Information-Hiding URLs for Easier Website Evolution

Charles Song and Vibha Sazawal
Department of Computer Science
University of Maryland
College Park, Maryland, USA
{csfalcon, vibha}@cs.umd.edu

Abstract

Many common elements of URLs do not adhere to the principle of information hiding. For example, filename extensions and parameter names can reveal volatile implementation details. As a result, when website implementations change, links between pages break. Bookmarks and code that generates URLs often break as well.

In this paper, we present two tools for information-hiding URLs. An information-hiding URL uses an alias to identify a web resource and appends parameter values into the hierarchical structure of the URL. The InformationHidingFilter uses a Java Servlet filter to facilitate the use of information-hiding URLs with JSP/Servlet web applications. Given a request, the filter identifies the JSP or Servlet being requested and identifies parameter values contained in the information-hiding URL. Required values not provided in the URL are automatically substituted with default values specified by the web developer. Thus, old links remain valid even when the website changes and new parameters have been added to the page. The InformationHidingChecker helps web developers adhere to information hiding by helping them identify JSPs or Servlets that lack URL information for the InformationHidingFilter or lack default values for parameters. We also discuss the performance cost of using information-hiding URLs.

1. Introduction

The uniform resource locator (URL) plays a public role in the Internet. Popular browsers display a web page's URL in a prominent, top center location. Users commonly enter URLs directly, bookmark them, and share them. Web pages link to other web pages via URLs. In effect, the URL serves as the *interface* between a web page and those who wish to see that page. The software engineering literature contains extensive research on interface design; the most influential of this work is David Parnas' principle of *information hiding*. In his seminal paper on information hiding, Parnas designed a software system such that each module's "interface or definition was chosen to reveal as little as possible about its inner workings" [13, p. 1056].

After even shallow examination, it becomes clear that many elements of URLs do not hide the inner workings of anything. Filename extensions, such as ".jsp," reveal details about how the website is implemented. Parameter names, such as "hl=en&q=website+evolution&btnG=Google+Search," also reveal server-side details. As a result, when website implementations change, URLs will change too. When these URLs change, links between pages break. Bookmarks and code that generates URLs may break as well.

Broken links matter [4, 6, 10, 15]. When links and bookmarks break over time, user experiences suffer. Customers lose the ability to navigate through sites smoothly. More importantly, information can get lost. Web developers can expend effort updating URLs as website implementation details change, but they can't control links created outside their organization.

To lower the effect of website change on URLs, we propose to apply Parnas' approach for designing software module interfaces to the URL. In this paper, we introduce the **information-hiding URL**. An information-hiding URL identifies a web resource indirectly via an alias and embeds parameter values into the hierarchical structure of the URL. If a programmer follows certain conventions, such as providing default values for parameters, *a link or bookmark defined using an information-hiding URL will not break even if many details about the page have changed.*

We also present two tools for information-hiding URLs that support their use with JSP and Servlet web applications. The **InformationHidingFilter** uses a Java Servlet filter; given a request, the filter identifies the JSP or Servlet being requested and then identifies parameter values contained in the information-hiding URL. Values not provided

in the URL are filled with default values specified by the web developer. Thus, old links remain valid even when the website changes and new parameters have been added to the page. The **InformationHidingChecker** is another tool that identifies JSPs or Servlets that lack URL information for the InformationHidingFilter or lack default values for parameters. While in a "debug mode," accessing such offending JSPs or Servlets with a browser will produce errors.

The contributions of this research are (1) a well-defined standard for information-hiding URLs that can be easily implemented across server platforms, and (2) a set of tools that enable both *support* and *enforcement* of information-hiding URLs for one extremely common server-side platform (Java Servlets and JSP).

In the next section, we present background information on Parnas-style interfaces and describe their application to the web domain. In section 3, we present implementation details about the InformationHidingFilter and the InformationHidingChecker. Section 4 discusses the performance implications of using information-hiding URLs. Section 5 discusses related work, Section 6 proposes future work, and Section 7 concludes.

2. The information-hiding URL

In *A Procedure for Design Abstract Interfaces for Device Interface Modules*, Britton, Parker, and Parnas elaborated on the properties of good interfaces [5] first introduced by Parnas in 1972. They defined the *abstract interface* of a module as the *list of assumptions* that clients may make about the module. This list should omit details that would change if the module is replaced or if the module evolves in likely ways.

Britton et al. suggest that the assumptions be written in two ways – first as a literal list of assumptions written in natural language, and then as a set of programming constructs that can be directly accessed by clients. The first list makes assumptions explicit; implicit assumptions can otherwise go unnoticed. The second way – one or more signatures in a programming language – is mandatory, because without it, client code cannot access module functionality.

Applying these ideas to the web domain, we present the *information-hiding URL*. The information-hiding URL is the analogue of the mandatory programming construct. Information-hiding URLs omit details that are likely to change, while still identifying a web resource of interest. Using information-hiding URLs, links and bookmarks are unaffected by likely changes.

We focus on three likely changes:

1. changes to server-side implementation technology
2. features are added to the webpage, causing parameters to be added
3. features are removed from the webpage, causing parameters to be removed

We chose these types of changes because we believe they are likely and because they are poorly handled by other strategies such as redirection.

How can a URL remain valid in the presence of these types of changes? An information-hiding URL uses an implementation-independent alias to identify a web resource and appends parameter values into the hierarchical structure of the URL in the order specified by the developer. For example, A Java Server Page URL typically looks like: http://www.example.com/directory/resource.jsp?param1=value1&param2=value2. The information-hiding URL counterpart of the above URL would be: http://www.example.com/directory/resource/value1/value2.

With the information-hiding URL, volatile implementation details such as the platform (e.g., JSP) and parameter names are hidden from the user of the web application. The implementation-independent alias used in the information-hiding URL does not change when the web resource's name or choice of implementation platform changes. Since names of the parameters are also hidden, web developers are free to change these names.

In addition, the information-hiding URL can handle the addition and retirement of parameters without breaking old links or bookmarks to previous versions. If a new parameter is required by the web resource, the new parameter is appended to an existing information-hiding URL to create a new URL. If the web resource is accessed with an old version of the URL, the missing parameter value is filled with a default value that web developers are required to provide. If a parameter is no longer needed by the web resource, the values of the retired parameter in the old information-hiding URL are simply ignored. In new links to that information hiding URL, the special keyword `nil` is used in the location of the retired parameter to hold its place.

To use information-hiding URLs properly, programmers must follow two conventions. First, they must design their web pages so that sensible default values can be assigned to parameters. Second, they should avoid modifying parameters; instead of modifying an existing parameter, a new parameter should be created and the old one should be retired. These conventions help maximize the likelihood that links to information-hiding URLs continue to work correctly even when changes to the underlying page have been made.

3. Tool support for information-hiding URLs

In this section, we present two tools that support the use of information-hiding URLs with JSP/Servlet applications. While it may seem strange to provide tool support that is

platform-specific when information-hiding URLs intend to hide platform changes, it is vitally important to separate the *concept* of the information-hiding URL from any one platform-specific tool that supports such information-hiding URLs. The tool support we present here is a proof of concept that we envision spreading to all platforms, as the JVM has for Java.

3.1. InformationHidingFilter

The InformationHidingFilter is (not surprisingly) implemented as a Servlet Filter. Java Servlet Filters [7] are entities that sit between an HTTP request and the JSP or Servlet being requested. When the web application is configured to allow it, these filters are invoked for every incoming request, and they can choose to manipulate the request or throw away the request before the intended target is reached.

At runtime, the InformationHidingFilter receives requests to information-hiding URLs, transforms these URLs into implementation-specific URLs the web application understands and then forwards the request to the web application. The InformationHidingFilter performs the transformation from information-hiding URL to non-information-hiding URL by reading metadata embedded in JSP files or Servlet mappings. We refer to that metadata as *information-hiding configuration.* To use the InformationHidingFilter, all web developers must do is add an entry to their web application's `web.xml` file to include the filter and enter the information-hiding configuration.

3.1.1 Configuration metadata

The information-hiding configuration metadata describes a web resource and the parameters it requires. Figure 1 shows the information-hiding configuration needed by the information-hiding URL in Section 2.

To separate the information-hiding configuration from other comments, the usual HTML comment tags `<!--` and `-->` are augmented with `:>` and `<:`. In other words, the InformationHidingFilter looks for comments demarcated by `<!--:>` and `<:-->`.[1]

The configuration is formatted in XML. The root element is the web resource. Inside the resource, a mandatory name element specifies the implementation-independent alias of this web resource. Then, an optional description element can follow the name element; in the description, developers can explain details about the web resource and document assumptions in natural language. Next, zero or more param elements follow to specify the parameters required by the web resource and the order in which they will appear in the information-hiding URL. Inside each param element are three required elements and one optional element: a name element, a type element, and a default-value element, followed by an optional description element.

```
<!--:>
<resource>
  <name>resource</name>
  <description>resource description</description>
  <param>
     <name>param1</name>
     <type>string</type>
     <default-value>def1</default-value>
     <description>parameter 1</description>
  </param>
  <param>
     <name>param2</name>
     <type>string</type>
     <default-value>def2</default-value>
     <description>parameter 2</description>
  </param>
</resource>
<:-->
```

Figure 1. Configuration for the information-hiding URL http://www.example.com/directory/resource/value1/value2**. Web developers enter this meta-data manually.**

In our current implementation, the name element can be any string, and the default value can also be any string, although the string "nil" has special meaning as described in Section 3.1.4 below. The type element must be entered by the web developer as documentation, but the InformationHidingFilter does not currently use that information; support for type checking is future work.

JSPs and Servlets have different locations to embed these configurations. For JSPs, the configurations are located inside the JSP file, preferably at the beginning of the file. For Servlets, the configurations are located inside the `web.xml` file of the web application[2]; a Servlet's configuration is inserted in the mapping for that Servlet. Figures 2 and 3 show examples of configurations that we added to an existing JSP and Servlet when modifying them to use information-hiding URLs.

Figure 2's metadata allows the InformationHidingFilter to translate the information-hiding URL of http://[hostname]/photodb/photo-list/colorado-national-monument/ to the URL that the web application expects: http://[hostname]/photodb/photo-list.jsp?name=colorado-national-monument. Similarly, Figure 3's metadata allows the filter to trans-

[1] The `:>` syntax was inspired by the "opaque signature" feature of Standard ML [17], because the opaque signature hides implementation details of a structure from clients using the signature. However, this source of inspiration does not imply that the information-hiding configuration offers any kind of SML-like features.

[2] `web.xml` describes deployment details of Java web applications.

```
<!--:>
<resource>
  <name>photo-list</name>
    <param>
      <name>name</name>
      <type>string</type>
      <default-value>nil</default-value>
    </param>
</resource>
<:-->

<!-- beginning of the rest of the jsp -->
<%@ page errorPage="error.jsp" %>
<%@ page import="com.magiccookie.photodb.PhotoL...
<%@ page import="com.magiccookie.photodb.Util" %>
<%@ page import="com.magiccookie.html.HtmlUtil" %>
```

Figure 2. Information-hiding configuration that we added to photo-list.jsp, a file in the PhotoDB application [8] by Ari Halberstadt.

late from http://[hostname]/best_sellers/ARTS/0/ to http://[hostname]/TPCW_best_sellers_Servlet?subject=ARTS&SHOPPING_ID=0.

3.1.2 InformationHidingFilter initialization

The InformationHidingFilter initializes when the web application starts. All of the initialization code is placed in the `init(FilterConfig)` method from the Servlet Filter interface. In this initialization method, the InformationHidingFilter builds a lookup table that enables fast translation from information-hiding URLs to the "regular" URLs that the web application expects.

The InformationHidingFilter builds the lookup table in two steps. In the first step, the filter generates mappings for Java Server Pages. The filter begins by recursively searching for JSP files in the web application directory. When a JSP file is located, the filter will extract information-hiding configuration from the JSP file and use it to store details about parameters for that JSP file in the lookup table. In the second step, the filter generates similar mappings for Servlets from `web.xml`.

3.1.3 InformationHidingFilter operation

When a request to an information-hiding URL is issued to the web application, the InformationHidingFilter will catch the request and perform a URL transformation. First, the InformationHidingFilter will try to identify the requested resource using the lookup table mentioned above. The filter will match the longest alias contained in the URL. This

```
<!-- beginning of the servlet mapping -->
<servlet-mapping>
  <servlet-name>
    TPCW_best_sellers_servlet</servlet-name>
  <url-pattern>
    /TPCW_best_sellers_servlet</url-pattern>

<!--:>
<resource>
  <name>best_sellers</name>
    <param>
      <name>subject</name>
      <type>string</type>
      <default-value>nil</default-value>
    </param>
    <param>
      <name>SHOPPING_ID</name>
      <type>string</type>
      <default-value>nil</default-value>
   </param>
</resource>
<:-->

</Servlet-mapping>
```

Figure 3. Information-hiding configuration that we added for the TPCW best sellers Servlet, a Servlet in the TPC-W benchmark [16].

is done by truncating the incoming URL from the rear one "directory" at a time until a mapping is found. When the mapping is found, the filter builds a new URL to the requested JSP file or Servlet.

The next step is to append the parameter values from the incoming request to the JSP file or Servlet. The "directories" not used during the resource lookup are treated as parameters. The first directory name is the value of the first parameter specified in the mapping and so on. If the filter runs out of "directories" before all parameters are assigned a value, default values are used. After the URL transformation is done, the InformationHidingFilter forwards the request to the transformed URL for the web application to process.

3.1.4 Nil values

A special value, `nil`, can be used in the information-hiding configuration and the information-hiding URL. In a configuration, `nil` can be specified as the default value of a parameter. If the incoming request does not specify a value for one or more trailing parameters in the information-hiding URL, and the default values for these parameters are `nil`, then the InformationHidingFilter will not pass these parameters to the web application.

In an information-hiding URL, `nil` can be used as an embedded value. When `nil` appears in an information-hiding URL, the InformationHidingFilter will not pass any value for the parameter that corresponds to that location. `nil` thus serves in this case as a place holder for an unused parameter.

3.1.5 Documentation of information-hiding URLs

To fully support Parnas' vision for information-hiding interfaces, web developers must also document assumptions in natural language text. To encourage programmers to write down and maintain such documentation, the InformationHidingFilter generates javadoc-like HTML documentation from information-hiding configurations in a format that is easy to access and read. The documentation displays a list of web resources available in the web application, and for each web resource it displays all the details and assumptions developers put in the information-hiding configuration. To view the documentation, web developers can access the special URL /docs in each web application. Developers can use this feature to inform others about a web application and the parameters it accepts.

3.2. InformationHidingChecker

The InformationHidingChecker is a tool that helps web developers adhere to information-hiding URL usage. We see the InformationHidingChecker as integral to the information-hiding URL concept – with it, the information-hiding URL is upgraded from a collection of optional language or framework features to an intentional use of such features to consistently adhere to information hiding throughout a web application. The InformationHidingChecker is integrated into the InformationHidingFilter; in fact, this checker is the debug mode of the InformationHidingFilter. This tool can be enabled or disabled by editing an initial parameter in the `web.xml` file of the web application.

When the checker tool is enabled, it can provide error messages when a web resource does not have any information-hiding configuration or when the configuration contains omissions or errors. When a request is issued to a web resource with missing or malformed information-hiding configuration, the error messages are displayed in the browser that is sending the request. The error message style was designed to mimic other JSP and Servlet errors sent by Apache Tomcat [2]. When one or more errors are detected, the InformationHidingChecker will discard the request and instead respond with an error message. Errors detected by the InformationHidingChecker are described in Table 1. Figure 4 shows a screenshot of an InformationHidingChecker error.

Table 1. InformationHidingChecker Error Messages

Error	Description of Error
URL mapping not found	information-hiding configuration is not found or name element inside configuration is not found
Duplicate parameter name found	two parameters have the same name in the same information-hiding configuration
Parameter name not found	param's name element is not found
Parameter type not found	param's type element is not found
Parameter default value not found	param's default value is not found

When the InformationHidingChecker is disabled, the InformationHidingFilter will only use well-formed information-hiding configurations to transform incoming URLs; if any error is detected, the configuration is ignored and requests to the corresponding web resource are forwarded to the web server unchanged.

Figure 5 summarizes how our system works. When a request comes in, the InformationHidingFilter looks to see if that request is for a resource it knows about. If the filter has valid information for that web resource, then it will construct a transformed URL and send that to the web application. If the filter does not have valid information for that resource and the debug mode is turned on, then an error message is displayed in the browser.

4. Performance evaluation

We measured the performance of two web applications to evaluate the performance cost of the InformationHidingFilter. For each of the web applications, we created two copies on the server; one copy is an unmodified version and the other copy was manually modified to support the InformationHidingFilter. Each version runs within its own Apache Tomcat server. The modified versions were created by manually adding information-hiding configurations to JSP files and Servlet mappings. The InformationHidingFilter itself was also added to each modified version.

We picked Apache JMeter [3] as our test client; this tool issues HTTP requests to the web applications and records the time to complete the requests. The Tomcat server and JMeter test client were running on the same machine for our evaluations; this approach removed network latency which can vary substantially.

We ran a JMeter "test case" fifty times for each version of

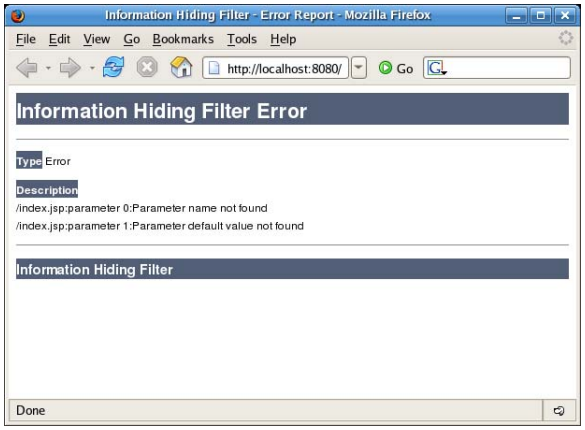

Figure 4. An error page served by the InformationHidingChecker. The offending file fails to list one parameter's name and doesn't include a default value for another parameter.

each web application. A test case consists of a set of HTTP requests (the same web pages are visited in each test case). We interleaved the test cases for the original and modified versions; the original version would execute a test case first, and then the modified version would run and so on. For each evaluation, we started the Tomcat server containing the version of the web application being tested, ran the JMeter test case, and then stopped the Tomcat server. We did this to ensure the initial compilation of JSPs and loading of Servlets are reflected in our tests.

The test machine was a dual core Dell XPS 600 with a 3.2 Ghz Pentium D processor, 2.0 GB RAM, and a 160 GB hard drive. Software running on the machine included RedHat Enterprise Linux WS release 4, Sun Microsystem Java SDK 1.5.0_07, Apache Tomcat 5.5.17, Apache JMeter 2.1.1, MySQL 14.7, and MySQL Connector 3.0.17 ga.

4.1. PhotoDB

To evaluate InformationHidingFilter's performance on a web application composed of JSP files, we chose PhotoDB, a photography database. The PhotoDB tested in this paper was photodb-20030523-114014 distributed by http://www.magiccookie.com/software/photodb/dist/. A MySQL database was populated by the scripts provided in the PhotoDB package, and the two versions of this web application shared the database.

A total of 28 unique URLs were gathered by a web spider [11] from PhotoDB: 12 that took no parameters and 16 that took 1 parameter. To create the modified version, information-hiding configurations were added to those JSP files (16 in total) that were intended for direct access via the web browser.

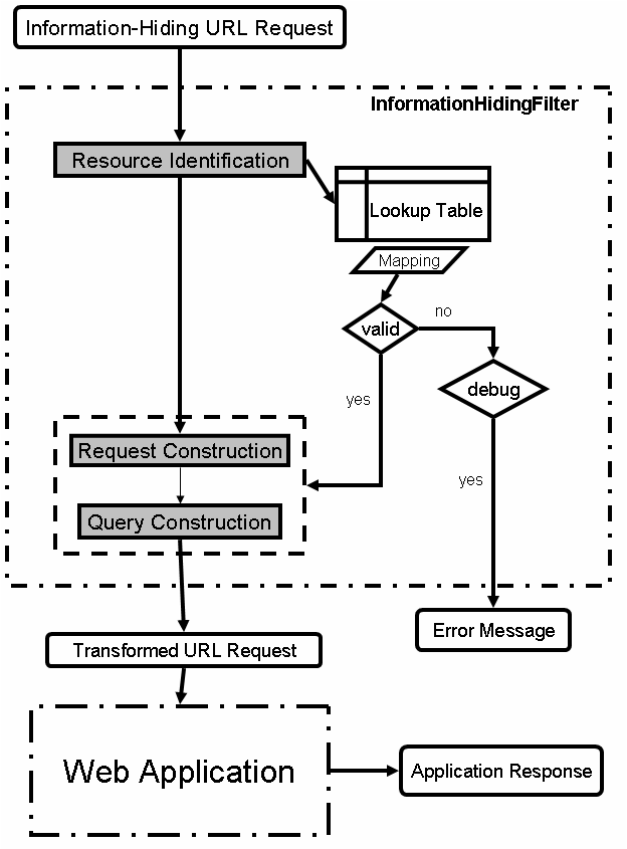

Figure 5. Processing of Information-Hiding URLs

The test case for both versions of PhotoDB consisted of 50 HTTP requests to each of the 28 URLs. For each test case, the amount of time to complete all 1400 HTTP requests was recorded.

4.2. TPC Benchmark W

We used TPC Benchmark W (TPC-W), a transactional web benchmark, as our Servlet web application. We installed the TPC-W Java implementation packaged by http://mitglied.lycos.de/jankiefer/tpcw/. A MySQL database was populated by the scripts provided in the TPC-W Java implementation package. Again, the two versions of this web application shared the database.

TPC-W contains 14 Servlets. To modify TPC-W to use the InformationHidingFilter, all 14 Servlet mappings were modified by embedding information-hiding configurations. A total of 1292 unique URLs were gathered by the web spider. Of the 1292 unique URLs, 1 had no parameters, 149 had one parameter, 1037 had two parameters, and 105

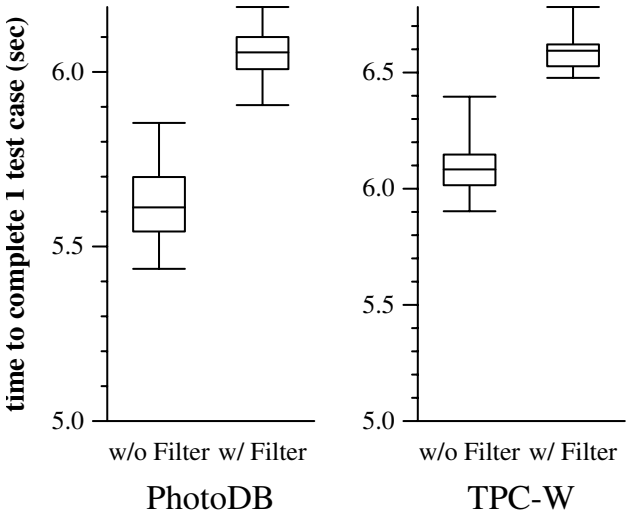

Figure 6. Box-and-whiskers plot of performance evaluation results

had four parameters.

The test case for TPC-W consisted of 1292 HTTP requests to the application. Due to an unstable JDBC connection observed for this web application, only one request for each unique URL was issued per test case. For each test case, the total amount of time to complete all 1292 HTTP requests was recorded.

4.3. Results

Figure 6 shows a box-and-whiskers plot of results from the tests. The whiskers are the minimum and maximum running times for each version, and the box outlines the middle 50% of running times. The line inside the box is the median running time across all test cases. The median is also shown in Table 2.

Given this experimental setup, the percent difference between the median times for unmodified and modified versions of both web applications is between 7.9 and 8.4%. We believe that this level of overhead is promising given the straightforward, unoptimized status of our current filter implementation.

5. Related work

5.1. URL rewriting with Ruby on Rails

Using routing [9], Ruby on Rails programmers can also pass parameters in the hierarchical structure of the URL. Instead of maintaining a single lookup table for the entire application as the InformationHidingFilter does, a parameter hashtable is maintained by each controller class. With Ruby code, parameters can be typed and also specified as optional.

version	# of test cases	# HTTP requests per test case	median run time (sec)	% Diff
PhotoDB (unmodified)	50	1400	5612	7.91%
PhotoDB (modified)	50	1400	6056	
TPC-W (unmodified)	50	1292	6083.5	8.38%
TPC-W (modified)	50	1292	6593.5	

Table 2. Median running times for test cases completed to evaluate the overhead of the InformationHidingFilter.

We believe that typical Rails URLs are quite close to the information-hiding URLs discussed in this paper. A crucial missing piece, however, is an InformationHidingChecker that enforces use of information-hiding URLs across an entire web application.

5.2. Non-URL means of sending form parameters

The HTML specification defines POST as a mechanism for sending form parameters without passing them on the URL. Web application frameworks such as Struts [1] also enable storage of form parameters directly into Java beans. While these methods do hide parameters from clients, they also limit the ability of clients to link and bookmark to certain kinds of functionality. The information-hiding URL enables links and bookmarks to parameter-passing URLs in a way that eases evolution.

5.3. Automatic refactoring for improved web evolution

Xu and Dean [18] present an automated technique for separating presentation markup from business logic. This work also promotes Parnas' information hiding principle because Parnas argued that unrelated design decisions should not co-mingle and instead should be separated by information-hiding interfaces. In the case of Xu and Dean's work, the information-hiding interfaces are custom JSP tags.

Ping and Kontogiannis [14] propose automatically restructuring websites to a controller-centric architecture. In

this architecture, web pages do not link to each other directly, but instead link to a controller that acts as a middleman between web pages. This system can lower the cost of changes to webpage names or to webpage locations, because only the controller needs to be notified of a change, not all the other web pages that link indirectly to the modified page.

Ping and Kontogiannis' approach is complementary to our approach. In addition to the use of a controller-centric architecture, information-hiding URLs could also be used to hide details about parameters and server-side implementation specifics.

5.4. Link versioning

Pan et al. [12] built Chrysant, a version control system for hypertext that supports HTML content. When used to store HTML, Chrysant versions both webpage content and the structure of links between web pages in a website. Link versioning support certainly helps web developers monitor the evolution of web links, and it is complementary to the InformationHidingFilter's goal of reduced overall URL change.

6. Future work

Type checking would be a useful addition to the InformationHidingFilter. Parameter types such as integers, doubles, and enumerations could be supported; if links include parameters of the wrong types, type mismatch exceptions could be sent to the web application for robust handling.

Another improvement to the InformationHidingFilter is support for runtime update of the information-hiding configurations. Currently, metadata are read once during the startup stage of the web application, but many web applications need to be updated while they are still live.

Finally, the tools could be implemented for other web platforms, such as Ruby on Rails.

7. Conclusion

Information-hiding URLs essentially ask the programmer to perform a small amount of work in order to receive a large amount of benefit. The programmer documents metadata as needed and designs their web applications in a way that default values can always be assigned to parameters. In return, links and bookmarks are highly likely to remain valid over time.

The InformationHidingFilter and InformationHidingChecker demonstrate how tools can support information-hiding URL use. The InformationHidingFilter efficiently translates information-hiding URLs into URLs that web applications expect. The InformationHidingChecker helps programmers enforce information-hiding across their entire web application. Future work includes expanding the functionality of these and other tools to enable even more expressive interfaces for web applications.

Acknowledgments

We gratefully thank Neil Spring, Jeff Hollingsworth, Chris Hayden, and Ken Yasuhara for their helpful advice and feedback.

References

[1] Apache Foundation. Struts. [http://struts.apache.org].
[2] Apache Foundation. Tomcat. [http://tomcat.apache.org].
[3] Apache Jakarta Project. JMeter. [http://jakarta.apache.org/jmeter].
[4] T. Berners-Lee. Cool URI's don't change. [http://www.w3.org/Provider/Style/URI].
[5] K. Britton, R. A. Parker, and D. Parnas. A procedure for designing abstract interfaces for device interface modules. In *Proc. of the 5th Intl. Conf. on Software Engineering (ICSE)*, 1981.
[6] R. P. Dellavalle, E. J. Hester, L. F. Heilig, A. L. Drake, J. W. Kuntzman, M. Graber, and L. M. Schilling. Going, going, gone: Lost internet references. *Science*, 302(5646), 2003.
[7] J. Falkner and K. Jones. *Servlets and JavaServer Pages*. Addison-Wesley, 2004.
[8] A. Halberstadt. Photodb. http://www.magiccookie.com.
[9] D. H. Hansson et al. Routing: Native ruby rewriting. [http://manuals.rubyonrails.com/read/chapter/65].
[10] S. Lawrence, D. M. Pennock, G. W. Flake, R. Krovetz, F. M. Coetzee, E. Glover, F. A. Nielsen, A. Kruger, and C. L. Giles. Persistence of web references in scientific research. *Computer*, 34(2), 2001.
[11] R. Miller. Websphinx. [http://www.cs.cmu.edu/ rcm/websphinx/].
[12] K. Pan, E. J. Whitehead, and G. Ge. Hypertext versioning for embedded link models. In *Proc. of HyperText*, 2004.
[13] D. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, Dec. 1972.
[14] Y. Ping and K. Kontogiannis. Refactoring web sites to the controller-centric architecture. In *Proc. of the European Conf. on Software Maintenance and Reengineering*, 2004.
[15] D. Spinellis. The decay and failures of web references. *Communications of the ACM*, 46(1), 2003.
[16] TPC Council. TPC-W benchmark. [http://www.tpc.org].
[17] J. Ullman. *Elements of ML Programming*. Prentice Hall, 1997.
[18] S. Xu and T. Dean. Modernizing JavaServer Pages by transformation. In *Proc. of the Fifth IEEE Intl. Symp. on Web Site Evolution*, 2005.