A Generic Platform for Addressing the Multimodal Challenge

Laurence Nigay, Joëlle Coutaz
Laboratoire de Génie Informatique (LGI-IMAG)
BP 53, 38041 Grenoble Cedex 9, France
Tel: +33 76-51-44-40 +33 76-51-48-54
E-mail: Laurence.Nigay@imag.fr Joelle.Coutaz@imag.fr

ABSTRACT
Multimodal interactive systems support multiple interaction techniques such as the synergistic use of speech and direct manipulation. The flexibility they offer results in an increased complexity that current software tools do not address appropriately. One of the emerging technical problems in multimodal interaction is concerned with the fusion of information produced through distinct interaction techniques. In this article, we present a generic fusion engine that can be embedded in a multi-agent architecture modelling technique. We demonstrate the fruitful symbiosis of our fusion mechanism with PAC-Amodeus, our agent-based conceptual model, and illustrate the applicability of the approach with the implementation of an effective interactive system: MATIS, a Multimodal Airline Travel Information System.

KEYWORDS: Multimodal interactive systems, software design, software architecture, I/O devices, interaction languages, data fusion.

INTRODUCTION

One new challenge for Human Computer Interaction (HCI) is to extend the sensory-motor capabilities of computer systems to better match the natural communication means of human beings. Towards this goal, multimodal interfaces are being developed to support multiple interaction techniques such as the synergistic use of speech and gesture. The power and versatility of multimodal interfaces result in an increased complexity that current design methods and tools do not address appropriately. As observed by B. Myers, "user interface design and implementation are inherently difficult tasks"[11]. Myers's assertion is even more relevant when considering the constraints imposed by the recent technological push. In particular, multimodal interaction requires [3]:

- the fusion of different types of data originating from distinct interaction techniques as exemplified by the "put that there" paradigm,
- the management of multiple processes including support for synchronization and race conditions between distinct interaction techniques.

Thus, multimodal interfaces make necessary the development of software tools that satisfy new requirements. Such tools are currently few and limited in scope. Either they address a very specific technical problem such as media synchronization [9], or they are dedicated to very specific modalities. For example, the Artkit toolkit is designed to support direct manipulation augmented with gesture only [7].

In this article, we propose a software architecture model, PAC-Amodeus, together with a generic fusion mechanism for designing and implementing multimodal interaction. The PAC-Amodeus model along with the fusion engine form a reusable global platform applicable to the software design and implementation of multimodal interactive systems.

The structure of the paper is as follow: first, we clarify the notion of interaction technique using the concepts of interaction language and physical device. We then present the principles of our software architecture model, PAC-Amodeus, and show how interaction languages and devices operate within the components of the architecture. Going one step further in the implementation process, we populate PAC-Amodeus with the presentation of our generic fusion mechanism. We conclude with an example that illustrates how PAC-Amodeus and the fusion engine function together. This example is based on MATIS whose main features are presented in the next section.

AN ILLUSTRATIVE EXAMPLE: MATIS

MATIS (Multimodal Airline Travel Information System) allows a user to retrieve information about flight schedules using speech, direct manipulation, keyboard and mouse, or a combination of these techniques [13]. Speech input is processed by Sphinx, a continuous speaker independent recognition engine developed at Carnegie Mellon University [10]. As a unique feature, MATIS supports both individual and synergistic use of multiple input modalities [13]. For example, using one single modality, the user can say "show me the USAir flights from Boston to Denver" or can fill in a form using the keyboard. When exploiting synergy, the user can also combine speech and gesture as in "show me the USAir flights from Boston to this city" along with the selection of "Denver" with the mouse. MATIS does not impose any dominant modality: all of the modalities have the same power of expression for specifying a request and the user can freely switch between them. The system is also able to support multithreading: a MATIS user can disengage from a partially formulated request, start a new one, and later in the interaction process, return to the pending request.

PHYSICAL DEVICES AND INTERACTION LANGUAGES

A *physical device* is an artefact of the system that acquires (input device) or delivers (output device) information. Examples of devices in MATIS include the keyboard, mouse, microphone and screen.

An *interaction language* defines a set of well-formed expressions (i.e., a conventional assembly of symbols) that convey meaning. The generation of a symbol, or a set of symbols, results from actions on physical devices. In MATIS, examples of interaction languages include pseudo-natural language and direct manipulation.

We define an *interaction technique* as the coupling of a physical device d with an interaction language L: <d, L>.

Interaction techniques supported by MATIS include speech, written natural language, graphic input and output:

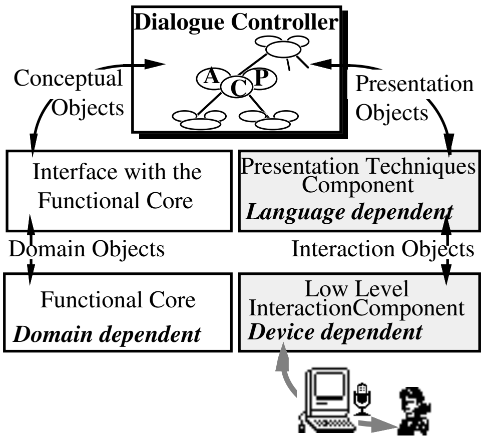
- speech input is described as the couple <microphone, pseudo natural language NL>, where NL is defined by a specific grammar,
- written natural language input is defined as <keyboard, pseudo natural language NL> (a MATIS user can also type in NL sentences in a dedicated windows),
- graphic input is described in terms of <mouse, direct manipulation>, and
- graphic output corresponds to the couple <screen, tables>. (Flight schedules returned by MATIS are always presented in a tabular format.)

Physical devices and interaction languages are resources and knowledge that the system and the user must share to accomplish a task successfully. They cover "the articulatory and semantic distances" expressed in Norman's theory [16]. Adopting Hemjslev's terminology [6], the physical device determines the substance (i.e., the non analyzed raw material) of an expression whereas the interaction language denotes the form or structure of the expression.

In [15], we demonstrate the adequation of the notions of physical device and interaction language for classifying and deriving usability properties for multimodal interaction. In this article, we adopt a complementary perspective and examine the relevance of these notions for software design.

SOFTWARE DESIGN

One important issue in software design is the definition of software architectures that support specific quality factors such as portability and modifiability. PAC-Amodeus is a conceptual model useful for devising architectures driven by user-centered properties including multithreading and multimodality. PAC-Amodeus blends together the principles of both Arch [18] and PAC [1]. Arch and its companion, the "slinky" metamodel, provide the appropriate hooks for performing engineering tradeoffs such as identifying the appropriate level of abstraction for portability, making semantic repair or distributing semantics across the components of the architecture [4]. In particular, the five component structure of Arch includes two adapters, the Interface with the Functional Core and the Presentation Techniques Component, that allow the software designer to insulate the key element of the user interface (i.e., the Dialogue Controller) from the variations of the functional core and of the implementation tools (e.g., the X window environment). The Arch model however, does not provide any guidance about the decomposition of the Dialogue Controller nor does it indicate how salient features in new interaction techniques (such as parallelism, fusion and fission of information [3]) can be supported within the architecture. PAC, on the other hand, stresses the recursive decomposition of the user interface in terms of agents, but does not pay attention to engineering issues. PAC-Amodeus gathers the best of the two worlds. Figure 1a shows the resulting model.



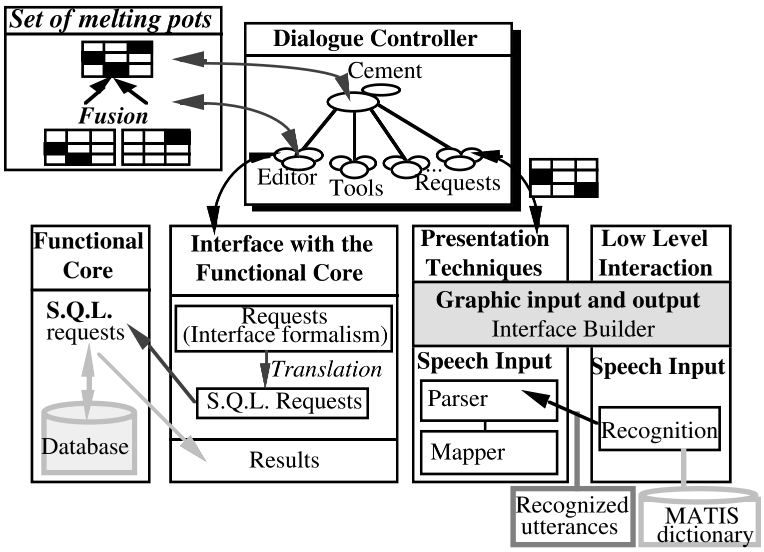


Figure 1: (a) The PAC-Amodeus software components. (b) PAC-Amodeus applied to the software design of MATIS.

A more detailed description of PAC-Amodeus can be found in [14]. Succinctly, the five components of the arch defines the levels of abstraction appropriate for performing engineering tradeoffs such as setting the boundaries between the levels of abstraction. We offer the notions of physical device and interaction language as criteria for setting these boundaries. For example, the designer may decide that the Low Level Interaction Component is device dependent. At a higher level of abstraction, the Presentation Techniques Component is device independent but language dependent. At the top of the Arch, the Dialogue Controller is both language and device independent.

PAC-Amodeus refines the Dialogue Controller into a set of cooperative agents that capture parallelism and information processing (e.g., data fusion) at multiple levels of abstraction. In turn, an agent is modelled as a three facet structure:

- the Presentation facet is in direct contact with the Presentation Techniques Component of the arch. It can be used to implement extensions on top of the Presentation Techniques Component;
- the Abstraction facet is in direct contact with the Interface with the Functional Core;
- the Control facet manages the links and constraints between its two surrounding facets (i.e., the Presentation and the Abstraction facets) as well as its relationships with other agents. As in ALV [8], the Control facet provides the hook for expressing constraints between different perspectives of the same concept.

In combining the Arch principles with PAC, one obtains an "engineerable" model that supports properties inherited from the agent paradigm. Figure 1b illustrates the application of PAC-Amodeus to the software design of MATIS. The Functional Core hosts the database of American cities, airline companies, flight numbers, departure and arrival times, etc. SQL requests are required to access information stored in the database. The Interface with the Functional Core (IFC) operates as a translator between the SQL formalism and the data structures used in the Dialogue Controller. In MATIS, the IFC serves as a communication bridge. As discussed in [2], it can also be used to restructure conceptual objects in a form suitable for the purpose of the interaction.

The Dialogue controller (DC) is organized as a two-level hierarchy of agents. This hierarchy has been devised using the heuristic rules presented in [15]. For example, because requests can be elaborated in an interleaved way, there is one agent per pending request.

At the other end of the spectrum, the Low Level Interaction Component (LLIC) is instantiated as two components inherited from the underlying platform: (1) The NeXTSTEP event handler and graphics machine, and (2) the Sphinx speech recognizer which produces character strings for recognized spoken utterances. Mouse-key events, graphics primitives, and Sphinx character strings are the interaction objects exchanged with the Presentation Techniques Component (PTC).

In turn, the Presentation Techniques Component (PTC) is split into two main parts: the graphics objects (used for both input and output) and the NL parser (used for input only). Graphics objects result from the code generation performed by Interface Builder. The Sphinx parser analyzes strings received from the LLIC using a grammar that defines the NL interaction language. As discussed above, the PTC is no longer dependent on devices, but processes information using knowledge about interaction languages.

Having presented the overall structure of PAC-Amodeus, we need now to address the problem of data fusion. As discussed in [14], fusion occurs at every level of the arch components. For example, within the LLIC, typing the option key along with another key is combined into one single event. In this article, we are concerned with data fusion that occurs within the Dialogue Controller.

THE FUSION MECHANISM

Within the Dialogue Controller, data fusion is performed at a high level of abstraction (i.e., at the command or task level) by PAC agents. As shown in Figure 1b, every PAC agent has access to a fusion engine through its Control facet. This shared service can be viewed either as a reusable technical solution (i.e., a skeleton) or as a third dimension of the architectural model.

Fusion is performed on the presentation objects received from the PTC. These objects obey to a uniform format: the melting pot. As shown in Figures 1b and 2, a melting pot is a 2-D structure. On the vertical axis, the "structural parts" model the composition of the task objects that the Dialogue Controller is able to handle. For example, request slots such as destination and time departure, are the structural parts of the task objects that the Dialogue Controller handles for MATIS. Events generated by user's actions are abstracted through the LLIC and PTC and mapped onto the structural parts of the melting pots. In addition, LLIC events are time-stamped. An event mapped with the structural parts of a melting pot defines a new column along the temporal axis.

The structural decomposition of a melting pot is described in a declarative way outside the engine. By so doing, the fusion mechanism is domain independent: structures that rely on the domain are not "code-wired". They are used as parameters for the fusion engine. Figure 2 illustrates the effect of a fusion on two melting pots: at time t_i, a MATIS user has uttered the sentence "Flights from Boston to this city" while selecting "Denver" with the mouse at t_{i+1}. The melting pot on the bottom left of Figure 2 is generated by the mouse selection action. The speech act triggers the creation of the bottom right melting pot: the slot "from" is filled in with the value "Boston". The fusion engine combines the two melting pots into a new one where the departure and destination locations are both specified.

The criteria for triggering fusion are threefold: the complementarity of melting pots, time, and context. When triggered, the engine attempts three types of fusion in the following order: microtemporal fusion, macrotemporal fusion, and contextual fusion.

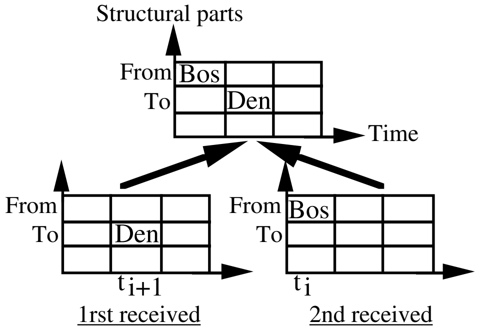


Figure 2: Fusion of two melting pots.

- *Microtemporal fusion* is used to combine related informational units produced in parallel or in a pseudo-parallel manner. It is performed when the structural parts of input melting pots are complementary and when these melting pots are close in time: their time interval overlaps. Figure 3 shows one possible configuration of temporal relationships between two melting pots m_i and $m_{i'}$ candidates for microtemporal fusion.

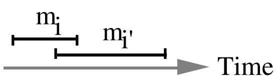


Figure 3: Two melting pots candidates for microtemporal fusion due to the intersection of their time intervals.

- *Macrotemporal fusion* is used to combine related informational units produced sequentially or possibly in parallel by the user but processed sequentially by the system, or even delayed by the system due to the lack of processing resources (e.g., processing speech input requires more computing resources than interpreting mouse clicks). Macrotemporal fusion is performed when the structural parts of input melting pots are complementary and when the time intervals of these melting pots do not overlap but belong to the same temporal window. Figure 4 illustrates the temporal constraints between two melting pots candidates for macrotemporal fusion.

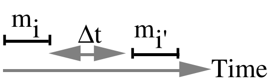


Figure 4: Two melting pots candidates for macrotemporal fusion.

- *Contextual fusion* is used to combine related informational units produced without attention for temporal constraints. For example, a MATIS user may specify the destination, then give a call, and resume the task a couple of minutes later. Contextual fusion is driven by the current active context. In MATIS, the current context corresponds to the current active request. Contextual fusion combines a new input melting pot m with the melting pots M of the current context if the content of m is complementary with one of the melting pots M. (Melting pots in M are ordered according to their recency.)

Having presented the driving principles of the fusion mechanism, we now focus on the technical details.

INSIDE THE FUSION MECHANISM

Our fusion algorithm has been implemented in C and embedded in a PAC-Amodeus architecture. We first introduce the metrics associated with each melting pot, then describe the three types of fusion in detail. Finally, we present the management of the set of melting pots and their transfer within the hierarchy of PAC agents.

Metrics for a Melting Pot

Figure 5 portrays the metrics that describe a melting pot m_i:

m_i=(p1, p2,... , pj,..., pn): m_i is comprised of n structures p1, p2, ...pn.
$info_{ij}$: piece of information stored in the structural part p_j of m_i.
$Tinfo_{ij}$: time-stamp of $info_{ij}$.
$Tmax_i$: time-stamp of the most recent piece of information stored in m_i.
$Tmin_i$: time-stamp of the oldest piece of information stored in m_i.
$Temp\_win_i$: duration of the temporal window for m_i.
Δt: Remaining life span for m_i.

A melting pot encapsulates a set of structural parts p_1, p_2,...p_n. The content of a structural part is a piece of information that is time-stamped. Time stamps are defined by the LLIC when processing user's events. The engine computes the temporal boundaries (Tmax and Tmin) of a melting pot from the time stamps of its pieces of information.

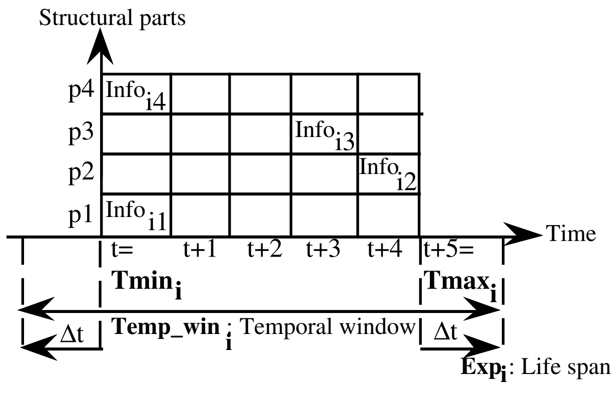


Figure 5: Metrics used to define a melting pot m_i.

So for $m_i=(p_1, p_2,...p_n)$, $Tmax_i=Max(Tinfo_{ij})$ and $Tmin_i=Min(Tinfo_{ij})$.

The temporal window of a melting pot defines the temporal proximity (+/- Δt) of two adjacent melting pots: for $mi=(p1, p_2,...p_n)$, $Temp\_win_i=[Tmin_i-\Delta t, Tmax_i+\Delta t]$. Temporal windows are used to trigger macrotemporal fusion.

The last metrics used to manage a melting pot is the notion of life span, Exp_i: $Exp_i=Tmax_i+\Delta t=Max(Tinfo_{ij})+\Delta t$. This notion is useful for removing a melting pot from the set of candidates for fusion.

The Mechanism

The fusion mechanism is driven by a set of rules.

Rule 1 deals with *microtemporal fusion*. Because priority is given to parallelism at the user's level, microtemporal fusion is first attempted on the arrival of a new melting pot from the Presentation Techniques Component. Since it models a user's action at instant t', this melting pot is composed of one column only. Rule 1 makes explicit the occurrence of microtemporal fusion: if the content of the new melting pot is complementary with a column (col_{it}) of an existing melting pot (m_i) and if the time-stamp of this column is close enough to t' (i.e., within Δmicrot), then microtemporal fusion is performed. Microtemporal fusion may involve undoing a previous fusion. This exception case will be discussed later.

<u>Rule 1</u> Microtemporal fusion (overlap of time intervals)

Given:
- $col_{it} = (p_1, p_2,... , p_j,..., p_n)$: one column at time t of an existing melting pot m_i.
- $col_{i't'} = (p'1, p'2, ..., p'j, ..., p'n)$ a one column melting pot $m_{i'}$ produced at time t'
- $i \neq i'$

col_{it} and $col_{i't'}$ are combined if:
- they are complementary: Complementary (col_{it} , $col_{i't'}$) is satisfied if: $\forall k \in [1..n] : \exists\ info_{ik}\ \Lambda\ \neg (\exists\ info_{i'k})$
- their time-stamps are temporally close: Close (col_{it},$col_{i't'}$) is satisfied if: $t' \in [t-\Delta microt, t+\Delta microt]$ (Intersection of time intervals)

Figure 6 illustrates the principles of microtemporal fusion with the example discussed earlier in Figure 2: The user utters the sentence "flights from Boston" at time t_i while selecting "Denver" with the mouse at time t_{i+1}. The melting pot m_i is produced as a result of the selection with its "t_{i+1}" column filled in. Later on, a new melting pot $m_{i'}$ arrives at the Dialogue controller resulting from the speech act. Column t_i of $m_{i'}$ is filled in with the information abstracted from the speech act. m_i and $m_{i'}$ are complementary (their content correspond to distinct structural parts). In addition, the time stamps of the two columns concerned in m_i and $m_{i'}$ are within Δmicrot (we suppose that Δmicrot is equal to 1 temporal unit). Thus microtemporal fusion can be performed.

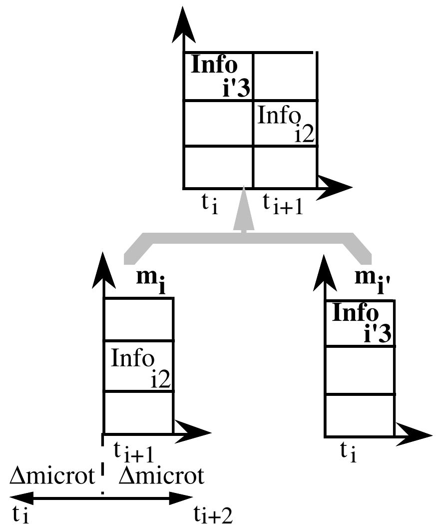


Figure 6: An example of microtemporal fusion.

One particular phenomenon in parallelism is *redundancy* [15]. As shown by the example of Figure 7, a MATIS user may utter the sentence "Flights from Boston" ($Info_{i'1}$ = [Boston]) while selecting "Boston" with the mouse ($Info_{i1}$ = [Boston]). One of the two user's actions must be ignored. i.e., the newly arrived melting pot must be discarded. Redundancy checking is performed before microtemporal fusion is attempted. Rule 2 makes this verification process explicit.

<u>Rule 2</u> Redundancy

Given:
- $col_{it} = (p_1, p_2,... , p_j,..., p_n)$: one column at time t of an existing melting m_i
- $col_{i't'} = (p'1, p'2, ..., p'j, ..., p'n)$ one column at time t' of a new melting pot $m_{i'}$
- $i \neq i'$

col_{it} and $col_{i't'}$ are redundant if:
- they contain the same information in the same slots: Redundant (col_{it}, $col_{i't'}$) is satisfied if: $\forall k \in [1..n] : \exists\ info_{ik}\ \Lambda\ \exists\ info_{i'k}\ \Lambda\ info_{ik} = info_{i'k}$ $\Lambda\ \forall k' \in [1..n] : \neg (\exists\ info_{ik'})\ \Lambda\ \neg (\exists\ info_{i'k'})$
- their time-stamps are temporally close: Close (col_{it}, $col_{i't'}$) is satisfied if: $t' \in [t-\Delta microt, t+\Delta microt]$

Macrotemporal fusion is driven by rules similar to those used for microtemporal fusion where Δmicrot is replaced by temporal windows. Whereas time has a primary role in micro- and macro- temporal fusions, it is not involved in contextual fusion.

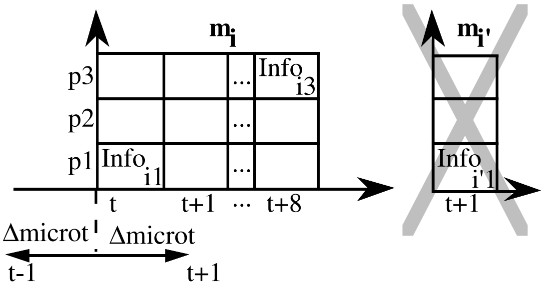


Figure 7: Redundancy: a new melting pot m_i' contains information $Info_{i'1}$ equal to $Info_{i1}$ of melting pot m_i produced nearly at the same time as m_i'.

As described in the above section, *contextual fusion* is the last step in the fusion process. The driving element for contextual fusion is the notion of context. In MATIS, contexts are in a one-to-one correspondence with requests. There is one context per request under specification and the current request denotes the current context. (The user may elaborate multiple requests in an interleaved way.) When a melting pot is complete (all of its structural parts have a value), and its life span expectancy Exp_i expires, it is removed from the set of candidates for fusion. Rule 3 expresses these conditions formally. Exp_i is used for making sure that incorrect fusions have not been performed: when a melting pot is complete, the engine keeps it for a while in the pool of candidates in case the next new melting pots trigger "undo" fusions.

<u>Rule 3</u> Conditions for removing a melting pot from the list of candidates for fusion:

Melting pot $m_i = (p_1, p_2, ..., p_j, ..., p_n)$ is removed if:

- m_i is complete: $\forall p_j \in m_i, \exists\, info_{ij}$
- and its span life is over: current date = Exp_i

Undoing erroneous fusions. Because our algorithm favors parallelism, it adopts an “eager” strategy: it does not wait for further information and therefore continuously attempts to combine input data. This approach has the advantage of providing the user with immediate feedback before the functional core is accessed. The drawback is the possible occurrence of incorrect fusions. Incorrect fusion may occur due to the different time scales required to process data specified through distinct languages and devices. As a result, the sequence of melting pots is not necessarily identical to that of the user's actions sequence. For example, in MATIS melting pots that correspond to direct manipulation expressions are built faster than those from voiced utterances. This situation will be illustrated with MATIS in the next section.

A melting pot removed from the fusion pool by the fusion engine is returned to the calling PAC agent for further processing. In the next paragraph we describe how melting pots relate to PAC agents.

The Fusion Engine and the PAC Agents

The PAC agents of the Dialogue Controller are in charge of task sequencing as well as processing the content of the melting pots. This activity is part of the abstraction and concretization processes as described in [3][14]. Abstracting involves multiple processing activities including the use of the fusion engine. When calling the engine, a PAC agent provides a melting pot as an input and receives a list of melting pots as output parameter. Depending on the current candidates in the fusion pool, the content of the input melting pot may or may not be modified by the fusion engine.

Data fusion is one aspect of abstraction. The enrichment of information is also performed by exchanging melting pots across the hierarchy of agents. There is one such hierarchy per task. The set of melting pots are partitioned according to the set of tasks. As a result, an agent hierarchy handles the melting pots that are related to the task it models. In addition the root agent of each hierarchy maintains a mapping function between the melting pots and the PAC agents interested by these melting pots. The benefit of this partitioning is that the fusion engine will not try to combine melting pots that belong to different task partitions. For example in MATIS, if the user utters "Flights from Pittsburgh to this city" while resizing a window, the two melting pots that model the users physical actions does not belong to the same set. As a result, the fusion mechanism does not attempt to combine them.

FROM MODEL TO REALITY: INTERACTING WITH MATIS

In this section we use MATIS to illustrate how PAC agents within the Dialogue Controller operate in conjunction with the fusion mechanism. Figures 8 and 9 show the message passing through the hierarchy of agents and the fusions performed in the context of the following example: the user has already specified the destination slot (i.e., Denver) as well as the departure slot (i.e., Boston) of the current request α. The result of this specification is modelled in Figure 8 as the melting pot m_1 as well as the existence of the *Request* α agent in charge of maintaining a local interaction with the user about this request. The user then utters the sentence "Flights from Pittsburgh" while selecting "TWA" using the mouse.

Because mouse clicks are processed faster than speech input, the mouse selection is first received by the Dialogue Controller through the Presentation facet of the *Tools* agent (<1> in Figure 8). The mouse click is modelled as the melting pot m_2 which contains [TWA]. The Presentation of the *Tools* agent performs a partial immediate feedback by highlighting the selection. Its Control facet calls the fusion mechanism (<2>): the new coming melting pot m2 is combined with m1 by contextual fusion. m_1, which now contains [BOS, DEN, TWA], is returned to the *Tools* agent (<3>). In turn, the *Tools* agent, which cannot perform any more processing on m1, sends m1 to its parent agent (<4>). As shown in Figure 8, the *Cement* agent which maintains the mapping between melting pots and the agents interested in these melting pots, transfers m1 to the *Request* α agent (<5>). *Request* α agent is then able to update its abstraction facet and its presentation facet (<6>): the request form on the screen (<7>) is updated accordingly with (Boston, Denver and TWA).

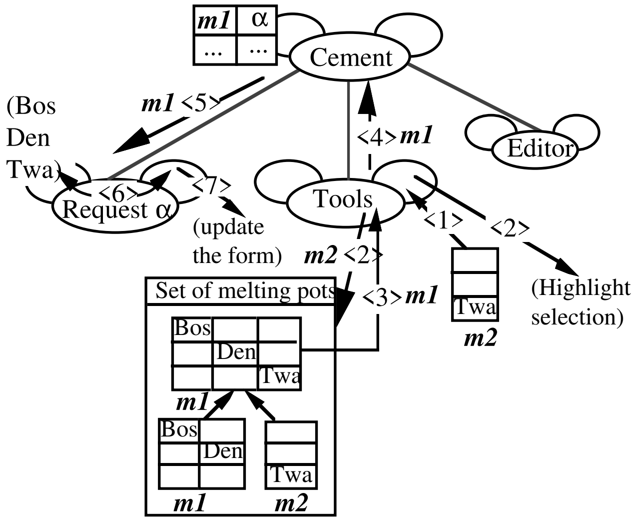


Figure 8: Interacting with MATIS: contextual fusion.

Meanwhile, melting pot m_3 which corresponds to the sentence "Flights from Pittsburgh", is received by the *Editor* agent (<1> in Figure 9). The *Editor* agent provides the user with a partial feedback by displaying the recognized sentence while calling the fusion mechanism (<2>). The current set of candidates for fusion is now {m1, m2, m3} (according to rule 3, m1 and m2, which have not reached their life span expectancy, have not been eliminated from the pool). Because the time intervals of m_2 [TWA] and m_3 [PIT] overlap, they are combined by microtemporal fusion and m2 becomes [PIT, TWA] (rule 1 applies). The previous contextual fusion [BOS, DEN, TWA] is undone: m1 [BOS, DEN] and m2 [PIT, TWA] are returned to the *Editor* agent (<3>) and reflected back to the *Cement* agent (<4>). The *Cement* agent dynamically creates a new agent *Request* β (<5>)because the new melting pot, m_2, [TWA, PIT] has no agent associated with itself (mapping table in the abstraction part of the *Cement* agent). The Presentation facet of the *Request* β agent displays a form containing the state of the new current request (<7>). From now on, the user has elaborated two requests. When completed, the content maintained in the abstract facet of a *Request* agent is transmitted to the Interface with the Functional Core for translation into the SQL format and submitted to the data base maintained in the Functional Core.

SUMMARY AND DISCUSSION

We have presented a software architecture model, PAC-Amodeus, augmented with a fusion mechanism to support the software design of multimodal systems. The platform defined by PAC-Amodeus along with the fusion mechanism fulfills specific requirements of multimodal systems such as data fusion and parallel processing. The fusion mechanism is responsible for combining data specified by the user through different modalities (i.e., a combination of devices and interaction languages). In particular, we have shown the benefits of the symbiosis between the hierarchy of agents of the architectural model and the fusion mechanism. Based on criteria such as time and structural complementarity, the mechanism is generic and reusable. Each melting pot processed may have any number of structural parts (e.g., lines) *that can be filled independently*. Consequently, the PAC-Amodeus model along with the fusion mechanism define a reusable platform for implementing multimodal systems. This property is a distinct advantage over most current tools which are limited in scope.

In a future work, we plan to enrich our fusion mechanism with a confidence factor attached to every slot of a melting pot. The notion of confidence factor provides a simple mechanism for modelling uncertainty and can be usefully exploited for solving ambiguities in deictic expressions. Figure 10 shows the relevance of confidence factors using the example of Figure 2.

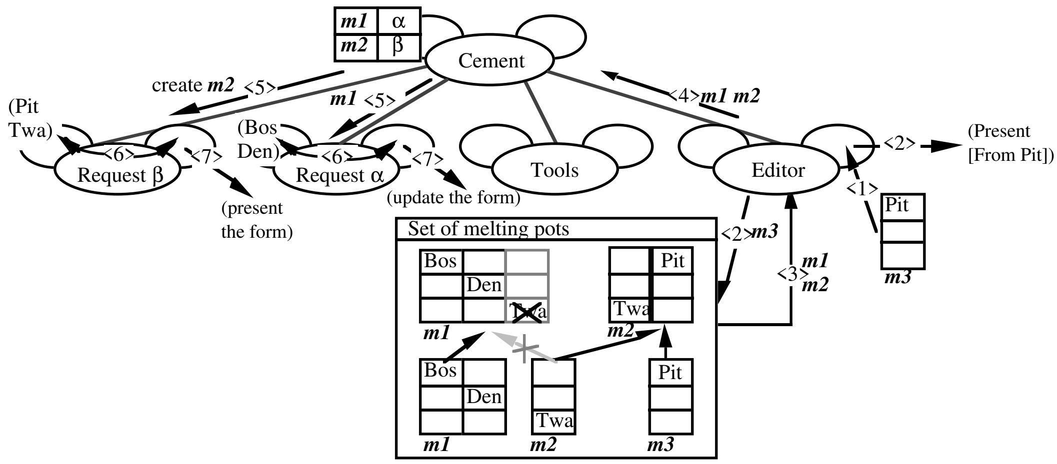


Figure 9: Interacting within MATIS: undoing fusion due to microtemporal fusion.

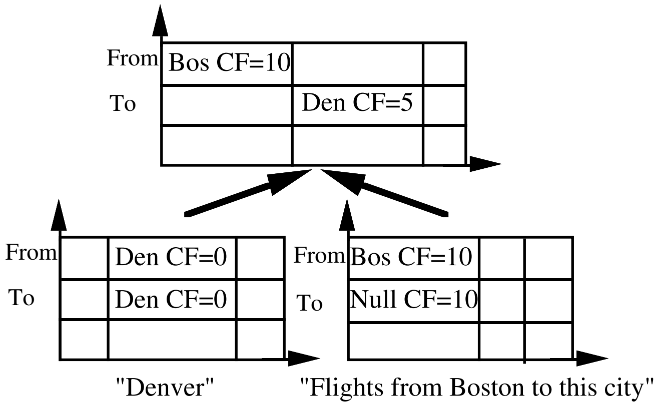


Figure 10: Confidence factor (CF $\in$ [1,10]): Example of a deictic expression (see figure 2).

Moreover Δt and Δmicro have been tuned experimentally. One can improve the setting of those parameters by letting the system compute the appropriate values depending on performance of the platform as well as on the behavior of the user. In addition, we will also examine systems that support multiple output modalities. This may lead to the development of a "fission" mechanism as introduced in MSM [3] and suggested in [17].

ACKNOWLEDGEMENTS

This work has been partly supported by project ESPRIT BR 7040 Amodeus II. Many thanks to G. Serghiou for reviewing the paper.

REFERENCES

1. Coutaz, J. PAC, an Object Oriented Model for Dialog Design, in Proc. Interact'87 (Stuttgart, 1-4 Sept. 1987), North Holland, pp. 431-436.
2. Coutaz, J., Balbo, S. Applications: A Dimension Space for User Interface Management Systems, in Proc. CHI’91 Human Factors in Computing Systems (New Orleans, April 27-May 2, 1991), ACM Press, pp. 27-32.
3. Coutaz, J., Nigay, L., Salber, D. The MSM framework: A Design Space for Multi-Sensori-Motor Systems, in Proc. EWHCI’93 (Moscow, Aug. 3-7, 1993), Springer Verlag (Lecture notes in Computer Science, Vol. 753, 1993, pp. 231-241).
4. Coutaz, J. Architectural design for user interfaces, Encyclopedia of Software Engineering, Volume 1 A-N, Wiley-Interscience, 1994, pp. 38-49.
5. Garfinkel, S., Mahoney, M. NeXTSTEP Programming, Springer-Verlag, 1993, 631 pages.
6. Hemjslev, L., Structural Analysis of language, Studia Phonetica, Vol. 1, pp. 69-78.
7. Henry, T.R., Hudson, S.E., Newell, G.L. Integrating Gesture and Snapping into a User Interface Toolkit, in Proc. Symposium on User Interface Software and Technology (Oct. 1990), ACM Press, pp. 112-121.
8. Hill, R.D. The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications, in Proc. CHI'92 Human Factors in Computing Systems (Monterey, May 3-7, 1992), ACM Press, pp. 335-342.
9. Little, T.D.C, Ghafoor, A., Chen, C.Y.R., Chang, C.S., Berra, P.B. Multimedia Synchronization, IEEE Data Engineering Bulletin, Vol. 14, 3 (Sept. 1991), pp. 26-35.
10. Lunati, J-M and Rudnicky, A. Spoken Language interfaces: The OM system, in Proc. CHI'91 Human Factors in Computing Systems (New Orleans, April 27-May 2, 1991), ACM Press, pp. 453-454.
11. Myers, B. Challenges of HCI Design and Implementation, Interactions, Vol. 1 No. 1 (Jan. 1994), pp. 73-83.
12. Nigay, L. A Case Study of Software Architecture for Multimodal Interactive System: a voice-enabled graphic notebook, TR LGI-IMAG (Oct. 1991), 31 pages.
13. Nigay, L., Coutaz, J., Salber, D. MATIS: A multimodal airline travel information system, SM/WP10, ESPRIT BRA 7040 Amodeus, (Feb. 1993).
14. Nigay, L., Coutaz, J. A design space for multimodal interfaces: concurrent processing and data fusion, in Proc. INTERCHI’93 Human Factors in Computing Systems (Amsterdam, April 24-29, 1993), ACM Press, pp. 172-178.
15. Nigay, L. Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales, PhD dissertation, University of Grenoble, France (Jan. 1994), 315 pages.
16. Norman, D.A. Cognitive Engineering, In User Centered System Design, New Perspectives on Computer Interaction, Hillsdale: Lawrence Erlbaum Associates, 1986, pp. 31-61.
17. Sutcliffe, A., Faraday, P. Designing Presentation in Multimedia Interfaces, in Proc. CHI'94 Human Factors in Computing Systems (Boston, April 24-28, 1994), ACM Press, pp. 92-98.
18. The UIMS Tool Developers Workshop, A Metamodel for the Runtime Architecture of an Interactive System, SIGCHI Bulletin, 24, 1 (Jan. 1992), pp. 32-37.