USING THE NaSr ARCHITECTURAL STYLE TO SOLVE THE BROKEN HYPERLINK PROBLEM

R. Bashroush, I. Spence, P. Kilpatrick, T. J. Brown
School of Computer Science,
Queen's University Belfast, Belfast BT7 1NN, UK
{r.bashroush, i.spence, p.kilpatrick, tj.brown}@qub.ac.uk

Abstract

According to a Web usability study [1] by the Georgia Institute of Technology, net users rate broken hyperlinks as the second-biggest problem online, right behind slow-loading pages. The problem of broken hyperlinks (or dead links) is a common problem within enterprise websites comprising hundreds or thousands of interconnected web pages that are continuously modified and updated [2]. Broken links can occur due to many reasons such as file rename, delete or path modification, which are likely events within enterprises. Many commercial tools were developed to deal with this problem [3][4]. In this paper, we present a novel architecture for linking web pages following the NaSr style [5] that provides a potential solution for the broken hyperlink problem.

Keywords
Hyperlinks, Software Architecture, Architectural Styles

1. Introduction

A key feature of the World Wide Web (referred to as web hereinafter) is its ability to take readers instantly to related documents through hyperlinks. Some consider it the soul of the medium. But as many as one in five Web links that are more than a year old may be out of date, according to Andrei Broder, vice president of research at search engine AltaVista [6].

According to an April 1998 Web usability study by the Georgia Institute of Technology [1], net users rate broken links as the second-biggest problem online, right behind slow-loading pages.

Broken hyperlinks can create a negative experience for visitors of a Web site which could be a serious problem for e-Business. As Web sites increase in size, it becomes increasingly difficult to keep an accurate inventory of content that has been changed over time. Many techniques and commercial tools [3][4] have been developed to validate websites for broken links (section 4), however, few of them treated the problem rather than fixing it after it happened.

In this paper, we present a novel technique for linking web pages following the NaSr [5] style to solve the problem of broken hyperlinks.

NaSr was designed to provide a framework of communication among architectural components within highly configurable architectures. Looking at a web page as a component within an architecture (the website) linked to other components (web pages) via hyperlinks (connectors), we used the NaSr framework for inter-component (web pages) connectivity as it provides a solution for connection (hyperlinks) management.

An overview of the NaSr style is given in section 2. Section 3 presents the web page connectivity framework that follows the NaSr style. Related work is treated in section 4. Section 5 concludes and highlights future directions.

2. Overview of the NaSr Architectural Style

In this section we present an overview of the NaSr [5] style and its framework.

The NaSr Framework consists of:
- Components
- Connection handlers
- Communication Protocols

NaSr architectures consist of concurrently executing OTS or user defined components wrapped inside NaSr *Components* (Section 2.1) that utilize a packet driven method of communication using defined *communication protocols* (Section 2.3). The communication management is looked after by *Connection handlers* (Section 2.2).

In the following, we use the term Component to refer to NaSr Components (OTS or user defined Component(s) plus a *Domain Adapter*, Figure 1).

Within NaSr, every component is identified by a *unique ID* and provides/requires a specific set of *service(s).* This is a key feature of the NaSr style that allows the separation of the services provided/required in the system from the components providing them. The separation allows any component in the system to be replaced (due to failure) or backed up (due to overload) by another component(s) th at provides the same set of services without the need to reconfigure or restart the system.

The newly added component(s) can make itself known to the Service Translation Center STC (Section 2.2.1) by sending an appropriate *registration* message identifying the services it requires/provides. Then new calls for that given service will be routed by the *Connection handlers* to the newly added component. The reader can see here the solutions and scenarios adopted from the real networking domain. This architecture strongly supports system's reconfigurability and increases system uptime. Also notice the separation of connection management from computational components.

Components are described in 2.1, connection handlers in 2.2, and communication protocols in 2.3.

2.1. Components

A component in NaSr is a separate thread of execution. Each component wraps a user defined or OTS component (Figure 1) that can be developed using any language and employ any interface types (event based, message based, etc.). The communication can be utilized by employing a NaSr *Domain Adapter* that translates the wrapped component's interface to NaSr packet based communication following a desired protocol.

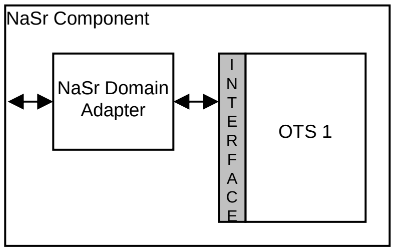

Figure 1. A NaSr Component wrapping an OTS Component (OTS 1) communicating with the system via the NaSr Domain Adapter

2.2. Connection handlers

Connection handlers are the objects of the NaSr style that handle the packet driven communication among components. Currently, we have identified three communication handling objects:

- Service Translation Center
- Communication Manager
- Broadcaster

These objects form the backbone for component communication using user-defined protocols. More objects and protocols can be developed in the future to allow more complicated communication services. The three objects are discussed next.

2.2.1. Service Translation Center (STC). This is a key object for building architectures using the NaSr style. It provides a translation table between services and the components providing them.

In general, the role of the STC in NaSr architectures is similar to the role of a DNS (Domain Name System) on an IP network. DNS provides translations between domain names and IP addresses whereas an STC translates between service names and components providing them.

The STC proved to be very useful in consistency and dependency architecture analysis for architecture verification.

2.2.2. Connection Manager (CM). The Connection Manager plays a similar role to a regular network router which forwards packets back and forth among the different machines on a given network.

In NaSr, the CM achieves this with the help of the STC. In a typical scenario, a component asks the CM for a desired service, the CM queries the STC and gets an ID for a component that provides this service. This ID is then passed to the requesting component and a direct communication among the two components takes place.

2.2.3. Broadcaster. A Broadcaster is similar to an Ethernet HUB. It can work without the need for an STC (compared to a Connection Manager) by broadcasting incoming packets to all components on the network. This could be useful on smaller architectures that do not require the overhead of an STC and packet routing. When a broadcaster is deployed to connect a group of components, the component communication will be similar to the Event-based Integration style [8].

2.3. Communication Protocols

Different communication protocols can be designed to serve different domain functional and nonfunctional attributes. For example, in concurrent time-critical systems, a time-stamp field would be included within the packet header to enable communication synchronization. In one system, we can have more than one protocol in use at the same time. To better understand that, consider two different sub-nets running different protocols to best suit their applications, but still they can communicate via a backbone structure.

For more information on the exact conceptual background and the structure of the NaSr style, please refer to [5].

3. Using the NaSr style as the framework for web page linking

As described in section 2, the NaSr style allows communication among components within the architecture via *Connection Managers* (message routers) that use

Service Translation Centers (an accurate inventory of the documents available) to locate other components.

Now let's see how this maps to our domain of concern. We modeled our domain as follows:

- A document (or web page) is a component
- Documents are connected using hyperlinks
- Every document has a unique service name that other pages use to connect (hyperlink) to it.
- No pages are allowed to be linked directly to each other, instead, they are linked using service names via *Connection Managers*.
- A database (*Service Translation Center* as referred to in NaSr) exists that contains an accurate inventory of the documents available, their service names, physical location on the server (or else where), etc.

To better understand how the model works, we will take an example. Consider a simple website consisting of 4 documents in a single folder (e.g. myWeb). The documents are:

- *Home*: the home page of the website, with a link to google.com, and a link from a friend
- *latestNews*: a page containing latest news
- *links*: a list of links to external (on a different domain) pages
- *Resume*: pdf file

The pages are linked to each other as shown in figure 2 below.

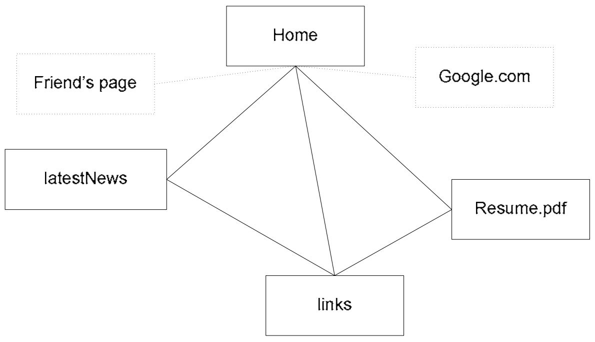

Figure 2. File structure view of the example website

We used dashed lines to express links to external web pages. The links page is also connected to other external web pages which we didn't show to preserve simplicity.

Assume you want to move the Home page to a different location (as your website is getting bigger and you want to keep the files tided). In this case, you have to go into each file that links to the Home page (and the links within the Home page itself) and modify the hyperlinks accordingly to reflect the new position of the Home page. With four files, this can be manageable, but imagine the case with hundreds and thousands of links. There are some tools that can help you automate this process to some extent, but still, what about the friend's link to the Home page.

Figure 3 below shows the same website but connected using the NaSr model we described before.

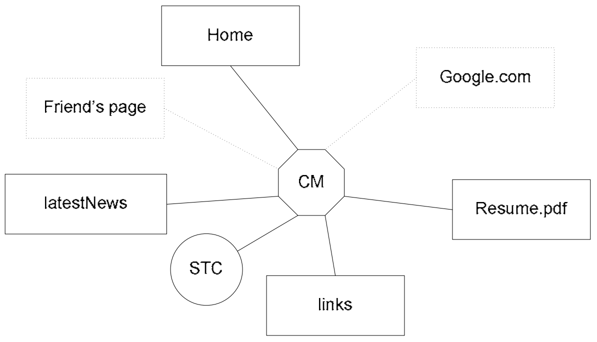

Figure 3. The example website using the NaSr style

In the figure above, we notice that no page is connected to the other. All pages are connected to one component, the *Connection Manager* CM, which handles the connectivity with the help of the STC (Service Translation Center). In the following, we discuss the CM and the STC and see how this architecture functions.

3.1. Connection Manager, CM

The Connection Manager, as explained before, is responsible of converting service calls to the physical location of the document assigned to the service.

In this example, we used PHP [9] to express the connection manager object. The different pages connect to the connection manager using hyperlinks similar to the ones shown in [figure 4 (a, b)] below.

When evaluated, the PHP statements return the physical location of the desired document. So from the user point of view, the process is transparent.

`route` is the function of the CM object that takes as an argument the service name and returns the physical link to the desired page. The `route` function in the CM calls internally the `getServiceNamePath` of the STC (section 3.2) to get the location of the document with the given service name.

The connection Manager can also be used to mark dead links (gray colored for example) to external pages using the `setColor` function [figure 4 (c)].

The `setColor` function returns `normal_color` if the link is *Active* (valid, see section 3.2) or `dead_color` if the link is *Inactive* (broken).

`setColor` can also be used in a different way to totally remove broken hyperlinks or forward the calls to a search web page for example, [figure 4 (d)].

Finally, figure 4 (e) shows how external pages can link to documents within our example website. No matter where the desired document is physically present, as long as the external call can reach the CM, the CM can query the STC and get the physical location of the desired resource and forward it to the surfer.

3.2. Service Translation Center, STC

The Service Translation Center is very similar in functionality to a Domain Name Servers, where it maps service names to resources. It also carries other information as shown in figure 5 below.

The STC can be implemented in different ways, in this example, we used a MySQL [10] database.

The table, shown in figure 5, could be automatically generated. When a page is added to the system, it is given a service name. Then, the page file is parsed and service names required are harvested and inserted in the services required field. The table is then checked for required services, and if all available, page is marked as broken links free, by unsetting the broken links field (b. Links) in the table.

If you want to deactivate a page for some reason (a dead link to an external page, being currently updated, it has crucial broken links, etc.), you unset the *Active* flag. Any incoming requests would then be redirected to a pre-set "Inactive page". The set or unset of the Active field of a page would cause the table to be re-evaluated so that the *b. Links* tags of other pages linking to the just deactivated page are set).

Load index can be used to indicate the load on the current page, it is not been used at the minute due to the fact that the load actually would be on the machine running the page rather than one page, and in this case, all the pages on the same machine would be overloaded. However, it was left there as it might be useful in some other implementations, for example, with mirrored web servers.

The final field, free textual description, can be used to explain more about a service.

The basic set of APIs the STC should provide is:

`- String getServiceNamePath( "Service_name" ):` returns the full path to the `Service_name` provided (contents of the `Location` field).

`- void addService("Document file name", "Service Name"):` It adds the service with the given name to the database, parses the file to get the services required, and then sets the *b. links* tag accordingly.

`- void deleteService("service name"):` deletes the service with the given name and calls `updateTable` to update the *b. Links* fields.

`- void updateTable():` updates the *b. Links* field for all records based on validating services required against service provided.

`- boolean checkConsistency():` checks for integrity and validity/existence of the different pages. It can be run, for example, every interval of time on the table to insure that dead links to external pages are detected.

Other functions can also be developed according to system needs and functionality, e.g.: `getServiceDetails`, `setServiceDetails`, etc.

Tools can be created to operate over the STC table to draw some information of the system, e.g.: a tool that would construct the site map (by following service connectivity within pages).

4. Related work

There have been many approaches followed to tackle the problem of broken hyperlinks.

Some researchers went for analyzing the semantics of a document content so that it can be tracked, even if it has been moved or renamed, based on its semantic content. Phelps and Wilensky [7] proposed a solution they called *robust hyperlinks* where a URL is augmented with a small *signature* computed from the document the URL points to. They claimed that five words are enough to uniquely identify a document and compute a unique signature for it. However, they didn't provide any mechanism for dealing with deleted documents, web pages that do not exist at all. Also selected words may later be edited out of the document, rendering the identifier moot.

Others went for developing tools that can analyze existing websites for broken hyperlinks, and in some cases, fix them. There are many commercial tools available in the market [3][4] that can spot broken hyperlinks within a website. These tools could be used with static (slowly

```
Hyperlink: <a href=<?php echo connectionManager.route("personal links page");?> > Links </a>          [a]

Form:    <form action=<?php echo connectionManager.route("login form handler"); ?>  > ... </form>      [b]

Color coded links: <a href=<?PHP echo connectionManager.route("latest news");?>                        [c]
                      color= <?PHP connectionManager.setColor(normal_color, dead_color)?> >
                                                                        Latest News </a>

Removing broken links: if(connectionManager.setColor(normal_color, dead_color) = dead_color)           [d]
                          echo connectionManager.route("error page");
                        else echo connectionManager.route("personal links page");

External links:  <a href="http://yourdomainname/connectionManager.php?service="Latest News">           [e]
                                                               News at yourdomainname </a>
```

Figure 4. hyperlink examples within the NaSr framework changing) websites; however, they become ineffective with active websites where documents are changed many times a day, and running the tools to check for broken links after each update becomes an expensive and inefficient solution.

5. Conclusion and future work

In this research, we looked at designing a framework for web page linking that follows the NaSr architectural style [5]. The framework solves the problem of broken hyperlinks by separating the physical document path from its service name. Documents are then linked together via service names and using the connection manager.

Using the NaSr style for web development can also be useful in capturing other information [11] (like quality attributes, design decisions) about the system architecture, which could be useful for evaluating and validating the overall system architecture [12][13].

We are currently working on developing a tool that can convert existing websites to NaSr compatible systems. The tool goes first over all existing files and adds them to the STC along with the corresponding service names. Then files are parsed and links are modified and changed from direct links to service oriented links via the connection manager.

6. Acknowledgment

This research is based on projects funded by British Telecom and Nortel Networks.

We would like to thank the IASTED SEA 2004 conference reviewers for their valuable and useful comments.

7. References

[1] *GVU's 9th WWW User Survey, Web and Internet Use Summary*. URL: http://www.gvu.gatech.edu/user_surveys/survey-1998-04/reports/1998-04-Use.html

[2] Dawn Anfuso, iMediaConnection news. *Broken Links are Prevalent in Marketing*. URL: http://www.imediaconnection.com/news/965.asp

[3] *LinkFixerPlus*. Commercial tool. URL: http://www.linkfixerplus.com/html/html-overview.htm

[4] *HTML Link Validator*. Commercial tool. URL: http://lithopssoft.com/hlv/

[5] R. Bashroush, I. Spence, P. Kilpatrick, and TJ Brown. A Network Architectural Style for Real-time Systems: NaSr. *Proc. of the 4th Working IEEE/IFIP International Conference on Software Architecture WICSA'04*, Oslo, Norway, June 2004.

[6] Andrei Broder, vice president of research at AltaVista, March 7, 2000. URL: http://www.altavista.com

[7] T. Phelps, R. Wilensky. Robust Hyperlinks cost just five words each. *UCB Computer Science Technical Report UCB//CSD-00-1091*, 2000.

[8] D. Garlan and M. Shaw. An introduction to software architecture. Ambriola & Tortola (eds.), *Advances in Software Engineering & Knowledge Engineering*, vol. II, World Scientific Pub Co., Singapore, 1993, pp. 1-39.

[9] PHP: an open source server side scripting language used mainly for web development. URL: http://www.php.net

[10] MySQL: an open source database management system. URL: http://www.mysql.com

[11] R. Bashroush, I. Spence, P. Kilpatrick, and TJ Brown. A Generic Reference Software Architecture for Load Balancing over Mirrored Web Servers: NaSr Case Study. *Submitted to the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, Egypt, January 2005.

[12] R. Bashroush, I. Spence, P. Kilpatrick, and TJ Brown. Towards an Automated Evaluation Process for Software Architectures. *Proc. of the IASTED international conference on Software Engineering SE 2004*, Innsbruck, Austria, February 2004.

[13] R. Bashroush, I. Spence, P. Kilpatrick, and TJ Brown. The Contribution of Architecture Description Languages to the Evaluation of Software Architectures. *Proc. of the National IEEE/IEE/ACM PREP 2004 Conference*, Hertfordshire, UK, April 2004.

service	services required	Location	Active	Load Index	b. Links	free textual description
Links	[Home Page, Resume, etc.]	./files/myWeb/links.php	1	0	1	The links page to other web pages
login page	[login form handler]	./users/login.php	1	0	0	The private section login page
login form handler	[]	./security/formHandler.php	1	0	0	The page that handles the login form data
Google.com	EXTERNAL SERVICE	http://www.google.com	1	N/A	N/A	Link to Google.com

Figure 5. The Service Translation Center