Lorenzo Cioni

A data base for speech signal processing

(paper accepted at the SST-94 International Conference, Perth, Western Australia 6-8 December 1994)

Abstract - The present paper gives a brief description of an approach to the creation and administration of large corpora of data to be used for speech signal processing. It contains, moreover, a definition of persistent and volatile objects together with the description of both their structure and of the programs that take care of their administration.

Introduction

The aim of this paper is to present an approach to the creation and administration of large corpora of data to be used for speech signal processing (SSP). Such data include speech files, pitch files, spectrogram files, text files and the like.

All these are, in many cases, correlated files either on their own or depending on the user's will. The aforesaid correlation is indeed partly a natural consequence of the obvious links existing, for instance, among a speech file and the files containing data that have been derived from such speech file with some algorithm (for instance the pitch or the spectrogram or even a set of labels within a text file) but can even depend on the aims of the person that is accessing the files. Such correlation can be implemented through the definition of sets of files stored as a whole, each set forming a complex entity. Therefore what we need is a way to store such complex entities as persistent objects so to make both their administration and their processing easier. Within an environment devoted to SSP, indeed, we may have several applications and each application defines the type of files it can access. From our point of view this means that an application can have access to one or more objects and each object can be accessed either partly or as a whole, depending on the type of the application. This fact requires the definition of a volatile object as dynamically composed by elements extracted from a set of persistent objects.

The main aim of the present paper is to describe both the structure of persistent and volatile objects and the programs that take care of their administration. As to the structure, a persistent object is defined by a linked list of pointers to files and tables. Such structure is called stored structure as opposed to the accessed structure, i.e. the structure of a volatile object accessed by an application. The accessed structure may be dynamically obtained by linking files extracted from a set of persistent objects to form a list of pointers and tables. As to the programs (see figure 5), basically we have a program that manages the stored structures (data base manager or DBM), a program that maps the stored structures on distinct accessed structures (data structure manager or DSM) and a program that both interacts with the applications and allows a user to display and edit the structure of the objects he/she is going to access (data base interface or DBI).

The nature of the data

Within the scope of SSP the basic entity is the file. Each file can contain either speech data or the result of the manipulation of such speech data. From our point of view, a speech file is the starting point of every subsequent operation and, therefore the starting point is the ability to perform acquisition and play-back. Then we have the need to transform the acquired speech data into other information (data analysis) and to present such information to the user (data presentation). The aforesaid transformation includes segmentation, pitch extraction, spectral analysis, measurements, labelling and so on and gives rise to a multiplicity of files that, however, can be grouped within few categories. The presence of such limited number of categories (speech, pitch, spectrogram, text and numbers) allows the definition of a set of classes. A class coincides with a category and is formed by files of the same type.

To each class, moreover, there corresponds a set of operations (or methods) that can be executed over the data, and so over the content of the files. The set of the available methods, however, is not partitioned among the classes so that a method (for instance visualisation) can belong to more than one class. On the other hand, methods are subdivided into non disjoint subsets and each subset corresponds to an application. An application is indeed defined by a set of methods and by a set of data types it can access to.

The basic pipeline of a SSP environment (data I/O,[1] data analysis and data presentation) is shown in the upper part of figure 1 and sees the file as the ultimate source/destination of every process.

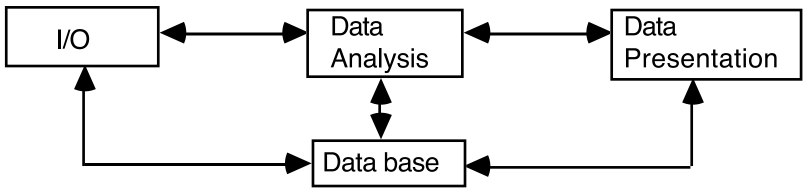

Figure 1. The basic/enhanced SSP pipeline.

From a traditional point of view, such pipeline is usually interfaced with a hierarchic file system that allows data storage. From our point of view, the process of data storage should be both more convenient and more efficient by using a data base (see next section). The presence of a data base fully extends the basic pipeline to the enhanced pipeline and requires the design of applications that are able to fully exploit the potentiality of the data base itself. The enhancement turns into the presence of the data base and the related programs as an element of any possible pipeline.

Why a data base and not a traditional file system?

The traditional approach is to collect speech files over which a user can perform some kind of analysis so to get pitch files or spectrogram files and the like that enrich the collection. According to this way of behaving, we consider a set of programs that face a set of files. Such programs, in many cases, have very little in common and often cannot share data, mainly because they deal with different file formats and/or data representation and coding. This, often, compels users to be programmers, or to rely upon programmers, so to write down bulk of codes that allow a dialogue among programs through the sharing of files. File storage and administration, on the other hand, are a duty of the users that must create and administer groups of directories, must remember where the files are stored, which are their versions and the existing links and relations among files.

Within this traditional approach, the classical hierarchical structure of a file system prevents users from establishing easy links and relations among files. Such a structure may be used to implement the physical structure of the data base but must be hidden to the users so to allow them to establish arbitrary relations among files.

The aim of the proposed data base is, therefore, twofold. It should allow users to access data directly through the data base interface so to know which data are available, of which type and which are the relations among them (this is particularly addressed to the use of applications that don't share the proposed data base philosophy, i.e. stand-alone applications). On the other hand it should interact with properly designed applications by exchanging with them the proper data. Figure 2 illustrates this situation. In such a figure with the term "User" we denote both a [real] user and an application.

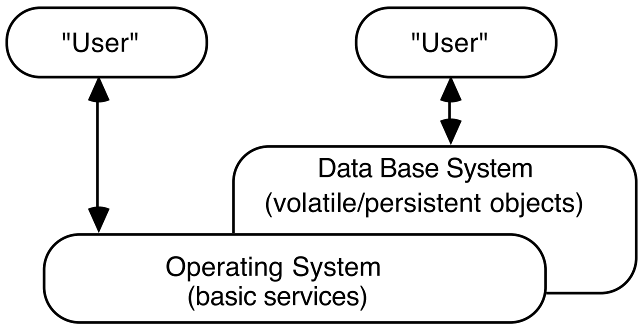

Figure 2. The Operating System and the Data Base System.

1 With I/O we mean both acquisition and play-back through any of the commercially available methods (IEE 488, plug-in boards, VXI bus boards and the like).

A user can either interact directly with the files through the traditional query language[2] provided by any Operating System or can access the files as elements of the data base through the data base interface. It is obvious that integrity and consistency are enforced if the access is mainly through the data base interface. As to the applications, the aforesaid situation is justified by the coexistence of applications designed to interact with the data base and applications designed as stand-alone modules. The aplications of the former type are able to access the data according to their conceptual/view level whereas the applications of the latter type are confined to the physical level and so to the access through the file system. Such applications access even more than one file (for instance speech and pitch) simply by name without performing any coherence control. The presence of the data base would help users to keep a clearer and more up to date situation of their data.[3]

The data base

The proposed data base is composed by the bulk of data and the aforesaid programs for their administration. The management of data requires the definition of the proper structures for their storage and flexible and effective mechanisms for their manipulation.

As to the data, we note the need to guarantee a data abstraction mechanism based upon the definition of physical, conceptual and view levels and schemes. At the physical level data are stored without any organisation, at the conceptual level data are grouped in a static way whereas at the view level they are dynamically structured.

Moreover, the data base should guarantee a physical data independence (i.e. the independence from the possibly distributed storage of files and from the underlying file systems provided by even different Operating Systems) and a sort of logical data independence, i. e. the possibility to change the inner composition of the persistent objects without modifying the applications that access them.

This means that the data type that describes the structure of a persistent object allows the presence or the absence of files belonging to certain classes. A persistent object is indeed a linked list of files and tables, and the type of each file is not prearranged or determined by its position along the list but is contained within the associated table.

The data base is characterised by a data definition language (DDL) and a data manipulation language (DML). The former is used for the definition of the basic data structures that implement the data base itself whereas the latter can be used by the users, either directly (though with the inter mediation of a shell, see figure 5) or through an application, to retrieve data stored in the data base, to create new data or to delete existing data from the data base. The DML should have the same structure in both cases (see figure 5) since it characterises the same data base interface.

Data retrieval bases itself on the execution of queries that allow an access either by name, by type or by even more complex patterns (for instance by type and after by name). Data creation involves the creation of either a new object or a new file. In the former case the operation can be executed even through a shell (see chapter about programs and figure 5), whereas in the latter case it is necessary to use one of the available applications that, either interacting with the hardware (acquisition) or processing an existing file, produces a new file with some type of data.

Data deletion of objects and files, on the other hand, can be performed only through the shell, see below). This is a precaution to enforce both consistency and integrity of the data base itself.

The structure of the data

At the physical level (physical scheme) we have only files, tables and links (not necessarily pointers), as shown in figure 3. The files can contain either data structures (see figure 4) or data produced by some a method, depending on the context. The element named key is a special table that allows an easy and fast access to each persistent object (see below).

The administration of files is a duty of the Operating System, whereas tables and keys (see figure 3) are managed through a data dictionary and represent instances of proper data types.

---

2 We denote with this term the set of commands for the administration of files and directories.

3 In spite of the fact that this situation is a real and interesting one, from now on we are going to consider mainly applications and users that access the data through the data base interface.

The data dictionary contains the definition of every data type that is present within the data base and so the definition of the persistent object data type; the definition of tables, keys and files data type; the definition of each class according to the methods it contains and so on. As to the tables, each of them contains information about the associated file, such as type (to which class it belongs to), owner, protections, size, date of creation and of last modification, exceptions and a link to the next element of the current persistent object.

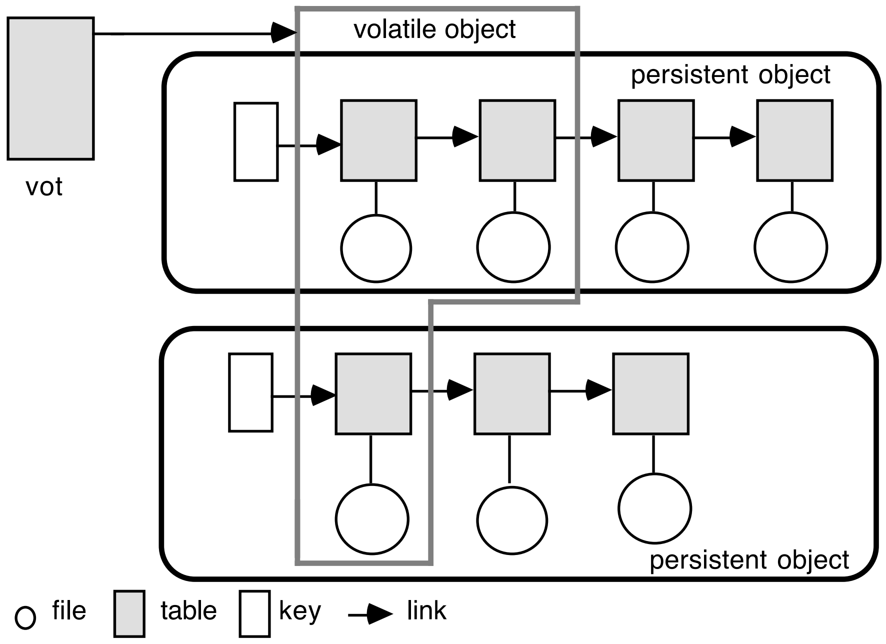

Figure 3. The structure of the data.

The exceptions are either local data or violations to the principle of inheritance. We consider exceptions the coding range, the sampling frequency, the inner file structure (sequential, indexed), the record structure and the presence of local methods or of locally disabled methods. Exceptions are required mainly for compatibility reasons with stand-alone applications, basically applications to perform I/O.

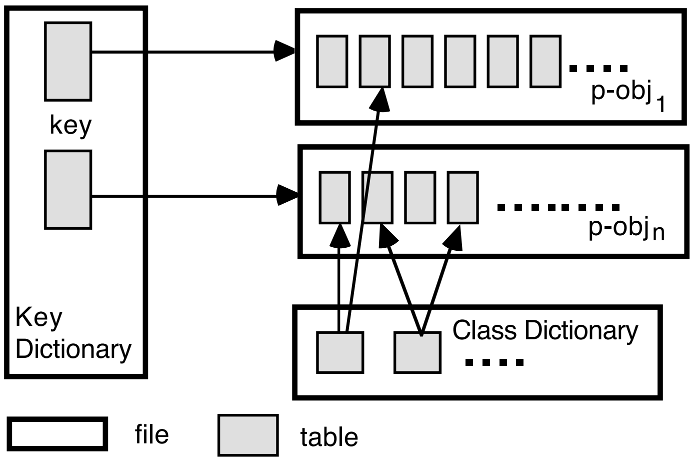

Figure 4. The Key Dictionary and the related structures.

The generic key contains the name of the associated persistent object, a list of the file types (and so a list of the applications that can access to that object), owner, protections and a link to the first table of the object. Keys are grouped within a file called Key Dictionary (see figure 4). The Key Dictionary contains the key of each persistent object within the data base and so a link to the corresponding file: the tables that define a persistent object are indeed stored within a file, called object file. On the other hand for each class there is a table within the Class Dictionary file (see figure 4) that contains dynamically allocated tables that allow an easy and fast access to the files belonging to a given class (access by type).

At the conceptual level (conceptual scheme) data are structured to form persistent objects, i.e. list of files and tables, each list being addressed by a key. At the view level (view scheme), users and applications can see either persistent objects or their dynamically created subsets. Such subsets are called volatile objects since they have no permanently stored counterpart. A volatile object can be composed by files either of the same persistent objects or of distinct persistent objects. Usually users require the direct access to both types of objects whereas an application can access only to certain file types, according to its methods, and so can access only to volatile objects.

The mapping of the conceptual scheme over the physical scheme is a duty of the DBM whereas the mapping of the view scheme over the conceptual one is performed by the DSM that also manages a set of tables of volatile objects (see vot, volatile objects table, figure 3): each of these tables can be used by the users for a fast and easy access to frequently used data.

The programs

As to the programs, the data base is characterised by a set of programs for its administration that interact either with other applications, with the users or with the Operating System, whose task is providing the most basic services on which the data base is built (for instance the file system).

Such programs are the DBM, the DSM and the DBI (see figure 5). The DBM and DSM can even define a distributed data base whereas every user and application can interact with the data base only through the DBI.

As to the single programs, the DBM manages the stored structures, the DSM performs the mapping of persistent objects over volatile ones whereas the DBI interacts with the applications and with the users.

So to enter in details, the DBM takes care of the definition and management of both the data base data structures (data dictionary, class dictionary, key dictionary, object files and the like) and the persistent objects. Beyond such tasks, the DBM performs both consistency check (it verifies that the objects are stored just as they are declared within the structures of the data base) and integrity management: it controls that the files that contain the data structures and definitions are managed only by authorised users and through the DBI. The latter task requires a protection of the basic structures of the data base, of the files within the objects and a control of the concurrent accesses to the data base itself: such a control depends on the nature of the requesting application and can be implemented even with object or file locking. The protection of the files aims at allowing file access only through the DBI so to enforce the data base consistency.

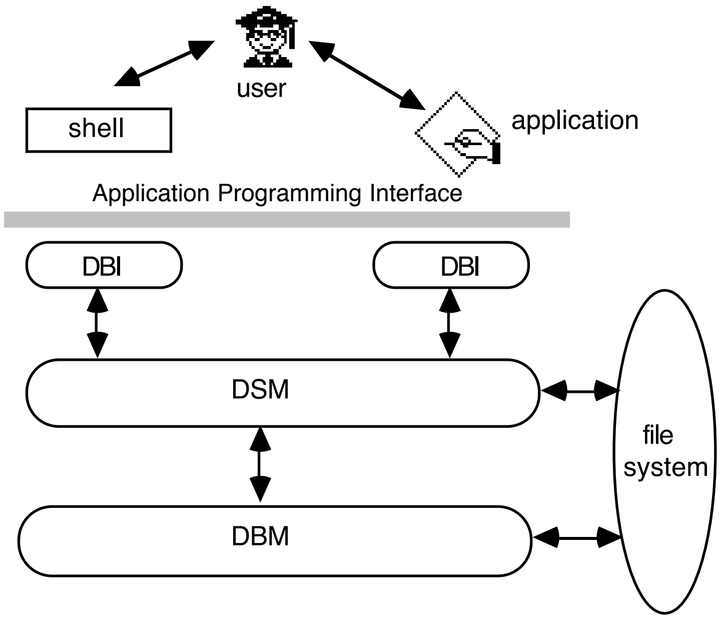

Figure 5. The programs for the data base administration.

The DSM performs a scheme translation and manages tables of volatile objects. Such tables contain references to files that belong to the same or to distinct objects and can be accessed by the same application. In a multitasking environment, the switching among the applications therefore requires the dynamic redefinition of such tables.

The aforesaid tables can be even stored by name in proper permanent structures so to allow a fast and easy access to frequently used data.

Last but not least, the DBI manages the access both through a shell and through one of the available applications. Its main tasks are the provision of a non procedural DML (so to define an uniform Application Programming Interface that enhances portability) and the translation of DML queries into commands for the access to volatile/persistent objects. Users can invoke DML commands either as embedded within the applications or through the command language provided by the shell.

The presence of the shell allows indeed the adoption of either a textual or a graphic user interface, respectively TUI and GUI. Depending on the nature of the user interface, the command language provided to the users varies. Anyway the shell allows an access to the objects and presents the applications as commands (TUI) or icons (GUI). The access to the objects requires a control to the ownership and access permissions to objects and files and includes both the visualisation and the editing of the structures of volatile/persistent objects. Editing operations allows the insertion of files (for instance those produced by stand-alone applications) within objects and the correct deletion of files and objects. The process of file deletion requires the deletion of the associated table and the updating of the persistent object to which the files belonged and of every table in which it was referenced. Similar considerations hold for the objects too. In such a case the persistent object is deleted as a whole whereas the deletion of a volatile object requires only the deletion of the associated table.

Conclusions

The present paper contains a brief description of a data base for the storage of data for SSP. Such data are characterised by an even short life cycle since they are created, processed, visualised and deleted very frequently.

The presence of the data base aims at the definition of an environment in which data are temporarily stored so to be correlated and accessed either by name, by type or by even more complex patterns.

The presence of the programs for data base management should guarantee both consistency and integrity of such data.

Future plans include both a deeper investigation of the potentialities of the Object Oriented paradigm and a feasibility study for the implementation of a prototype of the proposed environment over a Dec Alpha.

Bibliography

Bracchi, G. et al. (1980) *Sistemi per la gestione di basi di dati,* (Mondadori: Milano).

Bray, O. H. (1983) *Distributed Database Management Systems*, (Lexington Books: Lexington).

Ceri, S., Katsumi, T. & Shalom, T., eds., (1993) *Deductive and Object-Oriented Databases*, (Springer-Verlag: Phoenix).

Cioni, L. (1992) *A complex data-base for speech signal processing*, 3rd ERCIM Database Research Group Workshop on Updates and Constraints Handling in Advanced Database Systems, Pisa, Italy, 28-30 September 1992.

Cioni, L. (1992) *An Environment For Speech Signal Processing*, 4th Australian International Conference on Speech Science and Technology, Brisbane, Australia, 30 November-3 December 1992.

Fallside, F. & Woods, W. A. (1985) *Computer Speech Processing*, (Prentice Hall: Englewood Cliffs).

Gjessing, S. & Nygaard, K., eds., (1988) *ECOOP '88. European Conference on Object-Oriented Programming*, (Springer-Verlag: Oslo).

Glücksmann, R. (1984) *Il data base nel sistema informativo*, (ETAS: Milano).

Korth, H. F. & Silberschatz, A. (1991) *Database System Concepts,* (Mc Graw Hill: Singapore).