TR-IT-0012

Integration of Heterogeneous Components for Speech Translation: the "Whiteboard" Architecture and an Architectural Prototype

Christian Boitet

1993.8.30

A new software architecture for Speech Translation systems, based on the use of a "whiteboard", is presented. Unlike a blackboard, the "whiteboard" is accessed only by a "coordinator", and not by the "components", such as speech recognizer or syntatic analyzer. The main advantage of this architecture is to allow easy integration of existing or new components, without having to modify them in any way. Graphic examination of the state of the entire system can also be provided by the coordinator. An "architectural" prototype is under construction ans should be presented at the ATR workshop and the Waseda Symposium on Spoken Dialogue next November.

This report is an extension of a recent communication co-authored with M. Seligman [1]. Extensions concern essentially the rationalization of the kind of data structure proposed for the whiteboard, for which diagrams and examples from [2] have been used.

[1] Seligman, M. & Boitet, Ch. (1993) *A "Whiteboard" Architecture for Automatic Speech Translation.* Proc. International Symposium on Spoken Dialogue, Waseda University, Tokyo, 10–12 November 1993, to appear.

[2] Boitet, Ch. (1988): *Representation and Computation of Units of Translation for Machine Interpretation of Spoken Texts.* TR-I-0035 ATR, Osaka, 41 p.

The research reported here was conducted while I was staying at ATR Interpreting Telephony Research Laboratories, and then at ATR Interpreting Telecommunications Research Laboratories, as visiting researcher from GETA, IMAG, UJF&CNRS, France. I would like to heartly thank ATR for its constant support and very favorable research environment; and its members, from President through supervisors through researchers through secretaries to security personal, for all the personal help which they extended to me in so many ways at so many occasions.

Interpreting Telecommunications Research Laboratories
2–2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

Contents

Abstract

In the design of speech translation systems, there is room between highly complex blackboard architectures, where all component processes access the same unique data structure, and overly simple sequential architectures. We propose to allow component processes complete freedom of implementation, while at the same time integrating them under a coordinator, maintaining in a *whiteboard* an image of the input and output data structures of each component, at an appropriate level of detail.

Four main guidelines for building such systems are presented: (1) record overall progress of components in a whiteboard; (2) let a coordinator schedule the work of components; (3) encapsulate components in managers; and (4) use the managers to simulate incremental processing. Then, KASUGA, a rudimentary architectural prototype, is described. Its coordinator is written in KEE, an object-oriented expert system shell, and integrates only three components. It should be stressed that KASUGA is a feasibility study and demonstration, not an operational system.

Keywords: Speech Translation, Whiteboard, Incremental Processing in Translation.

Introduction

Speech translation systems must integrate heterogeneous components handling speech recognition, machine translation and speech synthesis. More components may be added in the future, for task understanding, multimodal interaction, etc.

If components are simply concatenated, as in ATR's ASURA speech translation system [12, 13], it is difficult for system components to share partial results. As a result, information is lost at subsystem interfaces and work has to be duplicated.

For example, ASURA uses context-free LR syntactic parsing to drive speech recognition; but syntactic structures found during the recognition parse are discarded when recognition candidates are passed to machine translation. Complete reparsing is thus needed.

Communication difficulties between subsystems may also damage robustness. During reparsing for MT in ASURA, if no well-formed sentences are found, partial syntactic structures are discarded before semantic analysis; thus there is no chance to translate partially, or to use semantic information to complete the parse.

On the other hand, experiments with blackboard architectures [8, 15, 24], in which each component directly manipulates the common blackboard, have not been encouraging. There are problems in controlling concurrent access; communication overloads (if components run on different machines); efficiency problems, due to the unspecialized character of the blackboard; and debugging problems, due to the complexity of writing each component.

Another possibility is to integrate heterogeneous component processes under a *coordinator.* Components may be written in various programming languages, may use their own data structures and algorithms, and may run on different machines. They are still logically arranged in a sequential order (or in a hierarchy if there are several input modalities), but the coordinator may send them input in an incremental way, so that they may actually run in parallel.

In such an architecture, the coordinator maintains in a *whiteboard* an image of the input and output data structures of each component, at a suitable level of detail. This global record fosters reuse of partial results and avoids wasteful recomputation. Further, the use of a unique object language for declaring, inspecting and manipulating all parts of the whiteboard permits transparent examination and should greatly aid experimention with, and integration of, future components, whatever they may be.

Each component process (e.g., Speech Recognition) is encapsulated in a *manager*, which transforms it into a server, communicating with external clients (including the coordinator) via a system of mailboxes. Managers handle the conversions between internal (server) and external (client) data formats. This protocol enhances modularity and clarity, especially if component developers are asked to explicitly and completely declare the appearance of their partial results on the whiteboard. Managers may also make batch components appear as incremental components by delivering outputs in a piecewise fashion, thus taking a first step towards systems simulating simultaneous translation.

In a first concrete exploration of these ideas, we (M. Seligman and myself) have produced a rudimentary architectural prototype, KASUGA. The coordinator is written in KEE, a powerful object-oriented expert system shell with very good graphic capabilities. With the three modules integrated so far, it is possible to demonstrate the above ideas.

In the first section, our four main guidelines are detailed: (1) record overall progress of components in a whiteboard; (2) let a coordinator schedule the work of components; (3) encapsulate components in managers; and (4) use the managers to simulate Incremental Processing.

In the second, some high-level aspects of the KASUGA prototype are described: its coordinator and its components; what it does in this preliminary state (a simple demonstration is discussed, in which incremental speech translation is simulated); and what kind of whiteboard it uses.

In the third, lower-level details are given on some internal aspects: the communication mechanism; the import and export of objects into the common object language; and the interface's present state and future possibilities.

I. Whiteboard architecture: Guidelines

1. Record overall progress of components in a whiteboard

The whiteboard architecture is of course inspired by the chart architecture of the MIND system [10] and later systems or formalisms for NLP [1, 7], as well as by the blackboard architecture, first introduced in HEARSAY-II [8, 15] for speech recognition. However, there is a significant difference: the components do not access the whiteboard directly, and need not even know of its existence.

1.1 Problems of the sequential and blackboard approaches

a. Sequential approach

There are two main problems with the sequential approach.

- P1: loss of information

  As the interface between two successive components is usually relatively poor, information is lost and work has to be duplicated. For example, ATR's ASURA speech translation system [12, 13] uses context-free LR syntactic parsing to drive speech recognition; but syntactic structures found during the recognition parse are discarded when recognition candidates are passed to machine translation. Complete reparsing is thus needed.

- P2: lack of robustness

  Communication difficulties between subsystems may also damage robustness. During reparsing for MT in ASURA, if no well-formed sentences are found, partial syntactic structures are discarded before semantic analysis; thus there is no chance to translate partially, or to use semantic information to complete the parse.

b. Pure blackboard approach

The pure blackboard approch solves P1, but not P2, and introduces other problems.

- P3: control of concurrent accesses

  In principle, all components are allowed to access any part of the blackboard at their convenience. Complex protection and synchronization mechanisms must be included, and fast components may be considerably slowed down by having to wait for permission to read or write.

- P4: communication overloads

  The amount of information exchanged may be large. If components run on different machines, such as is often the case for speech-related components, and may be the case for Example-Based MT components in the future, communication overloads may annihilate the benefit of using specialized or distributed hardware.

- P5: efficiency problems

  As components compute directly on the blackboard, it is a compromise by necessity, and can not offer the optimal kind of data structure for each component or algorithm.

- P6: debugging problems

  These are due to the complexity of writing each component with the complete blackboard in mind, and to the parallel nature of the whole computation.

c. Layered blackboard approach

In [3], I proposed to divide the blackboard in successive layers, each accessed by only one component, as illustrated in the figure below[1]. However, this solves P3, but not P4–P6.

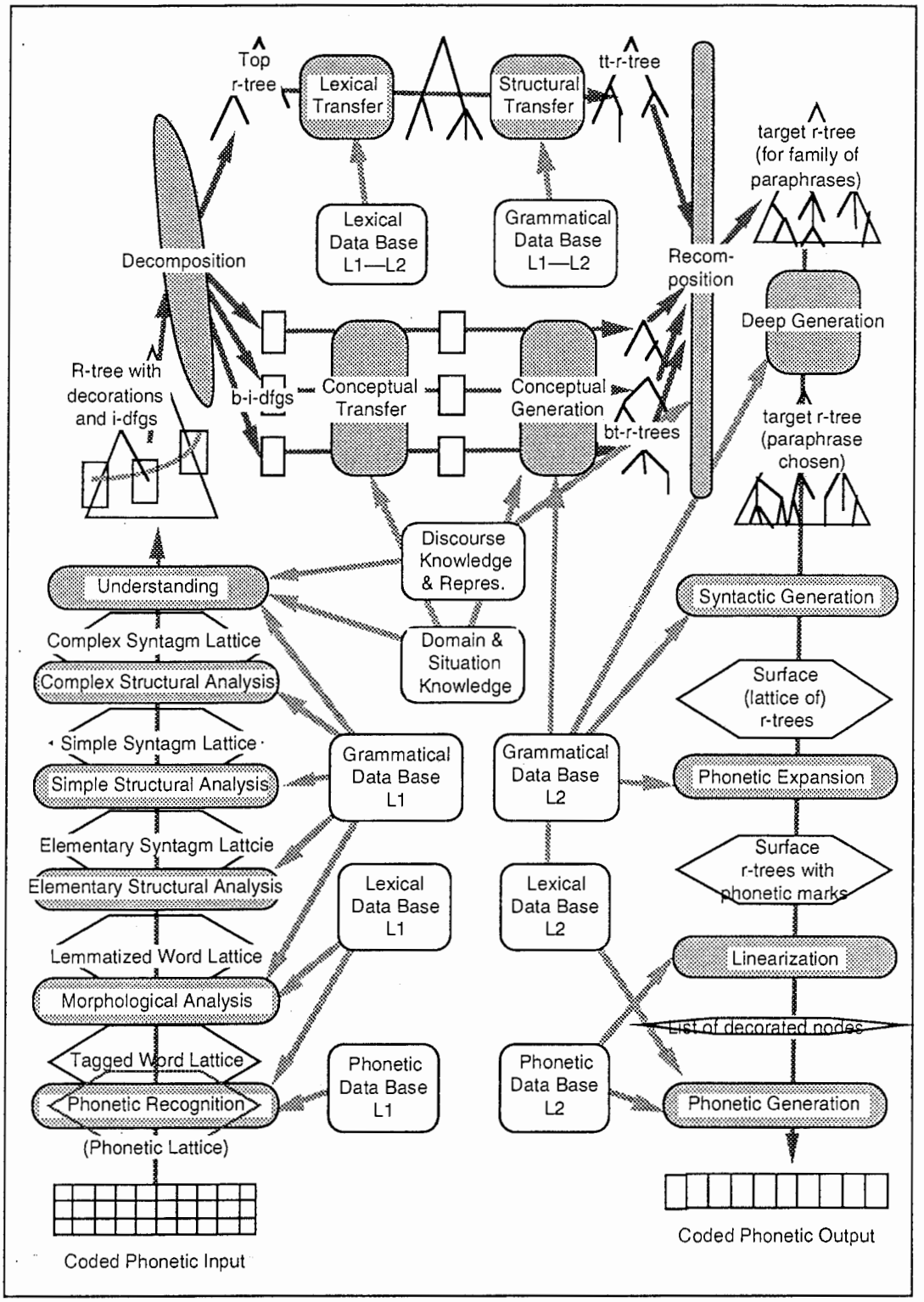

Figure 1: The layered lattice blackboard approach

[1] Many other ideas are illustrated here. Some, but not all of them, such as sharing not only dynamic states of the computation, but static sources of knowledge, are discussed in the report.

1.2 The "whiteboard" approach

In the "whiteboard" approach, the global data structure is hidden form the components, and accessed only by a "coordinator", as illustrated in figure 2 below (the whiteboard drawing is expanded later!).

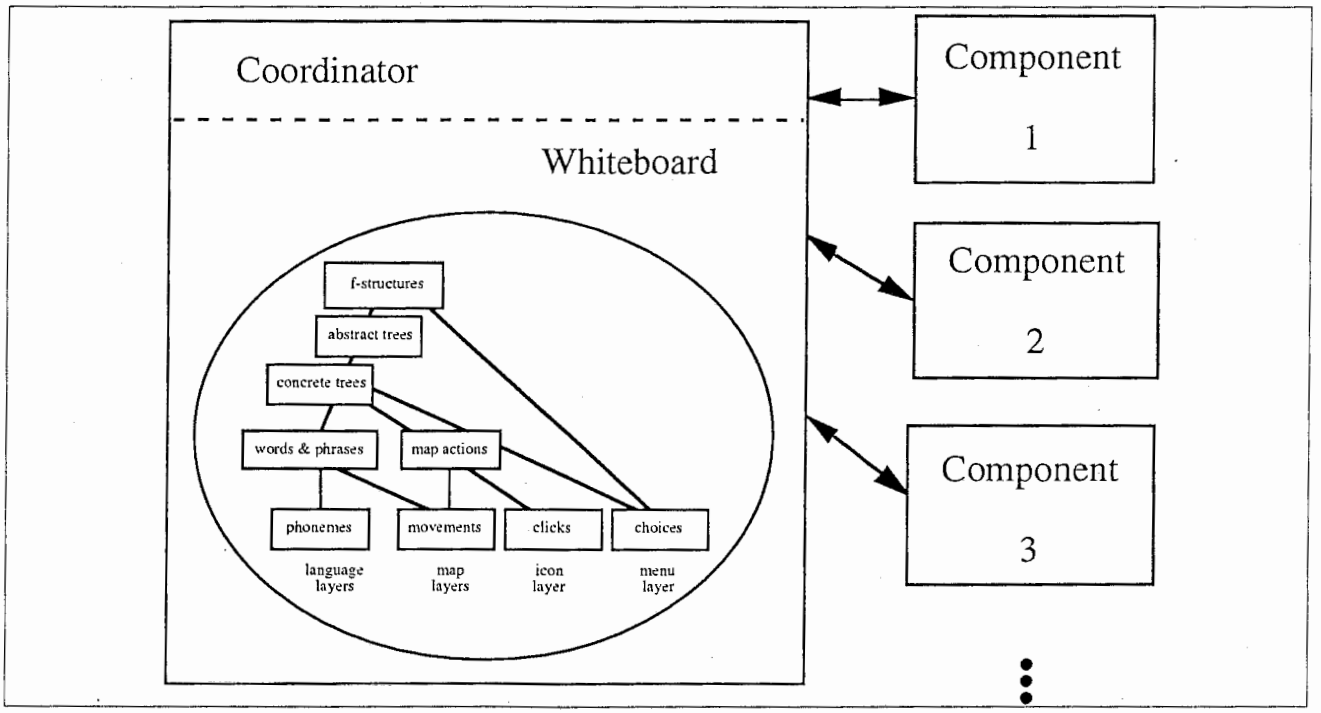

Figure 2: Coordinator, whiteboard and components

This simple change makes it possible to avoid problems P3–P6. It has also at least two good points.

- *It encourages developers to clearly define and publish what their inputs and outputs are,* at least to the level of detail necessary to represent them in the whiteboard.

  It is often a headache for "integrators" to find out what it is really that the components to integrate expect to receive and happen to deliver. Specifying these in a common structure or object description language is certainly worthwhile, not only for the integrators, but for the developers of each component, who will gain a clearer view of their own work.

- *The whiteboard may be the central place where graphical interfaces are developed* to allow for easy inspection, at various levels of detail.

  As this is a very time-consuming task and is looked upon as development and not research, researchers tend not to do it. By introducing the idea of generic interfaces for a class of components, interface building for NLP becomes a research topic in its own right, and developers gain access to better tools without having to build them.

1.3 Rationale for the proposed time-aligned, layered lattice

Beyond the general idea of a layered whiteboard, there are some arguments in favor of our design preferences for time aligned lattices and the use of "shaded" complex nodes (white, grey, black). A discussion of the kind of complex information to put into the nodes may be found in [3].

a. Charts, Q-graphs, lattices

In a *grid,* there are no explicit arcs. A node N covering [t1,t2] is implicitly connected to another node N' covering [t'1,t'2] iff [t1,t2] is anterior to [t'1,t'2], that is $t1 \leq t'1$, $t2 < t'2$, and $t2-e1 \leq t'1 \leq t2+e2$, e1 and e2 being respectively the *gapping* and *overlapping thresholds* [14]. Because of the first condition, there can be no cycles.

In a *lattice,* there are only explicit arcs. Cycles are forbidden, and there must be a unique first node and a unique last node. Figures 3 and 4 show two kinds of lattices used in natural language processing (NLP in the following), a chart and a Q-graph.

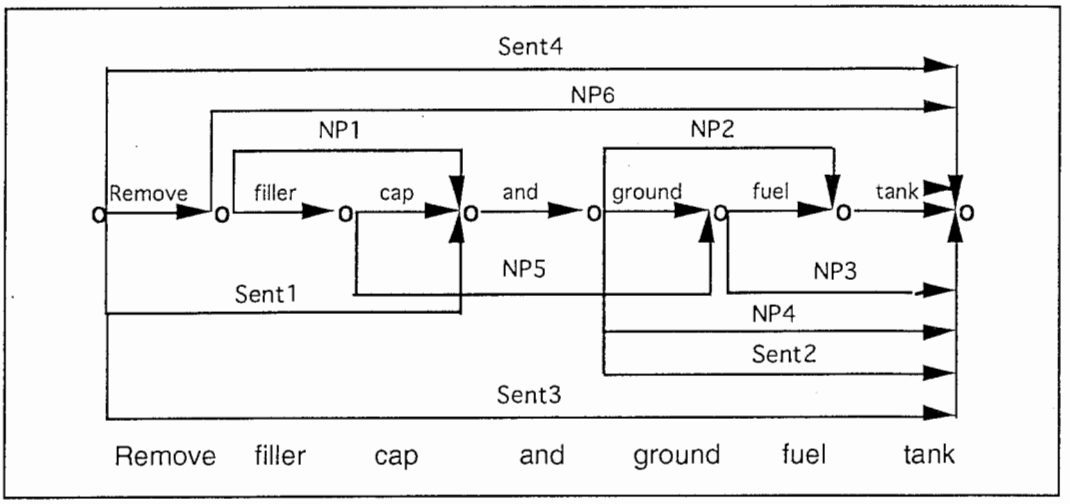

Figure 3: A chart (built on a syntactically ambiguous sentence)

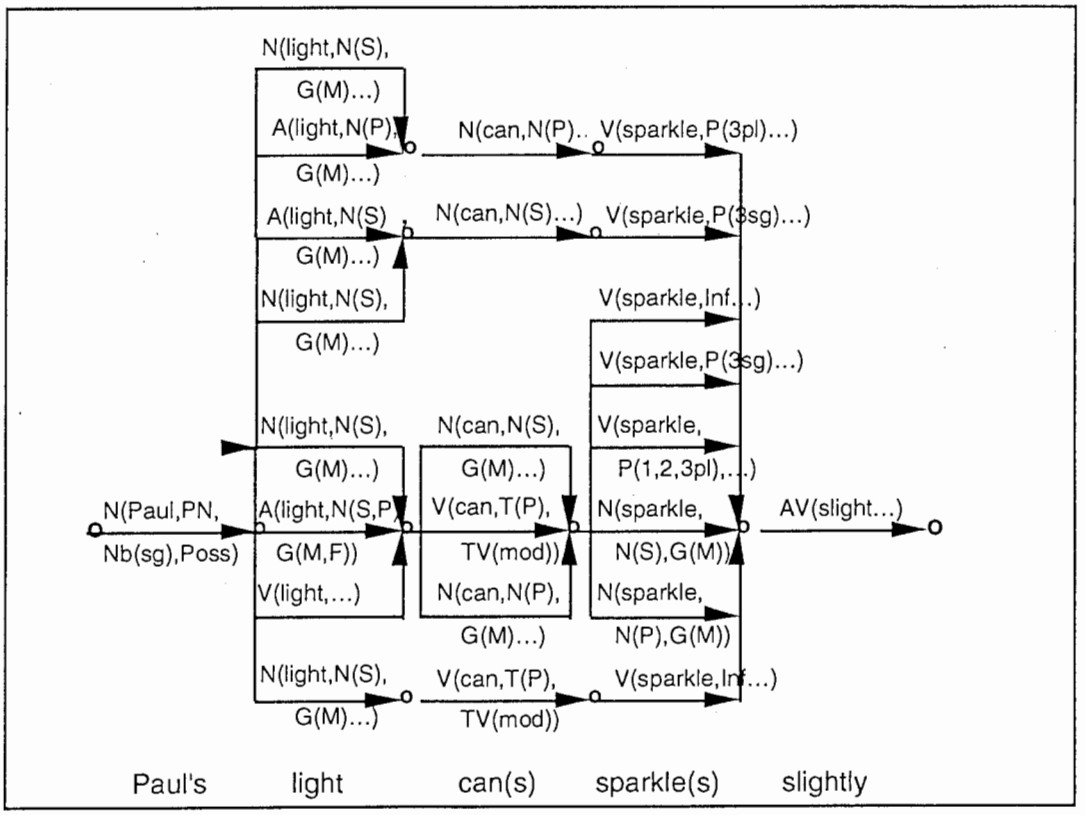

Figure 4: A Q-graph (built on a phonetically ambiguous sentence)

b. Why a lattice?

Both types of structures have been used in Speech Recognition. In the above reference ([14]), Quinton presented in 1980 that of the Kéal system, where the output of the phonetic component, the "lexical spectrum", is such a grid. The results of ATR's speech recognizer today are also presented as a grid. Each node contains a "detection", made of a start time, an end time, a label and a score. The difference is that Kéal's labels are words, while ATR's are phonemes. There are two special detections, initial and final, which contain special endmarkers instead of words or phonemes.

By contrast, the HWIM [24] system used a "phonetic lattice" on which an extended ATN operated.

Grids have only been used in MT to implement some working structures (like that of the Cocke algorithm). However, we may imagine to use them for representing an input text obtained by scanning a bad original, or a stenotypy tape [11].

On the other hand, lattices have been used extensively, in two varieties. First, the *chart structure* has originally been introduced by M. Kay in the MIND sytem (around 1965, see [10]). In a chart, the nodes are arranged linearly, so that there is always a path between any two given nodes, and the arcs bear the information, not the nodes. This data structure is also used by many unification-based natural language analyzers [16].

The *Q-graphs* of [7] and their extension [21] are the basic data structure for text representation in the METEO [4, 5] and TAUM-Aviation [9] systems. A Q-graph is a loop-free graph with a unique entry node and a unique exit node. It is possible for two nodes not to be on any common path from the entry to the exit. The information is beared on the arcs, in the form of simply labelled trees, or, in the extension, of trees with labels and binary features.

Actually, none of these structures is strictly a lattice, because two different arcs may link the same pair of nodes ; moreover, a Q-graph may contain two different arcs linking the same two nodes, and bearing the same tree. However, it is always possible to transform a chart or a Q-graph into an equivalent lattice (with the information on the nodes) by replacing arcs with nodes and creating appropriate arcs.

For example, ATEF (a SLLP, or Specialized Language for Linguistic Programming, developed at GETA for writing morphological analyzers) produced initially Q-graphs and decorated trees [6]. It was easily adapted [2] to also produce lattices, used as input to "algogrammars" (kinds of bilevel ATNs where one level describes the grammar and the other the heuristic control).

Q-graphs and lattices are also natural structures to represent a text with alternate formulations, like a first draft or a rough translation. For example, the following alternate formulations :

	it if you came			
I would like	you to	come	early	tomorrow
			earlier	
		be early		

Figure 5: A sentence with alternate formulations

could be represented by the lattice of figure 6, or by a Q-graph (with repetition of at least one arc), but not by a grid or a chart.

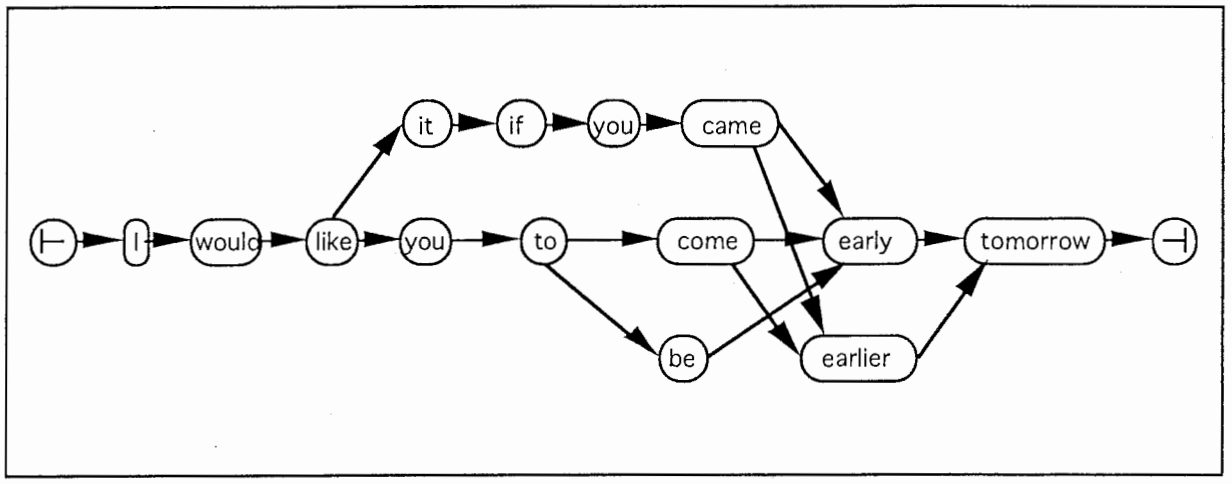

Figure 6: A word lattice (representing a sentence with alternate formulations)

Our favorite kind of structure, then, is a lattice. We record complex information in nodes, rather than in arcs as in charts. Our arcs bear only activation or inhibition weights, as in neural networks. This analogy with neural networks has the added advantage to pave the way for applying ideas from that fast developing field... as soon as we will understand how to do it!

The decomposition of the lattice in layers seems quite natural, and leads to more clarity. Each layer contains results of one component, selected to the "appropriate level of detail".

Its time-aligned character makes it possible to organize it in such a way that everything which has been computed on a certain time interval at a certain layer may be found in the same region.

Each layer has 3 dimensions, *time, depth* and *class.* The terms "class" and "label" are used interchangeably here. The idea is that a node at position (i,j,k) corresponds to the input segment of length j ending at time i and be of class (number) k[2]. All realizations of class k corresponding to this

---

[2] More precisely, we could say that a node at position (i,j,k) in the basic structure will cover the intervals [t1,2] ($0 \le t1 \le t2 \le tmax$) of the input such that $t1-e \le i-j+1 \le t1+e$ and $t2-e \le i \le t2+e$, e being some error margin associated to the node, and tmax being about 10000 if we take the basic unit of length to be the time between two successive frames (10 ms), a take a very safe maximum corresponding to 100 seconds, or 200–250 words at a speaking rate of 2–3 words/second. Although usual interpretation units are far shorter, they may consist of several utterances.

segment are to be packed in this node, and all nodes corresponding to approximately equal input segments are thus geometrically clustered.

In other words, the resulting structure is "factorizing", meaning that ambiguities are packed so that dynamic programming techniques may be applied on direct images of the whiteboard. Here is an example, where the main NP has been obtained in two ways.

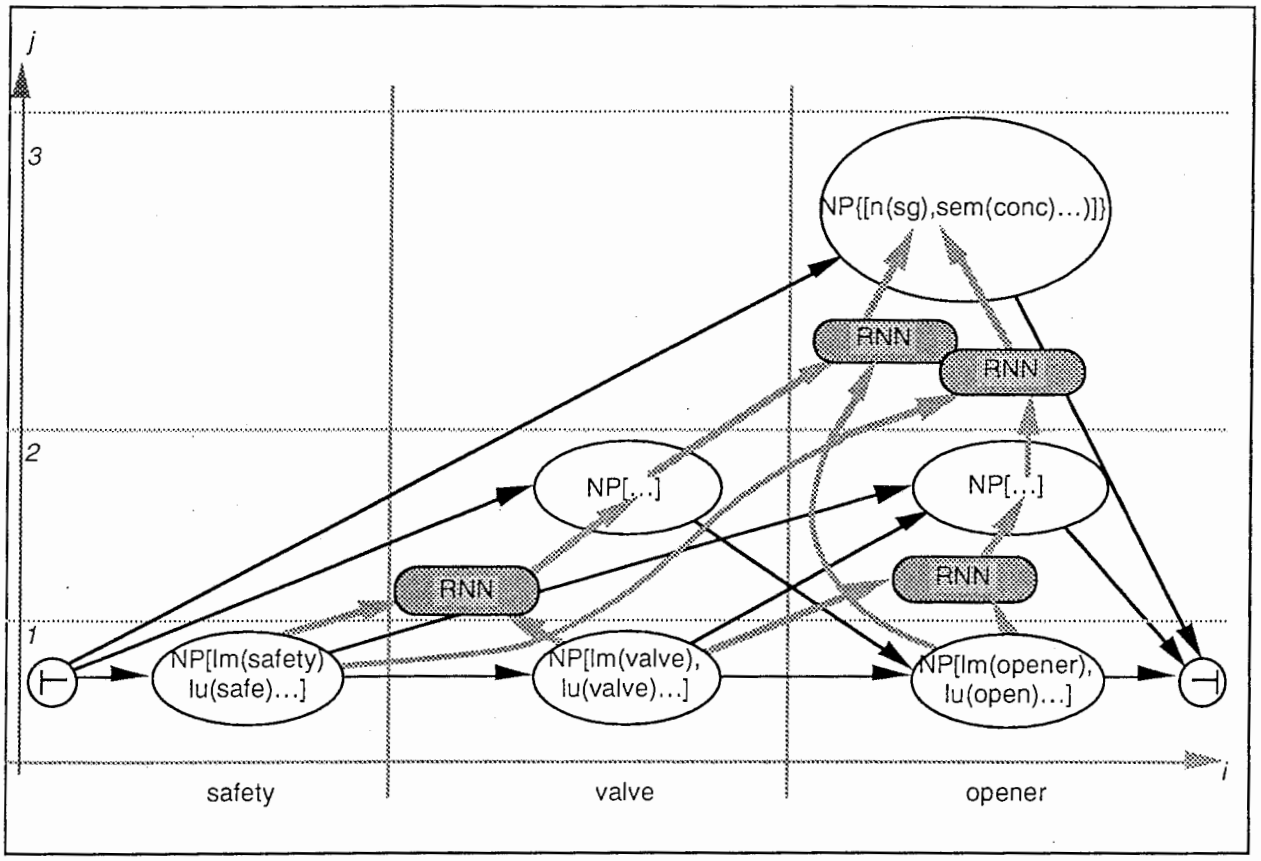

Figure 7: The whiteboard as a factorizing data structure

c. Degrees of detail: white, grey and black nodes

We said that the whiteboard could be a central place for transparent inspection, at suitable levels of detail. We use the notion of "shaded nodes" for this.

- "White" nodes are the real nodes of the lattice. They contain *results* of the computation of the component associated with their component: a white node contains at least a class, or label, legal in its layer, such as NP, AP, CARDP, VP... in the example above, and possibly more complex information, such as a "decoration" (bounded property list), an annotated tree, a feature structure, or... another latttice, as allowed by the declaration of the layer in the whiteboard.
- "Grey" nodes may be added to show how the white nodes have been constructed. They don't belong to the lattice proper. In the example above, they stand for *rule instances.* In other cases, they might be used to show the correspondences beetween nodes ot two layers. They may be used to represent general rewriting rules, such as X1...Xp –> Y1...Yq.

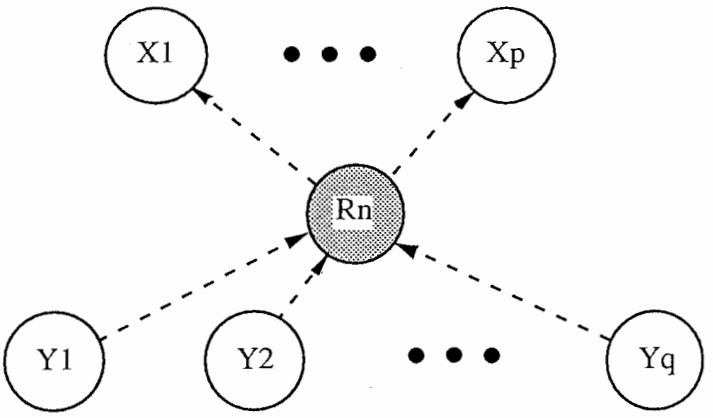

Figure 8: White and grey nodes corresponding to rule Rn: X1 X2...Xp –> Y1 Y2...Yq

- "Black" nodes may be used to represent finer steps in the computation of the component, e.g. to show the elementary steps of a parser, as in the example, where we represent the syntagmatic layer and the structures used by the Quinton algoritm [14].

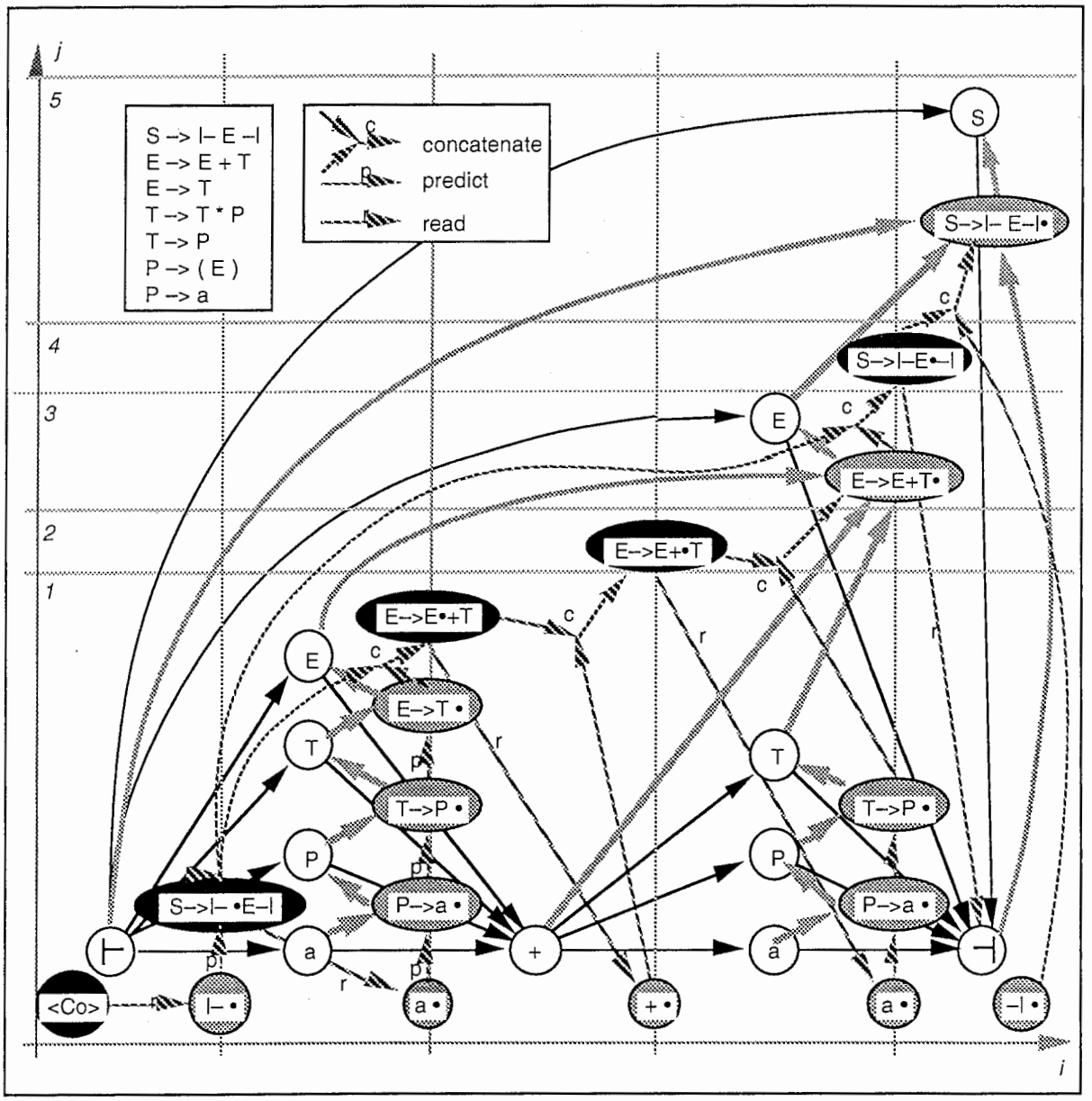

Figure 9: an example with black nodes

Whiteboard layers are organized in a loop-free dependency graph. Non-linguistic as well as linguistic information can be recorded in appropriate layers. For example, in a multimodal context, the syntactic analyzer might use selected information from a map layer, where pointing, etc. could be recorded. Interlayer dependencies should be declared, with associated constraints, stating for instance that only nodes with certain labels can be related to other layers. Here is an illustration of that idea, without any pretense to propose a realistic choice of layers, however.

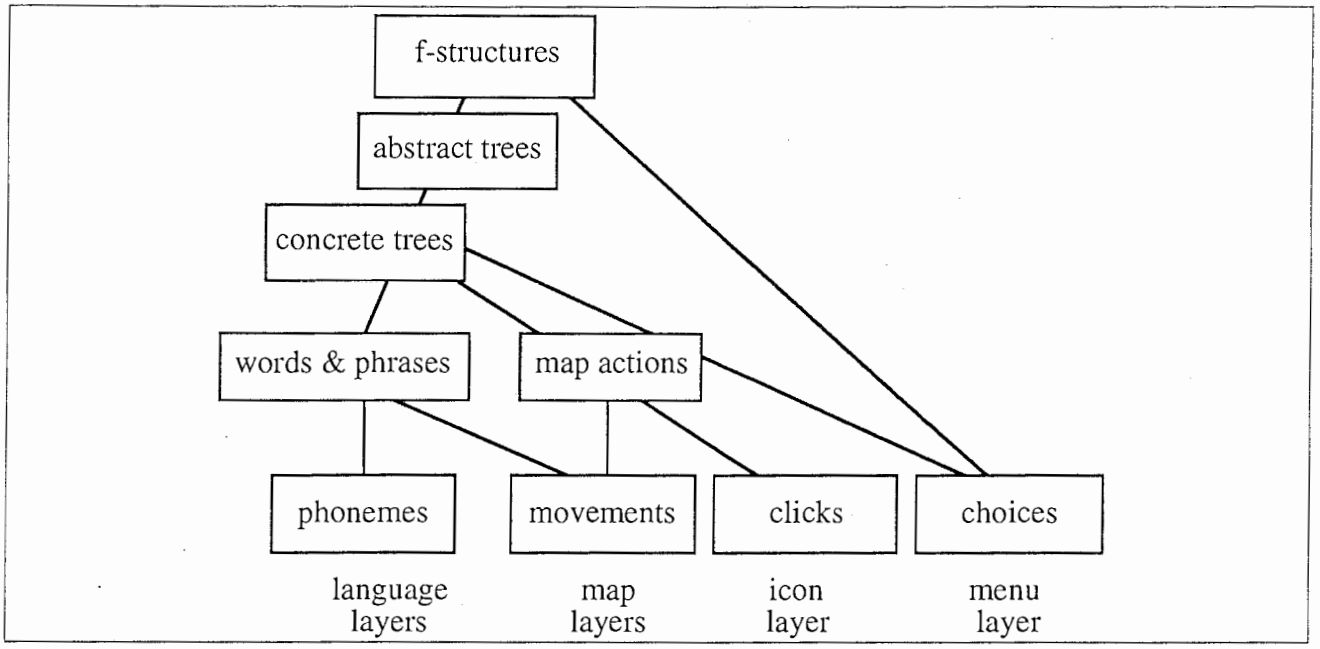

Figure 10: A hierarchy of layers in an hypothetical whiteboard

2. Let a coordinator schedule the work of components

In its simplest form, a coordinator only transmits the results of a component to the next component(s). However, it is in a position to carry out global strategies by filtering low-ranking hypotheses and transmitting only the most promising part of a whiteboard layer to its processing component.

Further, if certain components make useful predictions, the coordinator can pass these to other components as constraints, along with input. A process tracking entities in discourse focus, for instance, could produce constraints narrowing the set of rules used for word recognition.

3. Encapsulate components in managers

Developers of components should be free to choose and vary their algorithms, data structures, programming languages, and possibly hardware (especially so for the speech-related components).

Our approach is to isolate (or "encapsulate", in technical terms) existing components in *managers*, which hide them and transform them into servers. This strategy has the further advantage of avoiding any direct call between coordinator and components. To plug in a new component, one just writes a new manager, a good part of which is generic. Hence, we get the following updated diagram.

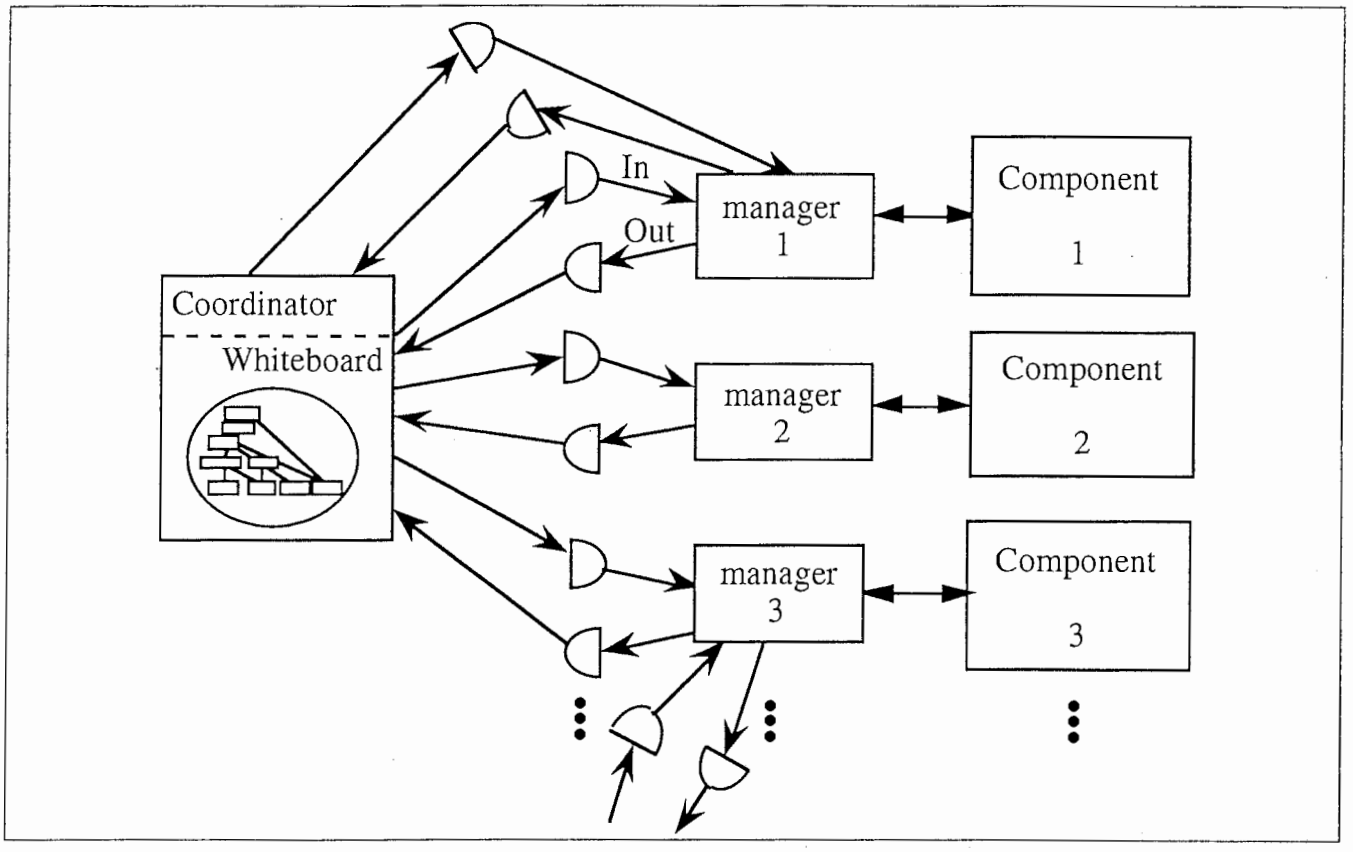

Figure 11: Coordinator, components and managers

A manager has a request box where clients send requests to open or close connections. A connection consists of a pair of in and out mailboxes, with associated locks, and is opened with certain parameters, such as its sleep time and codes indicating pre-agreed import and export formats.

The coordinator puts work to do into in-boxes and gets results in corresponding out-boxes. A managers periodically inspects its request box and in-boxes; executes the requests, transforms the contents of in-boxes and submits them to its associated component; and transforms the results of previous calls to the component before placing them in appropriate out-boxes.

As illustrated in figure 11 above, a manager can in principle have several clients, and a client can open more than one connection with the same server. For example, an on-line dictionary might be called for displaying "progressive" word for word translation, as well as for answering terminological requests by a human interpreter supervising several dialogues and taking over if needed. However, this potential is not used in KASUGA.

4. Use managers to simulate Incremental Processing

In real life, simultaneous interpretation is often preferred over consecutive interpretation: although it may be less exact, one is not forced to wait, and one can react even before the end of the speaker's utterance. Incremental processing will thus be an important aspect of future machine interpretation systems.

One subprocess should be able to begin work on the early output of another subprocess before the latter has finished processing an entire utterance. For instance, a semantic processor might begin working on the syntactic structures hypothesized for early parts of an utterance while later parts are still being syntactically analyzed [23].

Even if a component (e.g., any currently existing speech recognizer) has to get to the end of the utterance before producing any result, its manager may still make its processing appear incremental, by delivering its result piecewise and in the desired order. Hence, this organization makes it possible to simulate future incremental components.

II. KASUGA prototype: external level

1. The coordinator and the components

The coordinator (KAS.COORD) is written in KEE™, an object-oriented expert system shell with excellent interface-building tools. The whiteboard is declared in KEE's object language. KEE itself is written in Common Lisp.

Three components are involved:

- speech recognition (SP.REC) providing a phoneme lattice, programmed in C [19];
- island-driven syntactic chart-parsing (SYNT.AN) deriving words and higher-level syntactic units, programmed in C;
- word-for-word translation (WW.TRANS) at the word level, programmed in Lisp.

The managers are programmed in Lisp, and run independently, in three Unix processes. Each manager and the coordinator can run in different Unix shells. Although WW.TRANS is already accessible as a server on a distant machine, we had to create a manager for it to get the intended behavior.

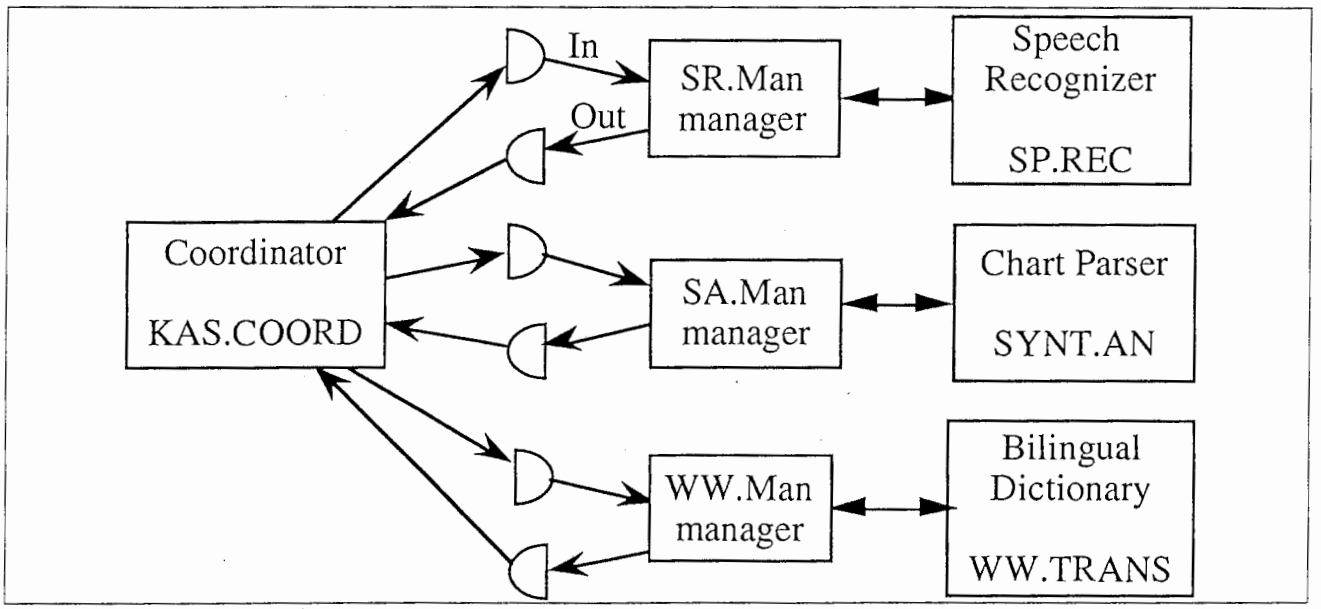

Figure 12: KASUGA's coordinator and components

2. What it does

With only these components, it is possible to produce a simple demonstration in which incremental speech translation is simulated and the transparency gained by using a whiteboard is illustrated. The phonemes produced by SP.REC are assembled into words and phrases by SYNT.AN. As this goes on, WW.TRANS produces possible word for word translations, which are presented on screen as a word lattice.

3. Whiteboard

KASUGA's whiteboard has only three layers: phonemes; source words and phrases; and equivalent target words. There is no layer to contain the speech wave itself, although a full system should have one for easier inspection.

At the first layer, the phoneme lattice is represented with phonemes in nodes. At the second layer, we retain only the complete substructures produced by SYNT.AN, that is, the inactive edges.

In KEE[3], we define a class of NODES, with subclasses WHITE.NODES, GREY.NODES, PHON.LAYER.NODES, and SYNT.LAYER.NODES in the syntactic layer. NODES have a generic display method, and subclasses have specialized variants (e.g., the placing of white nodes depends on their time interval, while that of grey nodes depends on that of the white nodes they connect).

III. KASUGA prototype: internal level

1. Communication Mechanism

When a manager receives a Make.Connection request from a client, it creates an in box and an out box (and associated locks, used to prevent interference between components), through which information is passed to and from the client[4]. The Make.Connection request includes codes showing in which format(s) the client is expecting to deposit data in the in box and read data from the out box, for that connection.

We believe that the overhead associated with message passing and import/export through files will be negligible in comparison with the actual processing time required by the components and coordinating processes. Data transfer could be programmed more efficiently, at the level of the operating system, e.g. using Unix sockets. But our method is more general, as it uses only the file system.

For each out box, the client (KASUGA) activates a reader process and the relevant manager activates a writer process. Conversely, for each in box, the client activates a writer process and the manager activates a reader process.

---

[3] Given suitable interface tools, other object languages, such as CLOS or C++, could serve equally well.

[4] In KASUGA, the coordinator is actually the only client of the managers, but this could change in a fuller system.

A reader process wakes up regularly (its SLEEPTIME is adjustable) and checks whether its mailbox is both non-empty and unlocked. If so, it locks the mailbox; reads its contents; empties the mailbox; unlocks it; and goes to sleep again.

A writer process, by comparison, wakes up regularly and checks whether its mailbox is both empty and unlocked. If so, it locks the box, fills it with appropriate data, unlocks it, and goes back to sleep. For example, the writer associated with SYNT.AN will deposit in the appropriate out box the image of all the inactive arcs created since the last deposit.

2. Import and Export of Objects

KAS.COORD writes and reads data to and from the managers in a LISP-like format, and handles the transformation into KEE's internal format. Each manager translates back and forth between that format and whatever format its associated component happens to be using. To enable these translations, the formats must be precisely defined[5].

For instance, the edges produced by the speech recognizer are of the form *(begin end phoneme score)*. The nodes and edges of the corresponding phoneme layer in the whiteboard are of the form *(node-id begin end phoneme score (in-arcs) (out-arcs))*, with arcs being of the form *(arc-id origin extremity weight)*.

In cooperation with M. Seligman, four algorithms are being specified and implemented. They will be presented in detail in a forthcoming technical report.

grid-to-lattice
: transforms the grid output by the speech recognizer into a lattice. A transition is established between two nodes containing detections iff these detections meet the condition given above (I.1.3.a, p. 8). A weight can also easily be computed from the gap or overlap.

lattice-to chart
: is quite interesting, as we try to produce an incremental factorization. As a matter of fact, a trivial solution consists in enumerating all possible paths in the lattice, and in creating as many chart arcs as necessary. But that leads to an explosion of the number of chart arcs. On the other hand, a solution based on minimizing the finite-state automaton obtained by the trivial method would be unduly costly, both in time and space, and not be incremental.

chart-to-lattice
: is quite straightforward, the only difficulty being to define when two arcs should be considered identical, and thus give rise to only one lattice node.

lattice-to-lattice-dictionary-expansion
: is used to produce the third layer (English word lattice) from the second (Japanese word and phrase lattice). Here, we use grey nodes to record which English nodes correspond to which Japanese nodes.

---

[5] Ideally, they should be defined with formal grammars. Then, a given transducer (or "filter" in the world of microcomputer software) can be realized as a syntax-driven translator, with a target syntax checker.

3. More on Interfaces

To take full advantage of the whiteboard, systems developers, too, must be able to see and operate on it. They need to visualize and monitor the overall system organization: to know which components are active or inactive; to view the data passed between components and the coordinator; and to check and reset current priorities and parameters.

KASUGA's current interface provides some, but not all, capabilities needed in a full system. The designer can view the phoneme lattice developing from left to right, as if the speech recognizer were incremental; the partial structures (islands), in near-real-time, as they are produced by SYNT.AN[6]; and word translation results.

The relations between these can also be made visible: we can see which Japanese words are related to which phonemes, or which English words are related to which Japanese words, etc.

By default, all nodes show their label only, but it is possible to inspect the rest of their content. Weights and scores are also displayed on demand. The view of any component can be changed for emphasis: one can for instance interactively select only the nodes above a certain confidence threshold. Overall processing can be interrupted for examination.

KEE offers very good interface building facilities, and in particular good tools for developing interactive graphic interfaces. If this architecture is to be further developed in the future, one could instead use a general-purpose, portable interface building toolkit in order to avoid the overhead and overspecialization associated with using a complete expert system shell.

Discussion

In this prototype, the static knowledge sources (automata, grammars, dictionaries) used by the components are distinct. Sharing occurs only dynamically, through the whiteboard. But sharing of static knowledge could also be envisaged, while still meeting the specific needs of each component.

Large MT systems have for some time used neutral lexical data bases, in which the dictionaries of particular components (analysis, generation) are compiled into the corresponding formats. In some experimental systems, analysis and generation use specialized rules compiled from a common "static" grammar. Likewise, it might be possible to extract a pure CFG from an augmented (e.g. unificational) grammar when desirable (e.g., as the CFG part of an HMM-LR speech recognizer).

A second idea concerning sharing is that predictions from a higher layer to a lower layer can be divided into fine-grained and coarse-grained. Fine-grained predictions (e.g., on the next possible phonemes, or

---

[6] White nodes and grey nodes. We may add a switch to SYNT.AN, so that it also produces its active edges, which would then appear as black nodes in the whiteboard.

morphosyntactic classes) should be static, that is, compiled in the running resources of the lower component. Otherwise, fast procedures would constantly be waiting for top-down predictions.

By contrast, coarse-grained predictions are more dynamic in nature, and may actually help improve the speed and accuracy of lower components. For instance, predicting a sub-task and an utterance type from a (possibly interactive) discourse analysis component could considerably reduce the set of possible rules, terms, and word senses.

Conclusion

The whiteboard architecture which has been researched here begins to be not only a concept on paper, but a reality on the computer. However, the present prototype has only been developed for illustration purposes. It would really take the good will and cooperation of more researchers to build this sort of "MI shell". But I am deeply convinced that this effort would be quite worthwhile, and lead to a state of affairs where a researcher could independently develop an original component, integrate it without too much effort by writing the corresponding manager[7], and experiment with it. Researchers would thereby gain twice: by getting a clearer view of what they (and others) are doing; and by being able to use generic interface tools provided by the coordinator for debugging and illustrating purposes.

Acknowledgments

Thanks should first and foremost go to M. Seligman, with whom I have been very happy to learn the basics of KEE and to develop the KASUGA prototype. I hope to continue working with him during the final development stages, using e-mail to beat the distance. Thanks also to M. Fiorentino from Intellicorp, Inc. and K. Kurokawa from CSK, Inc., for providing a demo copy of KEE™ and valuable technical support; Dr. Y. Yamazaki, President of ATR-ITL, and T. Morimoto, Head of Department 4, for their support and encouragement; H. Singer, Y. Kitagawa, and H. Kashioka, for their help in developing the components; and K. H. Loken-Kim, for stimulating discussions and proposing the term "whiteboard".

Bibliography

[1] **Barnett J., Knight K., Mani I. & Rich E. (1990)** *Knowledge and Natural Language Processing.* Comm. ACM, 33/8, 50–71.

[2] **Boitet C. (1976)** *Un essai de réponse à quelques questions théoriques et pratiques liées à la traduction automatique. Définition d'un système prototype.* Thèse d'Etat, Université de Grenoble.

[3] **Boitet C. (1988)** *Representation and Computation of Units of Translation for Machine Interpretation of Spoken Texts.* Comp. & AI, 6, 505—546.

[4] **Chandioux J. (1988)** *10 ans de METEO (MD).* In "Traduction Assistée par Ordinateur. Actes du séminaire international sur la TAO et dossiers complémentaires", A. Abbou, ed., Observatoire des Industries de la Langue (OFIL), Paris, mars 1988, 169—173.

---

[7] This should be facilitated by the fact that the part concerning communications through mailboxes is generic and reusable.

[5] **Chandioux J. & Guérard M.-F. (1981)** *METEO: un système à l'épreuve du temps.* META, 1, 17—22.

[6] **Chauché J. (1975)** *Les langages ATEF et CETA.* AJCL (American Journal of Computational Linguistics), microfiche 17, 21-39.

[7] **Colmerauer A. (1970)** *Les systèmes-Q, un formalisme pour analyser et synthétiser des phrases sur ordinateur.* TAUM, Univ. de Montréal, dec. 1970.

[8] **Erman L. D. & Lesser V. R. (1980)** *The Hearsay-II Speech Understanding System : A Tutorial.* In "Trends in Speech Recognition", W. A. Lea, ed., Prentice-Hall, 361–381.

[9] **Isabelle P. & Bourbeau L. (1984)** *TAUM-AVIATION: its technical features and some experimental results.* Comp. Ling., **11**/1, 18 27.

[10] **Kay M. (1973)** *The MIND system.* In "Courant Computer Science Symposium 8: Natural Language Processing", R. Rustin, ed., Algorithmics Press, Inc., New York, 155-188.

[11] **Mérialdo B. (1988)** *Multilevel decoding for Very-Large-Size-Dictionary speech recognition.* IBM Journal of Research and Development, **32**/2, March 1988, 227–237.

[12] **Morimoto T., Suzuki M., Takezawa T., Kikui G.-I., Nagata M. & Tomokiyo M. (1992)** *A Spoken Language Translation System: SL-TRANS2.* Proc. COLING-92, Nantes, juillet 1992, C. Boitet, ed., ACL, vol. 3/4, 1048—1052.

[13] **Morimoto T., Takezawa T., Yato F., Sagayama S., Tashiro T., Nagata M. & al. (1993)** *ATR's Speech Translation System: ASURA.* Proc. EuroSpeech'93, Berlin, 21-23/9/83, 4 p.

[14] **Quinton P. (1980)** *Contribution à la reconnaissance de la parole. Utilisation de méthodes heuristiques pour la reconnaissance de phrases.* Thèse d'Etat, Univ. de Rennes, 239 p.

[15] **Reddy R. (1980)** *Machine Models of Speech Perception.* In "Perception and Production of Fluent Speech", Cole, ed., Erlbaum, N.J., 215–242.

[16] **Schieber S. M. (1986)** *An introduction to unification-based approaches to grammar.* CSLI Lecture Notes , 4, CSLI, Stanford, 105 p.

[17] **Seligman M. (1991)** *Generating Discourses from Networks Using an Inheritance-based Grammar.* Ph. D. thesis, Univ. of California, Berkeley, 171 p.

[18] **Seligman M., Suzuki M. & Morimoto T. (1993)** *Semantic-level transfer in Japanese-German Speech Translation: Some Experiences.* IEICE, **5**/NLC93-13, 17–25.

[19] **Singer H. & Sagayama S. (1992)** *Matrix Parser and its Application to HMM-based Speech Recognition.* IEICE, **10**/SP92-76, DSP92-61, 21–26.

[20] **Singer H. & Sagayama S. (1992)** *Matrix Parsing applied to TDNN-based Speech Recognition.* Japanese Journal of Speech Processing, **1992**/3, 89–90.

[21] **Stewart G. (1975)** *Manuel du langage REZO.* TAUM, Univ. de Montréal, 120 p.

[22] **Tomita M. (1990)** *The Generalized LR Parser/Compiler V8-4 : a Software Package for Practical NL Projects.* Proc. Helsinki, COLING-90, 20-25 Aug. 1990, H. Karlgren, ed., ACL, vol. 1/3, 59-63.

[23] **Wahlster W. (1993)** *Planning Multimodal Discourse.* Proc. ACL-93, Columbus, Ohio, 22-26/6/93, 95–96.

[24] **Wolf J. J. & Woods W. A. (1980)** *The HWIM Speech Understanding System.* In "Trends in Speech Recognition", W. A. Lea, ed., Prentice-Hall, 316–339.

-o-o-o-o-o-o-o-o-o-o-