Advanced Golden Pages Web Service

David Whyte

B.A. (Mod) CSLL

Final Year Project, May 2004
Supervisor: Alexis Donnelly
ABSTRACT

While it is now possible to buy almost anything from the Internet, the fact remains that most people would sooner buy a car or a computer from a real bricks-and-mortar shop, where they can physically see what they are about to buy. Similarly, selection of a plumber, electrician or GP is generally made on the basis of proximity and availability. Business directories such as the myriad of Yellow/Golden Pages companies around the world are still popular in book form. Websites such as www.goldenpages.ie, www.yell.co.uk and www.gelbeseiten.de have, however, also become increasingly popular in recent years, offering as they do many functional enhancements over their respective paper equivalents.

The aim of this project is to further enhance the functionality of such a website, giving the user access to a much richer store of information not only about the businesses, but precise details of the products and services that they offer. Such a service should bridge the gap between geographically oriented business directories and the huge online marketplaces such as ebay.com. It will allow users to compare and contrast the products and services offered by all of the vendors in a specific geographical area or, to put it another way, one will be able to find the computer shop selling the cheapest Dell laptops in the Fermoy, Co. Cork area.
Declaration
I hereby declare that this thesis is entirely my own work and that it has not
been submitted as an exercise for a degree at any other university

____________________ 29th of May, 2004
David Whyte
Acknowledgements

I would like to thank my project supervisor, Alexis Donnelly, for coming up with an initial idea that I was able to make my own, and have fun while doing it. My thanks also to Dr. Carl Vogel for his unerring support during my time in college and also for teaching me the dark art of Speed Scrabble.

I would not have got to this stage without the love, support (both financial and moral) and an occasional prod from my parents, and I am indebted to them for having faith in me even at times when I didn’t have faith in myself.

Finally, for his expertise, generosity, encouragement and criticism without which this project would be half of what it is, I am eternally indebted to my friend, James Bligh. The quote on the next page is for him…
These cows are small; the cows out there are far away.

- Father Ted Crilly
ABSTRACT ................................................................................................................. 2

1. INTRODUCTION ...................................................................................................... 7
   1.1 Objectives ......................................................................................................... 7

2. BACKGROUND .......................................................................................................... 10
   2.1 Introduction ...................................................................................................... 10
   2.2 UDDI .................................................................................................................. 10
   2.3 UDDI AND GOLDEN PAGES? ........................................................................ 11
   2.4 THE “IN-HOUSE” ALTERNATIVE .................................................................... 12
   2.5 TECHNOLOGIES ............................................................................................... 12
       2.5.1 Server side ................................................................................................ 12
       2.5.2 Client side ................................................................................................ 14

3. DESIGN ..................................................................................................................... 16
   3.1 DESIGN INTRODUCTION ............................................................................... 16
   3.2 ACCESSIBILITY ............................................................................................... 16
   3.3 SEARCH OPTIONS ............................................................................................ 16
       3.3.1 Searching by area code .......................................................................... 16
       3.3.2 Searching by location ............................................................................. 17
       3.3.3 Searching by ad caption ........................................................................ 17
   3.4 FILTERING AND SORTING THE RESULTS ..................................................... 19
       3.4.1 Categories ................................................................................................ 19
       3.4.2 Sort by column .......................................................................................... 21
       3.4.3 Expanding the search .............................................................................. 21
       3.4.4 Filtering the search results ...................................................................... 21
       3.4.5 Summary .................................................................................................. 22
   3.5 RICHER INTERACTION: DIRECT INTERFACING WITH SHOPS AND SERVICES ... 23
       3.5.1 Overview .................................................................................................. 23
       3.5.2 Browsing and querying a single shop ...................................................... 25
       3.5.3 Browsing and querying multiple shops ................................................... 25
       3.5.4 Smart forms .............................................................................................. 25
   3.6 SUMMARY ......................................................................................................... 26

4. IMPLEMENTATION ................................................................................................... 27
   4.1 IMPLEMENTATION INTRODUCTION ............................................................... 27
   4.2 BACK END IMPLEMENTATION ....................................................................... 27
       4.2.1 The MySQL database ............................................................................. 27
       4.2.2 queries.php ............................................................................................... 30
       4.2.3 multishop.php .......................................................................................... 38
   4.3 FRONT END IMPLEMENTATION ...................................................................... 46
       4.3.1 The “GUIs” ............................................................................................... 46
       4.3.2 Accessibility .............................................................................................. 47
   4.4 SUMMARY ......................................................................................................... 47
1. Introduction

Irish Internet users generated over 8.1 million searches on the Golden Pages Online site in 2002, an increase of 16% on 2001 (http://www.goldenpages.ie/extras/products.html). The site went live in 1998 and has steadily grown in popularity since then, as more and more Irish homes and businesses have connected to the Internet. The site offers a number of advantages over the print editions, first and foremost that all six print editions are combined together online. Users can search for products and services anywhere in Ireland but within very targetted areas, be they bike shops in Ringsend, farming equipment in Salthill or office supplies in Athlone.

The current incarnation of the online service still has a number of problems, however. It is not accessible to visually impaired users with screen-readers without some difficulty because of its layout. In one sense it has less functionality than the print edition, in that it lacks the picture ads that make up so much of the company’s revenue in the book. The main search page also has some very slow-rendering graphics, which dramatically increase loading time.

1.1 Objectives

This project has two objectives: the first is to redesign the Golden Pages site to improve its accessibility. There are also a number of basic search enhancements I would like to add.

The second objective is to expand the service to provide richer information to its users by allowing them to view not only details about the advertisers, but also to browse, compare and contrast their products and services.
1.2 Report Layout

This report is divided into four major sections, excluding the introduction and the references.

- Chapter 2 – **Background** is a description of the current state of the art and contains an outline of the design strategy that was subsequently pursued.

- Chapter 3 – **Design** discusses with the features that an Advanced Golden Pages system should offer, from a user’s perspective.

- Chapter 4 – **Implementation** is a detailed walkthrough of the code that was finally implemented server-side and client-side.

- Chapter 5 – **Conclusion** is an evaluation of the project and also lists possible areas for future work.
The finished queries page.
2. BACKGROUND

2.1 Introduction

This section gives some background information on an existing web standard for business discovery and description and analyses its relevance, if any, to a potential advanced web directory for local Irish businesses. It outlines the grounds for pursuing an independent design format rather than adopting this web standard. Finally, an overview of the technologies to be used in the implementation is provided.

2.2 UDDI

The Universal Discovery, Description and Integration initiative (http://uddi.org) is an emerging web service. Web services are a loosely defined set of open technologies such as XML, SOAP, WSDL and UDDI, which are intended to allow businesses to find and interact with each other on the web through a set of standardised tools. The UDDI system is based around registries in which businesses describe their products and services. There are currently four major UDDI registries online, hosted by IBM (http://uddi.ibm.com), Microsoft (http://uddi.microsoft.com), SAP (http://uddi.sap.com) and, most recently, NTT Communications (http://www.ntt.com/uddi/index-e.html). Within the registry, listed businesses are referred to as “nodes” and they contain the following information:

* Business Entity. A business entity represents information about a business. Each business entity contains a unique identifier, the business name, a short description of the business, some basic contact information, a list of categories and identifiers that describe the business, and a URL pointing to more information about the business.

* Business Service. Associated with the business entity is a list of business services offered by the business entity. Each business service entry contains a business description of the service, a list of categories that describe the service, and a list of pointers to references and information related to the service.

* Specification Pointers. Associated with each business service entry is a list of binding templates that point to specifications and other technical information about the service. For example, a binding template might point to a URL that supplies information on how to invoke the service. The specification pointers also associate the service with a service type.
* Service Types. A service type is defined by a tModel. Multiple businesses can offer the same type of service, as defined by the tModel. A tModel specifies information such as the tModel name, the name of the organization that published the tModel, a list of categories that describe the service type, and pointers to technical specifications for the service type such as interface definitions, message formats, message protocols, and security protocols.

(http://uddi.org/faqs.html)

2.3 UDDI and Golden Pages?

Golden Pages is currently primarily a “discovery” service. It functions as a simple listing of businesses in Ireland, sorted by business type and, in the case of the print edition, by area. The idea of adding “discovery” and “introduction” to this service is an appealing one and, since the UDDI protocol is an open standard which is there to be used, integrating the Golden Pages into UDDI in some way was the initial goal of this project. To this end, two options presented themselves. The first route would be to set up an independent Golden Pages UDDI registry, using the UDDI system of Nodes, Business Entities, tModels et al. While maintaining the simple functionality of the original, businesses would be given the opportunity to greatly increase the amount of information they could supply about themselves and their products and services on the web. Additionally, because UDDI is an open standard, there would exist the possibility of later integration with other “Yellow Pages” UDDI registries around the world, should they exist. The second option would be to set up the Golden Pages as a “Registrar”, a third-party company which would enter and maintain its clients’ details on one of the global UDDI registries.

In the end, neither of these ideas proved feasible. The private registry idea was hampered by the fact, although tools such as jUDDI (an Apache project) and UDDI4J (by IBM) for creating registries do exist, they are both very much in the developmental stage and documentation in each case is sparse. Furthermore, both implementations run on Java servlets, the recommended implementation of which, Tomcat, proved very unreliable and difficult to set up (http://nagoya.apache.org/bugzilla/buglist.cgi?rep_platform=PC&op_sys=Windows+NT%2F2K&bug_severity=Critical&product=Apache+httpd-2.0). At the time of writing, there is no stable release of jUDDI, only automated nightly CVS “snapshots”, and the last major release of UDDI4J was in May of 2002. If and when development ever picks up on
either of these two projects again, there is certainly potential for further work. A Golden Pages UDDI registry is a very good idea in principle but, for now, it is hampered by a lack of tools.

Moving the Golden Pages database into a global UDDI registry would present several problems, the first of which would be speed. The global registries are currently all in experimental phases and none of the controlling firms have invested much in the way of bandwidth or server speed. As such, fast and reliable queries to any of these registries, be the queries directly from one of the standard web interfaces such as http://uddi.microsoft.com or through a specialised Golden Pages interface, cannot be guaranteed.

2.4 The “In-House” alternative

The decision was eventually made to go “in house” and design a self-contained system from the ground up. This would have the advantage of being faster and more reliable than a Java/JSP/UDDI-based system. Instead, the system will use a combination of MySQL and XML databases controlled by a series of PHP scripts. My main reasons for opting to go this route were that I was already reasonably familiar with PHP and, having been introduced to SQL and XML at the start of this academic year, I was also eager to put some of this new-found knowledge into action in a practical application.

2.5 Technologies

2.5.1 Server side

The system was implemented on Apache/MySQL/PHP, a very popular platform for websites involving any amount of hypertext preprocessing.

Apache is the currently most commonly used web server in the World Wide Web. It was written in 1995 as a series of patches to the then dominant NCSA httpd (Hypertext Transfer Protocol Daemon), after development on the latter had stalled, and it has remained at the top of the Netcraft Web Server Survey graph since early 1996. It is free software, maintained by the Apache Software Foundation

MySQL is an open source database management system (DBMS) owned by MySQL AB. As the name suggests, it is a relational database, with its own version of SQL (Structured Query Language). It is currently the most widely used DBMS.
**PHP** (PHP: Hypertext Preprocessor) is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl. Like Perl, it has no strong data-typing.

- If $a = “5” (the ASCII character, not a number) and $b = “3”,
- $a+$b will return 8,
- $a*$b will return 15 and
- $a.$b will return “53”

A PHP file looks like a normal HTML file except for everything between the `<?php` and `?>` tags. Everything here is executed and evaluated by the PHP engine before being sent out to a browser window. Being server-side, PHP code is completely invisible to the end-user. A PHP file might look as follows:

```html
<html>
<body>
Today’s date is:
<?php
    $today = date("F jS, Y");
    echo $today."."
?>
</body>
</html>
```

All that will appear in the browser window when the file is called will be: “Today’s date is: April 28th, 2004.” and if one decides to view the source of the page from the browser, all that can be seen is:

```html
<html>
<body>
Today’s date is:
April 28th, 2004.</body>
</html>
```

A PHP script can do many more things than just print the date. Given the appropriate permissions, it can read in HTML forms via HTTP POST or GET, send and receive email, read, write and execute files on the server, handle file uploads from client
machines, generate and manipulate images, generate PS and PDF documents, generate and compress archive files, generate and authenticate OpenSSL signatures, provide HTTP authentication for access control, handle cookies, query databases and parse XML files, to name just a few of its capabilities.

These three software elements interact as follows: PHP is run as a module in the Apache server, a background process (a “daemon” in UNIX or a “service” in Windows) which listens for network requests on port 80, the default HTTP port. In practise, this means that HTTP requests for all files with the .php (or .php3, or .php4 or .phtml or any arbitrary extension nominated in the config file) are run through the PHP engine, usually with the intention of outputting hypertext. This is then sent back to Apache and served out to the requesting user – hence “hypertext preprocessing”.

PHP has extensions to interact with various well-known DBMS’s, including Oracle, PostgreSQL, MSSQL and the above mentioned MySQL. Assuming PHP has been started with the appropriate module, a script can then connect to and query any available database on the network (usually localhost, in practice).

PHP also comes bundled with an event-based XML parser extension, which has a number of automated functions for parsing XML documents and dropping the results into easily accessible associative arrays. An event-based parser ignores DTDs and does not automatically take note of the document hierarchy. Instead it treats each occurrence of tag and its attributes as an event which triggers some sort of post-defined function.

2.5.2 Client side

While the server-side application is focused on the retrieval of information, an equally important part of any piece of software is the user interface, which is expressed in the languages of the web of HTML and CSS. The Hypertext Markup Language has come a long way since the first ever website went live from http://nxoc01.cern.ch/ around the end of 1990. (http://www.w3.org/People/Berners-Lee/FAQ.html#Examples). There has been an increasing usage of images and other embedded multimedia; the web today is as much a graphical environment as a textual one, which is in sharp contrast to the first version of HTML, which did not even have an <IMG> tag. Another issue has been that of competing browser manufacturers often opting to parse HTML in different ways or even using their own proprietary tags outside of the W3C specification, with the end result
being that a page could look radically different depending on what browser one was
using, if the page even displayed at all.

The W3C’s solution to this problem was to formalise the way in which elements should
be displayed in a browser, by introducing the CSS specification. Style sheets have long
been used in the printing and publishing industries to define standard layouts for
documents so as to ensure a consistent appearance. Academic papers such as this one
tend to conform to a basic set of formatting rules: The main text here is in 12pt Times
New Roman font, 1.5 line spaced, with 6pt spacing between paragraphs. Main headers
are 14pt bold Arial, and so on.

One of the primary aims of the W3C is to make the web as accessible to everyone as
possible and, as such, this is also a primary goal of this project. (http://www.w3.org/TR/WCAG10/) The de facto use of tables as a way to lay out
elements on a page plays havoc with screen-readers for the visually impaired. Now that
CSS-capable browsers are more widespread, the official policy of the W3C is that tables
should be used only for representing actual tabular data. Positioning of elements such as
navigation boxes containing links or multi-column newspaper style articles should be
done with <div> elements, all of whose attributes can be precisely defined with CSS.
3. Design

3.1 Introduction

This section is concerned primarily with the motivations behind the design. It describes the system from a user’s perspective, both in terms of what they might be looking for and how they would expect to interact with such a system.

3.2 Accessibility

Regardless of what features are offered by this system, it is vitally important that it be as accessible to as many people as possible. A clear layout and variable font sizes for those with poor eyesight and comprehensively labeled page elements (particularly forms, tables and images) for those using screen-readers will make sure that everyone gets the most out of the Golden Pages service.

3.3 Search options

3.3.1 Searching by area code

Anywhere in Ireland outside of Dublin, area codes are an effective way to divide up an area. For example, the 025 area in North Cork covers two towns, Fermoy and Mitchelstown, and numerous villages and townlands in an approximate 10-mile radius spreading out from Fermoy. The total area is significantly more than just the Fermoy area - meaning we are more likely to find what we need - but not as large as the entire county. Someone in Fermoy, Co. Cork (025) is unlikely to hire a plumber based in Skibbereen, Co. Cork (028), any more than someone in Dingle, Co. Kerry (066) would be willing to drive to a dentist in Limerick (061).

With the 01 area, the divisions are less clear cut. Some 1.5 million people live in the greater Dublin area, and all have the same area code, so something else is needed to split up this group and give its residents access to the same sort of localised searching afforded to their compatriots outside the Pale. The first three digits of a standard seven-digit Dublin telephone number do, in fact, serve as an indication of location, at least for standard residences and smaller businesses. Within each 3-digit sub-area, Eircom can generate 10,000 possible numbers, which should generally be enough to allow for further population growth within any of the sub-areas. Accordingly, the design allows the user to
search within each of these 3-digit domains and/or the adjacent ones. The user should not be expected to know which 3-digit domain corresponds to which area; it should be enough to know their own phone number. If the user specifically wants to look for a business in Clontarf, say, they can search by location.

Nonetheless, there are situations where the search area might need to be greater. If there happens not to be enough of a selection of used car dealers in the 025 area, the search can be expanded to encompass all of the surrounding area codes. This approach has a couple of advantages over searching by town and then by county. Mitchelstown is on the north-eastern border of County Cork so a resident of that town is more likely to be interested in businesses in Cahir, Co. Tipperary or Kilmalloch, Co. Limerick or Lismore Co. Waterford than Castletownbere in West Cork.

The first set of options presented to the user, then, will be either to search by town or street name or to search by area code (officially referred to as a National Dialling Code).

3.3.2 Searching by location

Searching by location is straightforward. A user should be able to type in any part of a postal address and get results back with addresses matching that. We are making an assumption that users will realise that there are multiple O’Connell streets in Ireland and that, while results for such a query would indeed be returned, they are likely to be relatively uninformative. A more common location query would be the name of a town, or a district or suburb of a city, such as Rathkeale (Limerick), Knocknaheeny (Cork) or Ballyfermot (Dublin).

3.3.3 Searching by ad caption

When leafing through the print edition of the Golden Pages, one’s eye is often drawn to the picture ads, for which advertisers pay premium rates, depending on the size of the graphic. A particularly eye-catching image will stay in the memory far longer than the actual name of the business with which it was associated. A customer who decides to deal with a business with a picture ad and finds the service satisfactory will, on returning to the book later on, probably be inclined to head straight for the picture. The more visually striking the picture, the likelier this is to happen.

The online service differs in a number of ways. Firstly, the graphics can now be made far more complex at no extra cost. Anyone should be allowed to scan in a photograph of
their shopfront, their staff or their fleet of delivery vans as a JPEG or a PNG and be asked
to pay no more than an appropriate filestorage fee. The dimensions of an image on the
web are far less relevant than in the print edition, where every column inch counts. The
print edition does allow for basic colours these days, but only for text and simple blocks.
A total of four businesses have full-colour photographs in the 2004 print edition of the
Golden Pages for the 01 area and these necessitate special glossy pages. A well-
compressed 640*480 pixel JPEG photograph weighs in at no more than 45 kilobytes,
which takes about eight seconds to download on a 56 Kb/s internet connection. While
larger companies will undoubtedly still pay professional graphic designers to create their
picture ads, an online service at least gives smaller vendors the opportunity to present a
better image at an affordable price.

An online picture, or at least the hyperlink to it, can also carry some useful metadata.
This metadata can be best used to summarise the picture. Bob’s Courier Service may
have a particularly eye-catching picture ad featuring a white van. The nearest an online
service can offer as far as the visual mnemonic for returning customers described above
is to give the customer the option to search for a “white van”, returning the business or
businesses with ads featuring a white van. Such a feature may even be useful from an
accessibility standpoint by giving a visually impaired user some way of discriminating
between different advertisers’ offerings.
3.4 Filtering and sorting the results

When a user is presented with a large amount of tabular data of varying relevance, it is important that they be allowed to filter and arrange this data as easily and intuitively as possible. To that end, several options are presented.

3.4.1 Categories

The print edition of the Golden Pages divides up its listed businesses by category. The categories are arranged alphabetically, much like the entries in a dictionary, thus allowing the user to quickly flip through and locate all of the businesses related to a particular category of products or services. There are, however, many categories pertaining to computers, for example. Some businesses will specialise only in selling computers, others in computer maintenance and repair and others in computer training. Web design firms, networking specialists and the like also fall into the sphere of “computers”. These each get a category to themselves, so it may not always be clear on opening the print edition exactly which category one should be looking out for. This problem is alleviated somewhat with an index of all of the categories, located towards the end of the book, in which all of the possible subcategories of computer-related businesses are listed side by side.

An online version has two ways of reproducing this. The first way is to provide a directory section, which lists all of the businesses and provides links to simple listings of all of the businesses in that category, with possible further targetted searches within that to narrow down the list of businesses.

A more convenient way is to show the user those categories most closely related to their search query in a separate panel on the results page. Thus, a search for computers in a specific area will show, as well as the appropriate business listing, all of the computer-related categories, such as Computer Software, Computer Maintenance, Computer Training, Web design and so on.

3.4.1.1 Note: The single page principle

Any web service offering some sort of search functionality will invariably have to deal with the prospect of a large number of search results being returned to one page. Perhaps because of bandwidth issues and loading times in the past, most designers consider the most appropriate method of displaying this large amount of data to be to split the results
up over a number of pages, displaying perhaps ten results on every page. The problem with this approach is that very few users are willing to look at more than two or three pages’ worth of results. If, as in the case of the Golden Pages site, the businesses are sorted alphabetically by business name by default, Acme Plumbing Co will appear on the first page by virtue of nothing else other than having a name starting with ‘A’, whilst McSweeney & Son Plumbers and Worthington Plumbers and Fitters are buried several pages further down the list, dramatically reducing their visibility to any potential clients.

Dedicated search engines such as Google have gone to great trouble developing sophisticated algorithms to sort the potentially hundreds of thousands of results generated by a search for, say, “Britney Spears”, so that the user rarely has to look beyond the first page of results. A Golden Pages style service, drawing from a much smaller and more tightly defined database, has less room for manoeuvre. Results will be pulled straight from a relational database table and there is no real practical or fair way to generate and store the kind of dynamic metadata to feed something like the PageRank algorithm.

The idea of user ratings has potential, but “peer review” on the Web does bring with it a number of major issues. So as to prevent either unfair promotion or sabotage of a particular business, either human editors would have to verify every rating as it came in, or users would have to create some sort of account on the service, allowing spurious ratings to be easily traced back. The first option is potentially expensive and the second is unlikely to be taken up by any more than a handful of users; few people would go to the trouble of submitting their personal details to an online service when all they want is to find a telephone number. The general question of reliable peer review was deemed to be outside the scope of this project. It is nonetheless a fascinating and very pertinent issue, and one worthy of further research.

Faced with this problem of not being able to meaningfully sort results, the only fair option (from the point of view of Mr. Worthington, the plumber) is to have all of the results on one page. From a user’s perspective, it means less clicking and, assuming the user were to otherwise load every single page’s worth of ten results, less loading time. The argument that the user would not bother to look through every one of the sets of ten serves only to highlight Mr. Worthington’s disadvantage. In this author’s admittedly subjective view, scrolling down one long page is a good deal less unappealing than wading through a series of individual pages which
A single page with, say, sixty names will load up in far less time than repeating the load-scroll-click cycle six times over, especially once one takes into account the time required to establish a connection to the webserver and then render each page in the browser at the other end. Lastly, a single HTTP request and corresponding database query is far simpler for the server to handle than six individual ones.

### 3.4.2 Sort by column

By default, the system would still the results by business name, but the user should have the option to arrange the results according to the columns of the tabular data, so as to be able to group the rows into meaningful clusters.

The titles of each column will then be links through which the user can sort the data. Arranging the results alphabetically by name should still be the default setting, for the want of any better criteria. It also makes sense to allow the user to group the businesses together by town - the most sensible geographical delimiter, since counties are too big and streets are too small an area to be useful. By the same token, grouping the rows together by area code is also a potentially useful option. If, for example, a user in Fermoy (025) wants to see if it is worthwhile making a trip to Youghal (024) or Macroom (026) to buy a used car, he or she can scan through all of the dealers outside of his or her area much more easily if they are grouped together.

### 3.4.3 Expanding the search

A user cannot be expected to know the likelihood of finding a chauffer-hire firm in the Dingle area, and it could be that there are none. Although there would be an option in the search form to search within a certain area code and its surrounding areas, the probability is that most users would only go for the latter option if the former yields nothing useful. Therefore, it should be made as simple as possible for the user to expand a search within a particular area code or Dublin 3-digit prefix to include the adjacent area codes - after the initial search has been performed.

### 3.4.4 Filtering the search results

If a long list of results is returned and the user wishes to refine the search to only show those from a particular town, they should be able to click on the name of any town in the results table to only see businesses from that town.
In the same way, it should be possible to narrow down a selection of results to one specific area code in the same way as they can be filtered by location. By clicking on the area code of one of the returned businesses, the selection will be filtered to show only the businesses within that area code which match the original query.

3.4.5 Summary

So as to maximise the potential visibility of all of the listed businesses some way equally, it is a good idea to let users shuffle, expand and filter the results as much as they want. This will also increase likelihood of the user to finding exactly what he or she is looking for.
3.5 Richer interaction: direct interfacing with shops and services

3.5.1 Overview

By and large, exactly which businesses we choose to patronise on a daily basis have very little to do with just the name of the business. If, for example, a particular product is cheaper in one shop than in another but otherwise equal in every way, we will go for the cheaper option. Our primary concerns as consumers can be summed up by one or more of the following criteria:

- **Proximity of vendor:** This is what a standard Golden/Yellow Pages directory is used to ascertain. The emphasis of such directories is to locate bricks-and-mortar businesses in close proximity to the user. This is why the six print editions of the Irish Golden Pages are distributed according to the six major telephone areas – 01, 02, 04, 05, 06 and 07/09. A resident of Cork (02) does not automatically receive any edition other than the 02 each year; this reflects the fact that location is still a major deciding factor in deciding which business one should pick up the phone and call.

- **Availability:** This applies to products in the obvious sense that if a given shop does not stock whatever the customer is looking for, the customer will be forced to look further afield. With services such as plumbers or doctors, it is a question of time, as in who is available at 8pm on a Sunday evening to fix your exploded boiler, or which of the doctors’ offices in Fermoy are open at the weekend.

- **Value:** If there are two supermarkets within easy reach, most consumers will clearly opt for that which sells the better value groceries. Assume “value” in this context to be some sort of heuristic based on the quality of the goods or service versus the price. Different consumers clearly have different benchmarks for what constitutes “good value”!

The current online and print versions of the Golden Pages do a perfectly good job of providing details related to the existence of and proximity of thousands of businesses in Ireland. However, they offer little or no information on the products and services offered by those businesses, and no information at all on how one business compares to another in terms of value.
What a direct user-to-product interface should offer is the chance to search for businesses that offer specific products and services according to the user’s needs in relation to proximity, availability and value, as defined above.
3.5.2 Browsing and querying a single shop

The system will offer several ways to see what one or more businesses are offering. To look at the stock of a single shop, the user should be able to click on a “shop” icon of any business that has one, which will open a page listing details of the products and/or services offered by that business. For example, a shop specialising in laptop computers will list the details of each machine in turn, such as make, model, price, processor and RAM.

In the case of a single service, the information presented is likely to be relatively static. There would be details of availability times, additional contact addresses, perhaps hourly rates. When browsing a long list of products offered by a retailer, however, it would be useful to perform searches within the list to narrow down the options. By entering search terms in a web form, the user could choose to display a specific type of product within the inventory, or even narrow the search further to show only those products less than or equal to a specified price. So, for example, he/she should be able to find all of the Compaq laptops offered by the vendor costing less than €2000, or all of the used Ford Fiestas with a 1999 registration or later costing no more than €8,500.

3.5.3 Browsing and querying multiple shops

Searching through one shop at a time can still be a tedious process. Far better would be the ability to look at all of the competing shops in one area at a time. This would, in a sense, “take out the middle man” and link the user directly to the products. The impression for the user would be comparable to looking at any Web-based trading site like Amazon or eBay, except that the product selection is made across the set of stock offered by local retailers.

3.5.4 Smart forms

One feature which materialised rather late into the implementation stage, but very much merited inclusion, was “smart forms” and it came from implementing the “filtering by category” idea above.

If a user performs a search of the main business database and a list of related categories is returned, we can take a good guess as to what it is they are looking for. This is an opportunity to further streamline the shop-browsing process. It should be possible to include a secondary form on the results page through which the user can search the shops
of all of the displayed businesses, thus dramatically cutting down on clicking and scrolling. The trick is knowing approximately what the user is looking for and, based on that, displaying a general form appropriate to the business type. The list of related categories sorted in order of number of occurrences are a good way to select which “smart form” to display. Almost every business category would have a certain type of basic form corresponding to it. In the case of a computer retailer, every product is guaranteed to at least be defineable by its description, make, model and price. The same goes for a used car dealer, a shoe shop, a bicycle shop etc. This generic product form can be used on most retail categories. However, should a particular category of business require a more specialised form, it would be a simple matter to design a new form for it. Examples include bookshops, estate agents and pizza delivery firms. A bookshop form would most likely have fields for Author, Title, Publisher and Year; an estate agent might have Property type (a select list with values like “plot”, “detached”, “semi”, “bungalow”, “office” etc), Location and Price. Pizza forms can have Base, Topping 1, Topping 2 and Topping 3, and a search would return every pizza offered by every pizza company within the specified area which has a deep-pan base and contains pepperoni, anchovies and olives! With call-out services such as doctors, electricians and plumbers, the emphasis is more on availability, so a typical catch-all form for them would revolve around times.

In all of the above cases, it would still be possible to carry out more refined queries on the subsequent page, where a fuller range of possible product attributes would be presented

3.6 Summary

This system is designed to offer a rich source of information to the user about products and services available in their area. With just a few mouse clicks, users are connected to businesses not just by arbitrarily picking a name from a list, but because the user knows that that business offers the exact product or service that he or she needs, at the right price and at the right time. Browsing the Golden Pages need no longer be a hit-and-miss affair.
4. IMPLEMENTATION

4.1 Introduction

This section describes the implementation of the whole system and is divided into four parts. The MySQL database section describes the structure of the database and explains how it was populated with 20,000 entries for testing. The sections on queries.php and multishop.php contain detailed run-throughs of the code and describe some of the more interesting features and issues that arose along the way.

4.2 Back end implementation

4.2.1 The MySQL database

The MySQL database consists of three tables: clients, category and areacode.

clients is the table containing the details of each advertiser. Its columns are:

- bizid – a unique identifier number of the business
- category – category id of the business; foreign key of cid in the category table
- bizname – the title of the business
- street1, street2, town
- countycode – either the name of a county or a Dublin post code
- areacode – either the national dialling code prefix without leading zero or “1-“ followed by a 3-digit Dublin phone prefix
- phone
- web, email
- picfile, pictext – file location of a picture ad, if there is one, plus some text describing said ad
- shop – URL of an XML database associated with this business, if one exists.
For testing purposes, a standalone script called `fclient.php` was used to populate this table with 20,000 random businesses in random locations around Ireland. The business names were plucked from a file of Irish surnames found on the internet. It was a long file, so I selected only the “K” surnames. A real implementation can of course handle business names beginning with any alphanumeric character. Each surname was prepended to one of a selection of random business types, which is consistent with its corresponding category id. The street names are also fake, but the towns, counties and area codes on every row are consistent with each other. The telephone numbers are completely random, except for the few areas of Dublin that were manually included, which have accurate 3-digit prefixes (I think) followed by randomly generated 4-digit suffixes. The web addresses are of the form `www.biznamebizcategory.ie` and the emails are all `info@biznamebizcategory.ie`. Approximately one business in ten has a picture ad and its filename is generated as bizid_bizname.png. Approximately one business in four has an XML shop and its generated filename is bizid_bizname.xml.

The category table has three columns:

- cid (primary key)– unique category identifier, used for category-specific searches
- category – text description of the category
- shopform – associated “smart form” used to query XML database(s) if this category is most closely related to user’s query

![Table showing area codes: 54 Records (22 retrieved)](image)

Again, for testing purposes, this table was given 65 rows, for 65 different business categories.

Lastly, areacode has two columns

- code (primary key)– an area code (national dialling code) without leading zero or a “1-“ followed by a three digit Dublin area prefix.
- adjacent – any area code which is adjacent to the code in the primary key

*Fig. 1: Detail of the areacode adjacency table*
Not all of the area codes were included, partly because this is only a test database, but also because it is impossible to find a map of all of the local area codes in Ireland. Whether this is because it is considered to be proprietary information or simply no such map exists is impossible to say.
The core of the system is written in PHP and is centred around two main scripts: queries.php, which runs and returns queries on the main database and multishop.php, which executes search queries across single or multiple XML databases. The following is a description of how these scripts work, from top to bottom.

4.2.2 queries.php

The queries script handles forms submitted by users, translates them into MySQL queries which it then runs on the MySQL database, before returning the results in HTML to the Apache server, which sends them out to the user’s browser window. Many of the larger chunks of code are split into separate files which are then included and evaluated inline by PHP, with a command of the form: include(“filename.inc”). The first step taken by the script is to check whether it has been requested with or without a HTTP query string – the web equivalent of parameters to a function, not to be confused with an SQL query. If there is no query string, just the search form and a welcome message will be returned, formatted in HTML, prompting the user to enter some search criteria in the form contained in form.inc. form.inc has a little PHP code to refill the elements of the form with the contents of whatever query has just been made. Once a query has been made, the HTML form sends its contents back around into the queries.php page in the form of a HTTP query string.

A typical HTTP request would look like this:

http://localhost/queries.php?what=computer&searchby=location&where=malahide&your area=01&radius=thisarea&sortby=default

This is how the request breaks down:

- **http://** - the Hypertext Transfer Protocol, as distinct from ftp:// (File Transfer Protocol), telnet:// (remote Unix login), imap:// (Internet Mail Access Protocol) etc.

- **localhost/** - the server name; could equally be www.advancedgoldenpages.com or 134.226.83.50.

- **queries.php** – the requested file, in this case a script that the Apache webserver will recognise and execute through its PHP module.

- **?** – denotes the start of the HTTP query string, which contains the following variables and their values, separated by ampersands
• what=computer
• &searchby=location
• &where=malahide
• &yourarea=01
• &radius=thisarea

Fig. 2: Detail of a typical query string and the corresponding form values

Each of the variables correspond to an element of the form above, and the two most logically important ones are $what and $searchby.

• $what must be set: its value is what will be compared against a list of categories and a list of business names. Any businesses whose names or categories, which contain, in this case, the string “computer”, will be included in the results table.

• $searchby is derived from the pair of radio buttons in the form; it will always be set to either “location” or “areacode” and this is what decides how the SQL query is to be generated. A number of functions are defined in dbq.inc: makareaquery(), makecatareaquery(), makewherequery() and makecatwherequery().

• If $searchby == “location”, we are then only concerned with the variables $what, $where and $sortby and we call makewherequery().

• If $searchby == “areacode”, we are only concerned with the variables $what, $yourarea, $radius and $sortby and we call makeareaquery().

• Finally, if the $cat variable is set (this will only happen after a user has clicked on a category in the “Related categories” panel), there exist two further functions, makewherequery() and makecatwherequery(), to additionally filter out only those businesses with the matching category id in the clients table of the database.
In the case of computers in Malahide, we will end up calling `makewherequery()`, which is defined in `dbq.inc`. This function takes three parameters, `$what`, `$where` and `$sortby`.

```php
function makewherequery($what, $where, $sortby) {
    if (!strlen($sortby) || $sortby != "shop") {
        if ($sortby == "picfile") {
            $upordown = "desc";
        } else {
            $upordown = "asc";
        }
    }
    $str = "order by ". $sortby . $upordown;
}
```

Fig. 3: The `makewherequery()` function, one of four which generate different SQL queries

- `$sortby` may or may not be set in the original HTTP request. If it is not explicitly set then, it will be assigned the value “default”.

- If `$sortby` is set to either “shop” or “picfile” (the field in the clients table containing the location of a picture ad), we would prefer MySQL to sort the results in descending order, so that the businesses with a shop/picture ad appear at the top of the list. Otherwise, the results should come back in normal ascending alphanumerical order.

- The resultant MySQL query (in English) then goes as follows: “Select all rows from the clients table where either ‘bizname’ or ‘pictext’ contain the string “computer” and any of the address fields contain the value “malahide”. Sort these results in ascending order”.

A more complicated scenario as far as constructing these SQL queries arises if the user has decided to search within a particular area code and the adjacent areas. In this case, `$searchby` is set to “areacode”, $yourarea is set to some area code and $radius is set to “adjarea” (the other option is “thisarea”). For this example, let us take $yourarea to be “021”, and we are searching for a “plumber”.

Thus, `makeareaquery()` is called. The first part of its output, dealing with `$what` is the same, but in order to find all of the relevant businesses in the adjacent areacodes too, it is first necessary to query the `areacode` table in the `getadj()` function.
- `getadj()` executes the following MySQL query:
  ```sql
  select adjacent from areacode where code = '21'
  ```

- It then formats the results, returning a string which looks like:
  ```sql
  ((areacode = '21') or (areacode = '22') or (areacode = '23') or (areacode = '24') or (areacode = '25') or (areacode = '26') or (areacode = '27') or (areacode = '28'))
  ```

- This string is inserted as a substring into the main SQL query-to-be, which will ultimately look like this:
  ```sql
  select *
  from clients, category
  where category.cid = clients.category
  and ((areacode = '21') or (areacode = '22') or (areacode = '23') or (areacode = '24') or (areacode = '25') or (areacode = '26') or (areacode = '27') or (areacode = '28'))
  and ((bizname like "plumber") or (pictext like "plumber")).
  ```

Just after the appropriate one of the four SQL query generating functions is called, a string variable called $foundstring is put together. This is the line that later appears as the caption of the results table saying, for example, “Searched for computer in the Malahide area, sorted by shop” or “Searched for plumber in the 021 area and surrounding areas, sorted by default”.

The next major step is to run the query and return the results in a table. If you, the reader are in a position to execute the source code on an appropriate webservice, it might be worthwhile to uncomment the line `echo $sql;` - this prints out the exact SQL query above the results table and was used during development for debugging purposes.
Fig. 4: The end result - all of the computer-related businesses in Malahide

\textbf{Shandle1=\texttt{mysql\_query(Sgp);}} queries the database and assigns the results in a two-dimensional associative array to \texttt{Shandle1}. As this pulls every column from both the \texttt{clients} and the \texttt{categories} tables, the following columns (listed here in no particular order) are returned: \texttt{bizid, category.cid} (or \texttt{clients.category} – same value), \texttt{bizname, street1, street2, town, countycode, areacode, phone, web, email, picfile} and \texttt{pictext}.

In summary, a while-loop processes each line in turn.

- As can be seen in any screenshot of the results table, \texttt{name, street1, street2, town, countycode, areacode} and \texttt{phone} are directly visible on the screen.

- The values for \texttt{web} and \texttt{email} are embedded in hyperlinks around respective “www” and “envelope” icons.

- If a value is set for \texttt{shop}, a hyperlink to “shop.php?xml=shop” is inserted around a “+” icon and a checkbox is placed beside it with its value set to the URL of the shop’s XML database. Each checkbox is part of a form which submission button, marked
“Browse checked shops” is at the bottom of the table. Checking one or more of the checkboxes and submitting this form will open `multishop.php`, displaying the inventories/details of the selected shops.

- If a picture ad exists, a hyperlink is inserted around an “Ad!” icon. The image/link code takes the following form (pseudocode): `<a href=picfile title="Picture ad for bizname: pictext"><img src="pic_icon.gif"></a>.

- The function `wg()` (as in “white or grey”), alternates the background colour of each row of table cells between white and grey, to make the table easier to read.

To browse through or perform a search across multiple shops, the user is given two options. Firstly, they can select an arbitrary number of businesses listed as having shops, and then click the “Browse selected shops” button at the bottom of the page. Alternatively (and what is probably the more convenient route), they can enter some basic search criteria into the “smart form” which appears over the search results, which immediately executes a query across all of the shops listed on the results page. This feature requires some additional database queries.

While the table is being traversed, two external arrays are also incrementally filled: $categorymenuarray[] and $shopnames[].

- $categorymenuarray[], contains the categories of each business as it comes up. When the loop terminates, this array is processed by a series of functions in `catdbq.inc`.

- The first two lines of code in `catdbq.inc` transform the data in $categorymenuarray into an associative array called $sortedcats of the form [category name] -> “# of occurrences”, sorted in descending order. This is basically what we see in the “Related categories” panel.

- The `pickaform()` function selects from the `categories` table the `shopform` value of the first element (i.e., the category with the most occurrences) of the $sortedcats array. The `shopform` value is the location of a file in the forms/ subdirectory which contains the code for the appropriate “smart form” to appear above the results table.

- Each category name is in a hyperlink which reposts the original HTTP query but with the added restriction that only the businesses in that specific category should be shown.
- $shopnames[] is a list of all of the shop URLs as they appear in the main while loop. This list forms part of the HTTP query sent by the selected “smart form” to **multishop.php**.

- Take forms/**genericproductform.inc** as an example: this form will contain four visible input fields, namely Product, Make, Model and Price. In addition, PHP generates a number of hidden input values, each of which are of the form: `<input type=”hidden” name=”randomshopname.xml” value=”on”>`

In the next section, we will see how shop.php and multishop.php parses the HTTP query string generated either by the “smart form” or by a selection of one or more shops in the results table.
4.2.2.1 Note: A question of flexibility

The main issue, which informed the design of `multishop.php`, is that almost anything can be encoded in the HTTP query string passed into it. Unlike a typical HTTP-form-into-PHP scenario, such as in the case of `queries.php` above, the search parameters cannot be predefined.

This is because we need to be able to describe and search for an infinite number of different products and services, which can be described in an infinite number of different ways. For example, a computer shop’s inventory and description thereof differs radically from an art gallery’s, or a butcher’s or a bookshop’s. A typical item in a computer shop might be:


A pet shop, whereas, describes its mice in a completely different way:


Creating set categories for every type of product and service on sale in the country would be completely impractical, so it was decided instead to make the system as flexible as possible. Critically, this rules out the use of DTDs, something which would seem to negate a basic rule of XML, namely that all documents be strictly structured. As explained above, however, it would be nigh impossible to define and formalise all of the tags and permitted attribute values for every possible item that the system is likely to come across.
4.2.3 multishop.php

multishop.php makes use of PHP’s event-based XML parser. While an XML file could have been manually parsed, there was nothing to be gained in this instance by reinventing the wheel. The parser takes much of the work out of traversing an XML file and extracting values from it, and it also makes it easier to adapt to different XML document hierarchies. Its key feature is flexibility, which it needs to deal with a HTTP query string such as:

- DESC=
- &MAKE=compaq
- &MODEL=
- &PRICE=2500
- &xml%2F1722_KIELTY.xml=on
- &xml%2F1437_Killian.xml=on
- &xml%2F13755_Killory.xml=on

From the start, one can see that the string is separable into two logical sections. The first deals with specific search parameters. This example string having come from the “smart form” on queries.php, it would seem that the user wishes to browse through all of the shops in a certain area which sell Compaq computers for not more than €2,500. In this case, there are three shops which lie in the appropriate area. Assuming that queries.php has correctly responded to a search for “computers” or something similar, these should all be computer shops.
Fig. 5: Detail of the “smart” computer form and the three businesses with shops that it will be querying

The first thing to do is to split the string into its two parts, namely the search criteria and the shops we will be searching through. The first function defined in `multishop.php` takes in the raw query string and splits it up into two arrays. The criteria array ($merits[]) is a plain two-dimensional array or, in simpler terms, an array of pairs.

Array (  

    [0] => Array (  
    [0] => MAKE  
    [1] => compaq  
    )  

    [1] => Array (  
    [0] => PRICE  
    [1] => 2500  
    )  

)

The second array ($shops[]), is an array of the locations of each of the XML files.
The next stage is to build an array of all of the items across the three shops, so that they can be compared to the search criteria. The function `multishop()` takes the `$shops[]` array as a parameter and runs each shop in turn through the parser, whose functions are all defined in `parse2.inc`.

Although DTDs are not feasible in such a system, some level of uniformity is nonetheless required. In this case, absolutely everything that is being sold or offered is to be categorised either as an “ITEM” or a “SERVICE”.

This is the expected layout of one of the XML databases. As will be discussed later, this layout also has a number of limitations.

```xml
<?xml version="1.0" ?>
<shop bizname="Kielty Computers">
  <item desc="laptop" make="Toshiba" model="Satellite" price="1699" />
  <item desc="laptop" make="HP" model="Omnibook" price="2100" colour="blue" />
  <item desc="laptop" make="Compaq" model="Armada" price="2349" colour="red" />
  <item desc="laptop" make="Dell" model="Latitude" price="1995" />
  <item desc="laptop" make="Toshiba" model="Tecra" price="1699" />
  <item desc="laptop" make="HP" model="Pavilion" price="2480" colour="blue" />
  <item desc="laptop" make="Compaq" model="Evo" price="2149" />
  <item desc="laptop" make="Apple" model="iBook" colour="white" price="2995" />
</shop>
```

The database starts and finishes with `<shop></shop>` tags and, in this case, is populated only with items. What is important to note here is that the attributes are completely arbitrary. For example, some of the items have a colour defined for them and some do not. The “desc” attribute is an abbreviation for “description”, but there is nothing to stop the database owner from deciding to use the full spelling of the word instead. As long as
there are <shop> and <item> and/or <service> tags present, the database can be read, displayed and searched through.

The `parse()` function in `parse2.inc` invokes a series of in-built XML-related functions in PHP. The coder creates two functions to define exactly what should happen when the parser meets either an opening XML tag or a closing one. The function for an opening tag, in this case called `startElement()`, must always take the handle of the XML parser ($parser), the name of the tag ($name) and the associative array of attributes ($attrs[])) as parameters. The corresponding closing tag function (`endElement()`) takes only the parser and the tag name as attributes. For the purposes of this system, `endElement()` did absolutely nothing at all, our only concern being with tag names and attributes.

When `startElement()` comes across a tag called either SERVICE, ITEM or SHOP, it pushes the attributes of that tag onto a global array called $shopsarray[]. It is the parser engine, incidentally, that automatically capitalises the attribute names – it proves to be convenient later on when displaying the names in a search panel on the left. Once each shop has been run through, the $shopsarray[] of our example shops will look something like this:

Array(
    [0] => Array(
        [BIZNAME] => Killian Computer Software Engineers
    )
    [1] => Array(
        [DESC] => laptop
        [MAKE] => Toshiba
        [MODEL] => Satellite
        [PRICE] => 1599
    )
    [2] => Array(
        [DESC] => laptop
        [MAKE] => HP
        [MODEL] => Omnibook
        [PRICE] => 2105
        [COLOUR] => blue
    )
)
etc etc....

[9] => Array(
    [BIZNAME] => Kielty Computers
)

[10] => Array(
    [DESC] => laptop
    [MAKE] => Toshiba
    [MODEL] => Satellite
    [PRICE] => 1699
)

Now that we have an array of items and their attributes from all of the requested shops ($shoparray[]), and a list of criteria to compare those attributes against ($mcrits[]), we call the `grep()` function, which takes these two arrays as parameters. `grep()` invokes `test_line()` on each of the items in of $shoparray[] (each item in turn being an associative array). If an item passes `test_line()`, the function `spit()` prints it to the screen. `grep()` also keeps track of the title of the current business, without which there would be a long list of products with no indication as to which shop they come from.

`test_line()` tests each of the key/value pairs of $mcrits[] (query criteria) against each of the key/value pairs in the item array (referred to as `line[]` within the function). A `$yescount` variable is incremented each time one of the query criteria matches one of the item attributes and the function only returns true if, at the end, `$yescount` is equal to the number of search criteria (and thus, by definition, the item has matched all of the criteria).
//test_line() tests the key/value pairs of $crits[] (query criteria) against each of the
key/value pairs in the $line[] array

function test_line($line,$crits){
    $yescount = 0; //incremented for each successful match; must reach the size of the
    $crits[] array for the function to return true
    for($c=0;($c)<(count($crits)-1);$c++){
        $targetkey = strtoupper($crits[$c][0]);
        $targetvalue = $crits[$c][1];
        if (array_key_exists($targetkey,$line)){
            // echo "key exists checking $line[$targetkey] and $targetvalue\n";
            if (strstr($targetvalue,"timefrom")) {
                // is this a timeframe value?
                if ($line[$targetkey] <= substr($targetvalue,0,-8)) {
                    // is that timeframe value less than or equal to what we want?
                    $yescount++;
                }else if (strstr($targetvalue,"timeeto")) {
                    // is this a timeframe value?
                    if ($line[$targetkey] >= substr($targetvalue,0,-6)) {
                        // do they close on or after the specified deadline?
                        $yescount++;
                    }else if (is_numeric($targetvalue)) {
                        // is it just a number (assuming a price for now)
                        if ($line[$targetkey] <= $targetvalue) {
                            $yescount++;
                        }else{
                            // it's a generic string - just match it
                            if (strstr($line[$targetkey],$targetvalue)) {
                                $yescount++;
                            }
                        }
                    }
                }
            }
        }
    }
    // echo "yescount is $yescount, crit# is ",count($crits),"\n";
    return ($yescount==count($crits));
}

Fig. 6: The test_line() function

test_line() was probably the most difficult piece of code to get to work, relying as it did
on so many steps previous steps to have executed correctly for it to do likewise. It also
exposes what I acknowledge to be a flaw in the initial design of the XML files. The
problem is one of functionality: if a user enters a price of €2,500, it is highly unlikely that
he or she will find very many computers for exactly that price. It is more likely that the
intended meaning is “less than or equal to €2,500”. Similarly, when someone requires a
plumber who is available from 8am on a weekday morning, it should be obvious that all
of the plumbers who start work before 8am are equally available. This issue came to a
head when trying to define how one would find a plumber at, say, 10pm, since there is no
truly efficient way to define a timespan in the current XML setup. test_line() currently
has a series of fixes to make a guess at what sort of test should be performed on a
particular attribute value, as in whether it should be greater-than-or-equal-to, less-than-or-equal-to or exactly equal to the current search criterion.

What is missing from the current XML design is metadata, in effect data types, that the program could use to distinguish between “2100” meaning €2100, 9pm or the part number of some computer accessory. For example:

```xml
<Item>
  <make>Dell</make>
  <model>Inspiron</model>
  <price type="price">2100</price>
</Item>

...or...

<Service>
  <name>Bob’s Plumbers</name>
  <telephone>01 2345678</telephone>
  <weekdays type="timespan">0600-1800</weekdays>
  <saturday type="timespan">1000-1600</saturday>
</Service>

...would be a much more sensible way of encoding the data. Time simply ran out before a parser for this XML structure could be fully implemented. Tweaking the startElement(), grep() and test_line() functions in parse2.inc would be straightforward; all of the intellectual effort required to conceptualise the algorithm first time around is second-nature at this stage. However, rather than running the risk of ending up with half-finished code, I have opted to submit the older, functional version - albeit less truly functional than the version just described.

The last significant piece of code in multishop.php is included from lm.inc, which prints the panel on the left of the page and generates the simple search form. The form generating code is in a further nested include file called genmultiform.inc. As in any of the “smart form” includes such as genericproductform.inc which are deployed on queries.php, the first piece of PHP code generates a series of hidden form inputs, one for each shop name, in this case from the array $shops[]. Next, a $fields array is declared and populated by scanning through all of the possible attribute names in $shopsarray[].
Lastly, those criteria for which search values have already been entered need to be displayed in the input fields on the page. Again, the references to the “timeto” and “timefrom” strings in the final foreach statement are a workaround for the abovementioned problem with defining timespans. A better XML document design would/will remove the need for this.

![Fig. 7: The end result of the “smart” form query across three shops for a Compaq laptop costing less than €2,500.](Image)

**4.2.3.1 Note: Autonomous XML sources**

Keeping track of every shop’s inventory is beyond the Golden Page’s remit for a number of reasons. Firstly, there is a question of liability: if a shop’s inventory claims to have a certain product at a certain price when in fact the price is much higher, thereby prompting a user complaint, it is hardly fair to blame the middle man. Secondly, a centrally stored inventory database potentially means much less flexibility for business owners who might wish to constantly update stock figures via their own database every time a
purchase is logged in-store. Thirdly, the cost of maintaining the hardware and software for such a huge amount of data would have to be passed down to the proprietors, something which would likely discourage smaller businesses from taking advantage of the service.

All things considered, it makes more sense to have each advertiser sort out their own hosting solution and make available a single file which can be requested from (and perhaps still cached in) the central web server.

4.3 Front end implementation

4.3.1 The “GUIs”

In the world of web development, the GUI of course refers to HTML. What makes describing the HTML somewhat complicated when dealing with PHP is that it is often difficult to tell where the PHP ends and the HTML begins. By definition, every line of PHP that begins with the command “echo” or “print” is an indicator that something will be output to the HTML document and hence to the user’s screen at that point.

When designing the page, it was important to keep in mind at all times that, especially for a web service such as the Golden Pages, the page is very much a tool. As such, its layout should be as intuitive as possible, or at least give enough information back to the user without drowning him or her in a sea of links and forms and buttons. While the interface is slightly more complex than the current Golden Pages site, it is certainly no more complex than the “inbox” page of an internet mail website such as Hotmail.

It is also important to make the page some way visually appealing. While queries.php has a only a minimal amount of images (five unique image files, in all) to help loading time, simple things such as alternating background colours for different panels serve to draw the user’s attention to the functionality in different parts of the page. All of the block elements (denoted by the <div> tag in HTML) which contain the main search form, the “smart form”, the “Related categories” panel and an invisible placeholder for the results table are “absolutely” positioned. This means that their position on the page, height and width are CSS values, which can be precisely specified to the nearest pixel. As well as making the page neater, they also serve as an effective replacement for tables used for layout (see the section on client-side technologies above).
4.3.2 Accessibility

Care has been taken to make the interface as accessible to all users. All of the images and links carry appropriately informative “alt” information. The only table used on the site is used for representing the results of the query on the main database and it is properly structured, with a caption and table headers indicating column titles.

The layout was tested for accessibility with the Bobby validator. It does not completely validate (i.e. qualify for “AAA” status) according to the W3C Web Content Accessibility Guidelines. The validator’s main complaints were related to “using colours to convey important information” – this refers to the alternating row colours in the queries.php results table. In response to this, an extra column was added to the left of the table to spell out, as explicitly as possible, which row is which out of how many in total.

In addition, the site was tested with the JAWS screen reader from Freedom Scientific. It is difficult to evaluate the success of the test simply because I am not used to navigating web pages guided only by a synthesised voice, but the synthesiser seemed to handle the site and the tabular data satisfactorily.

4.4 Summary

Implementation of the design specifications was, by and large, successful. Two completely separate scripts were developed, one in PHP/MySQL and the other in PHP/XML. The system as a whole functions well together and serves at least as a basis for a real-world application.
5. CONCLUSION

5.1 Overview

The two basic aims of the project were successfully identified and implemented. An online geographically oriented business directory system has been designed and implemented, which offers considerably enhanced functionality over a traditional directory service. The primary business look-up part of the system was tested thoroughly on a MySQL database with 20,000 generated test entries. The shop part was tested on a selection of XML database files describing both products and services and worked well, although, as explained earlier, the XML structure opted for on day one may have been too simplified. The smart form system has been tested for four different smart form types, and there is no reason to believe that this system will not work for more detailed smart form types. The user interface is intuitive, quick to load and, most importantly, accessible to all users of the web, having been tested on the JAWS 5 screen reader.

Overall, the system has a robust, modular design, making full use of the power and flexibility of PHP combined with MySQL and XML.

5.2 Future work

A potentially interesting academic avenue of research arising from this project could be, as mentioned earlier, the question of how to design a reliable user-ratings system that is some way resistant to abuse. Also worth a project unto itself would be designing a system for caching remote XML data sources on a central server, updating each either on a regular basis or when the remote file is changed.

The possibilities presented by utilising remote XML files as easily maintainable data sources are endless. Features such as online ordering, customisable shop layouts and even graphics could be implemented at the shop end while still preserving the system’s overall flexibility. Such features could be offered at minimal cost, thus giving even the smallest businesses, which may otherwise have no web presence, the chance to compete for customers on the web.
# 6. References

## 6.1 Web references

(all links were last checked and verified to be live on 2004-04-29)

<table>
<thead>
<tr>
<th>Netcraft</th>
<th>Web</th>
<th>Server</th>
<th>Survey</th>
<th>Ltd.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netcraft</td>
<td>Web</td>
<td>Server</td>
<td>Survey</td>
<td>Archive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>About</th>
<th>MySQL</th>
<th>AB</th>
<th>AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td></td>
<td><a href="http://www.mysql.com/company/">http://www.mysql.com/company/</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UDDI.org</th>
<th>FAQs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://uddi.org/faqs.html">http://uddi.org/faqs.html</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.w3.org/TR/WAI-WEBCONTENT/">http://www.w3.org/TR/WAI-WEBCONTENT/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Golden</th>
<th>Pages</th>
<th>website</th>
<th>(2004)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Golden</td>
<td>Pages</td>
<td></td>
<td>Ltd.</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.goldenpages.ie">http://www.goldenpages.ie</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## 6.2 Print references

<table>
<thead>
<tr>
<th>Golden</th>
<th>Pages</th>
<th>Directory</th>
<th>01</th>
<th>Area</th>
<th>2004</th>
</tr>
</thead>
<tbody>
<tr>
<td>Golden Pages Ltd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>