Hystrix It. J. Mamm. (n.s.) 18 (2) (2007): 185-194

COLLECTION, WAREHOUSING AND DISSEMINATION OF SPECIMEN INFORMATION: AN ADDED VALUE FOR THERIOLOGICAL COLLECTIONS

DAMIANO G. PREATONI

Dipartimento Ambiente-Salute-Sicurezza, Università degli Studi dell'Insubria, Via J.H. Dunant 3, 21100 Varese, Italy; e-mail: prea@uninsubria.it

Received 15 September 2007; accepted 7 December 2007

ABSTRACT - Recent achievements in the technologies for information management and sharing would allow to make more available and exploitable the wealth of data represented by theriological museum collections. Anyway, the scarce diffusion of Information Technology knowledge in the theriological field hinders the transition towards digital cataloguing of collections, often leading to the creation of data bases unable to last through time and without coherent information management policies. The aim of this contribute is to present a concise review of the existing practices and technologies used to design and implement information systems, in order to promote the increasing application of such technologies in the theriological and, in general, in the natural resource conservation field.

Key words: Museum collections, database design, information systems

RIASSUNTO - ***Raccolta e condivisione delle informazioni sui reperti: un valore aggiunto per le collezioni teriologiche***. I recenti sviluppi delle tecnologie per la gestione e la condivisione delle informazioni rendono oggi possibile una maggiore fruibilità e disponibilità del patrimonio costituito dalle collezioni teriologiche. Tuttavia, la scarsa diffusione nel contesto teriologico e museologico delle conoscenze nel campo dell'*Information Technology* rende difficoltosa la transizione verso la catalogazione in formato digitale, portando spesso alla creazione di banche dati che non garantiscono una ragionevole durata nel tempo né la necessaria coerenza nell'organizzazione delle complesse informazioni concernenti il catalogo di una collezione. Il presente contributo offre una concisa rassegna dei principi di base e delle pratiche più comuni nello sviluppo di sistemi informativi, con l'obiettivo di favorire una loro sempre maggiore applicazione nel campo della teriologia e della conservazione delle risorse naturali in generale.

Parole chiave: collezioni museali, progettazione di basi di dati, sistemi informativi

INTRODUCTION

Museum collections, in particular the theriological ones, represent a great asset for the whole scientific community. The last ten years achievements in technologies for networked information sharing, as well as the ongoing standardisation in information technology practices, would allow to make this wealth more available and usable by all users.

Information, by itself, can be defined as a peculiar kind of "renewable resource": it is an immaterial resource, and its efficiency seems to depend more on how information is organised rather than information content itself. As an example consider the differences between a computer-based and a paper-based catalogue: though the information contained in both contexts is exactly the same, its organisation makes the two systems almost totally different in their employment and performance.

In a paper-based catalogue, all relevant items are usually stored (i.e. written) together on a single card, drastically limiting their availability for more than a single person at a given time. On the other hand, trying to increment information availability using "replicates" (i.e. copies of each card) other problems are roused, e.g. the need of updating all the copies of a card, which can quickly become a hard task.

The use of information technologies to build information systems for museum collections makes information maintenance, updating and dissemination easier, opening further possibilities than the increase in availability: a remarkable trait of information as a "renewable resource" is its tendency to increase (and not to degrade or vanish) with use. This is especially true when the use of an information system allows to produce new information from existing items (either by analysis or synthesis), or to rearrange without harm the entire knowledge base, enhancing both the quality and quantity of the information handled by the system.

Therefore, it is possible to define some requirements to be met by a computer-based information management system, in order to efficiently handle information contents and flows typical of a museum collection catalogue.

The use of a computer-based catalogue, as a matter of fact, should be functional to the transformation of the existing (and supposedly working) catalogue into a shared and more available asset, able to grow and last in time. The main characteristics of such a system can be summarised as follows:

- independency from particular hardware and software architectures;
- guarantee to endure through time;
- simplicity in accessibility, availability and use;
- possibility to handle information supplied by users.

The apparently wide choice of software and hardware tools could suggest the existence of a simple and straight solution to the problem: in contrast, current situation is extremely complex and mutable, and really viable solutions, given the requisites listed above, are indeed few.

Several of the available systems to create and operate a database are designed to build small personal databases that can be produced in a short time and maintained with little effort. Notwithstanding this friendly appearance, these systems cannot guarantee any functionality either if the quantity of information raises or if data need to be shared. These systems also badly tolerate any rearrangement of the "rules" defined to store and organize data.

In addition, it is rather frequent that, just for commercial reasons and with no real advantages to the user, the format used to represent data (the so called "inner schema") of a given

personal database management system changes abruptly from year to year, severely compromising any possibility for the database to endure through time for more than five years (the so-called "backward incompatibility" problem). This and other factors suggest that the choice of a software tool based on its apparent simplicity and swiftness could lead in the long time to the realisation of an information system worse than an "old style", paper-based one.
Applying these concepts to the realisation of a computer-based collection catalogue, some solutions are here presented and discussed, focusing on good practices and pointing out the different and necessary steps defining the lifecycle of design, development and use of a database.

SYSTEMS ARCHITECTURE

Whilst personal database systems usually consist of a single program installed on a single computer, an efficient data base management system is often designed as a set of separate, interacting modules (Fig. 1). Each module is able to perform just a few well-defined and specific functions. This sort of distributed architecture can lead to several implementations, depending on the degree of separation among functions. Two main architectures are used: a two-tier architecture, involving a *server* system which handles data storage and management and a *client* system which only consumes (i.e. uses) data, and a three-tier system similar to that represented in Figure 1, where a second server (a web server for instance) joins the proper database server, acting as an inter-mediate proxy between the server and several (and diversified) minimal data consumers (such as web browsers).
The separation among functions achieved with a server-client architecture allows to build truly distributed and accessible data bases, since all the functions connected with data management are carried out by a single central computer, which is easily accessible by

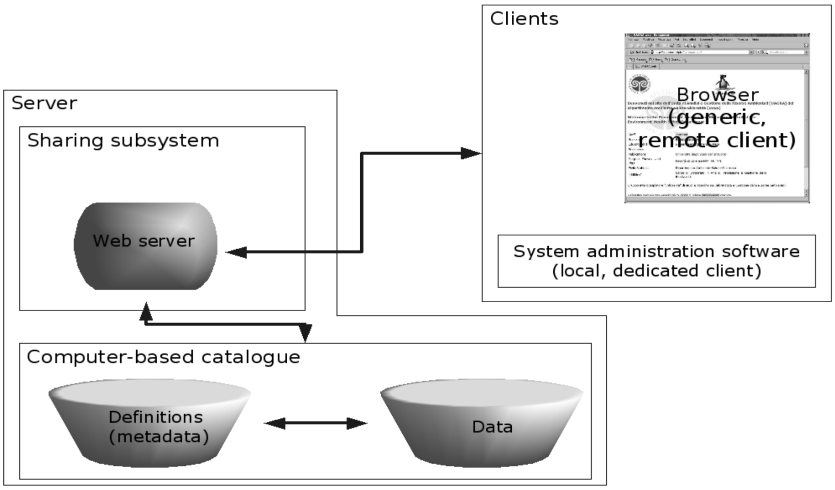

Figure 1 - Concept schema of a typical client-server architecture.

users. In this way, the problems linked to "monolithic" databases, that require the dissemination of copies to the various clients and then suffer, although in a more sophisticated fashion, from the same handicaps of paper-based catalogues, can be overcome.

SERVER MODULES

In a generic two- or three-tier architecture, the server is dedicated to storage functions, managing both data and rules to which data must adhere ("metadata" or "schema"). In most cases, server modules are highly specifically designed softwares, running under a given operating system and available as services to other programs (clients). They are not directly visible to users, allowing just low-level interactions, for base administrative functions such as the definition of the "schemas" to which data must conform (also called "outer schemas"), or other data organization rules, such as data access constraints (usually called "rights" or "grants") and bulk data transfers in raw formats (data pumping or data base population). The data base management system (DBMS) constitutes then the heart of an information system, with which users interact by intermediate programs that are either specialised data analysis packages or data base administration tools, used to make backup copies or to grant / revoke access rights to users.
An efficient DBMS is not only able to store data and metadata, but also allows to store all the relevant and recurrent procedures to manipulate information, supplying, by means of tools such as "views" and "procedures", further tools for widely manipulating (i.e. create, retrieve, update or delete) its contents. All these functions are assembled in a single container in a way that is totally transparent to the final user.
At a first glance, this solution could seem awkward and less effective than the "single program plus data file" monolithic solution.
Anyway, the total separation of data storage functions and tasks from data presentation and use, together with the capacity of storing views and procedures, that is of storing even complex analysis procedures, and the possibility to serve simultaneously several clients ("transational" approach), using a consistent language to dialogue (normally a declarative one such as SQL), make DBMS an instrument able to cope with the time endurance, scalability and coherence requirements discussed above.

CLIENT MODULES

If the typical characteristics of a DBMS server make this set of tools a real data warehouse and limit its functionalities just to data storage and retrieval, on the client side issues are exactly on the opposite: the possibilities of using data are in fact almost infinite and unpredictable, and several programs can share the same data warehouse, using it in many different ways.
Considering the specific case of a system devised to handle a museum catalogue, it is possible to imagine the existence of several specialised client softwares: apart from those for the simple database administration, it is possible to have softwares for handling

data about specimens, printing labels or paper catalogues, mapping specimen locations, performing biometrical and morphometrical analyses, or comparing and aligning DNA sequences coming from the specimens themselves. Possibilities are almost infinite, and especially in this case, the use of a two- or three-tier architecture allows first to concentrate into a single place (the server) all the data manipulation functions common to all clients, and, second, to develop, use and maintain each client system separately, with little or no influence at all on other "data consumer" programs. These two characteristics are almost impossible to obtain working with a monolithic database system, where data application is tightly connected with data repository, and the possibility of their true concurrent use is often not implemented.

DATA WAREHOUSE DESIGN

A database should reflect the information categories and flow of a given organisation (e.g. a museum collection catalogue), and therefore, to achieve a really functional product rather than just an addition of constraints, the design process should consider all the facets and details of the existing reality.

Modern database design techniques (Atzeni *et al.*, 2002) emphasize identification and formalisation of the "rules" to which data have to be conformed (both in the conceptual modelling phase and while defining the logical and physical database schemas), relegating data themselves to the background. To consider primarily "rules" than data, for example producing the so-called "schema" of a database, helps in the first place to verify how data should be organised and to evaluate the efficiency and coherence of the existing paper-based information system, pointing out any positive sides, which need to be effectively transferred and preserved in the computer-based information system.

A software tool that can only produce a graphical representation of a database model, even if a schematic representation of a database surely gives a deeper insight on data organisation, is not worth using. The next step should be the translation of such a schema into instructions readily available to the data base management software, or - better - to a range of different DBMS systems, so as not to force the designer to a particular DBMS software and to allow to reproduce the same schema on a different software product. Especially for complex databases, it is inconceivable to convert by hand a graphical representation of the schema into a machine-readable one, because it would be both a time-consuming and error-prone task. Actually, some database modelling softwares allow to automatically extract all the relevant information from the model, using the captions and descriptions linked to each single object, to produce text documents useful to properly describe the information system. This, of course, is far more than the possibility to store simple descriptions of each separate object constituting a database, a functionality that is present in most database systems.

In conclusion, the design process is

articulated in several consecutive phases, which will be described below. Although they are presented separately, in real cases the server and client designs are developed together, and influence each other. In particular, the design of a client system cannot be done without a sound knowledge of the data structures and services offered by the server.

THE SERVER DESIGN

1. Conceptual modelling

The first step in the modelling process includes the identification and understanding of all the peculiar characteristics of the information system and of all the possible user roles. Moreover, the relevant data flows and manipulation processes have to be pointed out. Usually, this phase is already made using a modelling tool, but can be carried out equally well with a simple paper-and-pencil approach, aimed at identifying the "actors" of the particular data context that has to be managed.

2. Logical modelling

On the basis of the outcomes of the conceptual modelling phase, it is possible to identify the data structures (tables and fields in each table, relationships between tables, data integrity constraints, etc.), which are necessary to guarantee a proper data storage and retrieval. Particular attention must be paid to identify and resolve any redundancy (using normalisation techniques, Codd, 1970). The outcomes must be reviewed with the future users of the systems, to check whether their necessities have been met or not.

3. Physical modelling

Until a precise definition of the type of data which have to be stored in each field is not available, the produced logical model should not depend on a specific DBMS software. Physical modelling involves the adaptation of the logical model to the constraints (or advantages) offered from any specific software. It is important to point out that today most data base management software share a number of commonly implemented data types and structures, freeing the designer from the constraints which depend on the choice of a particular program. The possibility of changing one's mind and reverting to a different DBMS while developing the physical model is not negated. This paradigm is valid, of course, for data base management software realised in compliance with the current standards (such as ISO/ANSI SQL-99 and SQL-2003, Melton and Simon, 1993), even if some small differences could exist.

4. Prototyping

The physical realisation of a DBMS has to be tested on a properly equipped system, also loading existent data to test the performance of the prototype.
The deployment of a prototype often involves the realisation of accessory programs (or simple operating procedures) to fill in the database itself with quantities of pre-existing data ("data population"; this process is different from the routine addition of few new

records at a time). When data migrate from a paper-based system to a computer-based one, the process involves the initial mass conversion of the existing material into a digital format, causing, as a by-product, the development of further procedures or the refinement of the database schema itself.

5. Prototype evaluation and testing

This phase includes all the tests on the prototype which, in compliance with the functional requisites defined in the conceptual modelling phase, are needed to check whether the new system meets the functionalities requested by final users, and also to identify any error both in the data structures organisation and in the procedures developed to manipulate the data.

6. Deployment into production

The final step in the development cycle coincides with the installation on a computer of the data base management software along with the physical schema and all the relevant data. This computer will serve data to all users, deploying the information system "into production".

THE CLIENT DESIGN

1. Functional analysis

In analogy with the conceptual design phase of the server system, in this first step of the client design all the functionalities and operations, which have been requested by final users, have to be identified, often working in collaboration with the users themselves. This step must be subject neither to a specific type of client, nor to specific programming languages or existing software development practices and tools.

2. Interface design

The functionalities needed by users are translated into a form, that is into textual or - more commonly - graphical user interface elements - such as text boxes, buttons, menus - defining the structures by which users will interact with the DBMS. Also in this phase choices are not mandatory, that is subject to a specific tool for software development. Even if standard interface elements (called “widgets” or “controls”) are present on any development platform, their degree of differentiation is higher than that described before for the available data types used in the physical design. The various tools used to realise a user interface offer a wide variety of user interface elements; available controls often work in different ways and have to be programmed differently, even if, from the user’s point of view, their look is identical. As a consequence, the interface design can often result a very complex task.

3. Interface realisation

Once the interface design has been defined, often by the discussion of interface mock-ups with the final users, a particular programming technique and language are used to create a working prototype. Several solutions are obviously possible, ranging from simple command-line based interfaces (CLI), useful to perform repetitive tasks, up to more complex graphical user interfaces (GUI), developed in high-level languages such as C or C++, Java, Python, Delphi, Ruby, PHP, etc. In this case, the approach is heavily influenced by the programming language chosen and by the programming libraries available for a given language, which could make the client application development easier.

4. Usability tests

As described before for the server, the client prototype has to be tested by users, more often a restricted group of users ("power users"), which can give to the application programmers some useful feedback or suggestions about some elements of the client design that result difficult to understand or poorly performing.

5. Deployment

As for the server design, the final version of the client is installed, often (but not necessarily) on the same computer that hosts the server system.

TOOLS

To list all the software tools available to carry out the steps outlined above is an almost impossible task: new tools are released rather frequently and new design and implementation techniques appear each year. The following list is thus inevitably based on the author's personal preference and experience, and must be not considered exhaustive. It is also worth stating that the author has no relationship with the developer or distributors of the cited software tools other than being a satisfied user.
A particular emphasis is given here to the programs realised according to the Free/Open Source Software (FOSS) paradigm, because they are widespread, cheap (or free) and, which is perhaps more interesting, available for different operating systems and computer types. The use of FOSSs is rewarding, since the characteristics of most open source programs correspond to the requisite of independency from hardware and software changes and guarantee a long life to the data base.
For a detailed description of the following software programs, address to the correspondent web sites.

TOOLS FOR DATABASE DESIGN AND REALISATION

Even if database design is a crucial step, a few general-purpose tools, which can run under different operating systems and work with different DBMS servers, are available. This probably happens because database designers are few (when compared to the number of database users), and one or two widespread tools predominate on the market. Actually, most of the available tools depends on a particular DBMS software, or runs only under one operating system.

1. Entity-relationships (Chen, 1976) modelling tools

DB Designer 4 (http://fabforce.net-/dbdesigner4/). Dedicated to development of data bases based on the MySQL DBMS.

MySQL Workbench (http://dev.mysql.com/downloads/gui-tools/5.0.html). Another tool for the MySQL DBMS, bundled together with MySQL itself.
TOAD Data Modeler (formerly called CASE Studio 2, http://www.casestudio.com/enu/default.aspx). Database design tool for several different DBMSs.
Clay Database design tool (http: //www.azzurri.jp/en/software/clay/Eclipse/). A multi-purpose database design tool, integrable as a plugin for the highly flexible Eclipse software development tool (http://eclipse.org/).

2. Data Base Management Systems (servers)

Several are the tools available to set up ANSI92 compliant data base servers. The three listed below are probably the best performing ones and the most widespread (in particular MySQL, the most used database backend for web-based applications). Firebird, and moreover Postgres, even if apparently less widespread, are far more robust and mature than MySQL.
MySQL (http://www.mysql.com/) Windows, Linux
Firebird (http://www.firebirdsql.org/) Windows, Linux
PostgreSQL (http://www.postgresql.org/) Windows, Linux

3. Data Base administration and maintenance tools

BlazeTop (formerly SQL Hammer, http://www.devrace.com/en/blazetop/) Windows, for Firebird DBMS.
FlameRobin (http://www.flamerobin.org/) Windows, Linux, for Firebird DBMS.
PgAdmin III (http://www.pgadmin.org/) Windows, Linux, for PostgreSQL.
MySQL Administrator (http://www-it.mysql.com/products/tools/administrator/) Windows, for MySQL.

4. Web server and related tools

A web server is needed when a three-tier architecture is chosen, and provides to users the graphical interfaces as dynamic web pages. These are often created by programs written in languages, such as PHP, which are read and interpreted by the web server itself. This technique is known as LAMP (Linux, Apache, MySQL, PHP) or WAMP (Windows, Apache, MySQL, PHP), depending on the operating system used.
In this case, the Apache Web server, available for almost all the existing operating systems, represents the only reliable choice.
Apache, particularly Apache2 (http: //www.apache.org/).
Several pre-packaged installation kits for Windows and Linux operating systems are available, in particular XAMPP (http://www.apachefriends.-org/en/xampp-windows.html) and MapServer for Windows (http://www.-maptools.org/ms4w/), the latter intended to develop web-based Geographic Information Systems using WAMP techniques.

5. Application program development

Programming languages: the number of development systems is rather high: among others it is worth citing the PHP (Personal Home Page) programming language (http://php.net/), particularly used for developing web-based applications. The libraries distributed in the PEAR (PHP Extension and Application Repository) network enhance the rapid development of PHP applications.

For stand alone (not web-based) application development, the Delphi development platform seems to be the most widespread solution (http://www.co-degear.com/tabid/122/Default.aspx).

Other solutions are available, such as the Python language (http://www.python.org/), or some systems modelled after the Visual Basic language, such as Gambas (http://gambas.sourceforge.net/).

REFERENCES

Atzeni P., Ceri S., Paraboschi S. and Torlone R. 2002. Basi di dati, modelli e linguaggi di interrogazione. McGraw-Hill. ISBN 88-386-6008-5.

Chen P. 1976. The entity relationship model: toward a undified view of data. ACM Transaction on DB system 1, 1: 9-36.

Codd E.F. 1970. A relational model of data for large shared data banks. Communications of the ACM, 13(6): 377-387.

Melton J. and Simon A.R. 1993. Understanding the New SQL: a complete guide. Morgan Kaufmann. ISBN 0-55860-245-3.