EPOS

PM 2004:01

2004-10-29

Electronic Publication Overlay Systems Project
Linköping University Electronic Press
Linköping university, Sweden

Erik Sandewall

High-Level Design of WWW Servers in Allegro Common Lisp

Related information can be obtained via the following WWW pages:

Linköping university:	http://www.liu.se/
EPOS Project:	http://www.ida.liu.se/ext/epos/
Author's Department:	http://www.ida.liu.se/
The author:	http://www.ida.liu.se/~erisa/

1 Introduction

When invoking a function or a procedure in an ordinary programming language, it is normally assumed that the arguments may be given as composite expressions, and that they are not restricted to atomic constants or variable symbols. However, although active web pages in HTML-based web servers can be viewed as a kind of procedures, they do not enjoy the same flexibility. The present paper reports on a software package that extends the embedded web server in the ACL (Allegro Common Lisp) system and that provides it with the kind of functional flavor just described. In passing, the software also adds a number of other convenience measures to the LHTML (Lisp-encoded HTML) of the ACL server.

In the sequel we assume basic knowledge of HTML, including the HTML syntax for active or dynamic web pages, besides of course knowledge of Lisp.

2 Background: the ACL web server

The ACL web server can be started and stopped at arbitrary times during an interactive session with the system. When running, it responds to http requests on a designated port (usually port 80) according to *specifications* that have been made before during the same session. There are three main types of server specifications. *File mapping* specifications map a URL to a corresponding file or directory, for example mapping `http://www.mysite.info/btn/plants/` to `C:/website/plants/` for outside requests if the proper host is designated as `http://www.mysite.info/`. (In the examples we shall assume Windows-type file paths but we use forward slash in the place of backslash, which is consistent with ACL conventions). Subdirectories and files contained in the specified directory are mapped accordingly. The files that are mapped in this way into the server-visible space may be in HTML or any other format that makes sense to a browser.

Static LHTML specifications are the second type. In this case a URL such as

```
http://www.mysite.info/btn/plants/listing
```

is mapped to an expression in LHTML, which is HTML coded as S-expressions. The LHTML encoding follows HTML very closely, so for example

```
<h2>Title with an <em>italic</em> word</h2>
```

is coded as

```
(:h2 "Title with an " (:em "italic") " word")
```

and

```
<a href="explain.html" target="sep">Explain</a>
```

is coded as

```
((:a href "explain.html" target "sep") "Explain")
```

Dynamic LHTML specifications, finally, allow one to associate a URL with arbitrary computation in the web server. The invocation of the dynamic “page” may contain parameters, separated using the HTML convention using question-mark and ampersand. These parameters are made available to the computation that results when the request reaches the web server, and the result of the computation may be expressed using LHTML as just shown.

For example, suppose one wished to provide the site visitor with distinct pages that present each one of many different plants, and each plant is characterized by its Linnaean latin name, such as “Taraxacum vulgare” for dandelion, or “Rosa canina” for wildrose. This may be done using file mapping, in which case the description of dandelion could be stored in the file system on

```
C:/website/plants/taraxacum/vulgare/descr.htm
```

and accessed accordingly as

```
/btn/plants/taraxacum/vulgare/descr.htm
```

(The site part of the URL will be omitted from now on). If instead the request is to be honored using a database, so that the exact HTML for the response is to be assembled dynamically for each request, then the access may instead be written as

```
/btn/plantsbase?gnd=taraxacum&spc=vulgare
```

invocing the procedure that has been associated with the dynamic webpage designator `/btn/plantsbase`, with the arguments “`taraxacum`“ and “`vulgare`“ in the obvious way.

The design of the ACL webserver provides the programmer with considerable power and flexibility for programming dynamic web pages and integrating them with statically stored information and with other software. It does have a number of annoying but minor problems which result from the 1-1 mapping from, and close adherence to HTML conventions, such as the lack of exchangeability between static and dynamic pages (as demonstrated in the dandelion example), and the lack of uniformity between the syntax for tables and for frames. Also, the convention for coding parameterized HTML commands using double left parentheses, as in the `:A` example above, leads to code that is quite difficult to read, for example when a FORM expression immediately contains a TABLE expression.

Our system provides solutions for these problems, in ways that will be described below. However, let us focus first on a more important aspect of the system, namely the methods for referring from one dynamic webpage to another one.

3 The main problem: restrictive invocation syntax in HTML

Suppose I have at one time defined a botany section of my website, using a dynamic web address that takes the Latin names for the gender and species of a plant as arguments, along the lines of the previous examples. Later on I wish to refer to this resource from a new web application, or even from a computational process via a wrapper. This is works well in simple cases, but not in more complex ones.

Consider the simple case first, by way of introduction. Suppose I define a new webpage containing a form where the user enters the two descriptors for a plant, maybe together with other information. With the existing ACL web server it is easy to define the new page so that it results in an invocation of the existing resource.

Similarly, suppose one argument for the existing resource is known in the computational environment of the new application, and another argument is entered by the user. In a somewhat contrived extension of the current example, for the purpose of illustration, we can suppose that the gender name is known whereas the species name is entered by the user in a form field. The known argument is coded using an HTML 'hidden' entity, and in fact LHTML allows it to be computed using an arbitrary Lisp form.

We proceed now to a case where the difficulties appear. Suppose we again change the example so that some processing of input data is needed in the new application in order to prepare for the invocation of the existing resource. This preparatory operation may be, for example, the use of a spelling corrector, or the use of a translator allowing input of plant names in English rather than in Latin, or making a selection from a menu. At this point the ACL webserver design becomes inconvenient. Following HTML conventions, it assumes that data that are entered into the fields of a form shall be sent 'as is' to the processor for that form. In our case it would mean that the botany resource is to receive the raw data that were input by the user, without the modifications (spelling correction, translation) that are part of our application.

There are two ways of handling such a situation in the ACL server. One possibility is to modify the receiving end, which in our example is the existing dynamic botany resource, and to add the application-specific services there. However, this violates important software structuring principles and leads to a messy overall software structure.

The other possibility is to insert an additional software layer. The webpage for the application, containing the form, invokes an auxiliary dynamic webpage with the form fields as the arguments. The auxiliary page checks the arguments, modifying them if necessary, and again invokes the existing (botany) resource. This method is globally well structured, but leads to code that is inconvenient and hard to read on the local scale.

The WRDL package provides a third solution, namely, a high-level representation that is natural and easy to read, and that will be described next. It is implemented by translating the WRDL expression to ordinary LHTML code along the lines of the second of the two possibilities described here, but the programmer does not need to be concerned about that translation.

4 High-level web queries in WRDL

We use the term 'resource' for an abstraction that subsumes both dynamic web pages and families of similar, static web pages. Instances of a resource are specified in functional style using a *resource designator* followed by a list of arguments. The static file page for dandelion in the botany example could be represented using the resource expression

```
(plant "taraxacum" "vulgare")
```

and the reference to a dynamic page for the same plant could be represented using the expression

```
(dynplant "taraxacum" "vulgare")
```

The resource designators `plant` and `dynplant` are defined using declarations specifying whether the resource is implemented statically or dynamically, what is the root directory in the case of a static implementation, and so forth.

The arguments in resource expressions can be arbitrary Lisp forms that are evaluated when needed, and that are not restricted to atomic values as were used in the example.

Resource expressions can be used in several ways. The basic usage is in webpage links, as in

```
(:link "Dandelion"
     resource (dynplant "taraxacum" v)
     target "window2" )
```

This WRDL expression translates to the following LHTML expression

```
((:a href (concatenate 'string "/btn/plantsbase"
                  "?gnd=" "taraxacum" "&spc=" v )
     target "window2" )
 "Dandelion" )
```

The string for the gender argument has been kept separate to make the treatment of the two arguments easier to follow. The LHTML expression is in turn translated to the following, if the value of `v` is `vulgare`:

```
<a href="/btn/plantsbase?gnd=taraxacum&spc=vulgare"
   target="window2">Dandelion</a>
```

The other main usage is for specifying the processing of data that have been entered into a web form. The following is a simple example in WRDL notation:

```
(:form
   (:formparams target "window2")
   "Gender: "   (field g text "") "<br>"
   "Species: "  (field v text "") "<br>"
   (:invoke "Enter"
            (dynplant (check (gpv g)) (gpv v)) ))
```

This WRDL expression produces the HTML for a webpage containing data input fields for two parameters `g` and `v`, preceded by the leading texts `Gender:` and `Species:`, respectively, as well as an invocation button labelled `Enter`. The second argument of the operation `:invoke` generates an `action` expression in the resulting HTML form which sends the input data, indirectly, to the dynamic webpage that the second argument represents. Notice that here the arguments for the resource designator `dynplant` can be arbitrary forms, which are are evaluated when the `Enter` button is clicked and in order to compute the arguments that are sent to the resource.

Implementationwise, such a form is realized using an auxiliary dynamic webpage that implements the (composite) argument expressions for the resource. The auxiliary webpage receives, as arguments, the data that are actually entered into the form, and invokes the existing resource (`dynplant`) with the computed values of the arguments.

5 Using WRDL to characterize remote resources

WRDL was designed for use in defining our own websites, which we wished to structure in terms of a number of generic resources that could then be used as building-blocks for several albeit related applications. In these cases, each WRDL 'resource' was to be implemented on our own server, either using mapped file directories, or using dynamic web pages in LHTML.

We soon realized, however, that the same design was also very suitable for interfacing to web resources at other sites. In this case, WRDL is only used for declaring how a particular resource name is to be mapped to a URL that is foreign to our own system. This provides additional convenience for those cases where webpages in our application are to contain links to foreign webpages.

There is an obvious further next step, not yet implemented, where the data that is returned from the foreign sites is considered as input to computations in our own server, rather than being directed to the user. This requires the use of wrappers that parse the HTML data provided by the foreign site, or whatever other format provided by it.

6 Related work, experience so far, and future plans

The initial idea for the WRDL system described here came from the work by McIlraith and Son (2002) using A.I. planning and plan execution techniques for building systems that solve problems by accessing multiple foreign web resources and combining the results. Several other systems of this kind are also being developed in the context of the semantic web; see e.g. Knoblock (2003).

In the WRDL package we have excluded such longer-range research aspects, at least for the present time, and focussed on building a system that meets practical needs in the development of the websites that we have in operation today. These are small-scale websites that serve a research group or a committee. Rapid adaptation of the system to the evolving needs of a small user community is the most important consideration in such cases. Good performance is a requirement but never becomes an issue.

It has been our goal, therefore, to develop a concise system that implements a few basic ideas: abstract resources, some legibility-motivated macro expansion on top of LHTML, and a few support packages, in particular for access control and password management. The databases in these applications have usually been represented as plain list structures that are kept on property-lists at runtime and as S-expressions on text files between runs. In one case we have used a MySQL database via the ACL-to-MySQL interface.

In summary, we have tried to keep the system simple and coherent, so that it can easily be put to use for additional applications.

Copies of the system are freely available, together with a moderately detailed documentation. Please email the author for additional details.

Our experience with the ACL webserver has been very positive. It is robust and flexible, and we have not had any problems with it at all during one year of operation. We use it both for sites that provide regular service over the Internet, and as an off-line tool while travelling: the author can access and edit data on his laptop using a copy of the software and the database that we have on the general server.

Our future plans do not necessarily exclude going in the direction of the 'semantic web'. However, our highest priority at this time is to use WRDL for web-enabling the 'software individuals' that are the topic of another but related recent research article (Sandewall, sfp).

6.1 References

McIlraith, S. and Son, T. "Adapting Golog for Composition of Semantic Web Services". Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), pages 482-493, April, 2002.

Knoblock, C. "Deploying Information Agents on the Web". Invited paper, International Joint Conference on Artificial Intelligence, 2003.

Sandewall, E "A Software Architecture for A.I. Systems Based on Self-Modifying Software Individuals". Submitted for publication.