1] More information on URNs at http://www.acl.lanl.gov/URN/.

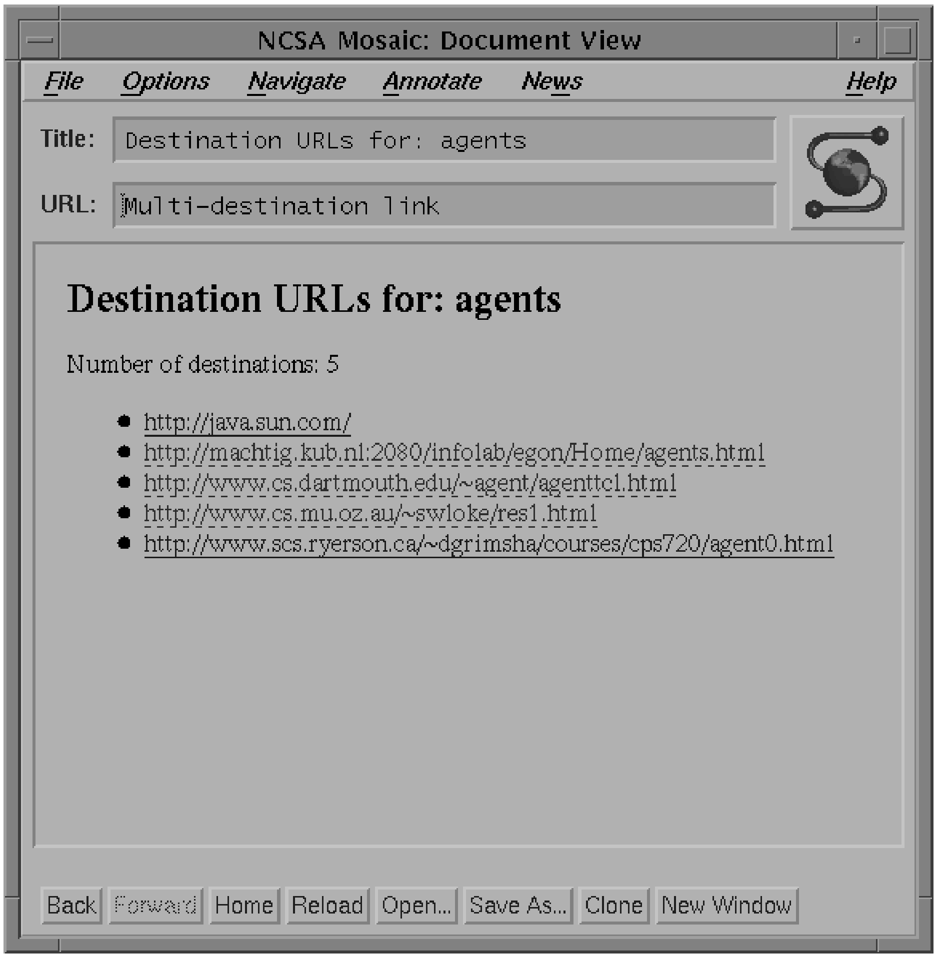

Figure 4: The page constructed when the link to "agents" is selected.

```
page_details("Seng's_Page", "http://www.cs.mu.oz.au/~swloke").
page_details("Andrew's_Page","http://fivedots.coe.psu.ac.th/~ad").
```

This database will be stored in a globally known Web page at:

```
http://www.db.com/details.html
```

We extend the URL syntax to include a page name reference:

```
http://www.db.com/details.html-$-<page-name>
```

e.g.

```
http://www.db.com/details.html-$-Seng's_Page
```

When such a URL is dereferenced on a page, we must define its meaning with `link_action/1`:

```
link_action(URLstring) :-
  append(DbURL, [0'-,0'$,0'-|PgName], URLstring),
  lw(get, DbURL)#>page_details(PgName, PgURL),
  show_page(PgURL).
```

The same technique can be used to simplify page referencing on a server. We assume that each server has a database called `map.html` of `page_map/2` facts. The database maps page names to actual pages on the server. For example:

```
page_map("LogicWeb", "http://www.cs.mu.oz.au/~swloke/logicweb.html").
```

Users can employ a particular server's `map.html` database by writing URLs of the form:

```
http://<server-address>/map.html-$-<page-name>
```

e.g.

```
http://www.cs.mu.oz.au/map.html-$-LogicWeb
```

This will return Seng Wai Loke's `logicweb.html` page.

This mechanism not only simplifies the information required to retrieve a page, but can also be utilised as a redirection and page protection device.

4.3 Links based on Page Information

The preceding examples have used information external to Web pages (i.e. an IS-A hierarchy, a page name database). The meaning of a link can also be determined from the source and/or destination pages, by executing their LogicWeb code, by parsing their text, or by looking at their meta-level information.

For instance, we can specify a link filtering rule which ensures that only up-to-date, interesting, and non-bookmarked pages are displayed:

```
link_action(URL) :-
  up_to_date(URL),
  interesting(URL),
  not(bookmarked(URL)),
  show_page(URL).
```

```
up_to_date(URL) :-
  lw(head, URL)#>about(last_modified, LastModifiedTime),
  getime(LastModifiedTime, '1997;04;31;01:00').

interesting(URL) :-
  lw(get, URL)#>h_text(Src),
  lw(get, "http://www.cs.mu.oz.au/~swloke/")#>interested_in(Keyphrase),
  contains(Src, Keyphrase).

bookmarked(URL) :-
  lw(get, "http://www.cs.mu.oz.au/~swloke/book_marks.html")#>link(_, URL).
```

`up_to_date/1` examines the page's meta-level information. `getime/2` checks if the last modified time is equal to or after 1 am on the 31st of April 1997. `interesting/1` assumes that Seng Wai Loke's home page contains an `interested_in/1` predicate (as in section 2), and checks if any of its key phrases are in the text of the page. `bookmarked/1` assumes the existence of a `book_marks.html` page which contains links to Loke's bookmarked pages. These are tested against the page being considered.

4.4 Dynamically Created Pages

A destination page can be dynamically constructed by composing pre-stored elements together. For instance, activating a link to a page may retrieve the page augmented with extra details, such as footnotes, background links, or advertising material.

As an example, we assume that the page

```
http://annotations.url.db/
```

contains facts relating the URLs of pages to notes that must be added to the bottom of those pages. The facts are in the form:

```
annotation_page(URL, NotesURL).
```

`NotesURL` is the URL of the page containing the notes about the `URL` page. For a given page, the following rule retrieves its notes and displays the page text followed by those notes:

```
link_action(URL) :-
  lw(get, "http://annotations.url.db/")#>annotation_page(URL, NotesURL),
  lw(get, URL)#>h_text(Src),
  lw(get, NotesURL)#>h_text(Notes),
  display_page("Annotated Page", [data(Src), data(Notes)]).
```

Customising Web pages during link traversal is similar to an application of *perspectives* in [12, 13]. Perspectives are graph structures that can be combined and operated upon in various ways. Applied to the Web, perspectives can represent combinations of pages or page fragments.

5 Handling Nondeterminism in Web Links

Due to the dynamic, unpredictable nature of the Web, link traversal is nondeterministic. Depending on the current state of the Web, selecting a link can result in a number of possible outcomes including a page being displayed, a redirection message, or an error message (e.g., the server is busy or down, or the page no longer exists). We can easily model the nondeterminism in link traversal with logic programs.

5.1 Automatic Redirection

A Web page that has been moved is often replaced by a redirection page, which typically contains a message like "This site has moved. We are now at...(the new URL)". It is possible to automate the task of following a redirection link so that the user need never see the redirection page.

`link_action/1` examines the target page before displaying it. If it is a redirection page, then the new URL is obtained, and its page is displayed:

```
link_action(URL) :-
  (redirection_page(URL) ->
    lw(get, URL)#>link(_, NewURL),
    show_page(NewURL)
  ;
    show_page(URL)
  ).

redirection_page(URL) :-
  lw(get, URL)#>title(Title),
  ( contains(Title, "has moved")
  ; contains(Title, "site moved")
  ; contains(Title, "redirection to new location")
  ; contains(Title, "redirect URL")
  ).
```

A redirection page is recognised by looking for specific phrases in the title.

5.2 Using a Mirror Site

Popular sites are often mirrored in order to reduce the load on the site's server. This also reduces the network traffic around the site, and improves the connection speed when a page is accessed from a closer source. Moreover, mirrored information means that if one site is down, then another source is available. The following rule fetches a page from an alternative site if the initial page request fails:

```
link_action(URL) :-
  (lw(get, URL)#>h_text(HTMLSrc) ->
    display_page(URL, [data(HTMLSrc)])
  ;
    mirrored_by(URL, MirrorURL),    % obtain the mirror site
    show_page(MirrorURL)
  ).
```

We assume the existence of `mirrored_by/2` facts which store the mirror URLs for a site. This simple rule can be modified in various ways, such as to also access a mirror site when a request time-outs.

6 Related Work

6.1 Hypertext Tools

Hypertext systems which utilise semantic networks to structure their information are surveyed in [10]. An important difference between these systems and our work is that they deal with a closed information base whereas the Web is open. With closed information, it is possible to build semantic networks to represent the entire corpus.

In the HyperBase hypertext tool [7, 4], Prolog code can be invoked when buttons are pressed. Applications include a system to help users fill in forms, and a system for teaching neuropathology. Our work is similar in that we also invoke Prolog-like code, but this is in the context of the Web, and the manipulation of pages as logic programs.

6.2 Web-based Tools

6.2.1 Structured Maps

Our knowledge-based structures are built above existing Web information, in a similar way to Structured Maps [9]. Structured Maps represent semantic information using an entity-relationship diagram (ERD) expressed in SGML, and have links to the underlying information. A three-level model is utilised: the top-most level is the entity-relationship schema, the middle level is an entity-relationship instance, and the bottom level is the information base. The reader browses the Maps, moving between levels to locate information. A query language for Structured Maps has not yet been developed.

In our approach, the query language is the knowledge representation language, and we have the additional data modelling power of rules.

6.2.2 CGI

A common technique for extending the meaning of a Web link is to use CGI scripts. A basic difference with our work is that CGI scripts execute on the server, whereas we utilise a client-based approach. This reduces the server load, and localises computation. Also code can be combined together in one place, and state information can be maintained easily.

6.2.3 Java

Java can be employed to add dynamic content to pages in the form of applets [2]. Much of the functionality provided by the knowledge-based structures described in §4 can be coded in Java. However, a LP-based language offers more direct ways of representing these structures and of querying them.

6.2.4 JavaScript

JavaScript is designed to allow the elements of a Web page (e.g. form input fields, links, text attributes) to be manipulated via a programming language [3]. For example, it is possible for the selection of a link to invoke a JavaScript function. Two restrictions with JavaScript are that all downloaded pages must be displayed, and only the forms and anchors of downloaded pages can be accessed within a program.

Our approach permits pages to be processed in more general ways. Also, its emphasis on the rule-based specification of link behaviour is particularly useful for representing and manipulating knowledge.

6.2.5 The PiLLoW/CIAO Library

PiLLoW/CIAO [5] is a library for writing Prolog Web applications, including predicates for fetching URLs, processing CGI form inputs, and HTML parsing. These utilities can be used with a Web browser to allow a click on a link to download Prolog code into a locally running Prolog engine. Once there, the code can be queried via a form. Alternatively, a query can be executed when the code is loaded, achieving the effect of invoking a Prolog goal by clicking on a link. This differs from LogicWeb where the code describing the link behaviour of a page can be part of the page. In the PiLLoW approach, the server must classify the separate Prolog code file as a new MIME type so that the browser can treat it differently from ordinary HTML pages. Such classifications are not required in LogicWeb, which does not distinguish between pages containing LogicWeb code and ordinary HTML text.

7 Conclusions

We have described a two-level Web model: the first level consists of Web pages, while the second is a link abstraction layer. This allows the meaning of a link to be extended in various ways, to utilise conceptual structures, page names, and information about pages. The model is implemented using LogicWeb, which allows Web pages to be viewed as logic programs.

The use of logic programming makes the development of knowledge-based structures in the link abstraction layer very straightforward. Also, the nondeterminism in logic programming is ideal for handling many aspects of the dynamic, rapidly changing nature of the Web.

Future work will investigate how to control the time and space resource usage of link computations. We are also looking into automatic means for building and maintaining knowledge-based structures like those described in §4.1.

The current LogicWeb implementation automatically stores a fixed subset of HTML tags in predicate form, but these are not always what are required by an application. The programmer must then resort to using the lower-level parsing utilities in LogicWeb to extract further information. We plan to make it easier to specify which page components should be automatically stored as facts as a page is downloaded.

Acknowledgements. We are grateful to Leon Sterling for valuable comments on an earlier version of this paper.

References

[1] HyperText Transfer Protocol version 1.0 Specification (RFC 1945). Available from *http://www.w3.org/pub/WWW/Protocols/Specs.html* and at *http://ds.internic.net/rfc/rfc1945.txt*.

[2] Java, Sun MicroSystems Inc. *Web site at http://www.javasoft.com/*.

[3] JavaScript for Netscape. Documentation available at *http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html*.

[4] HyperBase. Product out of distribution. Information available by email to info@amzi.com. July 1992.

[5] D. Cabeza, M. Hermenegildo, and S. Varma. The PiLLoW/CIAO Library for INTERNET/WWW Programming using Computational Logic Systems. In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP"96, Bonn, September 1996. Available from `http://clement.info.umoncton.ca/~lpnet`.

[6] A. Davison and S.W. Loke. LogicWeb: Enhancing the Web with Logic Programming (available at *http://www.cs.mu.oz.au/~swloke/papers/lw.ps.gz*). 1996.

[7] A. Littleford. Artificial Intelligence and Hypermedia. In E. Berk and J. Devlin, editors, *Software Engineering Series, Hypertext/Hypermedia Handbook*, pages 357 – 378. McGraw-Hill Publishing Company, Inc., 1991.

[8] S.W. Loke and A. Davison. Logic Programming with the World-Wide Web. In *Proceedings of the 7th ACM Conference on Hypertext, Washington D.C., USA. (*available at *http://www.cs.unc.edu/~barman/HT96/P14/lpwww.html).*, pages 235 – 245. ACM Press, March 1996.

[9] D. Maier, L. Delcambre, L. Anderson, and R. Reddy. Modeling Explicit Semantics Over a Universe of Information. In *PROCEEDINGS of the Third GCA International HyTime Conference (*available at *http://www.hightext.com/IHC96/lmd7.htm)*, August 1996.

[10] J. Mayfield. Two-level Models of Hypertext. To appear in *R. D. Semmel, ed., Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, World Publishing.*

[11] C.K. Nicholas and L.H. Rosenberg. Canto: A Hypertext Data Model. *Electronic Publishing*, 6(1):1–23, March 1993.

[12] V. Prevelakis. *A Model for the Organisation and Dynamic Reconfiguration of Information Networks.* PhD thesis, University of Geneva, 1995.

[13] V. Prevelakis. A Framework for the Organisation and Dynamic Reconfiguration of the World Wide Web. In *Proceedings of the 5th Hellenic Conference on Informatics (*available at *http://senanet.com/epy/18/www/epy.htm)*, 1996.