SWOOP: An application generator for ORACLE/WWW systems.

A. Hunter*
R.I. Ferguson
S. Hedges

Abstract:

The development of a software package (named Swoop) is described. Swoop is designed to support the generation and maintenance of WWW information systems which store information in ORACLE databases: a so-called *hyperbase* program. The biggest problem with hyperbases is that they require a sophisticated program to interpret Forms, query appropriate databases, and merge information into Hypertext. There is a clear need for application-generator tools which allow hyperbase programs to be constructed with minimal expertise on the part of the designer. It is this problem which Swoop addresses.

Keywords:

ORACLE, Interface, Hyperbase.

Introduction

Swoop is a software package designed to support the generation and maintenance of WWW information systems which store information in an ORACLE database. In these hybrid systems (which we refer to as *hyperbases*), information from a relational database is merged into hypertext documents for presentation.

The World Wide Web has facilities which make the provision of hyperbases possible. The basic capability of WWW is to download text files via hypertext links. Dynamic documents [1] are programs which can be invoked in place of a document download, and generate the text as output; a suitable program can hence extract information from a database and present it as HTML. The Forms interface, available using CGI [2] , allows Web-pages to be built which include user-interface elements such as fields, buttons and check-boxes: this can be used to provide user-input to dynamic-document pages.

Hyperbase programs may be presented to the user in two ways. First, the user may browse what appear to be normal web pages, with no Forms interface, although the pages are actually being dynamically constructed from the database. Second, the user may be presented with Forms which generate input: for example, to specify key-words to be used in searches. Forms could also be used to present output, although this is rarely done: most hyperbase programs present the results of searches as simple hypertext.

The biggest problem with hyperbases is that they require a sophisticated program to interpret Forms, query appropriate databases, and merge information into Hypertext. There is a clear need for application-generator tools which allow hyperbase programs to be constructed with minimal expertise on the part of the designer.

What is Swoop?

A number of tools have been developed which go some way towards merging WWW and ORACLE [4]. WOW [5] allows hyperbase programs to be written in PL/SQL and stored in an ORACLE database; it requires appropriate ORACLE programming skills. WOW programs may output a mixture of raw HTML and database information. DECOUX [6] supports an augmented form of HTML which includes some embedded SQL statements. This is interpreted by a forms-based interface, and is somewhat low-level in syntax. Nevertheless, it requires less programming skill than WOW and provides a simple method of describing a hyperbase page. WORA [7] dynamically constructs HTML forms as an interface to ORACLE tables; it provides sophisticated query facilities, but the information cannot be merged into a hypertext presentation.

Swoop supports the construction of hyperbases using an ORACLE database as a back-end. Information is presented to the user as simple hypertext pages; these pages are specified using a simple augmented HTML syntax. It also has facilities to aid in maintenance and specification of the database. Swoop provides a single, integrated solution to the generation of hyperbase systems.

Swoop has the following major components:

Swoopgen. This is an application generator which produces dynamic page programs from special augmented HTML files called *swoop-files*. Swoop files can include embedded pieces of SQL, the standard language for accessing Relational Databases [3]. The SQL inserts in swoop-files corresponds to points where information extracted from the database should be merged into the page. A tool called *swoopgen* translates these special swoop-files into PRO*C programs (PRO*C is ORACLE's pre-compiler to support embedded SQL statements within the C programming language), which when compiled act as dynamic page programs which will extract information from the database and present it merged into hypertext, as specified by the swoopfile. Thus, the system-designer using swoop needs only a rudimentary knowledge of ORACLE and HTML.

swoopform. Whereas *swoopgen* tool provides the facility needed to support browsing of the merged HTML/ORACLE pages, *swoopform* supports database maintenance. It provides a Forms-based interface which allows information to be Queried, Added, Updated and Deleted from tables in the Swoop database. It is provided primarily for the use of the system-maintainer, rather than system users, although in practice it has also been found useful for supporting limited user-input. *swoopform* can automatically produce a form for any Swoop table.

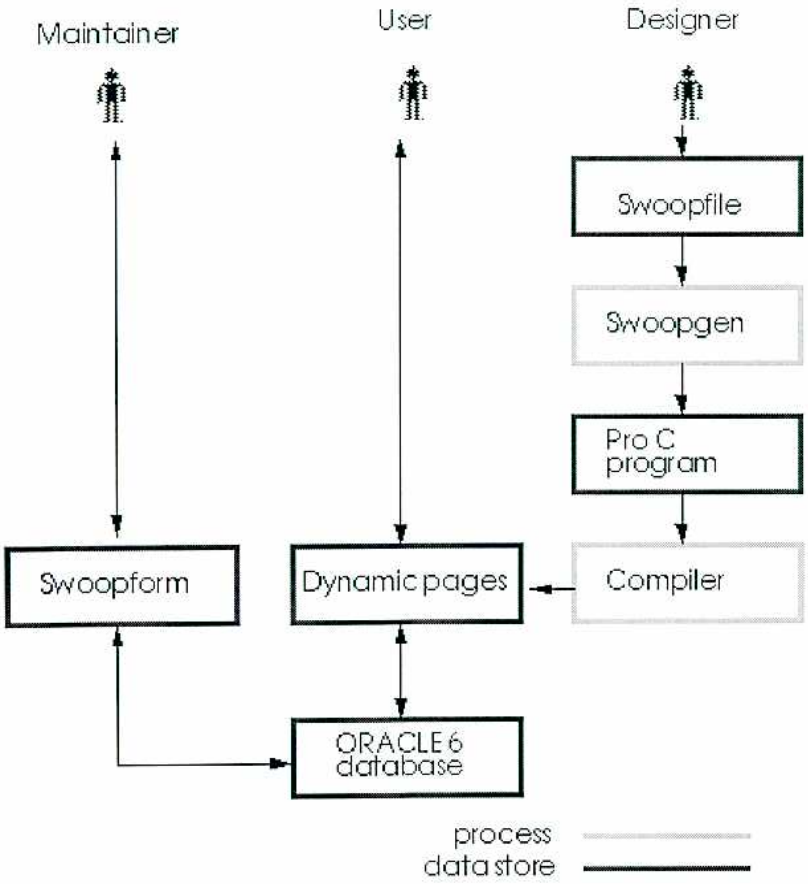

Figure 1: Major Swoop system components

Using swoopgen to build dynamic pages

One of the major components of Swoop is *swoopgen*: an application-generation program. Swoopgen takes as input swoop-files, which contain HTML and embedded *swoop statements*. Swoop statements may include pieces of SQL. Swoopgen produces the source code for dynamic page programs, in PRO*C. This source is then compiled to produce the executable dynamic page programs. When invoked from the Web, these produce HTML on standard output, which is displayed by the user's web browser. This HTML has inserted in it information extracted from the ORACLE database. The information inserted is determined by the embedded swoop statements.

This section discusses the format of swoop-files, and how they can be used to produce interlinked sets of Web pages.

To illustrate the discussion, a simple case study will be used: a system to track members of staff at an academic institution. It includes:

1. A home page for each member of staff;
2. A telephone list of all staff in the institution, with links to home pages
3. A list of departments;
4. A home page for each department, including a list of staff in the department with links to their home pages.

The database for this system is defined in Figure 2.

Table: **staff**

```
code      name     dept     tel         additional
GK        Khan     CIS      0225        Not to be argued with.
AH        Hunter   CIS      2778        Likes writing <i>software</i>!
SG        Garrick  ENG      3425        Hands-on worker.
CP        Porter   ART      2345
```

Table: **dept**

```
code     name         head
CIS      Computing    GK
ENG      Engineering  ATH
```

Figure 2: Departmental database

Note that the database has been left deliberately incomplete: the head of department for Engineering (ATH) hasn't been listed in the staff table, and the ART department has been ommitted from the departments table; CP has provided no additional information for his Home page; Swoop will support this incomplete information to the best of its ability.

Stage 1: producing a simple swoop-file

The first page required is a telephone list for all staff. The HTML for a simple version of this with no hyper-text links is presented in Figure 3.

```
<title>Telephone Listing</title>
<h1>Telephone Listing</title><hr>
Hunter, 2778<p>
Garrick, 3425<p>
Khan, 0225<p>
Porter, 2345<p>
```

Figure 3: HTML required to present staff telephone listing.

The above HTML can be produced by a dynamic page program called *tellist*, which has been automatically generated by Swoop from the swoop-file illustrated in Figure 4.

```
$sql staff order by name$
<h1>Telephone Listing</title><hr>
$repeat$
$name:0$, $telno:0$<p>
```

```
$endrepeat$
```

Figure 4: Swoop-file source for *tellist*

Swoop-files contains HTML, augmented by swoop statements. Swoop statements are enclosed in dollar '$' characters (the dollar character itself is available by using $$). Swoop supports the following types of statements:

- SQL statements
- SQL variable statements
- Repetition statements

Embedded SQL is divided into two parts within swoop-files. *$sql ...$* statements contain the tail end of SQL select statements, from the table name(s) onwards. They may be placed anywhere in the file, although it is most convenient to place them together at the top. *Sql variable statements* are embedded within the HTML text; they contain the name of a table column, a separating colon, and a number identifying a corresponding *$sql ...$* statement. *swoopgen* constructs appropriate SQL SELECT statements by matching the variables and sql statements together. The advantage of this dual representation is that a single SQL statement can be used to fetch information needed in a number of places.

In the example above, the *name* and *telno* columns are required, so *swoopgen* will construct the following SQL statement: `select name, telno from staff order by name;`

The values retrieved from the database by the program will be inserted in place of the sql variables at run-time.

$repeat$ ... $endrepeat$ statements can be used to produce repeated sections of HTML if the SELECT statement is expected to return more than a single row of information (as is the case here). If an sql variable is encountered outside *$repeat$ ... $endrepeat$* statements, then it is assumed that only a single value will be returned, and only the first value returned is shown.

Stage 2: producing interlinked pages using swoop-files

The *tellist* program is particularly simple because it produces a general list. However, many pages need to provide different output depending on some qualifying information. For example, the staff home pages can be supported by a single dynamic page program (called *homepage*), provided that we tell it *which* member of staff is required. We can do this by passing an appropriate piece of information (the staff code) to *homepage* as a parameter. The *homepage* swoop-file is shown in Figure 5. It uses a special specifier (*arg-0*) *within* an SQL statement to qualify the search, using the first (i.e. zero'th) argument passed to the page; this argument should be the code of the staff member whose home page is required. A *$arg-n$* tag may also be used anywhere within a file, if you want to output an argument rather than using it to select further information.

Swoop file for *homepage*:

```
$sql staff where code = arg-0 $
$sql staff s, dept d where s.code = arg-0 and s.dept = d.code$
<title>Home Page for $name:0$</title>
<h1>Home page for $name:0$</h1>
<hr>
<i>Code number:</i> $code:0$<p>
<i>Department:</i> <a href="dept?$d.code$">$d.name:1$<p>
<hr>
<h2>Additional information</h2>
$additional:0$
<hr>
```

Sample HTML output by *homepage*:

```
<title>Home Page for Hunter</title>
<h1>Home page for Hunter</h1>
<hr>
<i>Code number:</i> AH<p>
<i>Department:</i> <a href="CIS">Computing<p>
<hr>
<h2>Additional information</h2>
Likes writing <i>software</i>!
<hr>
```

Figure 5: Using arguments in Swoop-files.

To access the individual home pages from the telephone list, we can modify *tellist* to include URLs which invoke *homepage*, passing the staff code as a parameter. Figure 6 shows an updated version of the *tellist* swoop-file, and resulting HTML output, which contains links to each home page.

Swoop-file:

```
$sql staff order by name$
<h1>Telephone Listing</title><hr>
$repeat$
<a href="homepage?$code:0$">$name:0$</a>, $telno:0$<p>
$endrepeat$
```

HTML generated:

```
<title>Telephone Listing</title>
<h1>Telephone Listing</title><hr>
<a href="homepage?AH">Hunter</a>, 2778<p>
<a href="homepage?SG">Garrick</a>, 3425<p>
<a href="homepage?GK">Khan</a>, 0225<p>
<a href="homepage?CP">Porter</a>, 2345<p>
```

Figure 6: Swoop-file with hypertext links

The second select statement in the *homepage* swoop-file (see Figure 5) is used to get information about the user's department (specifically, its name rather than its code) from the department table, using a JOIN condition to ensure that the department corresponding to this particular person is located. In the

case of Porter, this statement will not find any information, since Porter's department (ART) is missing. Swoop will simply leave the department blank in this case. The department is also further hyper-linked to the *dept* page, which is listed in Figure 7.

Swoop-file for *dept* dynamic page.

```
$sql dept where code = arg-0 $
$sql staff s, dept d where d.code = arg-0 and s.dept = d.code and
s.code = d.head $
$sql staff where dept = arg-0 order by name$
<title>Department: $name:0$</title>
<h1>Department: $name:1$ ($arg-0$)</h1>
<hr>
<i>Head of department:</i> $s.name:1$<hr>
<h2>Staff</h2>
$repeat$
<a href="homepage?$code:2$">$name:2$<p>
$endrepeat$
<hr>
```

Sample HTML output:

```
<title>Department: Computing</title>
<h1>Department: Computing (CIS)</h1>
<hr>
<i>Head of department:</i> Khan<hr>
<h2>Staff</h2>
<a href="homepage?AH">Hunter<p>
<a href="homepage?GK">Khan<p>
<hr>
```

Figure 7: *dept* swoop-file and output

The interlinking of pages in this fashion is typical of Swoop, which can thus implement interfaces to quite complex database structures with minimal effort. Many hyperbase programs tend to concentrate on a particular, simple approach to structuring the search space, in order to reduce programming complexity. Swoop reduces that complexity to a level no worse than that encountered in any database system design problem.

Linking swoop-files and other HTML files

As a final part in this section, It is worth noting that swoop-files can be intermixed quite freely with separate HTML files The information which is stored in the database is itself in HTML format (it is inserted directly into the merged page when displayed), which means that particular pages can include links to other information. For example, the *additional* information section in the home pages above could contain links to further user-specific information. It is also possible to link to information stored, for convenience, outside the database. For example, by adding a link as follows to the home page, a picture of each member of staff can be included, where these are assumed to be stored in files *AH.gif*, *GK.gif*, etc. `<img src="$code:0$.gif"><p>`

Using Swoopform for Database Maintenance

The *swoopgen* tool produces dynamic page programs which are invoked through hypertext links to produce information on-screen; the user thus sees only hypertext, and need not even be aware that a database is being used at all. However, somebody must provide the information in the database in the first place. Although tables for use in Swoop must be created using special tools (since Swoop maintains some auxiliary information about tables in its own special tables), once a table has been created, it may be maintained using whatever database facilities are available. The maintainer might prefer to provide an sqlforms-based interface, or to upload from an Access database, for example.

Swoop also provides a facility to support table maintenance via WWW: *swoopform*. Swoopform is a program which generates a CGI Form containing fields for each column in a table. The form may be used to create new rows, to update or delete existing ones, and to search for existing rows using a query. The table to be queried is passed as a parameter when *swoopform* is invoked. Since all the tables known to swoop are themselves described in a special table, it is a trivial matter to produce a swoop-file for *swoopgen* which generates a list of all tables, with hypertext links to *swoopform* for each - such a swoop-file is provided with the Swoop system. Thus, the maintainer automatically has access to Forms-based facilities to maintain all Swoop tables. The downside of this powerful automatic facility, is that *swoopform* is not configurable - it always presents a Form in the same fashion, which can be used solely to update a single table.

Swoopform has the following features:

- There is a Submit button at the bottom, together with a set of radio buttons for the three options: Query, Insert and Delete. The action of the Submit button is determined by which radio button is pressed.
- A Query will retrieve the first row of the table matching the information in the fields; the match must be exact. This is used to retrieve table entries for update.
- Insert will update existing information or create new entries. Swoop tables have their Primary Key identified upon creation (and Primary Key fields have italicised prompts in *swoopform*, so that they stand out). Upon Insert, if the Primary Key matches an existing row in the table, then it is updated; if the Primary Key doesn't match, a new row is created. To avoid accidentally overwriting existing entries, the maintainer should fill in the Primary Key first, and execute a Query to check that nothing is retrieved.
- Delete will remove the first row of the table matching the information in the fields; multiple-deletions are not supported to avoid accidental damage to the database (these can always be done using the native database facilities).
- Fields are automatically sized to match the column width.
- Fields can contain HTML source, and *swoopform* will correctly store his in the database and merge it into pages upon display.
- A small amount of additional information can be included for use by *swoopform* when tables are created; this allows links to additional pages to be added in.
- A link to table-specific help is automatically provided, although the database designer needs to provide the HTML file which it attempts to access!
- A link to a general *swoopform* help page is automatically provided.
- Swoop tables can be marked as secure or insecure; if a table is marked as secure then *swoopform* includes a password field, and will not allow access to the table unless the password is entered. Insecure tables can be freely accessed: this allows *swoopform* to be used to gather information

from users. For example, we have used a simple *swoopgen* program to generate an Electronic Student Notice Board, which includes a link to *swoopform* to maintain it. Thus, students can both read the notice board and add messages to it. Messages can be assigned priority ratings and post/remove time-stamps, all of which are handled very simply through SQL in the swoop-file (the notice board source is provided as an example with the Swoop source pack).

Swoop maintenance utilities

Swoop provides tools at a number of levels; *swoopgen* is used by the system designer to generate dynamic page programs; *swoopform* is used by database maintainers (sometimes solely the system designer; sometimes a wider group) to add and update information. It also provides a small number of additional facilities to aid the system designer.

Although ORACLE maintains a great deal of information about individual tables, not all of it is easily retrieved, and there are some additional pieces of information (for example, the prompts used for fields on *swoopform*) which ORACLE doesn't support. Swoop therefore maintains two auxiliary tables in the database: SWOOPTABLES and SWOOPCOLUMNS. The information in these two tables must correspond to the profile of the tables accessed by Swoop. To make this correspondence simple to maintain, the Swoop distribution includes two scripts which create and delete swoop tables. The *createhtmltable* script takes as an argument a file name; the file may include a number of Swoop Create statements, which are essentially augmented SQL Create statements. The *deletehtmltable* script takes as an argument a table name, and deletes both the table and any references to it in the Swoop auxiliary tables.

Further details on the Swoop maintenance utilities can be found in the Swoop source distribution.

Future work

The purpose of Swoop is to make the generation of ORACLE hyperbase programs on WWW simple. In this it has succeeded; producing a Swoop-based database system is no harder than producing a conventional database system on any platform (actually, often easier since there is no need to generate any Forms-based interface). However, there are a number of enhancements which could be made:

SQL syntax.

The approach taken to embedding SQL in swoop-files is effective, but untidy. Currently, the $sql ...$ statement contains only the tail-end of the SELECT statement, which is constructed by assembling the variable tags associated with it to form the prefix. This has several consequences:

It is sometimes difficult, on first reading, to tell the exact purpose of an SQL statement (because it is effectively scattered through the swoop file).

If complex compound variables are to be fetched (e.g. SUBSTR(NAME,0,1) to extract an initial from a forename) then these must be included as variable tags, and are difficult to read.

The DISTINCT clause, which comes at the beginning of the SQL statement, necessitates an additional tag *$sql_distinct ...$*.

The approach taken was adopted because it precludes any need to parse the SQL statement: the construction of a prefix is relatively easy. A better approach would be to augment the SQL SELECT statement with tag names, and to remove these using a parser if necessary:

```
$sql select distinct d.name, substr(s.name,0,1) into dept, init from dept d, staff s
where d.code = s.dept$
```

- is translated into the SELECT statement:

```
select distinct d.name, substr(s.name,0,1) from dept d, staff s where d.code =
s.dept;
```

- and the tag variables *$dept:0$* and *$init:0$* refer to the first and second columns fetched respectively.

Interpretative Swoop.

swoopgen generates dynamic page programs, in PRO*C, which are compiled and run. This is the most time-efficient approach when actually using the dynamic page programs. However, PRO*C programs are extremely large, and take some time to compile. An interpretative version, which read a swoop-file and dynamically constructed pages from it would be useful, particularly during the developmental phase.

Integration into Web servers.

The requirement to integrate web pages with SQL-based databases is extremely common. It would be convenient if embedded SQL was supported within an extended version of HTML, with sophisticated Web servers able to interpret such statements. Obviously, the syntax of embedded SQL statements would be adjusted to fit into HTML standard; specifically, by replacing the $ tag delimiters with standard HTML <> delimiters. Swoop uses $ tag delimiters specifically to highlight the different interpretation that is placed upon them. A suggested standard is:

```
<sql select="<select statement>">
<sql repeat>
<sql endrepeat>
<sql variable="<variable name>">
```

Replacement for SWOOP

A replacement package, which uses an intrepretive approach and has a far simpler syntax, together with an improved *forms* interface, is currently under development and will be available in the near future.

Availability

Swoop is available solely via WWW at: *http://osiris.sund.ac.uk/ahu/swoop/home.html.*

If you need anonymous *ftp* to download it, you have no use for it anyway!

References

1. Ford, A. *Spinning the Web*, International Thompson, p.143.

2. McCool. *Web document*, http://hoohoo.ncsa.uiuc.edu/cgi/index.html

3. Sturner, G. *ORACLE 7: A User and Developers Guide*, Van Nostrand Reinhold.

4. OWWWIK. *The Oracle World Wide Web Interface Kit*, http://dozer.us.oracle.com:8080/index.html

5. WOW. *The WOW Gateway*, http://dozer.us.oracle.com:8080/sdk10/wow/

6. Decoux. *The Decoux Gateway*, http://dozer.us.oracle.com:8080/sdk10/decoux/

7. Ocrainets. *The WORA Gateway* http://dozer.us.oracle.com:8080/sdk10/wora/

About the Author(s)

Dr. Andrew Hunter
cs0ahu@sunderland.ac.uk
http://osiris.sunderland.ac.uk/ahu/home.html
Department of Computing and Information Systems, University of Sunderland, England.

Andrew Hunter is a Senior Lecturer, with interests in Genetic Algorithms, Neural Networks, and Interactive Software. He produces the http://osiris.sunderland.ac.uk/rif/welcome.html
Department of Computing and Information Systems, University of Sunderland, England.

Ian Ferguson is a Senior Lecturer with interests in Object Oriented software and the Internet.

Mr. Steven Hedges.
steve@maxx.co.uk
http://www.iisl.co.uk/
Internet Information Services, Ltd, 498 Dereham Road, Norwich, NR5 8TU, England.

Steven Hedges runs IISL, a WWW training and consultancy service.