AN ENVIRONMENT FOR SPEECH SIGNAL PROCESSING

Lorenzo Cioni*†

*Laboratorio di Linguistica, Scuola Normale Superiore

†Facoltà di Scienze dell'Informazione, Università degli Studi di Pisa

ABSTRACT - The aim of this paper is to describe a project that we are developing at our Laboratory. This project aims at the definition of an environment for speech signal processing in which a set of applications can co-operate through a library of user-defined data that represent either "the outcome of" or "the source for" such applications.

INTRODUCTION

The aim of this paper is to describe a project that we are developing at our Laboratory and whose scope is the definition of an environment for speech signal processing. We called it Mod-lab.
With speech signal processing we mean both analysis and synthesis of speech: analysis of sampled data files and synthesis from both text files and data deriving from analysis.
The environment is composed by a set of concurrent applications that can co-operate among themselves through a library of user-defined data (also data library in what follows) that represent either "the outcome of" or "the source for" such applications. This frame represents the logical architecture of the environment. Each application within the environment is devoted to a particular task and is characterized by a set of operations through which an user can handle the data contained within the data library. These operations allow the switching among the applications and enable the applications themselves to be linked together so that the data of a certain type can be processed in a sort of pipeline. In any case, each application must be as self-contained as possible and, therefore, must be characterized by a coherent set of operations. The applications within Mod-lab perform operations such as speech acquisition, disk storage and retrieval, playback, speech display and editing, speech analysis and synthesis, statistical analysis of data and so on.
Among the applications the lab-interface and the manager are two shells that simplify the interaction with any application and with the library of data, independently from their physical localization.
The data library is composed by a set of objects (basic logic entities) and is directly managed by the manager. The user, either through the manager or within an application, can create a new object (new) or open an existing one (open) so as to to manipulate it with one fitting application. The user can even select the icon of an existing object: the control passes to the manager that lets the user choose one of the applications that can handle that object.
An object contains a set of correlated but non homogeneous data that are the result of either acquisition (speech signals) or synthesis (synthesis signals) and of the subsequent processing of these data. An object is implemented as a linked list by using both files, that contain the data, and tables, that contain information about each file and about the object as a whole.
Every application can access a set of objects and, for each object, to a set of files while a certain object can be handled by a set of applications, depending on the type of data it contains. Every application, in its turn, can produce new data that enrich the object itself and that can be handled by some other application as well. Moreover, the objects are logically linked together, since distinct objects can contain data of the same type and, therefore, an application can act over a set of objects either to perform, for instance, comparisons or to establish correlations or to perform statistical operations and so on.

THE ARCHITECTURE OF THE ENVIRONMENT

The proposed environment is characterized by an underlying potentially distributed physical architecture composed by one or more computing systems linked together in some way (cf. figure 1). The Mod-lab is mapped over this physical architecture so that, within the environment, the composing elements (i.e. applications and objects) have the same behavior independently from their physical localization. With regard to this physical architecture we have that, from the system point of view, the

environment is characterized by two sets of elements: objects and applications (cf. figure 1). Each set is composed by elements that can be either local (i.e. that are localized on the physical system through which the user interacts with the environment, the so called Access Point or AP) or not (i.e. that are localized on a distinct physical system that is linked in some way to the AP). Objects and applications are told to be local only in relation to an AP and, therefore, in relation to a particular user that accesses the Mod-lab. In any case an AP must, at least, hold the application that defines the interface between an user and the Mod-lab (the Lab-Interface, cf. next section and figure 1) and can contain other local applications and even local objects, even if this is not necessarily the case. The Lab-Interface allows the user to interact directly with any application and, in an indirect way, with any object since, through it, the user can access the manager (cf. next paragraph).
With regard to the application whose main task is the management of the objects (the Manager, cf. next section and figure 1) it can be designed as a distributed application and, therefore, it may be seen as composed by a set of co-operating modules, each module being localized both on every AP (local or icon-bound modules) and on every other physical system that can contain objects (hidden modules). Each local module that stays over an AP is, indeed, associated to an icon (cf. figure 1) and can be accessed even through the Lab-Interface. Its task is the management of the local objects as a stand-alone module and of the non local ones in co-operation with the other hidden modules.
From the user point of view, on the other side, the architecture of the environment is quite simple and plain since every user accesses the physical architecture but he/she only knows something, often not everything, about the logical architecture of the system. Each user, indeed, accesses the environment through an AP (and so through a personal computer or a system like that) and directly interacts with either the icon of the Lab-Interface or the icon of a local object or the icon of a local application or the icon of the local module of the manager (cf. figure 1) to perform his/her own data processing but, in the same easy way, can as well interact even with non local objects and applications (cf. figure 1). One of the main goals of the proposed architecture is, indeed, that of freeing the users from the burden of data management so that they can concentrate only on data processing. Further details will be given in the next sections.

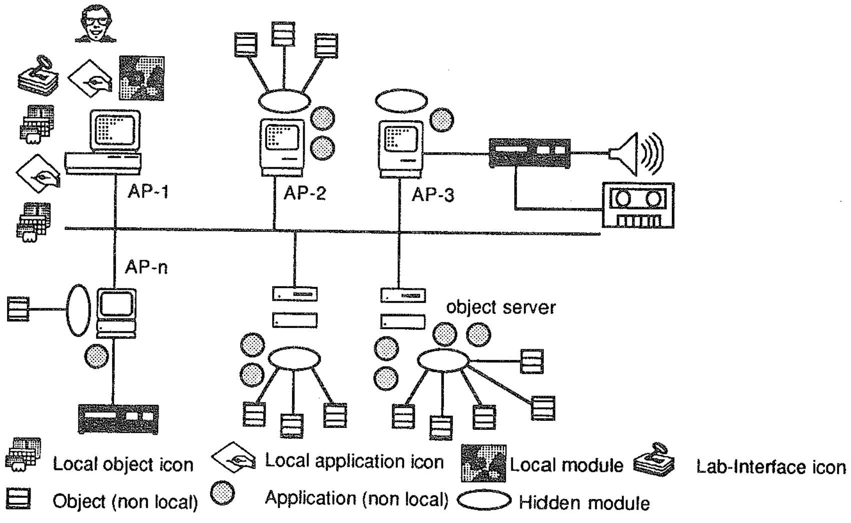

Figure 1. The physical architecture of the environment

CO-OPERATING APPLICATIONS

General framework

The Mod-lab is, therefore, characterized by a set of applications. This set can be subdivided either as before on the basis of a criterion of localization (local applications vs non local ones) or on the basis of the set of the characteristic operations of each application. Since, for our aims, the first criterion is somewhat meaningless, for reasons of symmetry, we are going to use the other one. According to this criterion, every application of the set is devoted to a particular task and is characterized by a set of operations through which an user can handle the data contained within the data library. The operations within this set are told to be inner operations if they are performed over the data by the current application while are told to be outer operations if they are performed over the data by a distinct application. The inner operations characterize each application by themselves and are available to the user even if the application is executed in stand-alone mode and so outside the Mod-lab (this behavior can be selected by the user through the Lab-Interface by means of an inner operation of the Lab-Interface itself). It is obvious that in stand-alone mode the outer operations of a generic application cannot be executed by the user. Through the outer operations, indeed, the user can carry out the switching from one application to another one (either with or without control passing) so that both applications can work on the same data (object passing) or not. In this way the applications themselves are linked together so that the data of a certain type can be processed in a sort of pipeline.
If the application switching occurs with control passing we have that the new application takes control and the user interacts directly with it, its own operations and its own user interface.
If, on the other side, the application switching occurs without control passing, the new application executes the requested operation over the current object and, when the execution is over, the calling application resumes control.
It the case of control passing, the new application works on the current object (object passing) but, obviously, can work even on different objects while, in the other case, it is forced to work only on the current object. The application switching mechanism gives us a way through which we can implement the outer operations of each application. To act in this way, within the set of the applications, some of them will be designed so that they can execute in background mode. In this way we can obtain an application switching without control passing. With this design methodology we obtain a multitasking environment that is able to manage both complex and persisting entities, such as the objects, and the switching among the applications.

The Lab-Interface and the Manager

The main task of the Lab-interface is the management of the interactions between the user and every application the user can need. The Lab-Interface allows the user to interact directly with the applications and, therefore, even with the manager itself and, through the manager, with the data within the data library. From this point of view, the Lab-Interface is simply a sort of shell that allows the user to interact easily even with non local applications i.e. without taking care of the physical localization of the application itself, but this does not keep the user from directly interacting with a local application through its icon (cf. figure 1). The Lab-interface is characterized by a set of inner operations and a set of outer operations through which it can pass the control to one of the applications. The inner operations allows an user both to customize the set of applications (but the manager) to which he/she can access through the AP and to access directly the manager through its local module. This local module holds the list of the addresses of the local objects and the addresses of every other hidden module of the manager to which it can ask for the list of the addresses of the objects that are local to each hidden module.
With regard to the interaction between the user and the applications we have that an user can select an application directly through its icon (every application has a corresponding icon) or through the Lab-Interface (the user can switch to one of the applications or to the manager). The first way is suitable only for local applications that can be executed even in stand-alone mode while the other one is suitable for every kind of application. In the first case the user opens the application and work with it but indirectly interacts with the local module of the manager so that the application is opened together with the set of local objects it can handle. This allows a straightforward access from that application to the local objects.

In the second case the Lab-Interface shows to the user the set of the actually active applications and let him/her either to open one of them or to tailor this applications set according to his/her own needs. As an example, if an user is interested only in display and editing of speech signals he/she can define his/her own set of active applications as composed only by those applications that perform those operations, discarding every other application.
We note that when the user is working with an application he/she can choose either to close it and so the Lab-Interface resumes control or to switch to another application among those that can be reached by the current application (control passing).
The manager, whose structure has been shortly sketched in one of the precedings paragraphs, is surely the most complex application within the Mod-lab since it is composed by a set of co-operating modules that can be physically distributed over both the set of APs and the set of "object servers" (cf. figure 1 and next paragraph). If we look at the environment from an AP, we have a local module and a set of hidden ones. The local module is the one running on that AP while the hidden modules are those running either on the other APs or even on some dedicated systems, the so called "object servers", whose main task is the permanent storage of objects but that can execute even some heavy CPU-bound application. We note, therefore, that the distinction is somewhat an arbitrary one because it is linked to a point of view and so, from now on, we are going to consider the set of modules as a whole and call it the manager, unless otherwise stated.
The manager, like the Lab-Interface, acts as a sort of shell that allows an easy access to the objects independently from their physical localization. The main task of the manager is the management of the direct interactions between both the user and the applications and the library of data. It manages the creation of the new objects and the passing of the either new or existing objects to the applications and, moreover, it puts at the user's disposal a set of operations for the direct handling of the objects. These operations are the so-called manager's inner operations. They allow the user either to create new objects or to open an existing object with one of the available applications or to delete any existing object or to examine the inner structure of any object, by listing every available information about its composing elements. The management of the objects and of the various kinds of interactions compels the manager to maintain a set of tables that allow the run-time management of the connections both among the objects and the applications and of the applications among themselves. This run-time tables are localized within every module of the manager and, together with a table that contains the physical addresses of every other module of the manager and some status information (such as if that node can be reached or not and so on), allow the fulfilment of requests in relation to non local objects or applications. Some of the tables are both the one that contains the addresses of the objects (ToO) and the one that contains the addresses of the applications (ToA) and the one that contains, for each application, information about which are the applications that implement its outer operations. The ToO sets up a link between the objects and the applications and, besides a set of status information, contains the physical address of every local object and of every non local object that has already been accessed through a local application. The ToA contains the physical address of every active application for a given user and, for every application, a set of pointers to the entries of the ToO that refers to objects that can handled by a particular application. The ToR, at last, contains the cross references (or rules) among the applications i.e. for each application which are the applications that implement its outer operations.

THE DATA LIBRARY

General framework

We can define the data library together with the manager and the others programs to handle the data as a data base management system (DBMS). Within this DBMS we have a distributed manager and a set of dedicated applications together with a distributed data library . Since the logical architecture of the environment must be able to hide, as much as possible, to the users the physical localization of the objects themselves in this section we put aside the distinction between local and non local objects.
The basic logic entity within the data library is the so called object while the basic physical entities are the file and the table. From a logical point of view (cf. figure 2) an object can be defined as an even empty subset of the set of data contained within the data library: it contains a set of correlated but non homogeneous data that are the result of either acquisition (speech signals) or synthesis (synthesis signals) and of the subsequent processing (with the more suitable application for a certain task) of

these signals such as F0, formants, statistical data and so on.
The users can create new objects or they can process the existing objects by using one of the available applications, the more suitable for a given task, and this processing can involve more than one application, even a set of applications linked together to form a pipeline.

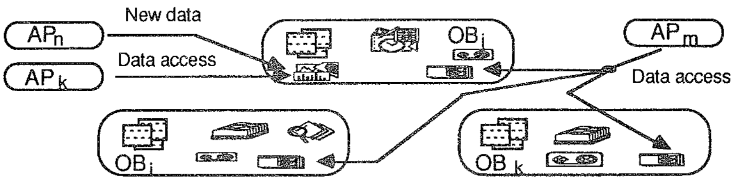

Figure 2. The data library and the data access

Every application of the set can have access to a set of objects and, for each object, to a set of files: a certain object, indeed, can be handled by one or more applications depending on the type of data it contains. For each object, on the other side, each of the fitting applications can produce new data that enrich the object itself and that can be handled by some other application as well. In this way we have that the objects are logically linked together, since distinct objects can contain data of the same type and, therefore, an application can act over a set of objects either to perform, for instance, comparisons or to establish correlations or to perform statistical operations and so on.
We, therefore, have a data library as composed by a set of objects that are directly managed by the manager. The users can handle the objects through the manager in order either to create a new object (command new) or to open an existing one (command open) or to handle an object with the manager's inner operations. An open or new command can be issued both within an application and within the manager. With regard to the new command, in the first case it causes the manager to create a new empty object and pass it to the calling application while in the second case the manager creates a new empty object and adds it in the set of the existing objects for a future usage.
In any case, the manager limits itself to the definition of the framework of the new object while the other applications (but the LI) have the task of filling the object itself with the actual data.
With regard to the open command, in the first case it causes the manager to open a certain object and pass it to the calling application while in the second case the manager lets the user choose at first among the existing objects and, after, among the applications that can handle a particular object.
The same things happen if the user selects the icon of an existing local object: the selection of an icon passes control to the manager that allows the user to choose one of the applications among those that can handle that particular object. In any case, the act of opening a particular object allows the user to manipulate it with one of the applications that can handle it.

The Objects

An object is implemented as a linked list of both tables and files: the files contain the data while the tables contain information about each file and about the object as a complex entity (cf. figure 3).
We have indeed one table that identify the object itself (the so called descriptor) and a table for each file within the object. An empty or new object consists of the only descriptor. The descriptor is therefore allocated whenever a new object is created and contains the physical address of the object as it is known to the module of the manager to which it is a local object. The global address of an object is composed by this local address and by the address of the module. In this way we have an one-to-one reference to the object. If the object is stored on an AP to the descriptor is also linked the icon of the object: in this case an open command through the icon passes the descriptor to the application through which the user want to handle the object itself. Each of the other tables contains both a pointer to the associated file and a set of information about what are the applications that can handle that file. Every application can access to an object through its global address and, for each object, to a set of its files depending on the information contained in each of the table that is associated to these files. We note that the environment contains applications that produce data for the objects (acquisition

and synthesis) and therefore the set of applications is a consistent set.
An application can, in any case, create both new objects and new files in existing objects simply interacting with the right module of the manager.

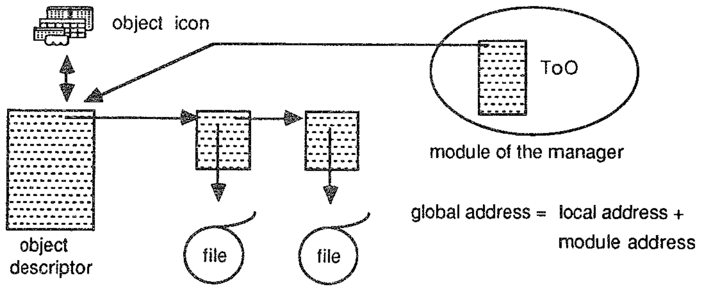

Figure 3. The physical structure of an object

In this way we have a set of physical lists of tables, each table referring a file and each physical list forming an object, and a set of logical lists of tables: each logical list contains the tables that refer to files that can be handled by the same application.
Every application, depending on the type of data it can handle, defines such logical lists according to the set of objects the manager passed to it and to the information contained within the individual objects.

CONCLUSIONS

The project shortly sketched in the present paper started at the end of 1991 and therefore it is yet in its infancy. Nowadays we are developing both the inner part of the architecture and an Hypercard demo. For what concerns the inner part of the architecture we are developing both the mechanism of application switching and that of object management. When this phase will be over we are planning to design the main applications for speech processing with their own user-interfaces in order to obtain the first working prototype, with a limited set of applications and on a single Macintosh, to be used as a basis for further developments. This environment is going to be embedded in a Macintosh based working environment but it could be embedded as well in any other multitasking environment with a window-like user interface. We chose Macintosh for its low cost, its easy-to-use user interface and its wide diffusion together with the availability of a lot of boards well suitable for speech signal processing.

REFERENCES

Agonigi, M. & Cioni, L. (1992) *Design of a Macintosh application for speech signal processing*, (Third Eurographics Workshop on Visualization in Scientific Computing, Viareggio, Italy, 27-29 April 1992).

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).

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

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