Modelling speech knowledges in a distributed complex object oriented architecture: REMORA

T. MARTELLI

Université de PARIS V - EHEI - 45 rue des Saints Pères 75006 PARIS

ABSTRACT.
This article presents the formal bases of our complex object oriented language, together with our distributed architecture, originally developed for a speech recognition application, generalized as a tool that can be used regardless its field of application to conceive evolutive multi-expert bases of complex knowledge.
The originality in this system lies in the definition of a complex object oriented language CODEX (Complex Object Dynamic EXpert language). We have defined a complex object to represent a complex knowledge, that can be dynamically modified, is composed of parts and that eventually has temporal characteristics. This representation grants standardization, so that within the same system declarative and procedural knowledge derived from different experts can be brought into collaboration. Our system takes into account temporal aspects of reasoning and of knowledge representation. Standardization of knowledge and distributed control forecast an extension to our knowledge base system of Distributed Artificial Intelligence.

I Introduction
It is a fact that in many fields, collaboration between sources of diversified knowledge has proved to be necessary. Numerous fields of application require the combination of expert knowledge or self organized methods to handle symbolic or numerical data. Moreover, in complex application such as recognition of continuous speech or automatic diagnosis, where different sources of data enter into action, it would be interesting either to compare several methods, simply to put in competition different expertise or to discover new knowledge.
The evolution towards Distributed Artificial Intelligence calling out multiple expertise, within the same system but also located in distant places has become a real necessity. For an easy and significant evaluation of resolution methods and expertises, it is important to have access into a larger base of knowledge. What really counts here is to conceive architectural models making possible the interaction of multiple source of knowledge, that eventually exchange information via one or more several transmission networks. The increasing search for the conviviality of a tool for the expert has led to the development of program interface using graphic tools thanks to object oriented programmation. It would be advisable to foresee simultaneously several formalisms for representing knowledge (such as logicales predicates, programs or production rules) allowing at the same time knowledge interaction. Furthermore, the representation mode for knowledge must consider the dynamic evolution throughout the time and during the reasoning process.

Firstly, we have build, specifically within the framework of an application to decoding of continuous speech, an operative distributed architecture in order to supply the expert with a real base of evolutive knowledge as the one we have already defined for our application (Martelli 87a). In other words, this knowledge base provides the experts with a set of knowledge and tools to explore a knowledge that, by its interaction generates new knowledge. Requirements for the conception of new knowledge, to test them and to work together have therefore lead to a definition of a work station and a knowledge base included in an architecture of Distributed Artificial Intelligence compatible with automatic learning mechanisms. Our distributed architecture allows automatic selection and collaboration of knowledge.

Secondly, as the representation of diverse knowledge categories within the same formalism turned out to be necessary to the set of an effective communication mechanism, we elaborated a standardized representation, considering knowledge as objects and specifying for this a complex object language CODEX (COnplex Dynamic EExpert object oriented language) (Martelli 88). Setting up mechanisms of standardization, we open a way towards an architecture of multi-sites distributed artificial intelligence.

II The choice of object formalism.
The study of different tools used for expertise extraction leads to mix several approaches. This observation can be applied to other fields of expertise. To take into account the expertise, it is necessary either to extract it a priori and build a model to represent it, or to produce structures permitting the experts a gradual description of the basic vocabulary he uses for the expertise. This can be made in various formalisms, but only the object formalism allows a modelisation of within the same concept, that is, the object having both declarative and procedural aspects, translated in terms of either attribute or method of an object (Cointe 85).
Our Complex Object Dynamic Expert Language CODEX

III.1 Description.

CODEX is designed to represent complex knowledge. In our application, complex knowledge are for instance, either signals and their interpretation, which involve numerical and symbolic aspects, or expert knowledge and algorithms designed for signal processing and pattern recognition within a multi-expert context (Martelli 88).

We generalize the concept of complex knowledge extending knowledge that can be composed by different parts, have a temporal dimension and is subject to dynamic modification. Knowledge can be automatically transformed or created and becomes a building tool for future knowledge. For its representation, we define original and specific tools proper to our formalism, as indicated below:

Complex object concept.

A complex object generalizes the notion of class and method, is constituted by complex attributes and complex methods and communicates through complex messages.

- A complex attribute has meta-selectors, which set specific mechanisms when accessing the attribute's value, respectively:
  - dom (domain), elem (element), and ref (referential).
  - Ref ensures the compatibility between information coming from different experts and allows standardization. Dom indicates either a type (integer, real ...) or the class of the value of the attribute. Elem reflects the composition into parts.
  - Named attributes are distinct attributes which refer to one of the main object modules, which compose our system's architecture. As they are considered in a generic way, they constitute a formal message receiver, which will be dynamically instantiated.

- A complex method is considered in itself as an object, which allows us to unify the class and the method concept setting up an inheritance mechanism at the method level. Thus a method is formally defined as a series of messages and dynamically instanciated through the selection message as a tool object. We give to this mechanism the name of unlinked procedural attachment.

Complex messages are specific classes: composition or selection or referenced messages, they offer tools to secure the compatibility of the exchanged information.

- A composition message permits to distribute the action described in the message throughout the whole or a part of the object components.
- A selection message performs the dynamic choice of a receiver to which the execution of the requested action is delegated.
- A referenced message makes possible the compatibility of information exchanged between sender and receiver through automatic conversion. If needed, the referenced message assumes conversion into the standard of a reference system.

Complex objects are composed by parts, that can be also objects; that is why we defined them in a recursive way. This structure establishes a search mechanism of complex objects specific to our system. Search is carried out by 'dom' and 'elem' meta-selectors.

In the case of objects of an significant size, the 'elem' meta-selector facilitates direct access to elements which are themselves considered as objects only if necessary, through the composition message class.

III.2 Example.

Let us consider an example of two objects linked by a referenced message. For two given objects Pierre (instance of man) and clock1 (instance of clock), we have a series of transmissions, including mes1 which result is the assignment of the value 21 to the attribute age. Mes1 is used in the 'age?' method:

```
(mesref year clock1 age=ref).
```

If we want to know the age of Pierre, the required value is automatically computed by using the particular method 'age?'. In fact for objects as simples as these, we must use a reference system for fully describe a given attribute. For example, the value of the attribute 'date' for the clock or year-of-birth can either be expressed in year or in months. The referenced message insures the compatibility of the operators to be subtract and assumes their conversion into the same reference system.

III The architecture of our system.

III.1 Description.

In order to realize the objectives outlined above we have built a distributed architecture characterized by a distributed mode of control and which offers the possibility of standardizing knowledge and reasoning. The originality of our approach lies in the possibility of describing the goals to be carried out in the system as a category of objects, allowing the sharing of problem solving strategies, which are considered as methods associated with goal objects. Since the attributes of an object can themselves be considered as objects, the domain meta-selector may be used to indicate a class of objects and the value of the attribute must be an
instance either of the class itself or of one of its subclasses. This allows goal generic classes such as improvement, comparison or combination to be defined. It also permits us to inherit, for each execution of a given goal, a certain number of properties, since the goal classes obtain their actual values according to the object classes manipulated.

Furthermore, the possibility of considering the execution of a goal as the result of an 'execute' method of the goal object class allows an execution strategy to be defined by default and to change strategies by simply redefining a method.

The comparison of different strategies is thus rendered possible by indicating at the method level the execution mode in conformity with the desires of the user. The 'goal' knowledge category, which is specific to our system, is very important because it is by means of the methods attached to it that the various knowledge evaluation strategies can be tested and the learning techniques applied.

The architecture is composed of five modules according to the descriptive needs of the user, in order to help him to input his knowledge and to allow him to create his own objects more easily by reusing the whole or part of the objects already in the system. These modules are also meant to ensure that the specified formalism is suited to the knowledge to be represented. For each of them, the knowledge structure is given prominence by means of a hierarchical tree showing the state of the various object classes (Martelli 88). The modules are:

- Initial data with which the analysis is carried out.
- Goals, to be executed.
- Entities are sets of the expert knowledge or the results of analysis.
- Programs represented as objects and written in various programming languages (Fortran, Pascal, Lisp, ...).
- Rules as packages or sets of rules, are attached to another knowledge category to avoid combinatorial explosion.

From a functional point of view, the system internally restructures the knowledge into three functional categories: goals, entities (initial data or entity) and tools (rules or programs). The tools are methods dynamically associated with other object categories, they build a set of reasoning. The linking up of one or several objects of these two categories to data (initial data or entity) or to goals permits either data driven or goal driven processing.

A main point for the evolution from Data Bases to Knowledge Bases is the ability to take into account the dynamic aspects. The main condition of such an improvement is to offer the users the ability to share programs in the same way as an ordinary data. In our modelisation one program is represented as one object. We consider that a programmer has an expertise on his own programs (implicit knowledge of program designer, possibility of improvement, description of program's results). A program must be reused by others users. An instance of a program object is associated to an execution of the program and dynamically linked either to a data or a goal. The user's program is here considered as a whole, in the opposite of the representation of a program as several basic objects used by object oriented interpreters for program analysis and instrumentation (Lieberman 87).

III-2 Control.

The distributed control is insured:

Internally by means of methods and of certain selectors, which represent the own control of the object.

Externally by means of messages which harmoniously organize the interaction between the various objects. The objects which receive the message are then selected in accordance with:

- The domain of the required objects: for example of rule triggering, the rule is a particular instance of a certain class of rules characterized by the type of objects it can match in order to satisfy its premise. It contains therefore its own control mechanism.
- The condition specified in the message which is dependent on a named attribute.

The messages are propagated from one to another by means of selectors or by the use of unlinked procedural attachment defined above, which allows dynamic selection either of the receiver object or of a method as well as delegation to other objects.

The objects are thus linked one to another by a dynamic graph. Formal links are established through named attributes: for a given named attribute of an object, the 'domain' contains a class of another module. The value of the named attribute becomes an instance of the class indicated by the domains (or of one of its subclasses), when the link is instantiated, in response to an occurrence either directly or by the fulfilling of predetermined conditions yielded by the selected message. The triggering of the execution is instigated by the initial message. When the message is received, it is interpreted by an object in its execution context and the object then retransmits messages to other objects either by reflex, by continuation (a new receiver is specified), or by conditional propagation (through selection messages). The system can create objects dynamically, for example by applying a tool object to an object entity, another object entity can be dynamically created, the latter being considered as the dynamic transformation of the initial object entity.

III-3 Execution example.

We want to obtain, as shown below, starting from a signal, a phonetic lattice using different tools in order to compare it. We can see instances of program object (P1, P2...), producing result object which are classified through the level of signal analysis performed: events (class 1), phonetic ground class (class 2) or lattice (class 3).

P3 is an example of global algorithm. The rules are not global, their are always attached to an another knowledge, for instance they are attached to P5, and are able to improve the results of P5. When a speech signal object is created (interpreted in terms of
object into the system), an analysis method is formally defined consisting of a series of messages designed to impose constraints on the attributes of the future convenient program objects. F0 is an instance of a signal object.

To satisfy the goal "compare", the system can trigger subgoals as well as the two given subgoals "combine" and "improve". Furthermore, under the user's desire the system can try the whole or part of the paths in the graph which lead to the expected result. This constitutes an example of exchanging strategies by simply redefining the execution method of the goal 'compare'. We can see the interconnection of various subgoals: for example, in order to obtain 'class 3' results, object program like P3 (which performs analysis completely) or a series of programs must be chosen. When an object program is being described, links are established between the output result class and the input data class with entity class belonging to a given level of analysis, by means of the named attribute. The specified result class can be either directly the result of the program or a more general class belonging to the same hierarchical tree. This program can be dynamically selected by matching its class of results with the class expected by the application of the goal.

For example for the signal F0, the system selects dynamically into the program knowledge module a (or a series of) convenient program that satisfies the selected message included both into the method 'analyse' of the signal object and into the goal or subgoal. It is done by propagation. Let us suppose the program P1 has been selected, it is linked to the 'class 2' as result class and to 'class 1' as data class, if we want to perform analysis starting from the signal instance F0 to obtain the result instance 'R2', if R2 is in the hierarchical tree of class 2 and satisfies condition propagated by P1 or the corresponding subgoal, an actual link is established between this three instances which becomes the values of the corresponding named attributes.

III 4 Temporal aspects.

Certain objects have the characteristic of being composed of elements which are temporal intervals. In the sense of Interval Temporal Logic (Manna 83). We have defined the temporal interval class (int) as the root class of temporal objects. We shall use two of these sub classes, namely segment and boundary.

IV Conclusion

We have presented the main characteristics of our system which is a tool for the acquisition of multi expert knowledge. CODEX, our complex object language written in Le_Lisp enables us to generalize the concept of class and method and to take into consideration the dynamic behaviour of the objects. The objectives we set ourselves were to be able to combine different knowledge and to make them work together within a user-friendly environment. These objectives have been the guiding principles for the development of our distributed architecture. In addition, various knowledge representation formalisms are possible (programs, production rules, temporal predicates). Our system is a work-station for the collaboration and production of new knowledge, in the meaning that the knowledge just created in turn becomes the construction tool for future knowledge. Building a distributed architecture as well as establishing the mechanism for the standardization of knowledge allows us to extend our system towards a multi-site distributed Knowledge Base.

The interest for the Knowledge Base Area is to allow a framework for the user to describe his knowledge as complex objects. The user is helped by the shell provided by the five knowledge module and their basic objects. We allow programs as well as complex knowledge (dynamic, temporal, composed by parts) to be part of the Knowledge Base. We assume the handling of complex knowledge by the means of our complex object language CODEX. The evolution towards Intelligent Knowledge Base is done by allowing the user to specify messages and goals objects as a complex query language.