Catalog: a WWW Gateway for DBMSs

José M. Martínez, Francisco Morán
Grupo de Tratamiento de Imágenes, Depto. Señales, Sistemas y Radiocomunicaciones
E.T.S. Ingenieros de Telecomunicación, Universidad Politécnica de Madrid
28040 Madrid, Spain
Tel: +34.1.336.7353 Fax: +34.1.336.7350
Email: `{jms,fmb}@gti.ssr.upm.es`
WWW: `http://www.gti.ssr.upm.es./staff/{jms,fmb}UK.html`

Abstract: The Catalog software package described here is designed to act as a gateway to the WWW for information systems based upon a DBMS: a so-called *hyperbase* program. There exist already several approaches to the problem, mainly working with ORACLE's RDBMS, although migration to most of other vendors' RDBMSs should be straightforward. Catalog handles the problem in a new way, and to serve a different primary purpose: it is a tool to dynamically extract items from the DB and group them together. No modification in the DB structure is required, and it is not necessary —nor possible— to program CGIs for querying. Therefore, its main role is the dynamic creation of *Cards:* HTML pages that are the result of searching and retrieving the contents of the DB.

1. Introduction

1.1. Hyperbases and the WWW

Hyperbases [Hunter et al. 95] are hybrid systems in which 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 files via hypertext links. Dynamic documents are programs which can be invoked in place of a direct file download, and generate the document to be downloaded as its output; a suitable program can hence extract information from a database and present it as an HTML document. For instance, the Forms interface, available under CGI, allows web pages to include user-interface elements such as fields, buttons and check-boxes: this can be used to provide data input to dynamic documents.

Hyperbase programs may be presented to the user in two ways. First, the user may be presented with forms which ask explicitly for input: for example, key-words to be used in a search. Forms can also be used to present output, although this is rarely done: most hyperbase programs present the results of a search as "simple" hypertext. The main problem with such hyperbases is that they require sophisticated CGI-compliant programs to interpret forms, query appropriate databases, and merge the resulting information into hypertext. There is a clear need for application-generator tools to allow hyperbase programs to be constructed with minimal expertise on the part of the designer. The other possible solution is to let the user browse what appear to be normal web pages (i.e., with no forms), but are in fact being dynamically constructed from the database.

1.2. Related Work

The WOW, DECOUX, ORAYWWW and WORA Gateways [Oracle 95] are some examples of tools that were developed before the release of the ORACLE Web Agent included in the ORACLE Web Server [Oracle 96], and go some way towards merging WWW and ORACLE. Another gateway that needs programming, but is open to different RDBMSs is GSQL [Jason 95].

Catalog does not need programming as the previous gateways do. Two other approaches which do not need programming either are SWOOP and MORE, which use ORACLE as a back-end, but are easily portable to other vendors' DBMSs:

- SWOOP [Hunter et al. 95] provides an application generator, namely a CGI programming language and environment. Information is presented to the user by means of simple hypertext pages; these pages are specified using an augmented yet simple HTML syntax. The system designer using SWOOP needs some knowledge of ORACLE and HTML, and the programmer should know about the DBMS schemes.

- MORE [Eichmann et al. 94] provides a meta-data repository: information about the artifacts under its management scope is stored and consulted to locate, load and compose those artifacts, stored by other means. MORE acts as a front end to organize assets based on a predefined meta-data repository, where the assets are incorporated. Browsing can be done by collection, classes, and natural language or pattern searches.

Catalog needs neither ORACLE nor HTML knowledge, as SWOOP does, but on the other hand, there is no flexibility for programming searches in the DB. It also leaks the complex searching facilities of MORE, but no reorganization into a meta-data repository is needed.

A closer approach to Catalog's is WDB [Rasmussen 95], which provides a software tool-set that simplifies the integration of SQL databases into WWW, providing access to the contents without writing a single line of code. The interface consists of a WDB script written in Perl and a set of *Form Definition Files (FDFs)*, each describing a different view of the database. These FDFs play a role similar to the Catalog *Templates*. Although Catalog does not include currently data conversion and computed values of fields, it provides access control, multilingual support, Card layout customization, and a more elaborated query mechanism.

2. Catalog system

The principal aim of Catalog is to provide an immediate and easy availability of data collections, as already available at several sites (museums, libraries, etc.). Besides the dynamic creation of queries, sets of results (Card descriptors), and Cards, it offers the means for:

- Customization of the Card layout (by the modification of the Templates).
- Dynamic information filtering over the data, thus allowing the retrieval of different parts of the data depending on the user profiles defined by the Information Provider — and independently of other access control mechanisms provided by the Service Provider.
- Supporting multilingual dynamic page creation.

Catalog consists of a *Templator* creating the raw templates of the user's tables in the database, and a set of compiled CGIs providing the set of services needed to search, retrieve and navigate through this information. The customization of the templates makes it possible to use fully all the available features; it follows a set of simple guidelines where no programming is required, nor DB or HTML knowledge. Every Catalog *Service* is divided into three software blocks [Fig. 1]:

- The *CGI interface* for that particular service, which interprets the input parameters (POST and GET methods), calls the needed functions, and formats the result into an HTML document.
- The *Functional Core*, composed by a set of functions for searching and retrieving the data items. These functions use the Database Interface for retrieving data from the storage system, and the user's Template for information filtering and object composition (including multilingual labelling).
- The *Database Inteface*, which allows the access to the available storage system, be it a DBMS, a file system, or any other. It is composed by a couple of simple C functions. Currently, it is available only for ORACLE, and it uses the Pro*C precompiler.

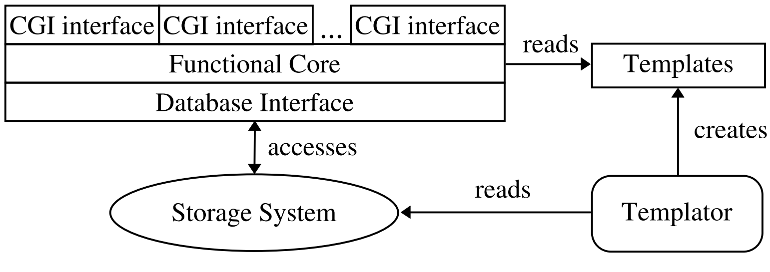

Figure 1: Catalog software blocks

2.1. Catalog services

There are several Catalog services, each with a separate CGI interface, to allow a flexible control over data retrieval, although not all of them are needed for all access configurations.

`entries` Reads the login and password for a user introduced via an HTML form. The dynamic document includes the available entries for this user.

`fields` Returns, for a given user and an entry, a dynamic document containing the available search fields. Several fields can be chosen from the list.

`items` Returns, for a set of search fields, a dynamic document containing the list of available items for each fields (if the field is of type `WRD` — see below), a free search box (types `FRE` or `TXT`), or both (type `STR`). Several items can be chosen for creating a query. The items of a same field are grouped by an OR condition, while different field groups are grouped by an AND condition.

`hits` Returns, given a query, the number of hits and a list with the *Card's labels*. If there are less than a pre-defined number of hits, the dynamic document contains a set of descriptors instead of the list of labels. Several items can be selected from the list of labels in order to ask for the descriptors.

`shorts` Returns a set of descriptors for a list of Card's labels. Each descriptor holds a link to the *full Card*.

`card` Returns a Card.

In order to create the dynamic document, each CGI service first reads the Template for information filtering and document composition and labelling instructions, and then retrieves data from the storage system. This is done using the Functional Core services.

2.2. Functional Core

The Functional Core provides the general functionalities needed by the particular services. For each CGI there is a corresponding Functional Core public service, which is in turn composed by one or several private methods in order to perform information filtering, access control, data locations and data retrieval. The formatting of the document into HTML is done within the public service code, in order to allow future reusability of the private services for other information servers supporting different document formats like MHEG.

Once it has read the Template and retrieved the document components, the Functional Core checks the permission level for each component with the user access level. When the Functional Core has identified the components that are accessible for that particular user, it accesses the Storage System through the Database Interface to retrieve the data, and then composes the document according to the Template structure.

Currently, access control is implemented by means of a simple, file based, security mechanism. When the user logs in the system through the `entries` CGI, the security agent checks the identity and finds the user profile associated to this user, which corresponds to a per site access level. This access level is used for information filtering using the permission levels associated to the different entries and fields via the Template customization. The user profile is the base for a *session ticket*, which includes information about the session language. Both values are passed among CGI and dynamic documents during the session, in order to obtain session state information. For security, the session ticket is encrypted with a key which varies from session to session.

2.3. Database Interface

In order to allow independence from the underlying DBMS, the Functional Core translates all the queries into a set of simple SQL calls which can be easily implemented for every SQL-DBMS. The required functions are: **`connect_db`**, **`disconnect_db`**, **`count_hits`**, **`get_hits`**.

For the moment, only the ORACLE Database Interface has been implemented using the Pro*C precompiler. INFORMIX and Ingres versions are planned as well, using the corresponding C precompilers. Some work is also planned for implementing this interface over a propietary DBMS not supporting SQL, but only keyword search. This will yield to a redefinition of the current interface, which is clearly RDBMS-compliant.

3. Catalog Templates

Catalog uses templates for both information filtering and document composition, including multilingual labelling. There must be a Templator application for each supported DBMS, that will create a master template for each DBMS user. This master template extracts all the tables, columns and cross-links for that user, so that the Catalog CGIs can run properly, although it may need customization for a full use of Catalog. For each table that will be accessible to the user, an entry is created in the master template with the following syntax:

```
**
IU_REF,IU_link,permission_level,label
number_of_components
{component}
...
{component}
IU_END,IU_link
**
```

where:

`IU_REF` Information Unit REFerence begin tag.

`IU_link` The table's primary key and the table name separated by the '~' character, as in `primary_key~table_name`.

`permission_level`
An integer assigning a minimun permission level for accessing the table.

`label` A label for the Information Unit or component. The Templator writes the table name or column name, removing '_' characters, if any. In the case of a link component (`REF:IU` type — see below), the Templator writes `LINK`*i*, where *i* is the number of the link in the IU's table.

`number_of_components`
The number of componentes found for this Information Unit.

`{component}` `component_type,component_link,permission_level,reference_flag,search_flag,document_flag,descriptor_flag,label_flag,label`

`component_type`
One of `STR`, `NUM`, etc.

`component_link`
If the component corresponds to a table column, the column name; if it corresponds to a link between tables (`REF:IU` type), the link definition including both anchor (current table's column name) and destination (column and table names), separated by the '~' character, as in `column_name~column_name~table_name`; if it corresponds to a group tag (`LAB` or `BAL`), a group style (see also below).

`reference_flag`
`I` or `R`, for inclusion of the component data or only a reference to it.

`search_flag` `SEARCH` or `NOSEARCH`, depending on whether the component is searchable or not.

`document_flag`
`DOC` or `NODOC`, depending on whether the component should be included in the Card's document or not.

`descriptor_flag`
`SHORT` or `NOSHORT`, depending on whether the component should be included in the Card's descriptor or not.

`label_flag` `LABEL` or `NOLABEL`, depending on whether the component should be included in the Card's label or not.

`IU_END` Information Unit reference END tag.

component types:

WRD Textual component available for list of terms searches.

FRE Textual data, with free text search.

STR Textual component presenting list of terms and free text searches.

TXT Textual preformatted data, with free text search.

NUM Numerical component, currently not searchable.

DAT Date component, currently not searchable.

BIN General binary data component.

IMA Image component.

VID Video component.

AUD Audio component.

REF:IU Link from a IU's component (entry field) to another IU, created when there are referential constraints between tables.

REF:{BIN,TXT,IMA,VID,AUD} Link to a binary, textual, image, video, audio component.

URL Internet resource locator component, which is presented as an active HTML link.

LAB Group begin tag.

BAL Group end tag.

group styles:

DEF_LIST The group components are presented as a definition list, with the label begin the definition term and the content the description. This is the default style.

ROW The group components are presented each in a row.

COL The group components are presented each in a column.

NUM_LIST The group components are presented in a table cell, each being an item of a numbered list.

LIST The group components are presented in a table cell, each being an item of a unnumbered list.

The page layout is based on HTML tables, in which each component is an item in a definition list consisting of a unique cell table (the document and descriptor Cards have some minor format differences). This is the default format style, which can be modified by using the group tags, although the table is still the base. Of course, the groups are recursive, that is, they can be combined in order to compose complex Card layouts.

The Templator converts the ORACLE types into the following Catalog types:

- `VARCHAR2`, `CHAR` → `STR`
- `LONG` → `TXT`
- `NUMBER` → `NUM`
- `DATE` → `DAT`
- `RAW`, `LOW RAW` → `BIN`
- Referential constraints between tables → `REF:IU`

Via customization, it is possible, for instance, to:

- Reorder the components within the IU.
- Group components and add grouping labels (via the label field of the `LAB` type component).
- Customize layout using `LAB` components with group styles in the component_link field.
- Identify `BIN` types as images, video, audio or other.

- Indicate that `STR` components are references (e.g., an image or video file path, that is, `REF:IMA` or `REF:VID`).
- Set permissions to the desired values.
- Change the labels to "natural language" ones.
- Add multilingual support. The current implementation uses the component's `label`, where several language labels are separated by the '%' character. The order of the different language labels, should correspond to the ones defined in the login form or in any other first form. The first one is used as default one.

4. Installation and Availability

To install Catalog, it suffices to copy the CGIs to the directory where the CGI binaries of the Web Server reside, and to run the Templator in order to create raw templates that can be eventually modified later on.

Currently, a prototype version of Catalog is available for consulting the database of our research group at "http://www.gti.ssr.upm.es./", and also that of the Museo Arqueológico Nacional (Spanish National Archaeological Museum) at "http://www.gti.ssr.upm.es./~man/".

5. Conclusions and Future work

The purpose of Catalog is the dynamic creation of HTML pages reflecting the search and retrieval of the contents of the DBMS, that can be interpreted as Cards. This has been fulfilled, with no need of modifying the storage structures already established by the data providers. It also allows information filtering and multilingual labelling.

The principal enhancement should focus on the searching mechanism, to avoid the "rigidity" of the present queries. A more friendly tool to edit templates would also be helpful, as well as a more powerful layout definition support. The access control is being enhanced by using a more elaborated security mechanism.

6. References

[Hunter et al. 95] A. Hunter, R.I. Ferguson, and S. Hedges. *SWOOP: an application generator for ORACLE/WWW systems.*
http://www.w3.org./pub/Conference/WWW4/Papers/207/

[Oracle 95] *The Oracle World Wide Web Interface Kit.*
http://dozer.us.oracle.com.:8080/index.html;
http://www.oracle.com/corporate/press/html/sdkrel.html

[Oracle 96] *Oracle WebServer: About Oracle Web Agent.*
http://www.oracle.com./products/websystem/webserver/html/ois3.html

[Jason 95] Jason Ng. *GSQL - a Mosaic-SQL Gateway.*
http://base.ncsa.uiuc.edu.:1234/gsqlsrc/starthere.html

[Eichmann et al. 94] D. Eichmann, T. McGregor, and D. Danle. *Integrating Structured Databases into the Web: the MORE System.* Computer Networks and ISDN Systems 27 (1994) 281-288.
http://rbse.jsc.nasa.gov./eichmann/www94/MORE/MORE.html

[Rasmussen 95] B.F. Rasmussen. *WDB - a Web Interface to SQL Databases.*
http://arch-http.hq.eso.org./bfrasmus/wdb/intro.html